SEARCHING AND RECONSTRUCTION: ALGORITHMS WITH TOPOLOGICAL DESCRIPTORS

by

Samuel Adam Micka

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2020

DEDICATION

For my friends, family, professors, and everyone who has supported my pursuit for knowledge. For Mallory, thank you for your love and endless patience.

ACKNOWLEDGEMENTS

Acknowledgments I would like to thank my parents, John and Kathy, who have provided constant reassurance and support. I thank my brother, Jack, for always being a source of optimism and comedy. I would like to thank my future wife Mallory, for her endless patience, for her feedback on presentations and writing, for her love and support during trying times, and for her positive attitude. I would like to thank my advisor, Brittany Terese Fasy for dedicating years of time and effort into my education. I would like to thank my committee: Brittany Terese Fasy, David L. Millman, Carola Wenk, Binhai Zhu, and Mike P. Wittie for all the help and feedback over the years. I would like to thank my collaborators Anna Schenfisch, Lucia Williams, Jordan Schupbach, Robin Lynne Belton, Rostik Mertz, Daniel Salinas, Xiaozhou He, Zhihui Liu, Stacey Hancock, Sally Slipher, Brad McCoy, Harish Thiruvalluvan, Ben Holmgren, Sean Yaw, Alan Cleary, Allison Theobold, and Ryan Hansen for all the help and advice. I would like to thank Clemente Izurieta and Sharlyn Gunderson-Izurieta for welcoming me to Bozeman during my REU many years ago.

I would also like to thank my funding sources from Montana State University (MSU): the MSU College of Engineering Benjamin Fellowship (2015–2016) and the MSU Graduate School Dissertation Completion Award. Additionally, this material is based upon work supported by the National Science Foundation (NSF) under Grant No. CCF 1618605 and Grant No. DRL 1657553. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

LIST OF ALGORITHMS

ABSTRACT

Topological data analysis and, more specifically, persistent homology have received significant attention as a method of describing the shape of complex data. Persistent homology measures the persistence (i.e., "relative size") of topological features such as connected components, holes, voids, etc. as a space is filtered. The persistence is often plotted in what is referred to as a persistence diagram. Persistence diagrams encode both topological and geometric information about shapes. Moreover, certain parameterized sets of persistence diagrams are sufficient for representing particular classes of shapes. In other words, a set of persistence diagrams can be substituted for the shape. Shape representation using persistence diagrams has shown promise in several learning and classification tasks on shape data. However, choosing a sufficient parameterized set of persistence diagrams is challenging. Previous solutions utilize exhaustive sampling approaches based on assumptions on the geometry of the shape and algebraic results. We consider the inverse problem of "reconstructing" the shape, using (augmented) persistence diagrams. Developing algorithms for reconstruction provides an alternative method of finding descriptors for representation that is optimized to minimize the number of diagrams and time complexity. In this work, we provide deterministic algorithms for reconstructing simplicial complexes in $\mathbb{R}^d$ and discuss challenges in reconstruction using other topological descriptors.

Shape reconstruction, and several other areas of research utilizing the persistence diagram, are predisposed to generating large numbers of diagrams. As such, developing efficient methods for searching in the space of persistence diagrams is also of great importance. We consider the bottleneck distance in the space of persistence diagrams. The bottleneck distance is used often in practice for comparing one diagram to another. However, the cost of computing the bottleneck distance can grow prohibitive for large sets of diagrams using a brute-force approach. We offer a data structure and algorithm for identifying the approximate nearest neighbor (and approximate k-nearest neighbors) in the space of persistence diagrams in less time than the brute-force approach for large sets of diagrams.

CHAPTER ONE

INTRODUCTION

As more data is digitalized, identifying and comparing shapes becomes increasingly common. However, shape comparison is a notoriously complex problem, drawing attention from several different fields; some approaches are summarized in the surveys [3, 22, 97]. Recently, topological data analysis (TDA), a field that focuses on studying the shape of data, has introduced several new methods for shape comparison. TDA has proved itself effective at identifying topological similarities and differences in data, as demonstrated in projects focusing on research in cancer [39, 74, 88], neuroscience [59], biology [93, 100], and road networks [2]. Moreover, tools in TDA are benefiting from techniques in other fields, such as statistics [16, 19, 32, 53, 72, 73, 81, 99].

A common theme in applications of TDA is using topological descriptors for quantifying differences and similarities between data. Topology investigates the "shape" of data and one important field in topology, used for algebraically describing shape, is homology groups. Homology groups associate abelian groups to "holes" in data, i.e., *topological features* that we colloquially call connected components, cycles, and n-dimensional voids. Persistent homology summarizes complex data using a filtration to describe the "size" of topological features. In this work, we focus on lower-star filtrations. For data in $\mathbb{R}^d$, the lower-star filtration tracks topological features based on when they first "appear" (are born) and "disappear" (die), referred to as the *persistence* of the feature, in a particular direction. These features are plotted in the extended plane in what is referred to as a *persistence diagram* (PD). The PD is a finite multiset of birth-death pairs along with points on the diagonal ($y = x$ line) with infinite multiplicity for purposes related

to computing distances between diagrams. Algorithms for computing PDs, such as the one described in [48, §VII], may generate pairings associated with points that have zero persistence (i.e., points that lie on the diagonal). We call the diagram an *augmented* persistence diagram (APD) when we utilize these "on-diagonal"points, see Chapter 2 for a more comprehensive overview.

Using PDs for shape representation is of particular interest, since the diagrams encode important topological information in addition to geometric properties (such as proximity or "heights", depending on the filtration). As such, one increasingly popular research objective is to *represent* a shape using topological descriptors that are sufficient for differentiating between shapes [39, 40, 58, 64, 79, 100]. We refer to the problem of finding these descriptors as the problem of *shape representation*. Turner et al. showed that a transform, mapping a simplicial complex to a family of PDs (or Euler characteristic curves (ECC)) parameterized by height filtrations, serves as a sufficient representation for simplicial complexes in $\mathbb{R}^2$ and $\mathbb{R}^3$ [100]. In other words, two simplicial complexes have identical transforms if only if they are are the same simplicial complex, i.e., the parametrized sets of diagrams are *sufficient* for representing the shapes.

Ghrist et al. and Curry et al., independently, showed that a generalization of the transform to $\mathbb{R}^d$ is invertible (also proving sufficiency) [40, 58]. Additionally, Curry et al. showed that a finite subset of PDs (and ECCs) can be chosen to infer the entire transform for simplicial complexes in $\mathbb{R}^d$ [40]. Both results build on the idea of constructible functions and the inversion formula of Schapira [94]. Betthauser proved similar statements for cubical complexes [14]. Maria et al. also recently developed intrinsic topological transforms that encode the intrinsic geometry of shapes and ignore their embedding [79]. These results prove that we can represent a shape using a finite set of topological descriptors. Moreover, these approaches are being applied to important research problems such as predicting clinical outcomes of Glioblastoma multiforme [39] and developing shape

spaces [64]. However, there is still room to explore techniques for choosing a finite set of descriptors that is reasonable in size and maintains the injectivity properties, and associated discriminative properties, proven by [40, 58, 100].

1.1 Research Questions

We are interested in a sub-problem related to the invertability of the transform that may offer insight into which directions to choose. When the transform was defined, Turner et al. proved injectivity by *reconstructing* the shape with a constructive argument using an infinite set of PDs [100, Theorem 3.1], motivating an interesting research question:

Research Question 1 (APD Shape Reconstruction)**.** *Can we develop efficient algorithms to* reconstruct *simplicial complexes embedded in* $\mathbb{R}^d$ *using APDs?*

The importance of developing algorithms for reconstruction is realized by the observation that a set of descriptors used to reconstruct a shape also differentiates the shape from others and, hence, represents the shape. As such, we can work towards minimizing the set of descriptors we generate for shape representation by developing reconstruction algorithms for computing the inverse and using the resulting diagrams. While Curry et al. developed a finite representation of the invertible transform for simplicial complexes [40], algorithms for reconstruction are not readily available.

In 2018, we developed an algorithm, generating a quadratic number of APDs, for reconstructing plane graphs in $\mathbb{R}^2$ [11] that was motivated by the construction in the proof of [100, Theorem 3.1]. We extended the results to embedded graphs in $\mathbb{R}^d$ and, even more recently, we provided an algorithm for reconstructing simplicial complexes in $\mathbb{R}^d$ [12, 56], which is the focus of Chapter 3. While some of our algorithms implement a brute force approach for reconstructing simplices, we have also developed more efficient methods that utilize information about the shape gathered throughout the reconstruction process.

For example, we developed an algorithm for reconstructing one-simplices that is output sensitive, i.e., we can use information about existing one-simplices to optimize the number of diagrams we generate [12]. We explore our results addressing Research Question 1 in Sections 3.2-3.5.

While we focus our attention on the APD for reconstruction, we do note that there exist several other topological descriptors. Many of these descriptors can be computed efficiently or can be smoothed and used for machine learning tasks that require an inner product structure. However, many of these descriptors encode less information about the shape than the APD, motivating a second research question.

Research Question 2 (General Descriptor Reconstruction)**.** *What challenges arise when reconstructing simplicial complexes using topological descriptors?*

The augmentation of the PD helps in encoding additional geometric information about vertices in the shape for reconstruction. For example, when an APD is generated using a height filtration, we can infer vertex heights of a shape relative to the chosen direction (see Section 2.5 for more details). However, for other descriptors, calculating the height of a vertex is only possible for a very small window of directions. Work detailing these challenges and exploring their prevalence is covered in Sections 3.6-3.5.

In Chapter 4, we shift our focus back to the PD. As more research focuses on the use of PDs for representing shapes, the desire for searching for a nearest neighbor in a large set of PDs has become more prevalent [2, 26, 43, 64, 76, 100]. The most common methods of comparing PDs to one another is by computing the bottleneck and Wasserstein between two diagrams. However, computing these distances is costly when comparing a large set of diagrams (e.g., thousands or millions). For example, querying for the nearest neighbor under the bottleneck distance in a set of n PDs, with at most m points per diagram, takes $O(nm^{1.5} \log m)$ using a state of the art approach developed by Kerber et

al. that utilizes k-d trees and geometric point matching (as opposed to combinatorial) [68]. Researchers have explored alternative methods of representing size functions, i.e., functions that track connected components throughout a continuous function (such as a height function) on a topological space [57], and extended these results to improve the speed of comparing persistence diagrams using a complex vector representation of the diagrams [44]. However, these approaches do not offer theoretical guarantees (i.e., approximation bounds) on the performance. Wang et al. provide stability results for persistence diagrams using a polynomial representation [101]. However, the approach does not offer lower-bounds on distances between the polynomials, limiting the applicability to nearest neighbor queries. Thus, we pose another research question.

Research Question 3 (Searching Persistence Diagrams)**.** *Given a set of n PDs Γ with at most m points per diagram and a query diagram Q, can we develop an efficient approach for finding an approximate nearest neighbor(s) for Q in Γ with bounds on the returned diagrams in terms of the nearest neighbor(s)?*

We explore this question further in Chapter 4, with our work from [51], and offer the first theoretical guarantees on searching for approximate nearest neighbors in the space of PDs. Specifically, we provide improved search time for the nearest neighbor and k-nearest neighbors, for large values of n, proving that the returned diagrams are a constant factor approximation, in terms of bottleneck distance, of the nearest (or k^{th}-nearest) neighbor.

CHAPTER TWO

TDA BACKGROUND

Research in TDA often focuses on tracking and quantifying the magnitude of topological features in data [2, 74, 88]. In this chapter, we introduce several definitions and develop a framework for generating various descriptors that encode topological and geometric information about data. For consistency, we follow many of Edelsbrunner's and Harer's definitions closely [48]. In Chapter 3, we consider the problem of identifying a set of topological descriptors that encode the geometric and topological properties of a "shape". Before venturing into these contributions, we make consistent our notion of "shape" that is utilized in this dissertation.

2.1 Data Representation

In this dissertation, we represent shapes as simplicial complexes.

2.1.1 Simplicial Complexes

First, we introduce geometric building blocks called *simplices*, see [48, §III.1]. A k-simplex is a k-dimensional generalization of a triangle defined by $k+1$ points, where the $k+1$ points $v_0, v_1, \ldots, v_k$ are *affinely independent*, meaning that $v_1 - v_0, v_2 - v_0, \ldots v_k - v_0$ are all linearly independent.

Definition 1 (Geometric Simplex)**.** *Let* $v_0, v_1, \ldots, v_k \in R^d$ *be* $k+1$ *affinely independent points. A* k-simplex *is the convex hull of the points* $v_0, v_1, \ldots v_k$.

Affine independence is necessary to avoid degeneracies such as three collinear points points defining a two-simplex (i.e., a flat triangle). We use simplices as components for constructing shapes by carefully linking them with no improper intersections. For example,

a one-simplex σ is composed of two zero-simplices τ_1 and τ_2. We say that τ_1 and τ_2 are *faces* of σ, written $\tau_1 \preceq \sigma$ and $\tau_2 \preceq \sigma$. In fact, for any k-simplex σ, if $\tau \preceq \sigma$, then τ is a face of σ.

Definition 2 (Geometric Simplicial Complex)**.** *A simplicial complex $\mathbb{K} \subset \mathbb{R}^d$ is a finite set of simplices with the following properties:*

1. *If $\sigma \in \mathbb{K}$ and $\tau \preceq \sigma$, then $\tau \in \mathbb{K}$.*

2. *If $\sigma, \sigma' \in \mathbb{K}$ and $\sigma \cap \sigma' \neq \emptyset$, then $\sigma \cap \sigma' \in \mathbb{K}$.*

While *abstract* simplicial complexes are also well-defined without an embedding (see [48, §III.1]), we focus our attention on the geometric form. If $\mathbb{K} \subset \mathbb{R}^d$ is a simplical complex, then we let $\mathbb{K}_i$ denote the i-skeleton of $\mathbb{K}$. For example, $\mathbb{K}_0$ are the zero-simplices (vertices), $\mathbb{K}_1$ are the one-simplices (edges), etc.. Moreover, we write $n_i = |\mathbb{K}_i|$ to denote the number of i-simplices. If $\sigma \in \mathbb{K}$ and $\tau \preceq \sigma$, then τ is a *face* of σ and σ is a *coface* of τ. Common examples of simplicial complexes utilized in practice include triangle meshes (see [96] for extensive examples and discussion), embedded graphs, and point clouds. Furthermore, efficient data structures exist for for storing (and simplifying) simplicial complexes [9, 41]. De Floriani and Hui offer insight into the different classes of data structures for simplicial complexes [42]. The availability and effectiveness of data structures for simplicial complexes make them an attractive choice for representing complex data.

2.1.2 Graphs

Throughout this work, we focus much of our attention on graphs. A graph is a simplicial complex composed only of zero-simplices and one-simplices; we denote a graph as $G = (V, E)$, where V and E are the zero- and one-simplices, respectively. When $G \subset \mathbb{R}^d$, we say that G is an *embedded graph*. When $d = 2$ we refer to an

embedded graph G as a *plane graph*. Note that plane graphs are planar graphs with a specified embedding.

2.2 Simplicial Homology

Simplicial homology groups associate algebraic structure with connected components, cycles, and higher-dimensional voids. We follow the definitions of homology presented in [48, §VII.1] but note that there are several well-established texts on the material, we recommend the following for further reading [48,62,87]. To define homology groups, we first define a method of relating the i-simplices to their $(i-1)$-dimensional faces. Defining this relationship enables us to talk about "boundaries" and "cycles". The linear combinations of i-simplices of $\mathbb{K}$ form an *i-chain*

$$C^i = \left\{ \sum_{\sigma_j \in \mathbb{K}_i} a_j \sigma_j \mid a_j \in \mathbb{Z}_2 \right\}.$$

For the sake of simplicity, we only consider coefficients in $\mathbb{Z}_2$, denoting the explicit inclusion or exclusion of some σ_j in the i-chain. The set of i-chains C_i forms an abelian group, called a *chain group*, with the operation of addition modulo two. Then, the *boundary* of a i-simplex is

$$\partial_i \sigma = \sum_{j=0}^{i} [u_0, \ldots, \hat{u_j}, \ldots, u_i],$$

where $\hat{u_j}$ denotes the exclusion of the simplex u_j [48, §IV.1]. By excluding a single zero-simplex at a time in the summation, we are summing the $(i-1)$-dimensional simplices that compose σ. For example, consider the two simplex τ with zero simplices v_0, v_1, v_2. Following the definition of a boundary, we have $\partial_2 \sigma = [v_1, v_2] + [v_0, v_2] + [v_0, v_1]$, which are exactly the edges of τ. Since the set of i-chains with modulo two addition is an abelian group, we can also consider the boundary of i-chain (i.e., a sum of simplices) ϕ is the sum

of the boundaries of the faces of ϕ,

$$\partial_i \phi = \sum_{\sigma_j \in \phi} a_j \partial_i \sigma_j.$$

The boundaries of i-chains form homomorphisms and we can define maps from C^i to C^{i-1}, relating chains to their respective boundaries:

$$\ldots \xrightarrow{\partial_{i+1}} C^i \xrightarrow{\partial_i} C^{i-1} \xrightarrow{\partial_{i-1}} \ldots.$$

Now, we return to the task at hand, identifying "cycles". Let ϕ be an i-chain, we call ϕ and *i-cycle* if $\partial\phi = 0$, i.e., the $(i-1)$-simplices bounding ϕ sum to zero with $\mathbb{Z}_2$ coefficients. We note that, intuitively, these are cycles because there are two copies of each $(i-1)$-bounding simplex which sum to zero in modulo two arithmetic. Again, consider the boundary of the two-simplex τ with vertices v_0, v_1, v_2; we recall from above that the boundary map $\partial_2\tau = [v_1, v_2] + [v_0, v_2] + [v_0, v_1]$. If we evaluate the sum further, we find that there are two copies of each vertex, making the sum zero in modulo two arithmetic.

We see that i-cycles lie in the kernel of ∂_i. We follow notation introduced in [48, Page 80] and refer to this group of i-cycles as Z_i. Next, we want to differentiate between cycles and "boundaries", i.e., cycles that bound a higher dimensional face, such as the example with τ above. Then, an i-boundary is a chain of i-simplices bounding an $(i+1)$ simplex in $\mathbb{K}$, i.e., elements of the image of ∂_{i+1}. Again, we follow the notation of [48, Page 80] and denote the i-boundaries as B_i. We note that B_i is a normal subgroup of Z_i, which is a subgroup of C_i. The quotient group

$$H_i = Z_i / B_i$$

is the ith *homology group*. In words, this group represents the set of i-cycles that do

not bound an $(i+1)$-simplex. Furthermore, H_i is a set of homology classes containing homologous i-cycles with arbitrarily chosen representatives.

2.3 Filtrations

In order to measure the magnitude of the features described in the homology groups for a simplicial complex $\mathbb{K} \subset \mathbb{R}^d$, we formalize a method for considering the components of $\mathbb{K}$ piece by piece and tracking topological and geometric changes as they occur. In the context of this dissertation, filtrations provide a means of assigning an ordering to different simplices. This ordering is critical for computing topological descriptors such as persistence diagrams and Euler Characteristic Curves (described in Section 2.4 and Section 2.6, respectively). Let

$$f : \mathbb{K} \to \mathbb{R}$$

be a function mapping simplices in $\mathbb{K}$ to the reals such that if $\sigma \preceq \tau \in \mathbb{K}$, then $f(\sigma) \leq f(\tau)$. For $i \in \mathbb{R}$, we also define an inverse $f^{-1} : \mathbb{R} \to \mathbb{K}$ with $\mathbb{K}^i = f^{-1}(\infty, i]$ composed of simplices $\sigma \in \mathbb{K}$ with $f(\sigma) \leq i$. Notice that $\mathbb{K}^i$ is also a simplicial complex (referred to as an *subcomplex*) because any simplex σ included in $\mathbb{K}^i$ has the property that $f(\tau) \leq i$ for all $\tau \preceq \sigma$. Then, f induces a sequence of subcomplexes

$$\emptyset = \mathbb{K}^0 \subseteq \mathbb{K}^1 \subseteq \ldots \subseteq \mathbb{K}^n = \mathbb{K},$$

referred to as a *filtration* and we may refer to f as a *filter function*.

One common filtration that we employ frequently in our work is the *lower-star filtration*. We note that this version of the lower-star filtration is presented as described in our previous work [11] but the definition we consider originates in [38, 48]. Let $\mathbb{S}^{d-1}$ be the d-dimensional unit sphere centered at the origin and let $s \in \mathbb{S}^{d-1}$ be a unit vector

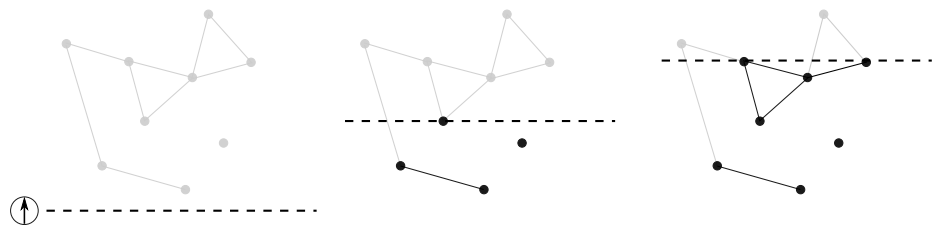


Figure 2.1: Example demonstrating the intuition behind a lower-star filtration on a simplicial complex in direction $(0, 1)$.

intersecting $\mathbb{S}^{d-1}$; we define the lower-star filtration with respect to direction s in two steps. First, we let $\mathbb{H}_s : \mathbb{K} \to \mathbb{R}$ be defined for a simplex $\sigma \subseteq \mathbb{K}$ by $\mathbb{H}_s(\sigma) = \max_{v \in \sigma} v \cdot s$, where $x \cdot y$ is the inner (dot) product and measures height in the direction of unit vector y where both x and y are treated as vectors in $\mathbb{R}^d$. Then, for each $j \in \mathbb{R}$, the subcomplex $\mathbb{K}^j := \mathbb{H}_s^{-1}((-\infty, j])$ is composed of all simplices that lie entirely below or at the height j, with respect to the direction s. Notice $\mathbb{K}^i \subseteq \mathbb{K}^j$ for all $i \leq j$ and $\mathbb{K}^i = \mathbb{K}^j$ if no vertex has a height in the interval $[i, j]$. The sequence of all such subcomplexes, indexed by $\mathbb{R}$, is the lower-star filtration, an example is provided in Figure 2.1. However we note, that with a finite simplicial complex there are a finite number of subcomplexes which include new simplices, as such, the number of subcomplexes in $\mathbb{H}_s$ for which we need to store is finite. We also simplify notation and simply refer to the lower-star filtration as $\mathbb{H}_s$ when the simplicial complex is clear from context. Since we limit our domain to simplicial complexes, we may also refer to this filtration as a *height filtration*.

2.4 Persistence Diagrams

Persistent homology is a method of quantifying the magnitude of topological features in a filtered space by tracking them in subcomplexes. We follow the definitions and descriptions of persistent homology introduced in [48, §VII.1]. Given a filtration on a topological space, the i-th persistence diagram tracks the homology classes of the i-

dimensional homology group across the different subcomplexes of the filtration. Moreover, we can consider the ith homology group of each subcomplex. The subcomplexes are connected by inclusion maps, meaning that if a simplex $\sigma \in \mathbb{K}^p$ then σ also must be in $\mathbb{K}^q$ for all $p \leq q$. As such, the inclusions in the filtration induce inclusions in the homology groups that, in turn, induce homomorphisms between the ith simplicial homology group for $\mathbb{K}^p$ and the ith simplicial homology group of $\mathbb{K}^q$ whenever $p \leq q$,

$$H_i(\mathbb{K}^0) \to H_i(\mathbb{K}^1) \to \ldots \to H_i(\mathbb{K}^n) = H_i(\mathbb{K}).$$

We denote the homomorphism from $H_i(\mathbb{K}^p)$ to $H_i(\mathbb{K}^q)$ as

$$g_i^{p,q} : H_i(\mathbb{K}^p) \to H_i(\mathbb{K}^q),$$

and can write

$$H_i(\mathbb{K}^0) \xrightarrow{g_i^{0,1}} H_i(\mathbb{K}^1) \xrightarrow{g_i^{1,2}} \ldots \xrightarrow{g_i^{n-1,n}} H_i(\mathbb{K}^n) = H_i(\mathbb{K}).$$

In accordance with [48, Page 151], the ith *persistent homology groups* are the images of the homomorphisms $H_i^{p,q} = \text{im} g_i^{p,q}$. We also define the ith *persistent Betti number* $\beta_i^{p,q}$ as the rank of the simplicial homology group $H_i^{p,q}$. Intuitively, the ith Betti number "counts" the number of i-cycles that persist between $\mathbb{K}^p$ and $\mathbb{K}^q$.

We utilize the concepts of persistent homology groups and persistent Betti numbers to track homology classes between different subcomplexes in a filtration. The first index for a homology class appears is referred to as the *birth* time of that class. When a class in $\mathbb{K}^{j-1}$ merges with another entering $\mathbb{K}^j$ then the class *dies* entering $\mathbb{K}^j$. Moreover, we see that $H_i^{p,q} = H_i(\mathbb{K}^p)$ if $p = q$. For $p < q$, the homology group $H_i^{p,q}$ contains homology classes that are alive in $\mathbb{K}^p$ and remain alive entering $\mathbb{K}^q$. When classes merge, we follow

the *Elder rule*, which requires the homology class with the later birth time to merge into the class with the earlier birth time. We note that birth or death times of a homology class may be infinite. For example, consider a lower-star filtration on a simplicial complex with a single connected component. In the zeroth homology group, we have a homology class born at the height of the first vertex and this class will persist infinitely since it will never be absorbed into another homology class due to the Elder rule. The *persistence* of a homology class is the death time of the class, minus the birth time, and, intuitively, measures the magnitude of particular homology classes.

To track the persistence of the homology classes, we plot their birth and death times as points in the extended plane $\overline{\mathbb{R}}^2$ (i.e., $\mathbb{R}^2$ along with an explicit point at ∞). Since two classes may appear and disappear in the same subcomplexes as one another, we count each point in $\overline{\mathbb{R}}^2$ with multiplicity equal to the number of classes that share the same birth and death time. To count the multiplicity of a point $(p, q) \in \overline{\mathbb{R}}^2$ representative of classes that are born in $\mathbb{K}^p$ and die entering $\mathbb{K}^q$, we count the classes that are alive in $\mathbb{K}^p$ that die entering $\mathbb{K}^q$. However, this sum may count classes that are alive strictly before $\mathbb{K}^p$ and die entering $\mathbb{K}^q$, so we subtract these classes from our sum. Formally, we follow the notation in [48, Page 152] and define, for i-dimensional persistent homology classes, the multiplicity of each point $(p, q) \in \overline{\mathbb{R}}^2$ for $p < q$ as

$$u_i^{p,q} = (\beta_i^{p,q-1} - \beta_i^{p,q}) - (\beta_i^{p-1,q-1} - \beta_i^{p-1,q}).$$

Thus, $u_i^{p,q}$ counts exactly the i-dimensional homology classes that are born in $\mathbb{K}^p$ and die entering $\mathbb{K}^q$. Moreover, we can consider these multisets of points for each value of $i \in \mathbb{Z}$.

Definition 3 (Persistence Diagram)**.** *Let f be a real-valued filter function on a simplicial complex $\mathbb{K} \subset \mathbb{R}^d$. Let Δ denote the points along the $y = x$ line. The ith persistence diagram, denoted $D_i(f) \subset \overline{\mathbb{R}}^2$, is the multiset of points (b, d) for $b < d$, each with*

multiplicity $u_i^{b,d}$ *along with points on the diagonal* Δ *with with infinite multiplicity.*

Next, we describe one method of measuring distances between persistence diagrams. Given persistence diagrams of arbitrary dimension $D_\cdot(f)$ and $D_\cdot(f')$, we consider a bijection $\phi : D_\cdot(f) \to D_\cdot(f')$ between the points of $D_\cdot(f)$ and the points of $D_\cdot(f')$. Recall that both $D_\cdot(f)$ and $D_\cdot(f')$ contain points along the $y = x$ line with infinite multiplicity and both diagrams contain a multiset of off-diagonal points. Thus, a bijection ϕ always exists since we can always match a point $p \in D_\cdot(f)$ with $\phi(p) \in D_\cdot(f')$ where $\phi(p)$ lies on the $y = x$ line. We measure the distance between $p \in D_\cdot(f)$ and $\phi(p) \in D_\cdot(f')$ by the distance between p and $\phi(p)$, $\|p - \phi(p)\|_\infty$.

Definition 4 (Bottleneck Distance)**.** *Let* $D_\cdot(f)$ *and* $D_\cdot(f')$ *be persistence diagrams. Let the function* $\phi : D_\cdot(f) \to D_\cdot(f')$ *be a bijection between the two diagrams. The* bottleneck distance *between* $D_\cdot(f)$ *and* $D_\cdot(f')$ *is*

$$d_B(D_\cdot(f), D_\cdot(f')) = \inf_{\phi} \sup_{p \in D_\cdot(f)} \|p - \phi(p)\|_\infty,$$

taken over all bijections ϕ*.*

We include an example of the bottleneck distance between two diagrams in Figure 2.2. In practice, computing the bottleneck distance relies on computing a perfect matching between the points of two diagrams and minimizing the length of the longest edge in the matching; see [48, §VIII.4] and [68], which use results from graph matching [50,65,71,78]. Moreover, the space of persistence diagrams under the bottleneck distance is a metric space (see discussion [48, §VIII.2] and [29, 30]). However, the metric space is not complete [81].

Persistence diagrams (PD) are stable, i.e., small changes in the underlying real-valued function f are reflected as small changes in the persistence diagram [37]. Moreover, points near the diagonal often represent topological noise and points far from the diagonal

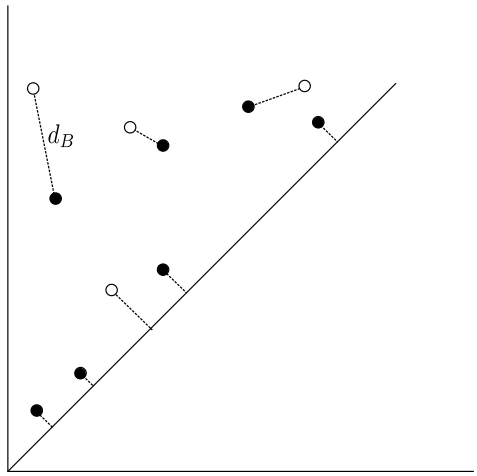


Figure 2.2: An example of a bijection between diagrams $D_\cdot(f)$ (the solid black points) and $D_\cdot(f')$ (the circles). Notice that $D_\cdot(f)$ and $D_\cdot(f')$ have a different number of off-diagonal points. However, points in either diagram may match with the diagonal.

typically represent topological features that are significant in practice; research in statistics has been conducted to separate "noise" from "signal" in persistence diagrams [53]. We provide an example of the zeroth APD on a plane graph in Figure 2.3. We also note that a persistence diagram can be represented as a set of half-open intervals (barcodes) in the form of $[b, d)$ as in [31, 103].

For a more in-depth review of persistent homology (and persistence diagrams) we refer the reader to [48, §VII]. Research into persistence continues, some recent areas of interest include improving runtimes for computing persistence [20,83,85], developing tools for computing persistence [1,10,52], and extending work into multidimensional persistence modules [23, 24, 27, 75, 95].

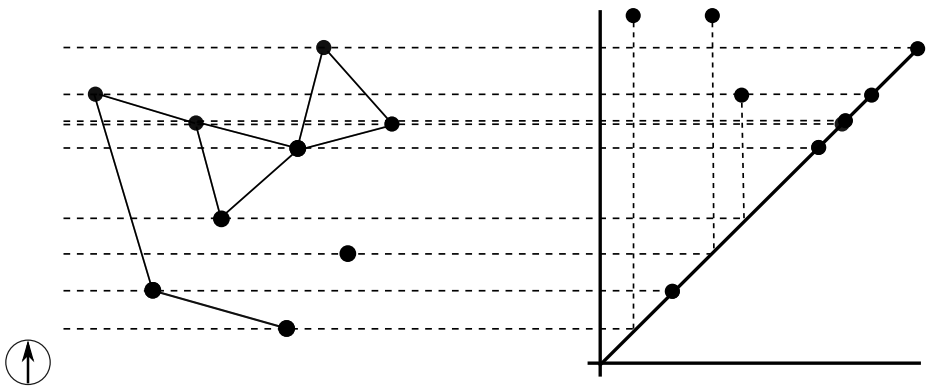


Figure 2.3: The zeroth augmented persistence diagram on a plane graph with height filtration chosen in direction $(0, 1) \in \mathbb{S}^1$. We follow connected components as they appear in the filtration (birth times, emphasized using horizontal lines at vertex heights) and are merged with other connected components (death times) in the diagram on the right side of the figure. The two points in the diagram with infinite death time are representative of the homology classes corresponding to the two different connected components.

2.5 Augmented Persistence Diagram

Next, we present a version of the persistence diagram that encodes additional geometric information in the form of "on-diagonal points". We refer to the descriptor as an *augmented* persistence diagram (APD). We defined and leveraged a version of the APD in our work in the past based on pairings from computing persistence [11, 12, 56, 80], and offer a formal definition for the APD here. First, we introduce a lemma relating simplices being included in a filtration and the change in the resulting Betti numbers. We omit the proof, but refer the reader to [48, pp. 120–121] for more details.

Lemma 1 (Adding a Simplex)**.** *Let $k \in \mathbb{Z}_{\geq 0}$. Let $L \subset K$ be simplicial complexes that differ by a single k-simplex. Then, exactly one of the following is true:*

1. $\beta_k(K) = \beta_k(L) + 1$,

2. $\beta_{k-1}(K) = \beta_{k-1}(L) - 1$.

Next, we define a filter function function that enables us to identify a correspondence between simplices and points in a persistence diagram. Let $\mathbb{K} \in \mathbb{R}^d$ be a simplicial complex with n simplices. Let $f : \mathbb{K} \to \mathbb{R}$ be a filter function on $\mathbb{K}$. Then, we define a corresponding *index filter function*:

$$id_f : \mathbb{K} \to \{1, 2, \ldots, n\},$$

such that for simplices $\tau, \sigma \in \mathbb{K}$ we have the following:

1. if $f(\sigma) < f(\tau)$, then $id_f(\sigma) < id_f(\tau)$,
2. if $f(\sigma) = f(\tau)$ and $\sigma \preceq \tau$, then $id_f(\sigma) < id_f(\tau)$.

Note that the choice of index filter is not unique and id_f is a bijection. We consider the filtration

$$\{\mathbb{K}^i := id_f^{-1}[0, i]\}_{i=1}^n.$$

Since, the subcomplex $\mathbb{K}^i \setminus \mathbb{K}^{i-1}$ is a single simplex, the number of distinct subcomplexes is equal to the total number of simplices in $\mathbb{K}$. Moreover, we have that for any two distinct simplices $\tau, \sigma \in \mathbb{K}$, $id_f(\tau) \neq id_f(\sigma)$. By Lemma 1, the filter function id_f on a simplicial complex results in all simplices being associated with birth or death event in the resulting persistence diagrams. Thus, all birth-death pairings in $D(id_f)$ have positive persistence, and the filter function id_f leads us to the following corollary.

Corollary 1 (Coordinate Bijection)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Let $f : \mathbb{K} \to \mathbb{R}$ be filter function and id_f be an index filter function corresponding total f. For $k \geq 0$, the k-simplices of $\mathbb{K}$ are in one-to-one correspondence with the finite birth coordinates of the points in $D_k(id_f)$ and the finite death coordinates in $D_{k-1}(id_f)$*

For reconstruction, this one-to-one correspondence is useful for identifying specific simplices. As such, we introduce the augmented persistence diagram to encode this pairing for general filter functions.

Definition 5 (Augmented Persistence Diagram)**.** *Let $\overline{\mathbb{R}}^2$ denote the extended plane and f a filter function on a simplicial complex $\mathbb{K} \subset \mathbb{R}^d$. Let id_f be the index filter function defined by f. Let*

$$A = \left\{ (f(id_f^{-1}(b)), f(id_f^{-1}(d))) \mid (b,d) \in D_i(id_f), b < d \right\}$$

and

$$B = \left\{ (f(id_f^{-1}(b)), f(id_f^{-1}(d))) \mid (b,d) \in D_i(id_f), b = d \right\}.$$

The ith augmented persistence diagram is the triple

$$\overline{D}_i(f) = (A, B, \Delta),$$

where Δ is the multiset of points (x,x) with infinite multiplicity.

The first set A includes the off-diagonal points present in a persistence diagram, the second set B, includes the points with equal birth and death times, and the third set Δ is the diagonal with infinite multiplicity. A *point* in the ith APD refers to a point in the set $A \cup B$, i.e., $(b,d) \in \overline{D}_i(f)$ is read $(b,d) \in A \cup B$.

Intuitively, the the ith APD is a PD that allows points (b,d) with $b = d$ (i.e., on-diagonal points). When we illustrate the ith APD, as in Figure 2.3, we include both sets and the diagonal to clearly differentiate between off- and on-diagonal points. Moreover, thinking about the APD as a PD that allows points with zero persistence may be helpful for those familiar with PDs. We often employ the lower-star filtration $\mathbb{H}_s$, for $s \in \mathbb{S}^{d-1}$, and we write $\overline{D}_i(s)$ when the context is clear. By Lemma 1, given an unknown simplicial complex $\mathbb{K}$ and a lower-star filtration in direction $s \in \mathbb{S}^{d-1}$, we can define a one-to-one correspondence between $\mathbb{K}_0$ and the birth coordinates in $\overline{D}_0(s)$.

The definition of the APD, along with Corollary 1, lead us to the following corollary, yielding a method of counting simplices using birth-death events [56, Corollary 5].

Corollary 2 (Simplex Count)**.** *Let $i \in \mathbb{N}$, $c \in \mathbb{R}$ with $c \neq 0$, and $s \in \mathbb{S}^{d-1}$. Then, the number of i-dimensional simplices of $\mathbb{K}$ that have height c is:*

$$\left| \left\{(a,b) \in \overline{D}_{i-1}(s) \mid b = c\right\} \cup \left\{(a,b) \in \overline{D}_i(s) \mid a = c\right\} \right|.$$

We bound the size of APDs when considering specific types of simplicial complexes. For example, if G is a graph, then the maximum number of edges in G is $n_0(n_0 - 1)/2$, and so $|E| = O(n_0^2)$. In the case when G is a plane graph, $|E| = O(n_0)$ due to the planarity of G. Furthermore, an APD has at least n_0 points from the vertices in G corresponding to births in the zero-dimensional diagram. These observations yield the following corollary (originating from our work [12, Corollary 2]) on the number of points in APDs for graphs.

Corollary 3 (Size of Augmented Persistence Diagrams)**.** *Let G be an embedded graph with n_0 vertices and n_1 edges. Then, an augmented persistence diagram computed for G has $\Theta(n_0 + n_1) = O(n_0^2)$ points. When G is a plane graph, the augmented persistence diagram has $\Theta(n_0)$ points. In general, if $\mathbb{K} \subset \mathbb{R}^d$, $i \geq 1$, and $s \in \mathbb{S}^{d-1}$, then the augmented persistence diagram $\overline{D}_i(s)$ has $O(n_0^{i+1}) = O(n_0 n_{i-1})$ points.*

Moreover, we transform an APD into a PD by dropping the on-diagonal points. This transformation enables us to extend the bottleneck distance to APDs.

Definition 6 (APD Bottleneck Distance)**.** *Let (A, B, Δ) and (A', B', Δ) be augmented persistence diagrams. Let $D_1 = A \cup \Delta$ and $D_2 = A' \cup \Delta$, and observe that D_1 and D_2 are persistence diagrams. The* bottleneck distance *between (A, B, Δ) and (A', B', Δ) is defined as*

$$\begin{aligned} d_B((A, B, \Delta), (A', B', \Delta)) &= d_B(D_1, D_2) \\ &= \inf_{\phi} \sup_{p \in D_1} \|p - \phi(p)\|_\infty, \end{aligned}$$

where ϕ ranges over all bijections between D_1 and D_2.

Moreover, the bottleneck distance is a pseudometric in the space of APDs.

Corollary 4 (APD Pseudometric)**.** *The bottleneck distance in the space of APDs is a pseudometric.*

Proof. Since the bottleneck distance is a metric in the space of PDs, all of the metric properties extend to the space of APDs, except for the identity of indescernibles. For example, let $x \in \mathbb{R}$ and consider the APD $(\emptyset, (x, x), \Delta)$ with one on-diagonal point and the APD $(\emptyset, \emptyset, \Delta)$ with zero points. Then, since (x, x) has persistence zero, the bottleneck distance $d_B((\emptyset, (x, x), \Delta), (\emptyset, \emptyset, \Delta)) = 0$ even though $(\emptyset, (x, x), \Delta) \neq (\emptyset, \emptyset, \Delta)$. $\square$

2.6 Euler Characteristic Curves

The Euler characteristic is an alternating sum of the number of k-dimensional faces of simplicial complex $\mathbb{K}$ and denoted as the function $\chi : \mathbb{K} \to \mathbb{Z}$. It serves as a topological descriptor and, moreover, a homotopy invariant. In this paper, we define the Euler characteristic in the context of simplicial complexes to maintain consistency. If $\mathbb{K} \subset \mathbb{R}^d$ is a simplicial complex, then

$$\chi(\mathbb{K}) = \sum_{i=0} (-1)^i |\mathbb{K}_i|.$$

Moreover, we can track the Euler characteristic over a filtered simplicial complex. We adopt the following definition from [14].

Definition 7 (Euler Characteristic Curve)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex, and f be a filter function on $\mathbb{K}$. The Euler characteristic is the piecewise constant function*

$$\chi_f : \mathbb{R} \to \mathbb{Z},$$

where, for $i \in \mathbb{R}$, $\chi_s(i)$ is defined by

$$\chi_f : i \mapsto \chi(f^{-1}(-\infty, i]).$$

In this dissertation, the Euler characteristic curve (ECC) is often used to track the Euler characteristic at each subcomplex of a lower-star filtration $\mathbb{H}_s$ in direction $s \in \mathbb{S}^{d-1}$. Thus, when the context is clear, and we wish to consider the entire simplicial complex $\mathbb{K}$, we may simplify notation to χ_s or simplify call it the ECC in direction s. We can see an example of an ECC in Figure 2.4. Since the ECC is a piecewise constant function, we can also view the curve as a set of pairs

$$S(\chi_f) = \Big\{ ([b, d), \chi_f(b)) \mid \text{there exists } \epsilon > 0,\ \chi_f(i) = \chi_f(i + \epsilon) \text{ and any } i \in [b, d) \Big\}.$$

Intuitively, this set contains the closed-open intervals for which the Euler characteristic remains consistent in the filtration. Moreover, we assume our simplicial complexes are finite and, as a result, the set of intervals is also finite.

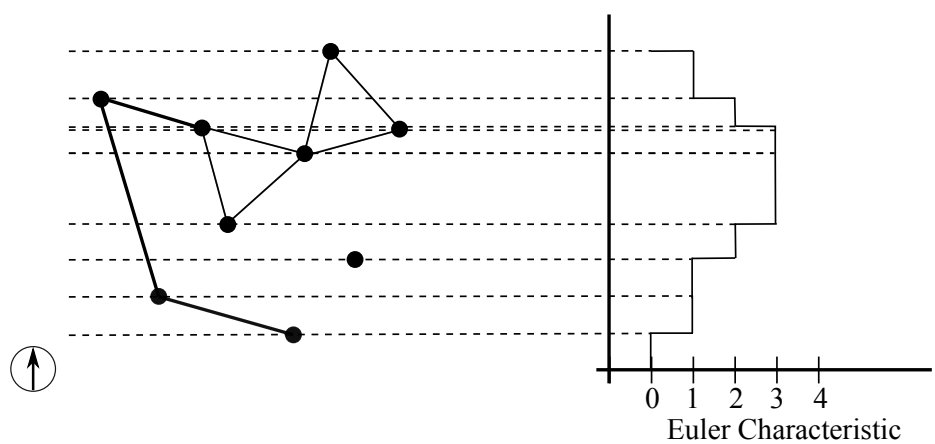


Figure 2.4: The Euler characteristic curve on a plane graph with height filtration chosen in direction $(0, 1) \in \mathbb{S}^1$. We track the Euler characteristic χ over each subcomplex in the filtration, plotting the values in the (rotated) diagram on the right.

2.7 Augmented Euler Characteristic Curve

Next, we define an augmented version of the Euler characteristic curve in the same way that we defined the APD. Let $f : \mathbb{K} \to \mathbb{R}$ be a filter function and recall the index filter $id_f : \mathbb{K} \to \mathbb{Z}_+$ from Section 2.5. Then, we consider the interval representation of an ECC, using the index filter, and generate a set of intervals representing the *augmented* Euler characteristic curve (AECC).

Definition 8 (Augmented Euler Characteristic Curve)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex, and f be a filter function on $\mathbb{K}$ and $i \in \mathbb{R}$. Let id_f be the index filter function defined by f. The Augmented Euler characteristic is the function*

$$\overline{\chi_f} : i \mapsto \chi(f^{-1}(-\infty, i]),$$

with interval representation

$$\overline{\chi_f} : \left\{ \big(\big[f(id_f^{-1}(b), f(id_f^{-1}(d)) \big), \chi_f(b) \big) \mid [b, d) \in S(\chi_{id_f}) \right\}.$$

The primary difference between the AECC and the ECC is that the AECC may contain empty intervals of the form

$$\big(\big[b, b \big), \chi_f(b) \big).$$

The inclusion of these intervals, like with the APD, enables us to establish a correspondence between the values of the intervals in the AECC and the vertices in the simplicial complex. Moreover, by Lemma 1, there is a one-to-one correspondence between the vertices and interval values of χ_{id_f}. In other words, the AECC encodes vertex heights relative to a particular filtration direction.

CHAPTER THREE

RECONSTRUCTING SIMPLICIAL COMPLEXES

Shape representation and comparison has been a longstanding research problem in the field of computer science. Many geometric approaches have been developed for the purposes of shape comparison (see surveys [3, 15, 22, 97]). Additionally, topology-based methods have received substantial attention; some focus on shape representation [11,12,15, 25, 40, 43], others focus on methods for general shape comparison [76, 91, 100], and some focus on developing comparison methods for specific problem domains [39,64]. Several of these approaches utilize the persistence diagram (or augmented versions of the persistence diagram) [11, 12, 40, 43, 64, 76, 91, 100]. In data where capturing information about connectivity and density is valued, topological features are indispensable. For example, research has been conducted using a topological transform (the smooth Euler characteristic transform) on magnetic resonance images of Glioblastoma multiforme tumors [39], the persistent homology transform on microCT scans of primate heel bones [100], and persistence on porous materials [43].

We focus our attention on the persistent homology transform, a new technique introduced by Turner et al. that is gaining popularity for shape representation [100]. The transform maps a simplicial complex (in $\mathbb{R}^d$ for $d = 2, 3$) to an uncountably infinite set of PDs. Each PD is generated by a height filtration from a direction chosen intersecting $\mathbb{S}^{d-1}$. As a result, we can think of the uncountably infinite set of PDs as being parameterized by directions in $\mathbb{S}^{d-1}$. Curry et al. proved that there is a discrete representation of the of the persistent homology transform for simplicial complexes in $\mathbb{R}^d$ [40]. Furthermore, independent, but simultaneous, research proved that many topological transforms are invertible on simplicial complexes [40, 58] and on cubical complexes [14]. The parameterized set

of persistence diagrams is sufficient for representating a shape, i.e., we can differentiate between shapes using their respective parameterized sets.

However, for simplicial complexes in $\mathbb{R}^d$, current discrete representations of the persistent homology transform are dependent on the local "flatness" (i.e., embedding) of shapes [40]. Furthermore, the transforms described by Curry et al. [40] and Ghrist et al. [58], while invertible, do not offer an explicit algorithm for reconstructing the shape itself and the inversion method used by [14] is developed for voxel-based cubical complexes. Turner et al. proved the sufficiency result in $\mathbb{R}^2$ and $\mathbb{R}^3$ using a constructive argument but a formal algorithm has not yet been described for simplicial complexes in $\mathbb{R}^d$ [100]. Motivated by the construction introduced in [100, Theorem 3.1], we provide an algorithm for reconstructing embedded graphs in $\mathbb{R}^2$ (and later in $\mathbb{R}^d$) [11, 12, 56]. For a more detailed look into the state of inverse problems in TDA see the survey by Oudot and Solomon [90]. The problem of reconstruction of general interest because the resulting set of descriptors used for reconstruction can also be used to differentiate the shape from other shapes. As such, algorithms reducing descriptors used for reconstruction also reduces the number of descriptors for representing the shape.

In this chapter, we work towards developing algorithms for reconstructing simplicial complexes using finite sets of APDs. More generally, this chapter focuses on addressing Research Question 1 using APDs. We also consider the problem of reconstruction using other descriptors, offering insight into Research Question 2. The chapter is organized as follows: we provide preliminaries and necessary background in in Section 3.1, develop predicates for reconstructing simplices in Section 3.2, describe two algorithms for reconstructing vertices in $\mathbb{R}^d$ in Section 3.3, utilize predicates to identify edges in Section 3.4, and generalize these techniques to simplices in $\mathbb{R}^d$ in Section 3.5. These sections address Research Question 1. We go on to discuss challenges in reconstruction when using various descriptors in Section 3.6 and Section 3.7, addressing Research

Question 2. We also experimentally investigate the prevalence of some of these challenges in Section 3.8. Finally, we provide insights into the cost of failing to reconstruct some simplices in particular classes of simplicial complexes in Section 3.9.

3.1 Preliminaries

In what follows, we consider simplicial complexes in $\mathbb{K} \subset \mathbb{R}^d$, for $d \geq 2$. For vertices and edges, we may let $V = \mathbb{K}_0$ and $E = \mathbb{K}_1$ in some circumstances in this chapter. We denote the *degree* of a vertex $v \in \mathbb{K}_0$ as $\deg(v)$. A k-simplex $\sigma \in \mathbb{K}_k$ is uniquely identified by $k+1$ vertices, which we denote by $\mathrm{verts}(\sigma) = \{v_0, v_1, \ldots, v_k\}$. Let e_i be the ith *standard unit basis vector* in $\mathbb{R}^d$. That is, e_i is a unit vector with the value zero as all its coordinates but coordinate i and has the value one as its ith coordinate. When we consider a vertex $v \in \mathbb{R}^d$ we write the coordinates of $v = (v^{(d)}1, v^{(d)}2, \ldots, v^{(d)}d)$. We adopt our general position assumption from [56].

Assumption 1 (General Position)**.** *For any two vertices $v = (v^{(1)}, v^{(2)}, \ldots, v^{(d)})$ and $v' = (v^{(1)'}, v^{(2)'}, \ldots, v^{(d)'})$ embedded in $\mathbb{R}^d$ and for all $1 \leq i \leq d$, $v^{(i)} \neq v^{(i)'}$; also, v, v' are not equidistant from the origin. Any set of $d+1$ vertices are affinely independent. Finally, no three points are collinear in the subspace $\mathbb{R}^2 \subseteq \mathbb{R}^d$ spanned by the e_1 and e_2 standard basis vectors.*

The first assumption makes height filtrations on simplicial complexes in axis-parallel directions (i.e., e_i for $i \in \{1, 2, \ldots, d\}$) easier, since no two vertices will lie at the same height. The assumption that no two vertices are equidistant from the origin allows us to apply an optional parabolic lifting map that preserves the first property if the maximum simplex dimension in the complex is equal to d (see Definition 13). The assumption that any $d+1$ points are affinely independent ensures that we can choose initial, and subsequent, directions to isolate simplices when developing predicates for simplex reconstruction.

The final assumption provides a method of considering a subspace of $\mathbb{R}^d$ where we can efficiently reconstruct one-simplices. We note that, in practice, these general position assumptions can be realized using well-studied perturbation techniques [21, 49, 102].

Note that any set of $k+1$ vertices $X = \{v_0, v_1, \ldots, v_k\}$ satisfying Assumption 1 has the property that span$\{v_1 - v_0, v_2 - v_0, \ldots, v_k - v_0\}$ defines a k-dimensional affine subspace of $\mathbb{R}^d$, denoted $\mathrm{aff}(X)$. Given a simplex σ, we may use the notation $\mathrm{aff}(\sigma)$ to mean the affine space $\mathrm{aff}(\mathrm{verts}(\sigma))$.

To highlight the relationship between the birth-death pairs in an APD and the simplices in $\mathbb{K}$, we define a structure (from our work [12, Definition 3], we include an updated version from [56, Definition 6]) that will be used throughout the remainder of the dissertation to talk about lower-star filtrations on $\mathbb{K}$. This structure provides a method of visualizing the problem and gives geometric intuition for several proofs that follow.

Definition 9 (Filtration Hyperplane)**.** *Let $s \in \mathbb{S}^{d-1}$ be a unit vector, and let $c \in \mathbb{R}$. We define the* filtration hyperplane at height c *as the $(d-1)$-dimensional hyperplane, denoted $\ell(s, c)$, that passes through the point $cs \in \mathbb{R}^d$ and is perpendicular to s. We define the closed half-spaces above and below this hyperplane by $H^{\uparrow}(s, c)$ and $H^{\downarrow}(s, c)$, respectively. Given a finite set of vertices $V \subset \mathbb{R}^d$, let $f\colon V \to \mathbb{R}$ be the height function in direction s. The* filtration hyperplanes of V *are the set of hyperplanes*

$$\mathbb{L}(s, V) := \{\ell(s, f(v))\}_{v \in V}.$$

Figure 2.3 from Chapter 2 offers an example of filtration lines in R^2 corresponding to the points in an APD. We note that all hyperplanes in $\mathbb{L}(s, V)$ are parallel to each other and perpendicular to the direction s. The hyperplane $\ell(s, f(v))$ defines all potential locations for v. Since the births in the zero-dimensional diagram are in one-to-one correspondence with the vertices of the simplex complex $\mathbb{K}$ by Lemma 1, a single diagram suffices to

construct $\mathbb{L}(s, V)$ and can be done in $O(n_0)$ time.

However, not all descriptors provide such a descriptive relationship to the simplices in a complex (we pull the following results from our work [55]). For example, the ECC does not always generate full sets of filtration hyperplanes. We see examples of this in Figure 2.4 from Chapter 2, where some of the lines do not correspond to changes in the ECC, these lines would not be included as filtration lines. As such, we refine our definition of filtration hyperplanes (filtration lines in $\mathbb{R}^2$) for the ECC:

$$\mathbb{W}(s, V) = \{\ell(s, h) \mid \exists \epsilon_0 > 0 \; \forall \epsilon \in (0, \epsilon_0) : \chi_s(h - \epsilon) \neq \chi_s(h + \epsilon)\}.$$

This set corresponds to the subset of vertices in the vertex set V that are *witnessed* from s through the ECC χ_s. As such, we refer to these lines as *witness lines*. Some examples of situations in which vertices are not witnessed are if the vertex is included in the filtration at the same time as an edge because the vertex being added will be canceled out by the inclusion of the edge in the Euler characteristic value. Similarly, a vertex will not be witnessed if included in the filtration at the same time as two edges and a two-simplex. Furthermore, we note that a vertex v lying on a filtration line from s does not necessarily imply that v is witnessed from s, i.e., it could lie on a witness line for another vertex if they lie at the same height from s. This distinction between filtration hyperplanes and witness lines is important because vertices which are not witnessed can not be reconstructed. We realize this observation by noticing that we can not determine any coordinates for a vertex without an associated witness line from a particular direction. This observation leads us to the following lemma which proves that there exist simplicial complexes in $\mathbb{R}^2$ for which the ECC can not be used for reconstruction.

Lemma 2 (Witness Lines)**.** *There exist simplicial complexes $\mathbb{K} \subset \mathbb{R}^2$ and vertices $v \in \mathbb{K}_0$ such that the ECC does not witness v from any direction.*

Proof. Let $\mathbb{K}$ be any straight-line embedding of K_4 meeting the requirements in Assumption 1 and let each cycle bound a two-simplex (i.e., the complex has three two-simplices). $\mathbb{K}$ is a convex polygon with a single interior vertex (i.e., on the boundary) $v \in \mathbb{K}_0$. We prove that there does not exist a $s \in \mathbb{S}^1$ such that v is witnessed by χ_s. Assume, for contradiction, that there does exist a $s \in \mathbb{S}^1$ such that χ_s witnesses v. We note that v is interior and from any direction $s \in \mathbb{S}^1$, there exists a $v' \in \mathbb{K}_0$ such that $s \cdot v' < s \cdot v$.

We consider the two possible scenarios for χ_s at time $s \cdot v$: the situation in which there exists one vertex below v and the situation in which two vertices lie below v with respect to direction s. If one vertex $v' \in \mathbb{K}_0$ lies below v then so does a single edge $(v, v') \in \mathbb{K}_1$. Since $\mathbb{K}$ is a complete graph, the edge (v, v') is included in the filtration at time $s \cdot v$. The inclusion of (v, v') cancels out the inclusion of v in the Euler characteristic and χ_s does not change at time $s \cdot v$, a contradiction. Then, in the case where two vertices $v', v'' \in \mathbb{K}_0$ lie below v with respect to the direction s, we also have two edges and a two-simplex, included at time $s \cdot v$ by the construction of $\mathbb{K}$. Then, the single zero-simplex and two-simplex cancel out the two one-simplices and χ_s does not change at time $s \cdot v$, a contradiction. $\square$

The presence of codimension zero simplices lead to the types of problems described in Lemma 2. Oftentimes, these situations can be avoided with general position assumptions. Next, we consider more general results that motivate the research in this chapter relating filtered simplicial complexes and their respective diagrams.

3.1.1 Persistent Homology Transform

The persistent homology transform (PHT) was first introduced in 2014 [100]. Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. When considering general dimensions, the PHT considers a simplicial complex $\mathbb{K} \in \mathbb{R}^d$ and the parameterized set of all directions corresponding to unit vectors $s \in \mathbb{S}^{d-1}$. This uncountably infinite set of directions correspond to an uncountably infinite set of persistence diagrams (one for each dimension),

parametrized by the directions and resulting height filtrations.

Definition 10 (Persistent Homology Transform)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Let $\mathbb{S}^{d-1}$ be the unit sphere in $\mathbb{R}^d$ and $\mathcal{D}$ denote the space of persistence diagrams. The persistent homology transform is a function*

$$\textit{PHT}(\mathbb{K}) : \mathbb{S}^{d-1} \to \mathcal{D}^d,$$

where, for each $s \in \mathbb{S}^{d-1}$, the function maps

$$\textit{PHT}(\mathbb{K}) : s \mapsto D_i(\mathbb{H}_s),$$

for all $i \in \{0, 1, \ldots, d\}$.

3.1.2 Euler Characteristic Transform

Recently, Turner et al. and Crawford et al. investigated and utilized the Euler Characteristic Transform (and Smooth Euler Characteristic Transform) [39, 100]. Like persistence diagrams and the PHT, we can consider the ECCs and define an Euler Characteristic Transform (ECT). Intuitively, we are considering a transform from a simplicial complex to the space of ECCs where the set of ECCs generated is parameterized by directions on $\mathbb{S}^{d-1}$. Each direction determines a filtration on the simplicial complex and a resulting ECC.

Definition 11 (Euler Characteristic Transform)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex and $\mathbb{S}^{d-1}$ be the unit sphere in $\mathbb{R}^d$. The Euler characteristic transform is a function*

$$\textit{ECT}(\mathbb{K}) : \mathbb{S}^{d-1} \to \mathbb{Z}^{\mathbb{R}},$$

where, for each $s \in \mathbb{S}^{d-1}$, the function maps

$$\textit{ECT}(\mathbb{K}) : s \mapsto \chi_s$$

3.1.3 Properties of the PHT and ECT

The PHT and ECT are injective when the domain is simplicial complexes in $\mathbb{R}^2$ and $\mathbb{R}^3$ by [100, Theorem 3.1 and Corollary 3.2]. In other words, we can differentiate between two simplicial complexes using the PHT. The proof of injectivity uses a constructive approach where it is shown that you can determine the simplices in a simplicial complex using the diagrams of the PHT. Both transforms were also shown to be injective in $\mathbb{R}^d$ by simultaneous, but independent, work (Curry et al. [40] and Ghrist et al. [58]) using an inversion formula of Schapira [94] applied to the PHT and ECT. Furthermore, Curry et al. went on to prove a finite bound on the number of directions necessary to represent an unknown geometric simplicial complex $\mathbb{K}$ in $\mathbb{R}^d$ from Euler characteristic curves, i.e., using a finite subset of curves from $ECT(\mathbb{K})$ [40]. First, the embedding for the vertices of $\mathbb{K}$ are determined using an assumed lower bound δ on the local geometry around any given vertex (discussed more in Section 3.7). The vertices are discovered by generating topological summaries (i.e., Euler characteristic curves or persistence diagrams) by choosing directions in $\mathbb{S}^{d-1}$ based on δ to ensure that the embedding of each vertex is identified. Then, the $(d-1)$-dimensional sphere is stratified using hyperplanes that intersect pairs of vertices. Directions from each stratum are sampled to generate a parameterized set of Euler characteristic curves. Theorem 7.14 of [40] shows that this parameterized set of Euler characteristic curves sufficiently represents $\mathbb{K}$ with a finite number of Euler characteristic curves (or persistence diagrams). However, these bounds are dependent on assumptions about the curvature of the complex and are exponential in the dimension.

While ECCs require only linear time to compute, we focus our attention on a variant

of the PHT using APDs instead of PDs. APDs encode additional geometric information that allows for bijections between points in the diagram and vertices in the complex, permitting different algorithmic approaches to reconstruction. As such, we also introduce the augmented persistent homology transform (APHT).

3.1.4 Augmented Persistent Homology Transform

Like with the PHT and ECT, we can also consider a transform mapping a simplicial complex to a set of APDs parameterized by directions on the unit sphere. Each direction determines a filtration on the simplicial complex and a resulting APD.

Definition 12 (Augmented Persistent Homology Transform)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Let $\mathbb{S}^{d-1}$ be the unit sphere in $\mathbb{R}^d$ and $\overline{\mathcal{D}}$ denote the space of augmented persistence diagrams. The augmented persistent homology transform is a function*

$$\textit{APHT}(\mathbb{K}) : \mathbb{S}^{d-1} \to \overline{\mathcal{D}}^d,$$

where, for each $s \in \mathbb{S}^{d-1}$, the function maps

$$\textit{APHT}(\mathbb{K}) : s \mapsto \overline{D}_i(\mathbb{H}_s),$$

for all $i \in \{0, 1, \ldots, d\}$.

One of the major contributions of this dissertation is providing an algorithm for reconstructing simplicial complexes in $\mathbb{R}^d$ using a finite set of diagrams from the APHT. By reconstructing the simplicial complex, we provide an inverse for the APHT, in the form of an algorithm with time and diagram complexity analysis, and prove the injectivity of the APHT, see Theorem 8. Next, we introduce an oracle that provides APDs for the purposes of our reconstruction algorithms.

3.1.5 Diagram Oracle

In this chapter, our focus is to develop algorithms for identifying finite representations of the APHT. As such, we introduce our oracle from [56, Section 2.2] that, when given a direction, a dimension, and a simplicial complex (which is often clear from the context), returns an APD. The oracle framework makes it possible to accommodate various implementations of generating APDs while making no explicit assumptions about the runtime of computing a particular APD. The oracle is assumed to be a globally accessible function for the remainder of the chapter.

Definition 13 (Oracle)**.** *For a simplicial complex $\mathbb{K} \subset \mathbb{R}^d$ and a direction $s \in \mathbb{S}^{d-1}$, operation* $\texttt{Oracle}(s)$ *returns diagram $\overline{D}(s)$. Moreover, if a dimension $i \in \mathbb{Z}$ is specified, the oracle* $\texttt{Oracle}_i(s)$ *returns $\overline{D}_i(s)$ restricted to the i-dimensional points.* [1] *We define $\Theta(\Pi)$ to be the time complexity of computing the diagram.*

Our primary reconstruction algorithm (Algorithm 3.11) discovers simplices by identifying wedges in which properties, extracted from APDs, differ. Unfortunately, one cannot form the wedges around the d-simplices in $\mathbb{R}^d$. As such, for the remainder of this work we assume that the highest dimensional simplex of unknown complex is $\kappa < d$. However, we also provide an alternative suggested workflow that integrates a parabolic lift to apply this technique to simplicial complexes with $\kappa = d$.

We define two simplicial maps for the optional parabolic lift. First, for simplicial complex K in $\mathbb{R}^d$ and vertex $v \in \mathbb{K}_0$ with $v = (v^{(1)}, v^{(2)}, \ldots, v^{(d)})$, the *lifting map* $\mathscr{L}$ is the simplicial map induced by $v \mapsto (v^{(1)}, v^{(2)}, \ldots, v^{(d)}, v \cdot v)$. Second, for simplicial complex K' in $\mathbb{R}^{d+1}$ and vertex $v' \in \mathbb{K}'_0$, the *projection map* $\mathscr{P}$ is the simplicial map induced by omitting the $(d+1)^{\text{st}}$-coordinate of v'.

[1] While, we can return different dimensions as separate diagrams, all persistence points computed from a given direction are computed for one diagram. In particular, we sometimes request the zeroth and first dimensions of a diagram and refer to them as separate diagrams. However, when calculating diagram complexity, we count one diagram not two.

For the unknown complex $\mathbb{K}$, we define the lifting map $\texttt{Oracle}_{\uparrow}(s, i)$ that returns APDs of $\mathscr{L}(\mathbb{K})$ and accepts directions in $\mathbb{R}^{d+1}$. If we choose to apply the lifting map, we can reconstruct $\mathscr{L}(\mathbb{K})$ with Algorithm 3.11 using $\texttt{Oracle}_{\uparrow}$. As $\mathscr{P}(\mathscr{L}(\mathbb{K})) = \mathbb{K}$, we can then apply the projection map to the result of the reconstruction and output $\mathbb{K}$. The lifting map is not necessary when $\kappa < d$ and so we continue with the remainder of the work using the standard oracle.

3.1.6 Contributions to Section 3.1

The preliminaries described in this section have developed significantly over the years and pulled from papers [12, 55, 56] with new concepts being added since the original work that our group conducted on this problem in [11]. However, the primary contributions to this section are the carefully selected properties in Assumption 1 as well as the idea and definition of the oracle. Many of the properties included in Assumption 1 were developed specifically for interesting degeneracies discovered through discussions between members of the group. For a while, the problem stopping progress on this work was co-dimension zero simplices and the idea of the oracle was presented by Samuel Micka. Samuel Micka made an initial attempt at writing up Definition 13 which was updated by David L. Millman and further refined by Samuel Micka.

3.2 Simplex Predicates

In this section, we develop the constructions and a predicate needed for reconstructing simplicial complexes. We note that the results in this section originate from in our recent work [56]. The predicate, presented in Algorithm 3.3, determines whether or not a set of $k + 1$ zero-simplices is a k-simplex of the underlying simplicial complex. Before we are able to introduce the predicate, we described several helper functions in Section 3.2.1.

3.2.1 k-Indegree

The key piece of machinery we develop for determining whether a simplex exists is the *k-indegree* of a simplex, which is the count of k-dimensional cofaces of a simplex σ below σ in a particular direction. First, we introduce two preliminary lemmas which assist with the Face Isolation operation described in Lemma 5 and Algorithm 3.1. The first lemma allows us to pack additional points into the affine space defined by a set of vertices. We describe this plane filling operation with Lemma 3.

Lemma 3 (Plane Filling)**.** *Let $V \subset \mathbb{R}^d$ be k affinely independent points with $k \leq d$, let $s \in \mathbb{S}^{d-1}$ be orthogonal to $\mathrm{aff}(V)$. We can produce $d-k$ points V' in $\mathbb{R}^d$ such that s is orthogonal to $\mathrm{aff}(V \cup V')$ and $\dim(\mathrm{aff}(V \cup V')) = d-1$ in $O(d^3)$ time.*

Proof. Label the k vertices of V $\{v_0, v_1, \ldots, v_{k-1}\}$ and define $d \times k$ matrix A such that $v_i - v_0$ are the ith columns for $0 < i \leq k-1$ and s is the kth column. Creating A takes $\Theta(kd)$ time. Let Q_N be the basis vectors of the null space of A. We can compute these vectors via a QR-decomposition using Gram-Schmidt in $O(d^3)$ time [98]. As the vertices in V are affinely independent and s is orthogonal to $\mathrm{aff}(V)$, the dimension of the column space of A is k and so there are $d-k$ vectors in Q_N. Label the vectors $\{q_k, q_{k+1}, \ldots, q_d\}$ and let $V' = \{(q_k + v_0), (q_{k+1} + v_0), \ldots, (q_d + v_0)\}$. Constructing V' takes time $O(d^2)$. Computing the QR-decomposition dominates the algorithm, hence the running time is $O(d^3)$

By our construction, $\dim(\mathrm{aff}(V \cup V')) = d-1$. To show that s is orthogonal to $\mathrm{aff}(V \cup V')$, it suffices to show that $s \cdot v_0 = s \cdot v_i'$ for all $v_i' \in V'$. Indeed, since q_i is orthogonal to s, we have $s \cdot v_i' = s \cdot (q_i + v_0) = s \cdot v_0$. $\square$

To prove that the ordering of vertices in Algorithm 3.1 remains consistent, we introduce Lemma 4 to assist in the proof of the properties in Lemma 5.

Lemma 4 (ϵ-Perturbation)**.** *Let $s_1, s_2 \in \mathbb{S}^{d-1}$ be two directions and $V \subset \mathbb{R}^d$. Let $n = |V|$. If $w \in V$ such that for all $v \in V \setminus \{w\}$ we have $s_1 \cdot w \neq s_1 \cdot v$, then we can compute an $\epsilon > 0$ such that*

1. $v \cdot s_1 < w \cdot s_1 \iff v \cdot (s_1 + \epsilon s_2) < w \cdot (s_1 + \epsilon s_2)$ *and*

2. *for all $v_1, v_2 \in V$: if $v_1 \cdot s_1 < v_2 \cdot s_1$, then $v_1 \cdot (s_1 + \epsilon s_2) < v_2 \cdot (s_1 + \epsilon s_2)$*

in $O(n \log n)$ time.

Proof. Let S be the set of line segments

$$S := \left\{ \big((0, x \cdot s_1), (1, x \cdot (s_1 + s_2)\big) \right\}_{x \in V}.$$

Each line segment in S represents a linear interpolation between points $(0, v \cdot s_1)$ and $(1, v \cdot (s_1 + s_2))$ corresponding the height of each $v \in V$ in directions s_1 and $s_1 + s_2$, respectively. Moreover, we can interpret each point along the line segment between $(0, v \cdot s_1)$ and $(1, v \cdot (s_1 + s_2))$ as $v \cdot (s_1 + \epsilon s_2)$ for some $\epsilon \in [0, 1]$. Let V' denote the vertices of V with unique heights in direction s_1. Then, we want to identify an $\epsilon > 0$ such that the ordering of the dot products of the vertices of V' with $(s_1 + \epsilon s_2)$ is consistent with the ordering of the vertices of V' with the dot product of s_1. It suffices to find the left most intersection p in $(0, 1]$ of the segments in S, denoted $p^{(1)}$, and then choose epsilon to be smaller than $p^{(1)}$. We can identify $p^{(1)}$ in $(0, 1]$ using standard segment intersection algorithms in $O(n \log n)$ time [13, §2.2].

Let $\epsilon = p^{(1)}/2$. Let $v \in V \setminus \{w\}$. To show Part 1($\Rightarrow$), assume that $v \cdot s_1 < w \cdot s_1$ then, by our choice of ϵ, $v \cdot (s_1 + \epsilon s_2) < w \cdot (s_1 + \epsilon s_2)$, as required.

Let $v \in V \setminus \{w\}$. To show Part 1 ($\Leftarrow$), assume that $v \cdot (s_1 + \epsilon s_2) < w \cdot (s_1 + \epsilon s_2)$. We recall that ϵ was chosen such that no two lines in S had crossed before ϵ in the x-direction, and since $v \cdot s_1 \neq w \cdot s_1$, we can replace ϵ with zero and maintain the inequality $v \cdot s_1 < w \cdot s_1$.

The proof of Part 2, follows from the proof of Part 1 ($\Rightarrow$). Let $v_1, v_2 \in V$. Assume that $v_1 \cdot s_1 < v_2 \cdot s_1$ then, by our choice of ϵ, $v_1 \cdot (s_1 + \epsilon s_2) < v_2 \cdot (s_1 + \epsilon s_2)$, as required. Note that the converse statement does not necessarily hold because $v_1 \cdot s_1$ could equal $v_2 \cdot s_1$. $\square$

Now, we develop a method for choosing directions that "isolate" a face of a simplex. The method for isolating simplices in Lemma 5 is described in Algorithm 3.1.

Lemma 5 (Face Isolation)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Let $V \subset \mathbb{K}_0$ be k vertices with $k \leq d$, and let $W \subseteq V$. Let $s \in \mathbb{S}^{d-1}$ be orthogonal to $\mathrm{aff}(V)$ and for $v \in V$ and $u \in \mathbb{K}_0 \setminus V$, $s \cdot v \neq s \cdot u$. Then, Algorithm 3.1 finds a direction $s' \in \mathbb{S}^{d-1}$ such that there exists a constant $c \in \mathbb{R}$ with the following properties:*

1. *For all $w \in W$, $s' \cdot w = c$.*
2. *If $v \in V \setminus W$, then $s' \cdot v > c$*
3. *No vertex in $\mathbb{K}_0 \setminus W$ has height c in direction s'.*
4. *If $v \in \mathbb{K}_0 \setminus V$ and $w \in W$, then $s \cdot v < s \cdot w \iff s' \cdot v < c$.*
5. *The runtime of finding such an s' is $O(n_0(d^5 + \log n_0))$.*

Proof. Algorithm 3.1, has an "early exit" in which we return s when $V = W$. Observe that when $s = s'$, Property 1 and Properties 3–4 are trivially true and Property 2 is vacuously true. Next, we consider the more interesting path of the algorithm.

(Property 1) To show Property 1, it suffices to show that all vertices in W have the same height in direction s'. At termination of algorithm, we return vector $s' = s + \epsilon s_P$. Recall that for any $v_1, v_2 \in V$, $s \cdot v_1 = s \cdot v_2$. Furthermore, we note that s_P is the normal to hyperplane P. Therefore, for any points $p_1, p_2 \in P$, $s_P \cdot p_1 = s_P \cdot p_2$. Moreover, $W \subset P$, so for all $w_1, w_2 \in W$, we have that $h_{s'}(w_1) = s' \cdot w_1 = (s + \epsilon s_P) \cdot w_1 = s \cdot w_1 + \epsilon s_P \cdot w_1 = s \cdot w_2 + \epsilon s_P \cdot w_2 = h_{s'}(w_2)$.

(Property 2) Recall that $\mathrm{aff}(W)$ is orthogonal to s by definition and s' by Property 1. Thus, all vertices in W have the same height in direction s and s'. As such, it suffices to show that for all vertices in $v \in W' = V \setminus W$ and any $w \in W$, $w \cdot s' < v \cdot s'$. To prove this claim, we introduce the following loop invariant: if P^i is P entering iteration i, then let the vector s_{P^i} be the normal to $\mathrm{aff}(P^i)$. There are two choices for s_{P^i} and we choose the one that has the property that $s_{P^i} \cdot w < s_{P^i} \cdot v$ for any $w \in W$ and all $v \in W'$. It is true on initiation since W' is empty. Assume the invariant holds entering iteration i. We remove v from P^i and replace it with v' creating the new affine space P^{i+1}. Furthermore, by construction, one of the two vectors $s_{P^{i+1}}$ normal to $\mathrm{aff}(P^{i+1})$ has the property that $v' \cdot s_{P^{i+1}} < v \cdot s_{P^{i+1}}$, so we choose $s_{P^{i+1}}$ to satisfy the property. Next, assume, for contradiction, that a point $w' \in W'$ lies below $\mathrm{aff}(P^{i+1})$. Then, $\mathrm{aff}((P^i \setminus \{v\}) \cup \{w'\})$ would have an a intersection point with ℓ in $L_\downarrow$ nearer v than the argmin x chosen on Line 12, a contradiction. Upon exiting the loop, we have that $w \cdot s_P < v \cdot s_P$ by our inductive assumption. Then, the ϵ returned from Lemma 4 with arguments s, s_P and, $(\mathbb{K}_0 \setminus V) \cup \{w\}$ for any $w \in W$ (since all vertices in W have the same height in directions s and s_P) is positive. Then, since $\epsilon > 0$, $w \cdot s = v \cdot s$, and $w \cdot s_P < v \cdot s_P$, we have that $w \cdot (s + \epsilon s_P) < v \cdot (s + \epsilon s_P)$. Since $s' = s + \epsilon s_P$, the claim holds. The algorithm terminates since $|\mathbb{K}_0|$ is finite.

(Property 3) Let $v \in K_0 \setminus W$ and $w \in W$ and observe that ϵ is chosen as in Lemma 4 with arguments s, s_P, and $(\mathbb{K}_0 \setminus V) \cup \{w\}$. Then, by Lemma 4, since $v \cdot s \neq s \cdot w$, we have $v \cdot s < w \cdot s \iff v \cdot s' < w \cdot s'$. Moreover, since $v \cdot s \neq w \cdot s$, we have that no vertices in $\mathbb{K}_0 \setminus V$ lie at height c in direction s'. Finally, any $v \in V \setminus W$ does not have height c in direction s' by Property 2.

(Property 4) Let $v \in K_0 \setminus V$ and $w \in W$. Let ϵ be chosen as in Lemma 4 with arguments s, s_P, and $(\mathbb{K}_0 \setminus V) \cup \{w\}$ for any $w \in W$. Then, by Lemma 4, since $v \cdot s \neq s \cdot w$, $v \cdot s < w \cdot s \iff v \cdot (s + \epsilon s_P) < w \cdot (s + \epsilon s_P)$. Furthermore, $s' = s + \epsilon s_P$ and since

$w \cdot s' = c$ for all $w \in W$ by Property 1, the condition holds for all $v \in K_0 \setminus V$ and any $w \in W$.

(Property 5) In Algorithm 3.1, Lines 1–2 returns s if $V = W$. As $|V| < d$, checking if the two sets are equal takes $O(d^2)$ time. Lines 3–4 initialize P, and by Lemma 3 take $O(d^3)$. Line 6 initializes $W_\downarrow$ and $W_\uparrow$ by computing dot products with a subset of $\mathbb{K}_0$ and takes $O(n_0 d)$.

For each vertex in $V \setminus W$ we perform the loop on Lines 7–15. The loop is the crux of the algorithm. At the beginning of each iteration, P defines a $d-1$ dimensional hyperplane. In the iteration, we remove a point v from P and find a new point that defines a hyperplane with specific properties (shown in other parts of the proof). As the time for updating hyperplane is dominated by Line 10, we will focus the analysis on that line. Label the points of P on Line 9 as $\{p_1, p_2, \ldots, p_{d-1}\}$. Recall that any point q on the hyperplane defined by $P \cup \{x\}$, satisfies the equation

$$\begin{vmatrix} {p_1}^{(1)} & {p_1}^{(2)} & \ldots & {p_1}^{(d)} & 1 \\ {p_2}^{(1)} & {p_2}^{(2)} & \ldots & {p_2}^{(d)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ {p_{d-1}}^{(1)} & {p_{d-1}}^{(2)} & \ldots & {p_{d-1}}^{(d)} & 1 \\ x^{(1)} & x^{(2)} & \ldots & x^{(d)} & 1 \\ q^{(1)} & q^{(2)} & \ldots & q^{(d)} & 1 \end{vmatrix} = 0$$

Moreover, notice that the ith coordinate of ℓ can be written parametrically as $\ell^{(i)}(t) = v^{(i)} + ts^{(i)}$. Thus, we can compute the intersection by solving for t in the polynomial

obtained by expanding by minors

$$\begin{vmatrix} {p_1}^{(1)} & {p_1}^{(2)} & \dots & {p_1}^{(d)} & 1 \\ {p_2}^{(1)} & {p_2}^{(2)} & \dots & {p_2}^{(d)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ {p_{d-1}}^{(1)} & {p_{d-1}}^{(2)} & \dots & {p_{d-1}}^{(d)} & 1 \\ x^{(1)} & x^{(2)} & \dots & x^{(d)} & 1 \\ \ell^{(1)}(t) & \ell^{(2)}(t) & \dots & \ell^{(d)}(t) & 1 \end{vmatrix} = \sum_{i=1}^{d} \ell^{(i)}(t)|M_{d+1,i}| + |M_{d+1,d+1}| = 0.$$

There are $d+1$ minors each of size d. Using LU-decomposition, we can compute each determinant in $O(d^3)$ time [98]. The resulting polynomial, that is linear in t, can be solved in $O(d)$ time. Thus, each intersection takes $O(d^4)$ time. We compute the intersection at most once per vertex of $\mathbb{K}_0$, therefore, Line 10 takes $O(n_0 d^4)$ time. Finally, we get our running time by observing that we compute L once for each entry of $V \backslash W$ and $|V \backslash W| \leq d$. Therefore Lines 7–15, takes $O(n_0 d^5)$ time.

In the last steps of the algorithm, Line 16 computes the normal to $\mathrm{aff}(P)$ by computing determinants of $d+1$ of size d (similar to above) in $O(d^4)$ time. Line 17 computes the scaling of the s_P in $O(n_0 \log n_0)$ by Lemma 4. As Lines 7–15 and Line 17 dominate the computation, Algorithm 3.1 takes $O(n_0(d^5 + \log n_0))$ time. □

Next we develop a predicate that uses these isolated simplices to test for k-simplices. Intuitively, we identify one-simplices by checking all pairs of zero-simplices with this predicate, then identify all two-simplices by checking all triples of one-simplices, etc. using a "bow tie" technique like the approach found in [11]. For this predicate to generalize for reconstruction of higher-dimensional simplices, we provide the following definition:

Definition 14 (k-indegree for Simplex)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex and $\sigma \in \mathbb{K}$ be a j-simplex such that $0 \leq j < k \leq d$. Let $s \in \mathbb{S}^{d-1}$ be a direction perpendicular to*

Algorithm 3.1: `FaceIsoDirection`(V, W, s)

Input: V, a set of k vertices, W, a set of vertices with $W \subseteq V$, and $s \in \mathbb{S}^{d-1}$ orthogonal to $\mathrm{aff}(V)$

Output: direction s' satisfying Lemma 5 Properties 1–4

if $V = W$ **then**
    return s
$V' \leftarrow$ value returned from Lemma 3 with parameters V and s
$P \leftarrow V \cup V'$
$W' \leftarrow \{\}$
$W_\uparrow, W_\downarrow \leftarrow$ vertices in $\mathbb{K}_0 \setminus V$ above (below) $\mathrm{aff}(V)$ in direction s
for $v \in V \setminus W$ **do**
    $\ell \leftarrow$ line through v perpendicular to $\mathrm{aff}(P)$
    $P \leftarrow P \setminus \{v\}$
    $L \leftarrow \{\ell \cap \mathrm{aff}(P \cup \{x\}) | x \in W' \cup W_\uparrow \cup W_\downarrow\}$
    $L_\downarrow \leftarrow \{x \in L | x$ is below v on $\ell\}$
    $x' \leftarrow \arg\min_{x \in L_\downarrow} \|v - x\|_2$
    $v' \leftarrow (v + x')/2$
    $P \leftarrow P \cup \{v'\}$
    $W' \leftarrow W' \cup \{v\}$
$s_P \leftarrow$ normal of $\mathrm{aff}(P)$ with $w \cdot s_P < v \cdot s_P$ for all $w \in W$ and $v \in W'$.
$\epsilon \leftarrow$ value from Lemma 4 with parameters s, s_P, and $(\mathbb{K}_0 \setminus V) \cup \{w\}$ for any $w \in W$
return $s' \leftarrow \frac{s + \epsilon s_P}{||s + \epsilon s_P||}$

$\mathrm{aff}(\sigma)$. *Then, the* k-indegree of σ in direction s *is the number of* k*-dimensional cofaces of* σ *that have the same height as* σ *in direction* s.

Since s is perpendicular to $\mathrm{aff}(\sigma)$, all zero-simplices of σ are at the same height in

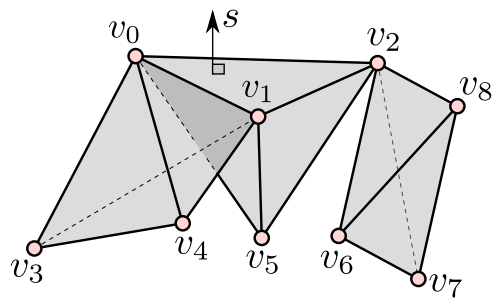


Figure 3.1: The computation of the three-indegree of $\sigma = [v_0, v_1, v_2]$ with respect to direction s. Here, let $s \in \mathbb{S}^3$ be normal to $\mathrm{aff}(\sigma)$ such that v_5 is below σ in direction s. Note that only one three-simplex is a coface of σ, the simplex defined by $\mathrm{verts}(\sigma) \cup \{v_5\}$. However, three faces of σ have positive three-indegree; namely $[v_0, v_1]$ and $[v_2]$ both have three-indegree equal to one. Thus, we compute the three-indegree of σ using an inclusion-exclusion argument ($3 - 1 - 1 = 1$) described in (3.4).

direction s. However, as shown in Figure 3.1, not all k-simplices at this height contribute to the k-indegree of σ. As such, we introduce the following lemma to prove that we can isolate faces of a simplex and count their k-indegree independently of the original simplex.

Lemma 6 (Unique Face Isolation)**.** *Let $k \leq d \in \mathbb{N}$. Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Let $\tau \preceq \sigma \in \mathbb{K}$. Let $s \in \mathbb{S}^{d-1}$ be orthogonal to $\mathrm{aff}(\sigma)$ such that for all $v \in \sigma$ and $u \in \mathbb{K}_0 \setminus \mathrm{verts}(\sigma)$, $s \cdot v \neq s \cdot u$. Let $\sigma' \in \mathbb{K}$ be a k-simplex at the same height as σ in direction s. Then, σ' contributes to the k-indegree of τ in direction* $\texttt{FaceIsoDirection}(\mathrm{verts}(\sigma), \mathrm{verts}(\tau), s)$ *if and only if $\tau = \sigma \cap \sigma'$.*

Proof. Let s' be the direction returned by $\texttt{FaceIsoDirection}(\mathrm{verts}(\sigma), \mathrm{verts}(\tau), s)$. Let $f \colon \mathbb{K} \to \mathbb{R}$ ($f' \colon \mathbb{K} \to \mathbb{R}$, respectively) be the filter function for direction s (s', respectively).

($\Rightarrow$) Suppose that σ' contributes to the k-indegree of τ in direction s'. By the definition of k-indegree, $\tau \preceq \sigma'$. Since $\tau \preceq \sigma$ by assumption, we have $\tau \preceq \sigma \cap \sigma'$. We must now show that $\sigma \cap \sigma' \preceq \tau$.

By contradiction, suppose that $\sigma \cap \sigma' \not\preceq \tau$. Then, there exists a vertex $v \in \sigma \cap \sigma'$ such that $v \notin \tau$. However, $v \in \sigma \cap \sigma'$, $v \in \sigma \setminus \mathrm{verts}(\tau)$. Therefore, $s' \cdot v > c$ by Property 2 of

Algorithm 3.2: $\texttt{ComputeIndegree}(\sigma, s, k, T)$

Input: $\sigma \in \mathbb{K}$, $s \in \mathbb{S}^{d-1}$ such that $\exists c \in \mathbb{R}$ where $v \cdot s = c$ for all $v \in \sigma$ and for any $u \in \mathbb{K}_0, \backslash \mathrm{verts}(\sigma)$, $s \cdot v \neq s \cdot u$, $k \in \mathbb{N}$ such that $k > \dim(\sigma)$, table T for memoization

Output: the k-indegree for σ

```
1: D̄_{k-1}(s), D̄_k(s) ← (k-1)^st and k^th APDs from Oracle_{k-1}(s) and Oracle_k(s)
2: c ← height of σ in direction s
3: numDeaths ← number of deaths in D̄_{k-1}(s) at height c
4: numBirths ← number of births in D̄_k(s) at height c
5: doubleCounts ← 0
6: for τ ≺ σ in non-descending order by dimension do
7:     if T[τ] was not computed yet then
8:         s' ← FaceIsoDirection(verts(σ), verts(τ), s)
9:         T[τ] ← ComputeIndegree(τ, s', k, T)
10:    doubleCounts ← doubleCounts + T[τ]
11: return numDeaths + numBirths − doubleCounts
```

Lemma 5, a contradiction to the claim that σ' contributes to the k-indegree of τ in direction s'. Therefore, $\sigma \cap \sigma' \preceq \tau$ as required.

($\Leftarrow$) Suppose that $\tau = \sigma \cap \sigma'$. Let s' be the direction returned by $\texttt{FaceIsoDirection}(\sigma, \tau, s)$, and let $f' \colon K \to \mathbb{R}$ be the filter function for direction s'. Since σ' is a k-simplex and $\tau \preceq \sigma'$, we can write: $\tau = [v_0, v_1, \ldots, v_j]$ and $\sigma' = [v_0, v_1, \ldots, v_k]$ where $j \leq k$ and $v_i \in K_0$. Then,

$$f'(\sigma') = \max_{i=0}^{k} f'(v_i) = \max\left(\max_{i=0}^{j} f'(v_i), \max_{i=j+1}^{k} f'(v_i)\right) = \max\left(f'(\tau), \max_{i=j+1}^{k} f'(v_i)\right). \tag{3.1}$$

Since σ' is at the same height as σ in direction s and $\tau \prec \sigma$, σ' is also at the same height as

τ in direction s, meaning that $f(v_i) \leq f(\tau)$ for all $0 \leq i \leq$ k. Since v_i not in τ for $i > j$, it must also be the case that $f(v_i) < f(\tau)$. By Property 4 of Lemma 5, any vertex below τ in direction s is also below τ in direction s'. Thus, $f'(v_i) < f'(\tau)$ for all $j < i \leq k$ and

$$\left(\max_{i=j+1}^{k} f'(v_i)\right) < f'(\tau).$$

Hence, by Equation 3.1, $f'(\sigma') = f'(\tau)$. Since $\tau \preceq \sigma'$, we have shown that σ' contributes to the k-indegree of τ. □

Note that Figure 3.1 is an example of a case where only one three-simplex contributes to the three-indegree of the two-simplex in question.

Since Lemma 6 shows that a single diagram is not sufficient to compute the k-indegree, we use an inclusion-exclusion style argument to compute the k-indegree in Algorithm 3.2. Note T is a table used for memoization and the first time this algorithm is called, we have not computed any entries of T. We prove the correctness of this algorithm in the following theorem:

Theorem 1 (Computing k-indegree)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Let $\sigma \in \mathbb{K}$. Let $s \in \mathbb{S}^{d-1}$ be a direction such that $\exists c \in \mathbb{R}$ where $v \cdot s = c$ for all $v \in \sigma$ and for any $u \in \mathbb{K}_0 \setminus \mathrm{verts}(\sigma)$, $s \cdot v \neq s \cdot u$. For $k > \dim(\sigma)$, $\texttt{ComputeIndegree}(\sigma, s, k, \cdot)$ returns the k-indegree of σ in direction s.*

Proof. We prove the claim inductively on $j = \dim(\mathbb{K})$. For the base case ($j = 0$), let $k > j$ and consider the 0-simplex $[v]$. We note that this base case is a generalization of [12, Lemma 11]. Let $f \colon \mathbb{K} \to \mathbb{R}$ be the filter function for direction s. We note that, unlike in [12, Lemma 11], we are only making an argument for the k-indegree at a single vertex and not all vertices. As such, we can relax the requirement that no two vertices in $\mathbb{K}_0$ have the same height in direction direction s and just require that no require that no vertex in

$\mathbb{K}_0 \setminus \{v\}$ has the same height in direction s as v. Thus, we have that k-indegree of σ is equal to the number of k-simplices that have height $f(v)$, which, by Corollary 2, is:

$$|f^{-1}(f(v))| = \left|\{(a,b) \in \overline{D}_{k-1}(s) \ s.t. \ b = f(v)\}\right| + \left|\{(a,b) \in \overline{D}_{k}(s) \ s.t. \ a = f(v)\}\right|. \tag{3.2}$$

In Algorithm 3.2, notice that if σ is a single vertex, we do not enter the loop that starts on Line 6. Thus, the return value is exactly the number given in (3.2).

For the inductive assumption, let $j \geq 0$. We assume that Algorithm 3.2 returns the k'-indegree of τ in direction s, for all $\tau \in \mathbb{K}_j$ and all $k' > j$.

For the inductive step, let $\sigma \in \mathbb{K}_{j+1}$. Let $k > j+1$. Now, we compute the k-indegree of σ in direction s. Let $f\colon K \to \mathbb{R}$ be the lower-star filtration for direction s. Using Corollary 2, we know that the number of k-simplices with height $f(\sigma)$ in direction s is:

$$\delta := \left|\{(a,b) \in \overline{D}_{k-1}(s) \ s.t. \ b = f(\sigma)\}\right| + \left|\{(a,b) \in \overline{D}_{k}(s) \ s.t. \ a = f(\sigma)\}\right|. \tag{3.3}$$

Let F_σ denote this set of simplices, let $\sigma' \in F_\sigma$, and let $\tau \prec \sigma$. By Lemma 6, the k-simplex σ' contributes to the k-indegree of τ if and only if $\tau = \sigma \cap \sigma'$. Then, we can isolate each face of $\tau \prec \sigma$ and compute the k-indegree using (3.3) of τ and subtract it from the k-indegree of σ. This will ensure that a coface of some face $\tau \prec \sigma$ will not contribute to the k-indegree of σ. We formalize this idea with an equation for the k-indegree of σ

$$\delta - \sum_{\tau \prec \sigma} \delta_\tau, \tag{3.4}$$

where δ_τ is the k-indegree of τ in direction s' returned from `FaceIsoDirection`$(\mathrm{verts}(\sigma), \mathrm{verts}(\tau), s)$.

In Algorithm 3.2, `numDeaths`+`numBirths` is equal to δ, and the values δ_τ are computed in Line 9 of Algorithm 3.2. Thus, the return value matches (3.4). $\square$

The next lemma provides the runtime of Algorithm 3.2.

Theorem 2 (Algorithm 3.2 Complexity Bounds)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Let σ be an i-simplex in $\mathbb{K}$ for $0 \leq i < d$ and let k be a fixed integer $0 < k \leq d$. Let $s \in \mathbb{S}^{d-1}$ be a direction such that $\exists c \in \mathbb{R}$ where $v \cdot s = c$ for all $v \in \sigma$ and for any $u \in \mathbb{K}_0 \setminus \mathrm{verts}(\sigma)$, $s \cdot v \neq s \cdot u$. Then, the following properties hold:*

1. *if $i = 0$ and $k = 1$, then Algorithm 3.2 uses one diagram and runs in $O(\Pi + n_0^2)$ time.*
2. *Else, if $i > 0$, then Algorithm 3.2, uses $2^{i+1} - 1$ diagrams and runs in $O(2^{i+1}(n_0(n_{k-1} + d^5 + \log n_0) + \Pi + 2^i))$ time.*

Proof. If $i = 0$ and $k = 1$, then we prove **Property 1**. The loop is never entered and we compute one diagram in time $\Theta(\Pi)$ and count $O(n_0^2)$ points in the diagram.

To prove **Property 2**, we focus our attention on the for loop that begins on Line 6 of Algorithm 3.2. In this loop, we iterate through the proper faces of σ. The i-simplex σ has $2^{i+1} - 2$ faces so the loop iterates $2^{i+1} - 2$ times. However, when processing a new simplex $\tau \preceq \sigma$ for the first time, we need to create the table entry $T[\tau]$. To compute the entry, we get a new direction that will isolate τ on Line 8 of Algorithm 3.2 and then recursively call Algorithm 3.2 on Line 9. Since faces of σ are processed in non-descending order, by the time we process τ, all of the faces of τ (which are also faces of σ and have dimension less than $dim(\tau)$) have table entries. As a result, the recursive calls never go more than one level deep. Since recursive calls are only made when iterating through the faces of σ, there are $O(2^{i+1})$ calls made on Line 9 of Algorithm 3.2. Each call gets the kth and $(k-1)^{\text{st}}$ diagrams on Line 1 and counts the births and deaths on Lines 3–4 of Algorithm 3.2, taking $O(n_{k-1}n_0 + \Pi)$ time. Then, since $\tau \preceq \sigma$, we iterate through the $O(2^i)$ faces of τ, never entering the conditional on Line 7 of Algorithm 3.2, to sum up the entries from T in time $O(2^i)$. Moreover, to make each recursive call, we also call `FaceIsoDirection`$(\mathrm{verts}(\sigma), \mathrm{verts}(\tau), s)$ on Line 8 of Algorithm 3.2, which runs in

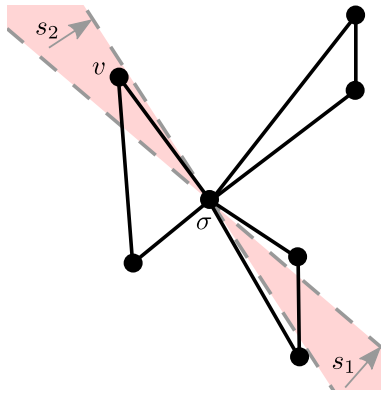


Figure 3.2: A wedge (shaded in pink) centered at a k-simplex σ (here, a single vertex) that isolates v. In Theorem 3, we notice that the $(k+1)$-indegrees for σ in directions s_1 and s_2 differ by one, indicating that $\mathrm{verts}(\sigma) \cup \{v\}$ defines a simplex in K.

$O(n_0(d^5 + \log n_0))$ by Lemma 5. Summing these values, we arrive at the total runtime for Algorithm 3.2 is $O(2^{i+1}(n_0(n_{k-1}+d^5+\log n_0)+\Pi+2^i))$ for an i-simplex and $2^{i+1}-1$ new APDs are generated, one for each recursive call and one for the initial k-below count. □

3.2.2 Simplex Predicate

Using the k-indegree, we are able to isolate and determine the presence of k-simplices between two hyperplanes centered at a simplex. This idea is a generalization of the "bow tie" technique used for identifying edges in [11]. The generalization of a bow tie is a double-cone shaped region that contains exactly one vertex called a *wedge*; see Figure 3.2.

Definition 15 (Wedge)**.** *Let $P \subset \mathbb{R}^d$ such that $|P| < \infty$ and $\mathrm{aff}(P) \neq \mathbb{R}^d$. Let $s_1, s_2 \in \mathbb{S}^{d-1}$ be orthogonal to $\mathrm{aff}(P)$. Let $p, p' \in P$ and note that $s_1 \cdot p = s_1 \cdot p'$ and $s_2 \cdot p = s_2 \cdot p'$. The* wedge *between s_1 and s_2 at P, and without loss of generality for any $p \in P$, is the closure of the symmetric difference between $H^{\downarrow}(s_1, p)$ and $H^{\downarrow}(s_2, p)$, denoted $H^{\downarrow}(s_1, p) \triangle H^{\downarrow}(s_2, p)$.*

Turner et al. also developed an approach that "rotates" around potential simplices, watching for changes in the Betti number, to identify links in $\mathbb{R}^2$ and $\mathbb{R}^3$ in a constructive proof of injectivity [100, Theorem 3.1]. However, the wedge we describe is able to utilize

information from the APD, through the concept of k-indegree, and we provide deterministic approaches for determining the wedges and using them for reconstruction in $\mathbb{R}^d$.

Next, before we develop our simplex predicate. We need a method of determining an initial direction to test for our simplex. Specifically, in Algorithm 3.3 (Line 3) of Section 3.2.2, we need to choose an initial direction s that is orthogonal to the k-plane $\mathrm{aff}(\sigma \cup \{v\})$ but not orthogonal to any other subspace spanned by zero-simplices of $\mathbb{K}_0$.

Lemma 7 (Orthogonal Vector and Full Set)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex with zero-simplices $\mathbb{K}_0$. Given a set of zero-simplices $W = \{w_0, w_1, \ldots, w_k\} \subseteq \mathbb{K}_0$ for $k < d$, we can find $V \subseteq \mathbb{K}_0$ and a direction $s \in \mathbb{S}^{d-1}$ such that the following conditions are met;*

1. *$W \subseteq V$*
2. *For all $q \in \mathbb{K}_0 \setminus V$ and $v \in V$, we have $s \cdot q \neq s \cdot v$,*
3. *s is orthogonal to $\mathrm{aff}(V)$,*
4. *and $|V| \leq d$.*

This s and V can be found in $O(dn_0 + d^2)$ time.

Proof. We find s and V constructively. First, choose $s \in \mathbb{S}^{d-1}$ orthogonal to $\mathrm{aff}(W)$ (observe that this can be done using a single iteration of Grahm Schmidt orthogonalization). This means that every $w \in W$ has height $s \cdot w = c$ for a single constant $c \in \mathbb{R}$. We define W' to be a set of vertices that is initially empty. Then, for every $q \in \mathbb{K}_0 \setminus W$, we set $W' = W' \cup \{q\}$ whenever $s \cdot q = c$. Iterating through all such $q \in \mathbb{K}_0 \setminus W$, we can then define $V = W \cup W'$. Observe that the first two parts are met immediately by construction. To see that the third part is met, we observe that, since $s \cdot v_i = c$ for all $v_i, v_j \in V, i \neq j$, it must be that $s \cdot (v_i - v_j) = (s \cdot v_i) - (s \cdot v_j) = 0$, i.e., s is orthogonal to $\mathrm{aff}(V)$ as required. Part four is a consequence of the general position assumption (Assumption 1).

The runtime of this procedure can be calculated as follows. Finding s takes one iteration of the Grahm Schmidt algorithm, and thus takes $O(d^2)$ time [98]. Checking if $s \cdot q = c$ takes $O(d)$ time, and is repeated for each of the vertices in $\mathbb{K}_0 \setminus W$. Thus, the total runtime is $O(dn_0 + d^2)$, as desired. $\square$

In Algorithm 3.3, we use our initial direction and the difference in indegree between the two filtration hyperplanes (wedge) to test for the presence of a $(k+1)$-simplex.

Theorem 3 (Correctness of Simplex Predicate)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex, $k \in \mathbb{N}$, $\sigma \in \mathbb{K}_{k-1}$, and $v \in \mathbb{K}_0 \setminus \mathrm{verts}(\sigma)$. Then, Algorithm 3.3 returns* `True` *iff* $\mathrm{verts}(\sigma) \cup \{v\}$ *defines a k-simplex in K in time $O(2^k(n_0(n_{k-1} + d^5 + \log n_0) + \Pi + 2^{k-1}))$ with $2^{k+1} - 2$ diagrams.*

Proof. Let $\Sigma = \mathrm{verts}(\sigma) \cup \{v\}$. We are testing if Σ defines a k-simplex in K. First, we compute an initial direction s_t using Lemma 7 in $O(dn_0 + d^2)$ time. Since s' may be orthogonal to extra zero-simplices (in V, but not in Σ), we call `FaceIsoDirection` with parameters V, $\mathrm{verts}(\Sigma)$, and s_t to get our initial direction s that guarantees that for all $v, v' \in \mathbb{K}_0 \setminus \mathrm{verts}(\Sigma)$ and all $w, w' \in \mathrm{verts}(\Sigma)$, $s \cdot v \neq s \cdot w$, $s \cdot w = s \cdot w'$, and $s \cdot v \neq s \cdot v'$. Using Algorithm 3.1, let $s_1 = \texttt{FaceIsoDirection}(\Sigma, \mathrm{verts}(\sigma), s)$, as in Figure 3.2. By Properties 1 and 2 of Lemma 5, there exists $c_1 \in \mathbb{R}$ such that all vertices of σ are at height c_1 and v is above c_1 (in direction s_1). Next, let $s_2 = -\texttt{FaceIsoDirection}(\Sigma, \mathrm{verts}(\sigma), -s)$. Note that we are negating s as well as the direction returned by `FaceIsoDirection` (Algorithm 3.1). Again by Properties 1 and 2, there exists $c_2 \in \mathbb{R}$ such that all vertices of σ are at height c_2 and v is above c_2, in $-s_2$. Thus, $v \notin H^{\downarrow}(s_1, \sigma)$ and $v \in H^{\downarrow}(s_2, \sigma)$, which means that $v \in \mathbb{W} := H^{\downarrow}(s_1, \sigma) \triangle H^{\downarrow}(s_2, \sigma)$.

Next, we show that v is the only vertex from $\mathbb{K}_0 \setminus \mathrm{verts}(\tau)$ in $\mathbb{W}$. Assume, for contradiction, that there exists a vertex $v' \in \mathbb{K}_0 \setminus \Sigma$ such that $v' \in \mathbb{W}$. Then, $c_1 < s_1 \cdot v'$ and by Property 3 and Property 4 of Lemma 5, $s \cdot w < s \cdot v'$ for all $w \in \mathrm{verts}(\tau)$. Furthermore,

$c_2 < -s_2 \cdot v'$ which, by Property 3 and Property 4 of Lemma 5, implies that $-s \cdot w < -s \cdot v'$ which simplifies to $s \cdot w > s \cdot v'$ for all $w \in \mathrm{verts}(\tau)$, a contradiction. Therefore, v is the only vertex in $\mathbb{W}$, as required.

The previous two paragraphs show that v is the only vertex in the wedge $\mathbb{W}$. We now show that the k-indegree can be used to determine if Σ defines a simplex in $\mathbb{K}$. Recall that every k-dimensional coface of σ must contain all of the vertices of σ, plus exactly one more. Since $v \notin H^{\downarrow}(s_1, \sigma)$ and $v \in H^{\downarrow}(s_2, \sigma)$, every simplex that contributes to the k-indegree of σ in direction s_2 also contributes to the k-indegree of σ in direction s_1. In addition, the only potential simplex contributing to the k-indegree of σ in direction s_2 that does not contribute to the k-indegree in direction s_1 is the one defined by $\mathrm{verts}(\sigma)$ and v. Thus, Line 6 of Algorithm 3.3 returns `True` iff Σ defines a k-simplex.

We compute a direction normal to $\mathrm{aff}(\tau \cup \{v\})$ in $O(dn_0 + d^2)$. Then, we call Algorithm 3.1 three times, each costing $O(n_0(d^5 + \log n_0))$ time and computing no new diagrams by Lemma 5. Finally, we compute the k-indegree from two directions, each call taking time $O(2^{i+1}(n_0(n_{k-1} + d^5 + \log n_0) + \Pi + 2^i))$ and $2^{i+1} - 1$ diagrams by Theorem 2. Moreover, we see that $i = k - 1$ and we simplify our complexity to $O(2^k(n_0(n_{k-1} + d^5 + \log n_0) + \Pi + 2^{k-1}))$ time and $2(2^k - 1) = 2^{k+1} - 2$ diagrams. $\square$

3.2.3 Contributions to Section 3.2

In Section 3.2, we have presented predicates that make it possible to detect a simplex of arbitrary dimension, the results originally appeared in [56]. This idea was, initially, an extension of the "wedge" formulation used in [11, 12]. However, Samuel Micka realized that the notion of indegree did not naturally extend to higher dimensions by discovering scenarios where extra simplices were counted in the computation. Samuel Micka found that an inclusion-exclusion approach could be used to properly compute the indegree and made initial attempts at providing an algorithm and proofs to support the idea. However, an error

Algorithm 3.3: `CheckSimplex`(σ, v)

Input: $\sigma \in \mathbb{K}_{k-1}$, a vertex $v \in \mathbb{K}_0 \setminus \mathrm{verts}(\sigma)$

Output: `True` if $\sigma \cup \{v\} \in \mathbb{K}_k$, `False` otherwise

1: $k \leftarrow$ dimension of $\sigma + 1$

2: $V, s_t \leftarrow$ value returned from Lemma 7 with parameter $W = \mathrm{verts}(\sigma) \cup \{v\}$

3: $s \leftarrow$ `FaceIsoDirection`$(V, \mathrm{verts}(\sigma) \cup \{v\}, s_t)$

4: $s_1 \leftarrow$ `FaceIsoDirection`$(\mathrm{verts}(\sigma) \cup \{v\}, \mathrm{verts}(\sigma), s)$

5: $s_2 \leftarrow -$`FaceIsoDirection`$(\mathrm{verts}(\sigma) \cup \{v\}, \mathrm{verts}(\sigma), -s)$

6: **return** $|$`ComputeIndegree`$(\sigma, s_1, k, [\,]) -$ `ComputeIndegree`$(\sigma, s_2, k, [\,])| = 1$

was found by David L. Millman and Brittany Terese Fasy, who helped correct the problem. Samuel Micka was deeply involved in proving the statements in this section, writing them up, and refining them. The updated statements required some additional lemmas (such as Lemma 7 and Lemma 3) which were primarily addressed by David L. Millman and Anna Schenfisch. It should be noted that the predicates developed in this section were the key to reconstruction in higher dimensions and this served as a bottleneck for moving past embedded graphs for over a year. As such, developing the work in Section 3.2 was a significant breakthrough in this research. We summarize more specific contributions from Samuel Micka as follows and note that statements not listed here are statements that Samuel Micka feels he did not make significant contributions to:

- Samuel Micka, David L. Millman, and Brittany Terese Fasy all worked closely together on developng the theory for Lemma 5 to ensure that all of the properties needed for the predicate would be provided by this lemma. Samuel Micka spent significant time in this lemma formalizing the properites, removing/combining properties, and refining proofs and statements after reviewer comments. It should be

noted that the runtime for this lemma was determined primarily by David L. Millman.

- The theory for Lemma 6 was developed by the group with Brittany Terese Fasy making an initial attempt at writing up the lemma and further revisions were made by Samuel Micka. An error was found in the lemma by anonymous reviewers and the proof was corrected by Samuel Micka.

- The theory for Theorem 1 was initially developed and written by Samuel Micka and further refined by Brittany Terese Fasy using a new formulation of Lemma 6. The proof was reviewed and refined by Samuel Micka.

- Theory for Theorem 2 was developed and written by Samuel Micka. Specifically, Samuel Micka suggested the the idea of using memoization to simplify the proof of complexity was wrote up the results.

- Theorem 3 was worked on primarily by Brittany Terese Fasy and Samuel Micka, building on the previous lemmas to generalize the idea of a "wedge" to higher dimensions. It should be noted that this idea has been around since we decided to generalize to higher dimensions. However, it was not until the predicates in this section were developed that we were able to formalize the idea.

3.3 Reconstructing Zero-Simplices

In this section, we consider the problem of reconstructing the zero-skeleton of a simplicial complex using APDs returned from the `Oracle`. Specifically, given a simplicial complex $\mathbb{K}$ embedded in $\mathbb{R}^d$, we will describe two algorithms which return $\mathbb{K}_0$. These two algorithms trade off between diagram and time complexity and are described in Section 3.3.2 and Section 3.3.3 respectively. First, we introduce proofs from our previous work in [11] to assist with the proofs in Section 3.3.3.

3.3.1 Previous Work in $\mathbb{R}^2$

In this section, we consider lemmas and theorems from our original work [11] that offer a method of reconstructing $V = \mathbb{K}_0$ of a simplicial complex $\mathbb{K} \subset \mathbb{R}^2$ which we build on for higher dimensions. Some of these statements and proofs also appear (in an updated form) in [12]. We include our proofs to offer insight into how the approach developed to higher dimensions, i.e, in [12, 56]. First, we introduce [11, Lemma 3].

Lemma 8 (Vertex Existence Lemma)**.** *Let $\mathbb{K}$ be a simplicial complex with vertex set V of size n_0. Let $s_1, s_2 \in \mathbb{S}^1$ be linearly independent and further suppose that $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ each contain n_0 lines. Let A be the collection of vertices at the intersections of lines in $\mathbb{L}(s_1, V) \cup \mathbb{L}(s_2, V)$. Let $s_3 \in \mathbb{S}^1$ such that for all $u, v \in A$, $\ell_3(u) = \ell_3(v) \iff u = v$. Then, the following two statements hold true:*

(1) $v \in V \iff \ell_3(v) \in \mathbb{L}(s_3, V)$ *and* $A \cap \ell_3(v) = \{v\}$

(2) *For all* $\ell \in \mathbb{L}(s_3, V)$, $A \cap \ell \neq \emptyset$.

Proof. First, we prove Part (1).

($\Rightarrow$) Let $v \in V$. Then, $\ell_i(v) \in \mathbb{L}(s_i, V)$ and $v \in \ell_i(v)$ for $i = \{1, 2, 3\}$. Hence, $\ell_1(v) \cap \ell_2(v) \cap \ell_3(v) = \{v\}$, as desired.

($\Leftarrow$) Assume, for the sake of contradiction, that $\ell_3(v) \in \mathbb{L}(s_3, V)$ and $A \cap \ell_3(v) = \{v\}$, yet $v \notin V$. Since $\ell_3(v) \in \mathbb{L}(s_3, V)$ and $v \notin V$, some other vertex $u \in V$ must have height $\mathbb{H}_{s_3}(v)$. Since $u \in V$, we know $\ell_i(u) \in \mathbb{L}(s_i, V)$ for $i \in \{1, 2, 3\}$. And, by ($\Rightarrow$) applied to u, we know $u \in A$. Since $\ell_3(u) = \ell_3(v)$, both u and v are in A and on the line $\ell_3(v)$, but $u \neq v$, which is a contradiction.

Next, we prove Part (2) of the lemma. Assume, for contradiction, that there exists $\ell \in \mathbb{L}(s_3, V)$ such that $A \cap \ell = \emptyset$. As $\ell \in \mathbb{L}(s_3, V)$, a vertex $v \in V$ exists such that $\ell = \ell_3(v)$ and v lies on ℓ. However, $v \in \ell_1(v) \cap \ell_2(v) \subset A$, which is a contradiction. $\square$

Next, we explain how to pick a third direction (with the first two being e_1 and e_2) satisfying these properties so that we can determine vertex locations. This result comes from our work [11, Lemma 4].

Lemma 9 (Vertex Localization)**.** *Let L_H and L_V be n_0 horizontal and n_0 vertical lines, respectively. Let w (and h) be the largest (and smallest) distance between two lines of L_V (and L_H, respectively). Let B be the smallest axis-aligned bounding box containing the intersections of lines in $L_H \cup L_V$. For $0 < \varepsilon < h$, let $s = (w, h - \varepsilon)$. Any line parallel to s can intersect at most one line of L_H in B.*

Proof. Note that, by definition, s is a vector in the direction that is at a slightly smaller angle than the diagonal of the box of size w by h. Assume, by contradiction, that a line parallel to s may intersect two lines of L_H within B. Specifically, let $\ell_1, \ell_2 \in L_H$ and let ℓ_s be a line parallel to s such that the points $\ell_i \cap \ell_s = (x_i, y_i)$ for $i = \{1, 2\}$ are the two such intersection points within B. Notice since the lines of L_H are horizontal and by the definition of h, we observe that $|y_1 - y_2| \geq h$. Let $w' = |x_1 - x_2|$, and observe $w' \leq w$. Since the slope of ℓ_s is $(h - \varepsilon)/w$, we have $|y_1 - y_2| < h$, which is a contradiction. $\square$

Finally, combining the previous two lemmas, we describe a method for reconstructing vertices in $\mathbb{R}^2$ from [11, Theorem 5]. We add the word augmented to make the results consistent with the rest of the work described in this dissertation.

Theorem 4 (Vertex Reconstruction)**.** *Let $\mathbb{K}$ be a plane graph. We can can compute the coordinates of all n_0 vertices of $\mathbb{K}$ in $O(n_0 \log n_0)$ time from three directional augmented persistence diagrams.*

Proof. Let $s_1 = e_1$ and $s_2 = e_2$, which are linearly independent. We compute the filtration lines $\mathbb{L}(s_i, V)$ for $i = 1, 2$ in $O(n_0)$. By our general position assumption, no two vertices of $\mathbb{K}$ share an x- or y-coordinate. Thus, the sets $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ each contain n_0

distinct lines. Let A be the set of intersection points of the lines in $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$. The next step is to identify a direction s_3 such that each line in $\mathbb{L}(s_3, V)$ intersects with only one point A, so that we can use Lemma 8.

Let w (and h) be the greatest (and least) distance between two adjacent lines in $\mathbb{L}(s_1, V)$ (and $\mathbb{L}(s_2, V)$, respectively). Let B be the smallest axis-aligned bounding box containing A, and let $s_* = (w, \frac{h}{2})$. By Lemma 9, any line parallel to s_* will intersect at most one line of $\mathbb{L}(s_2, V)$ in B. Thus, we choose $s_3 \in \mathbb{S}^1$ that is perpendicular to s_*. By the second part of Lemma 8, we now have that each line in $\mathbb{L}(s_3, V)$ intersects A. Thus, there are n_0 intersections between $\mathbb{L}(s_2, V)$ and $\mathbb{L}(s_3, V)$ in B, each of which also intersects with $\mathbb{L}(s_1, V)$.

The previous paragraph leads us to a simple algorithm for finding the third direction and identifying all the triple intersections. In the analysis below, steps that do not mention a number of diagrams, use no diagrams. First, we construct $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ in $O(n_0)$ time using two directional augmented persistence diagrams. Second, we sort the lines of $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ by their x- and y-intercepts respectively in $O(n_0 \log n_0)$ time. Third, we find s_3 by computing w and h from our sorted lines in $O(n_0)$ time. Fourth, we construct $\mathbb{L}(s_3, V)$ in $O(n)$ time using one directional augmented persistence diagram. Fifth, we sort the lines in $\mathbb{L}(s_3, V)$ by their intersection with the leftmost line of $\mathbb{L}(s_1, V)$ in $O(n_0 \log n_0)$ time. Finally, we compute coordinates of the n_0 vertices by intersecting the i-th line of $\mathbb{L}(s_2, V)$ with the i-th line of $\mathbb{L}(s_3, V)$ in $O(n_0)$ time. (Observe, this last step works since the vertices correspond to the n_0 intersections in B, as described above).

Therefore, we use three directional augmented diagrams (two in the first step and one in the fourth step) and $O(n_0 \log n_0)$ time (sorting of lines in the second and fifth steps) to reconstruct the vertices. $\square$

3.3.2 Minimizing Diagram Complexity

Here, we describe a method of reconstructing $V = \mathbb{K}_0$ in $\mathbb{R}^d$ using APDs from the `Oracle` which minimizes diagram complexity at the cost of higher time complexity. The results in this subsection were originally introduced in [12].

First, we present a lemma which enables us to identify points at which vertices lie. Lemma 10 proves that we can choose a direction with n_0 filtration hyperplanes in which each of the hyperplanes intersect at least one potential vertex location.

Lemma 10 (Generalized Vertex Existence)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a straight-line embedded graph in $\mathbb{R}^d$. Let $s_1, s_2, \ldots, s_d$ be linearly independent directions in $\mathbb{S}^{d-1}$ and further suppose that $\mathbb{L}(s_i, V)$ contains n_0 filtration hyperplanes for each $i \in \{1, 2, \ldots, d\}$. Choosing one hyperplane in each set $\mathbb{L}(s_i, V)$, the intersection of these hyperplanes is a point. Let A denote the set of n_0^d such intersection points. Let $s_{d+1} \in \mathbb{S}^{d-1}$ such that each $a \in A$ has a unique height in direction s_{d+1}. Then, the following equality holds: $V = \mathbb{L}(s_{d+1}, V) \cap A$.*

Proof. We prove this equality in two steps. First, we prove $V \subseteq \mathbb{L}(s_{d+1}, V) \cap A$. Let $v \in V$. Then, for $i \in \{1, 2, \ldots, d+1\}$, there exists $\ell_i(v) \in \mathbb{L}(s_i, V)$ such that $v \in \ell_i(v)$. Thus, $v \in \cap_{i=1}^{d+1} \ell_i(v)$, as was to be shown.

Next, we prove $V \supseteq \mathbb{L}(s_{d+1}, V) \cap A$. Assume, for contradiction, that there exists a filtration hyperplane $\ell \in \mathbb{L}(s_{d+1}, V)$ such that $A \cap \ell = \emptyset$. Since $\ell \in \mathbb{L}(s_{d+1}, V)$, there exists a vertex $v \in V$ such that $\ell = \ell_{d+1}(v)$ and v lies on ℓ. However, since $V \subset A$ from above, v is in $\cap_{i=1}^{d} \ell_i(v) = A$, contradicting the hypothesis $A \cap \ell = \emptyset$. $\square$

Next we describe a method for choosing a direction (Algorithm 3.4) which will enable us to reconstruct the vertices. The correctness of Algorithm 3.4 is proven in Lemma 11.

Lemma 11 (Generalized Vertex Localization)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Choosing one hyperplane in each set $\mathbb{L}(e_i, V)$ for $1 \leq i \leq d$, the intersection of these*

Algorithm 3.4: `findDGMVertsDirection`(L)

Input: L a list, where L_i contains the heights of $\mathbb{L}(e_i, V)$.

Output: Direction $s \in \mathbb{S}^{d-1}$ satisfying conditions of Lemma 11.

```
h ← ∞
w ← 0
for i ∈ {1, 2, ..., d} do
    H_i = {h_1^(i), h_2^(i), ..., h_{n_0}^(i)} ← V^(i) sorted in ascending order
    minTemp ← min_{j∈{2,3,...,d}} h_j^(i) − h_{j−1}^(i)
    maxTemp ← h_{n_0}^(i) − h_1^(i)
    if w < maxTemp then
        w ← maxTemp
    if h > minTemp then
        h ← minTemp
h ← (1/2) h
x ← [−1/w, ..., −1/w, (d−1)/h]^⊤
return s ← x/||x||
```

hyperplanes is a point. Let A denote the set of n_0^d such intersection points. Then, Algorithm 3.4 returns a direction $s \in \mathbb{S}^{d-1}$ such that each hyperplane in $\mathbb{L}(s, V)$ intersects at most one of the n_0^d points in A in $\Theta(dn_0 \log n_0)$ time.

Proof. Let $L_i = \{h_1^{(i)}, h_2^{(i)}, \ldots, h_n^{(i)}\}$ be the ordered set of heights of hyperplanes in $\mathbb{L}(e_i, V)$. Let $w^{(i)} = h_{n_0}^{(i)} - h_1^{(i)}$; in other words, $w^{(i)}$ is the largest distance between any two hyperplanes in the set $\mathbb{L}(e_i, V)$, and let $w = \max_{1 \le i \le d} w^{(i)}$. Let $h^{(i)} = \min\{h_j^{(i)} - h_{j-1}^{(i)}\}_{j=2}^{n}$; in other words, $h^{(i)}$ is the smallest height difference between any two (adjacent) hyperplanes in the set $\mathbb{L}(e_i, V)$, and let $h = \frac{1}{2} \min_{1 \le i \le d} h^{(i)}$. Then, we

consider the hyperplane that intersects the origin and each point $we_i + \langle 0, \ldots 0, \frac{h}{d-1} \rangle$ for $i \in \{1, 2, \ldots, d-1\}$. Next, we choose a vector orthogonal to the hyperplane. Without loss of generality, we choose

$$x = \left\langle -\frac{1}{w}, \ldots, -\frac{1}{w}, \frac{d-1}{h} \right\rangle^{\top}$$

and observe that $(we_i + \langle 0, \ldots, 0, \frac{h}{d-1} \rangle) \cdot x = 0$ for each $i \in \{1, 2, \ldots, d-1\}$. Finally, we define $s = \frac{x}{||x||}$.

We now show that s satisfies the claim that each hyperplane in $\mathbb{L}(s, V)$ intersects at most one of the n_0^d points in A and that we can find s in $\Theta(dn_0 \log n_0)$ time. Let $p = (p^{(1)}, p^{(2)}, \ldots, p^{(d)})$ and $q = (q^{(1)}, q^{(2)}, \ldots, q^{(d)})$ be points in A with $p \neq q$, and assume, for contradiction, that they lie on the same hyperplane in $\mathbb{L}(s, V)$. Then, $p \cdot s = q \cdot s$. By the definition of dot product, we have the following equation (after multiplying s by $||x||$):

$$\frac{d-1}{h}(p^{(d)} - q^{(d)}) - \sum_{i=1}^{d-1} \frac{1}{w}(p^{(i)} - q^{(i)}) = 0.$$

Since $\frac{d-1}{h}$ and w are positive numbers, we can rearrange this equality to obtain:

$$\left|p^{(d)} - q^{(d)}\right| = \frac{h}{w(d-1)} \left| \sum_{i=1}^{d-1} (p^{(i)} - q^{(i)}) \right|. \tag{3.5}$$

Recall that $h = \frac{1}{2} \min_{1 \leq i \leq d} h_i \leq \frac{1}{2} h_d$. Therefore, we know that $2h \leq h_d \leq |p^{(d)} - q^{(d)}|$. Applying Equation 3.5 to this inequality, we obtain:

$$\begin{aligned} 2h &\leq \frac{h}{w(d-1)} \left| \sum_{i=1}^{d-1} (p^{(i)} - q^{(i)}) \right| \\ &\leq \frac{h}{w(d-1)} (d-1) \max_{1 \leq i \leq d-1} |(p^{(i)} - q^{(i)})| \\ &\leq \frac{h}{w} w. \end{aligned}$$

Thus, we have $2h < h$, which is a contradiction when $n \geq 2$. For the case where $n = 1$, the vertex is $(h_1^{(1)}, h_1^{(2)}, \ldots, h_1^{(d)})$ and the $(d+1)$st direction is not needed.

We analyze the complexity of computing w and h. For each direction e_i, we perform three steps. First, we sort the heights of $\mathbb{L}(e_i, V)$ in $\Theta(n_0 \log n_0)$ time. Second, we compute $w^{(i)}$ in constant time (as it is the maximum value minus the minimum value of the heights). Third, we compute $h^{(i)}$ in $\Theta(n_0)$ time. As there are d dimensions, computing the sets $\{w^{(i)}\}$ and $\{h^{(i)}\}$ takes $\Theta(dn \log n)$ time. Computing w and h from the sets $\{w^{(i)}\}$ and $\{h^{(i)}\}$ is $\Theta(d)$ time. Thus, the bottleneck is sorting in each direction, which makes the total runtime $\Theta(dn_0 \log n_0)$. □

Equipped with the above method for finding a suitable $(d+1)^{\text{st}}$ direction to locate vertices in higher dimensions, we conclude this section with a theorem describing the algorithm to compute the coordinates of the vertices of the original simplicial complex.

Algorithm 3.5: `FindVerticesDGM()`

Input: None (but makes calls to global `Oracle` for unknown simplicial complex)

Output: $V = \mathbb{K}_0$

```
L ← []
for i ∈ {1, 2, ..., d} do
    L_i ← heights of filtration hyperplanes in L(e_i, V)
s ← findDGMVertsDirection(L)
A ← all n_0^d intersection points of L(e_i, V) for i ∈ {1, 2, ..., n_0}
for ℓ ∈ L(s, V) do
    for p ∈ A do
        if p ∈ ℓ then
            add p ∩ ℓ to V
return V
```

Theorem 5 (Higher-dimensional Vertex Reconstruction)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. We can can compute the coordinates of all n_0 vertices of $\mathbb{K}$ with Algorithm 3.5 using $d+1$ directional augmented persistence diagrams in $\Theta(dn_0^{d+1} + d\Pi)$ time.*

Proof. Let `Oracle` be the oracle that takes a direction $s \in \mathbb{S}^{d-1}$ and returns the $\overline{D}_0(s)$ in $\Theta(\Pi)$ time. For $i \in \{1, 2, \ldots d\}$, we use this oracle to obtain $\overline{D}_0(e_i)$. Note that, by Assumption 1 (General Position) and Corollary 3, for each of these directions, we have exactly n_0 distinct filtration hyperplanes, in one-to-one correspondence with the vertices. Note that, for a given direction e_i, we store the filtration hyperplanes as a list of the vertex heights. Choosing one hyperplane in each direction yields d pairwise orthogonal hyperplanes; their intersection is a point in $\mathbb{R}^d$ and this point is a potential vertex location. In total, we have n_0^d potential vertex locations, of which only n_0 are actual vertices. We denote this set of n_0^d potential vertex locations by A. By Corollary 3, $\overline{D}_0(s)$ has at least n_0 points. Thus, computing these lists of vertex heights takes $\Theta(\Pi + n_0)$ time per dimension on Line 3 to account for computing and listing the points of the APD.

Let s be chosen as in Lemma 11 in $\Theta(dn_0 \log n_0)$ time. By Lemma 11, each hyperplane $\ell_s(v)$ intersects at most one point in A for each $v \in V$. Thus, by Lemma 10, there are exactly n_0 distinct intersections between $\mathbb{L}(s, V)$ and A, in one-to-one correspondence with the n_0 vertices of $\mathbb{K}$.

Then, to identify vertex locations in $\mathbb{R}^d$, we employ the following brute force algorithm. We check each element $v \in A$ for intersections with any hyperplane $\ell \in \mathbb{L}(s, V)$, this operation is seen on Line 8. Since $|A| = n_0^d$ and $|\mathbb{L}(s, V)| = n_0$, we have n_0^{d+1} checks that we must perform in the nested loops on Line 6 and Line 7, with each check taking $\Theta(d)$ time. Thus, the total time complexity of calculating V from the $d+1$ sets of filtration hyperplanes is $\Theta(dn_0^{d+1})$ and no additional diagrams are computed.

In total, this algorithm uses $d+1$ directional diagrams. The time complexity of constructing the $d+1$ sets of filtration hyperplanes is $\Theta(dn_0 \log n_0 + d\Pi + dn_0)$, and

an additional $\Theta(dn_0^{d+1})$ time to compute the actual vertex locations. Thus, the total time complexity is $\Theta(dn_0^{d+1} + d\Pi)$. $\square$

Algorithm 3.5 yields a method for reconstructing $\mathbb{K}_0$ while minimizing the number of diagrams used in the reconstruction. Next, we consider an alternative approach to reconstructing $\mathbb{K}_0$ which uses a larger number of diagrams, but has a much smaller time complexity in terms of n_0.

3.3.3 Minimizing Time Complexity

In this section, we introduce Algorithm 3.7, which provides an alternative means of reconstructing $\mathbb{K}_0$ which minimizes time complexity at the cost of diagram complexity. Algorithm 3.7 starts by choosing an initial direction. This choice can be arbitrary, so we choose the last cardinal direction, e_d. Next, for each coordinate position i, we call Algorithm 3.6 to find the ith coordinates of all vertices, denoted $V^{(i)}$, using only two APDs specifically chosen for those coordinates. These results initially appeared in [56].

Lemma 12 (Two-Diagram Vertex Coordinate Localization)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex for $d \geq 2$ and $i \in \{1, 2, \ldots, d-1\}$. If the dth coordinate of all vertices in $V = \mathbb{K}_0$ are known, i.e., we have $V^{(d)}$, then Algorithm 3.6 returns $V^{(i)}$ using two additional directional augmented persistence diagrams in time $\Theta(n_0 \log n_0 + \Pi)$.*

Proof. The main idea behind the correctness of Algorithm 3.6 is that the direction $s \in \mathbb{S}^{d-1}$ chosen on Line 6 maintains the vertex ordering as in direction e_d. Then, we can solve a single equation on Line 10 for the ith coordinate of each vertex by intersecting corresponding hyperplanes from $\mathbb{L}(e_i, K_0)$ and $\mathbb{L}(s, K_0)$.

We first examine the construction of s and show how it maintains the vertex ordering from e_d. In Line 2, we let $h = \frac{1}{2}\min_{j=2,\ldots,n_0}\{v_{j-1}^{(d)} - v_j^{(d)}\}$; in other words, h is half the minimum distance between any two hyperplanes in $\mathbb{L}(e_d, K_0)$. Similarly, in Line 5, we let

Algorithm 3.6: `FindCoordinates`$(i, V^{(d)})$

Input: $i \in \mathbb{N}$ such that $0 \leq i < d$, $V^{(d)}$ the dth coordinate of all vertices

Output: $V^{(i)}$

1: $H_d \leftarrow V^{(d)}$ sorted in ascending order

2: $h \leftarrow \frac{1}{2} \min_{j=2,\ldots,n_0} \{H_d[j-1] - H_d[j]\}$

3: $\overline{D}_0(e_i) \leftarrow \texttt{Oracle}_0(e_i)$

4: $H_i \leftarrow$ births in $\overline{D}_0(e_i)$ sorted in ascending order

5: $w \leftarrow$ smallest distance between elements of H_i

6: $s \leftarrow$ normalized vector with $-h$ as i^{th} coordinate, w as d^{th} coordinate, and zeros elsewhere

7: $\overline{D}_0(s) \leftarrow \texttt{Oracle}_0(s)$

8: $H_s \leftarrow$ births in $\overline{D}_0(s)$ sorted in ascending order

9: **for** $j \in \{1, 2, \ldots, n_0\}$ **do**

10: $\quad v_j^{(i)} \leftarrow \frac{1}{h}(wH_d[j] - H_s[j])$

11: **return** $V^{(i)}$

w be the maximum distance between hyperplanes in $\mathbb{L}(e_i, K_0)$. Then s is the normalized vector with $-h$ as its ith coordinate, w as its dth coordinate, and zeros elsewhere. We now consider $\pi(s)$, $\pi(e_d)$, and $\pi(\mathbb{L}(s, K_0) \cup \mathbb{L}(e_d, K_0))$, the projection of s, e_d, and $\mathbb{L}(s, K_0) \cup \mathbb{L}(e_d, K_0)$ into the (e_i, e_d)-plane. By construction, the lines $\pi(\mathbb{L}(s, K_0))$ are parallel to (w, h) in the (e_i, e_d)-plane, and so by combining Lemma 9 and Lemma 8 as in the proof of Theorem 4, $\pi(\mathbb{L}(s, K_0))$ intersect each of $\pi(\mathbb{L}(e_d, \mathbb{K}_0))$ in the same order. Furthermore, since s has zero in all coordinates except i and d, the order is maintained in $\mathbb{R}^d$ as well.

Now, we show how we can use this ordering to compute $v^{(i)}$. Assume that the hyperplanes in $\mathbb{L}(e_d, K_0)$ and $\mathbb{L}(s, K_0)$ are ordered by their heights in e_d and s respectively as in Line 1 and Line 8. Suppose, without loss of generality, that v is the jth vertex with

respect to the e_d direction. Then, v lies on the jth hyperplane of $\mathbb{L}(e_d, K_0)$ and the jth hyperplane of $\mathbb{L}(s, K_0)$, meaning that v must lie in their intersection. To compute $v^{(i)}$, we consider the equations of the two hyperplanes. Recall that the jth hyperplane in $\mathbb{L}(s, K_0)$ is the set of all points $x = (x^{(1)}, x^{(2)}, \ldots, x^{(d)}) \in R^d$ at some height. Call this height $c_{j,s}$. Then, the jth hyperplane in $\mathbb{L}(s, \mathbb{K}_0)$ is described by the equation

$$-hx^{(i)} + wx^{(d)} = c_{j,s}. \tag{3.6}$$

Let the height of the jth hyperplane in $\mathbb{L}(e_d, \mathbb{K}_0)$ be c_{j,e_d}. Then, the equation for that height is

$$c_{j,e_d} = x \cdot e_d = x^{(d)}. \tag{3.7}$$

Combining Equations 3.6–3.7 and solving for $x^{(i)}$, we get

$$x^{(i)} = \frac{1}{h}(wc_{j,e_d} - c_{j,s}).$$

Because v is in the intersection of these hyperplanes, $v^{(i)} = \frac{1}{h}(wc_{j,e_d} - c_{j,s})$. Since j was arbitrary, the same process works for all zero-simplices in $\mathbb{K}_0$. Thus, the for loop on Line 9 computes the ith coordinate of all zero-simplices and $V^{(i)}$ is returned on Line 11.

It takes $\Theta(n_0 \log n_0)$ time to sort the vertices on Line 1, Line 4, and Line 8 and $\Theta(\Pi)$ time to compute the two diagrams on Line 3 and Line 7. All other operations in Algorithm 3.6 take linear or constant time. Thus, the total runtime is $\Theta(n_0 \log n_0)$ and only two APDs are generated on Line 3 and Line 7, as required. □

We are now ready to present Algorithm 3.7 for reconstructing all of the vertices of $\mathbb{K}$.

Theorem 6 (Reconstructing Zero-Simplices)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Then, Algorithm 3.7 reconstructs the vertex locations of all $v \in \mathbb{K}_0$ using $2d - 1$ diagrams in*

Algorithm 3.7: `FindVerticesTIME()`

Input: None (but makes calls to global `Oracle` for unknown simplicial complex)

Output: $V = \mathbb{K}_0$

1: $\overline{D}_0(e_d) \leftarrow \texttt{Oracle}_0(e_d)$

2: $V \leftarrow$ list of n d-dimensional points, with all coordinates set to `null`

3: $V^{(d)} \leftarrow$ birth times in $\overline{D}_0(e_d)$

4: **for** each i in dimensions 1 to $d-1$ **do**

5: $\quad V^{(i)} \leftarrow \texttt{FindCoordinates}(i, V^{(d)})$

6: **return** V

$\Theta(dn_0 \log n_0 + d\Pi)$ *time.*

Proof. To prove the correctness of Algorithm 3.7, we define a loop invariant. We show that at the start of iteration i of the main loop of the algorithm (Lines 4–5), we know the coordinates $v^{(1)}, v^{(2)}, \ldots, v^{(i-1)}$ and $v^{(d)}$ for all $v \in \mathbb{K}_0$.

In the base case (iteration $i = 1$), we know only the dth coordinate of each vertex and the invariant is satisfied. Then, for $1 < k \leq d-1$, assume that we know all coordinates for all $v \in \mathbb{K}_0$ up to coordinate $k-1$ in iteration $k-1$. Then, we show that the invariant holds true for k as well. By Lemma 12, the call to Algorithm 3.6 on Line 5 returns $V^{(k)}$. Furthermore, by the inductive assumption, we know all coordinates for each vertex up to coordinate $k-1$. Thus, after the kth iteration, we know all coordinates for each vertex up to the kth coordinate, as required. The algorithm terminates after $(d-1)$ iterations. Since the loop invariant holds in the base case and in the inductive step, and the algorithm terminates, the algorithm is correct.

Finding the dth coordinate of each vertex requires a single diagram, $\overline{D}_0(e_d)$ on Line 1. By Lemma 12, each call to Algorithm 3.6 generates two APDs and takes time $\Theta(n_0 \log n_0 + \Pi)$. Algorithm 3.6 is called $d-1$ times, generating a total of $1+2(d-1) = 2d-1$ diagrams

and taking time $\Theta(dn_0 \log n_0 + d\Pi)$. $\square$

Now, with two methods of reconstructing $\mathbb{K}_0$, we move on to describe an efficient algorithm for reconstructing $\mathbb{K}_1$ in embedded graphs in Section 3.4 and then a general reconstruction algorithm for simplicial complexes in Section 3.5.

3.3.4 Contributions to Section 3.3

In Section 3.3, we have presented two methods for reconstructing vertex locations using APDs. Both ideas are generalized from the vertex reconstruction approaches and conjectures in our previous work [11,12]. In a broad sense, the initial idea for the algorithm prioritizing time-complexity in Section 3.3.3 was developed through discussions with the research group and our two-dimensional approach in [11]. Many discussions were led by Samuel Micka throughout the Fall of 2018 and included in the paper [56]. Samuel Micka formalized the idea and initial write ups were done by Lucia Williams. Further updates were made by Samuel Micka and others from the research group. The algorithm prioritizing diagram complexity in Section 3.3.2 was an idea conjectured to be true by the entire research group upon completing the submission of [11]. However, Samuel Micka, assisted by discussions with Brittany Terese Fasy, formalized the idea, wrote the statements and proofs, and revised the work found in Section 3.3.2 and [12]. Lucia Williams and Anna Schenfisch helped with revisions and further discussions on later drafts of the statements and proofs found in Section 3.3.2. We note that the contents of Section 3.3.1 come from [11] and are included to add clarity to proofs in Section 3.3.3 and later in Section 3.6.

3.4 Reconstructing One-Simplices

In this section, we consider reconstructing the one-skeleton (or edges) of simplical complexes. We may refer to the one-skeleton as embedded graphs without loss of generality. We note that the results of this section follow primarily from the work found

in [56]. Using one of the algorithms from Section 3.3, we reconstruct the vertices of an unknown simplicial complex and assume that these are known apriori in the algorithms to follow. Results for embedded graphs are stated in Corollary 5 and Corollary 7.

First, we note that an edge reconstruction algorithm for embedded graphs was described in our research [12] with diagram complexity $n_0^2 - n_0$. Here, we describe an output sensitive algorithm, first introduced in our unpublished paper [56], with diagram complexity $O(n_1 \log n_0)$. Intuitively, we use a hyperplane sweep in the e_2 direction where events occur at vertices; thus, we may use the word "above (below)" as shorthand for "above (below) with respect to the e_2 direction". We discover edges incident to each vertex $v \in V$ by determining regions where potential edges lie and logarithmically search the region with information about edges already discovered below v.

The hyperplane sweep may be easier to visualize as a line sweep in $\mathbb{R}^2$. Many of our descriptions utilize this tool, so we begin by defining a projection of $\mathbb{K} \subset \mathbb{R}^d$ to $\mathbb{R}^2$:

Definition 16 (Projection to $\mathbb{R}^2$)**.** *For $d \geq 2$, let $\pi \colon \mathbb{R}^d \to \mathbb{R}^2$ be the projection onto the first two coordinates; that is, $\pi(x^{(1)}, x^{(2)}, \ldots, x^{(d)}) = (x^{(1)}, x^{(2)})$. Note that the image of π is the plane spanned by e_1 and e_2, and $\pi(x)$ is the orthogonal projection of x onto that plane.*

All operations for reconstruction *are* taking place in $\mathbb{R}^d$. However, the general position ensures that $\deg(v) = \deg(\pi(v))$ and, moreover, that no three vertices are collinear in $\pi(\mathbb{R}^d)$ by Assumption 1, so discussing edge reconstruction in $\pi(\mathbb{R}^d)$ is reasonable.

To keep track of regions containing edges incident to a vertex v, we introduce an *edge interval object* `eI`, which contains an ordered list of vertices radially sorted cw about $\pi(v)$ that are in the interval `eI`, denoted `eI.V`, and a count `eI.count` representing the number of vertices in `eI.V` that are adjacent to v. Intuitively, the `eI` object allows us to determine intervals where potential edges lie. Then, using information we know about edges below

v, we split the wedges, efficiently searching for intervals containing exactly one vertex that have a positive `eI.count` value. In order to search the edge interval in logarithmic time, we define Algorithm 3.8 and use it to converge on these edge intervals containing a single vertex with a count of one. We converge by searching intervals in a clockwise fashion and may refer to vertices and intervals to the "left" as meaning they lie before the current interval or split point in the clockwise ordering or to the "right" to mean that they lie after.

Algorithm 3.8: $\texttt{SplitInterval}(v, \texttt{eI}, \texttt{E}_v, V_v)$

Input: $v \in \mathbb{K}_0$, edge interval `eI`, and $\texttt{E}_v$, a list of known vertices adjacent to v sorted cw about $\pi(v)$, V_v containing all vertices sorted cw around $\pi(v)$

Output: edge intervals $\texttt{eI}_\ell$ and $\texttt{eI}_r$.

1: $\texttt{split}^\uparrow \leftarrow \lfloor \texttt{eI.V}.size()/2 \rfloor$

2: $\ell \leftarrow$ line in e_1, e_2-plane intersecting v, not intersecting any $v' \in V_v$, and bisecting $\texttt{eI.V}[\texttt{split}^\uparrow], \texttt{eI.V}[\texttt{split}^\uparrow + 1]$

3: $s \leftarrow \ell^\perp$ such that $s \cdot \texttt{eI.V}[\texttt{split}^\uparrow] < s \cdot \texttt{eI.V}[\texttt{split}^\uparrow + 1]$ with zeros in all dimensions larger than two

4: $\texttt{split}_\downarrow \leftarrow$ index of v' in $\texttt{E}_v$ where v' is the first vertex in $\texttt{E}_v$ with $v' \cdot s < v \cdot s$

5: **if** $\texttt{split}_\downarrow$ is undefined **then**

6: $\quad \texttt{split}_\downarrow \leftarrow \texttt{E}_v.size() + 1$

7: $\texttt{E}_v^\ell \leftarrow \texttt{E}_v[\texttt{split}_\downarrow :]$

8: $\texttt{E}_v^r \leftarrow \texttt{E}_v[: \texttt{split}_\downarrow - 1]$

9: $\texttt{eI}_\ell\texttt{.count} \leftarrow \texttt{ComputeIndegree}(v, s, 1, [\,]) - \texttt{E}_v^\ell.size()$

10: $\texttt{eI}_\ell\texttt{.V} \leftarrow \texttt{eI.V}[: \texttt{split}^\uparrow]$

11: $\texttt{eI}_r\texttt{.count} \leftarrow \texttt{ComputeIndegree}(v, -s, 1, [\,]) - \texttt{E}_v^r.size()$

12: $\texttt{eI}_r\texttt{.V} \leftarrow \texttt{eI.V}[\texttt{split}^\uparrow + 1 :]$

13: **return** $\texttt{eI}_\ell, \texttt{eI}_r$

Lemma 13 (Interval Splitting)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex, $v \in \mathbb{K}_0$ a vertex, and* $\mathtt{eI}$ *an edge interval containing* $\mathtt{eI.count}$ *unknown edges. Let V_v be the vertices of $\mathbb{K}_0$ ordered radially cw around $\pi(v)$. Let* $\mathtt{eI.V}$ *be the sorted vertices from V_v contained in* $\mathtt{eI}$*. Let $\mathtt{E}_v$ contain all known adjacent vertices v' from V_v, i.e., $(v, v') \in \mathbb{K}_1$ and v' comes before the vertices in* $\mathtt{eI.V}$ *in the radial ordering of all vertices around $\pi(v)$. Then, Algorithm 3.8 uses two diagrams and $O(\Pi + n_0^2)$ time to split* $\mathtt{eI}$ *into two new edge intervals* $\mathtt{eI}_\ell$ *and* $\mathtt{eI}_r$ *with the properties:*

1. $\mathtt{eI}_r.\mathtt{V} + \mathtt{eI}_\ell.\mathtt{V} = \mathtt{eI.V}$ *(where $+$ denotes concatenation),*
2. $\mathtt{eI}_\ell.\mathtt{V}.size()$, $\mathtt{eI}_r.\mathtt{V}.size() \leq \lceil \frac{\mathtt{eI.V}.size()}{2} \rceil$,
3. $\mathtt{eI}_\ell.\mathtt{count} = |\{v' \in \mathtt{eI}_\ell.\mathtt{V} \mid (v, v') \in \mathbb{K}_1\}|$, *and* $\mathtt{eI}_r.\mathtt{count} = |\{v' \in \mathtt{eI}_r.\mathtt{V} \mid (v, v') \in \mathbb{K}_1\}|$.

Proof. We first show Properties 1 and 2. Algorithm 3.8 first creates two vertex sets from $\mathtt{eI.V}$, one containing the first half up to and including index $\mathtt{split}^\uparrow$ of $\mathtt{eI.V}$ ($\mathtt{eI}_\ell$) and the other containing the second half ($\mathtt{eI}_r$) including index $\mathtt{split}^\uparrow + 1$ to the end of $\mathtt{eI.V}$ (Lines 1–4). Thus, $\mathtt{eI}_\ell.\mathtt{V} + \mathtt{eI}_r.\mathtt{V} = \mathtt{eI.V}$ and Property 1 holds. Furthermore, by choice of split point $\mathtt{split}^\uparrow$ as $\lfloor \mathtt{eI.V}.size()/2 \rfloor$, $\mathtt{eI}_\ell.\mathtt{V}.size() = \lfloor \frac{\mathtt{eI.V}.size()}{2} \rfloor$ and $\mathtt{eI}_r.\mathtt{V}.size() = \lceil \frac{\mathtt{eI.V}.size()}{2} \rceil$, proving Property 2.

We now prove Property 3 for $\mathtt{eI}_\ell$ and note that the proof of $\mathtt{eI}_r$ follows the same argument. Assume ℓ is chosen as on Line 2 (we demonstrate how to choose ℓ at the end of this proof). We observe that ℓ intersects v and partitions the vertices in $\mathtt{eI.V} + \mathtt{E}_v$. We choose s orthogonal to ℓ (Line 3). By Theorem 1, the value returned from $\mathtt{ComputeIndegree}(v, s, 1, \cdot)$ counts all edges incident to v and below $v \cdot s$ in direction s. This is exactly equal to the number of edges in $\mathtt{eI}_\ell$ and the number of edges $(v, v') \in \mathtt{E}_v$ for which $v' \cdot s < v \cdot s$ (the edges below ℓ in eV). Thus, we can compute $\mathtt{eI}_\ell.\mathtt{count}$ by

subtracting the number edges in $\mathtt{E}_v$ that are counted in this k-indegree calculation. We find the greatest index of any vertex on the right side of ℓ and in $\mathtt{E}_v$ in Lines 4–6. Then, the set of verticesbelow ℓ in $\mathtt{E}_v$, denoted $\mathtt{E}_v^\ell$, is everything in $\mathtt{E}_v$ after that index, as in Line 7.

We can now compute $|\{v' \in \mathtt{eI}_\ell.\mathtt{V} | (v, v') \in \mathbb{K}_1\}|$ as the k-indegree of v from direction s minus the number of elements in $\mathtt{E}_v^\ell$, and we set $\mathtt{eI}_\ell.\mathtt{count}$ to that value on Line 9. Then, Property 3 is satisfied. See Figure 3.3 for a demonstration of two calls to the algorithm.

By Theorem 2, the calls to $\mathtt{ComputeIndegree}(v, s, 1, \cdot)$ and $\mathtt{ComputeIndegree}(v, -s, 1, \cdot)$ each generate a single persistence diagram, so a total of two diagrams are used. Since v is a zero-simplex, the runtime for each of these calls is $\Theta(\Pi + n_0^2)$ Furthermore, searching for the above and below split each take $O(\log n_0)$ time.

Determining an ℓ with the property that ℓ only intersects v and bisects the wedges can be done by binary searching for the first vertex $v' \in V_v \setminus \mathtt{eI.V}$ in cw order that has a line defined by v and v' that splits $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ and $\mathtt{eI.V}[\mathtt{split}^\uparrow + 1]$. If no such v' exists then ℓ can be set to the line intersecting v and the point midway between $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ and $\mathtt{eI.V}[\mathtt{split}^\uparrow + 1]$ in constant time. Then, we consider the case where such a line between v and v' exists. We must find a suitable way to split $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ and $\mathtt{eI.V}[\mathtt{split}^\uparrow + 1]$ without intersecting any other vertices in V_v with our line ℓ. If the line intersecting v and $V_v[V_v.indexOf(v') + 1]$ still falls between $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ and $\mathtt{eI.V}[\mathtt{split}^\uparrow + 1]$, we choose our ℓ to intersect the v and midway point between $V_v[V_v.indexOf(v') + 1]$ and v'. If the line intersecting v and $V_v[V_v.indexOf(v') + 1])$ does not split $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ and $\mathtt{eI.V}[\mathtt{split}^\uparrow + 1]$, then we choose our line to intersect v and the midpoint between $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ and the orthgonal projection of $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ onto the line intersecting v and v'. Thus, computing ℓ can be done in $\log(n_0)$ time, dominated by searching for a vertex $v' \in V_v \setminus \mathtt{eI.V}$ that has a line bisecting $\mathtt{eI.V}[\mathtt{split}^\uparrow]$ and $\mathtt{eI.V}[\mathtt{split}^\uparrow + 1]$. Then, the total complexity $O(\Pi + n_0^2)$. $\square$

Now, we use Algorithm 3.8 to efficiently identify intervals that contain edges.

Lemma 14 (Finding Edges Above a Vertex)**.** *Let $v \in \mathbb{K}_0$ be a vertex. If all edges $(v, v') \in \mathbb{K}_1$ with $v' \cdot e_2 < v \cdot e_2$ are known then Algorithm 3.9 finds the set of $(v, v') \in \mathbb{K}_1$ with $v' \cdot e_2 > v \cdot e_2$ using $O(\deg(v) \log n_0)$ augmented persistence diagrams in $O(\deg(v) \log n_0(\Pi + n_0^2))$ time.*

Proof. Let $\mathtt{V_v}$ be a list of vertices sorted radially cw around $\pi(v)$, beginning with the vertex with smallest angle below the horizontal in the bottom-left quadrant. Throughout Algorithm 3.9 we maintain a stack `eIStack` of edge intervals objects, which we think of as holding *potential adjacent vertices* – vertices that may or may not participate in an edge with v. Then, we define $\mathtt{E_p}$ be the concatenation of all vertex lists of edge intervals in `eIStack` in order. We denote the index of a vertex v' in $\mathtt{E_p}$ or $\mathtt{V_v}$ as $\mathtt{E_p}.index(v')$ or $\mathtt{V_v}.index(v')$, respectively. Having defined $\mathtt{E_p}$ and $\mathtt{V_v}$, we now give a loop invariant to prove the correctness of Algorithm 3.9. At the beginning of each iteration of the loop on Line 6,

1. all $v' \in \mathbb{K}_0$ adjacent to v with $\mathtt{V_v}.index(v') < \mathtt{V_v}.index(\mathtt{eIStack}.peek().\mathtt{V}[1])$ are in $\mathtt{E}_v$ (all edges to the left of the stack are known),

2. if $v_1, v_2 \in \mathtt{E_p}$ and $\mathtt{E_p}.index(v_1) < \mathtt{E_p}.index(v_2)$ then $\mathtt{V_v}.index(v_1) < \mathtt{V_v}.index(v_2)$ (the edge stack is cw-ordered),

3. all $\mathtt{eI} \in \mathtt{eIStack}$ have $\mathtt{eI.count} > 0$,

4. if $(v, v') \in \mathbb{K}_1$ then v' is exclusively in $\mathtt{E}_v$ or $\mathtt{E_p}$.

Initially, the stack contains only one edge interval object containing all vertices above v (Lines 1–5) and $\mathtt{E}_v$ contains all adjacent vertices below v (as input). Both are ordered cw. Thus, all four invariants are satisfied. (In the case that there are no vertices above v, `eIStack` is empty and the invariants are still trivially satisfied.)

Assume the invariant holds entering the ith iteration of the loop on Line 6. We now show that it holds entering the $(i+1)^{\text{st}}$. To begin i^{th} iteration, we pop an edge interval off of `eIStack` into the variable `eI` on Line 7. We consider the case where `eI` contains only one vertex and the case where it contains more than one vertex separately.

Suppose $\texttt{eI.V}.size()$ is one, as in Line 16. By Property 3, $\texttt{eI.count} > 0$, and by Lemma 13 `eI.count` gives the number of `eI.V` that participate in edges with v. Then, the single vertex in `eI.V` must participate in an edge with v, so we add it to $\texttt{E}_v$ on Line 17. By Property 1 and Property 2, $\texttt{E}_v$ contained all edges to the left of the stack at the beginning of the iteration, and `eI` contained the farthest-left potential vertex. Thus, after adding v, $\texttt{E}_v$ still contains all known vertices, and `eI` remains sorted, so both conditions are maintained. No edge intervals are pushed onto `eIStack` so Property 3 holds trivially, and v is moved from $\texttt{E}_\texttt{p}$ to $\texttt{E}_v$ so Property 4 holds.

Next, we consider the scenario on Line 8, where $\texttt{eI.V}.size() > 1$. We first split `eI` into two new edge intervals $\texttt{eI}_\ell$ (left) and $\texttt{eI}_r$ (right) on Line 9 using Algorithm 3.8. Note that by Lemma 13, vertices in $\texttt{eI}_\ell$ and $\texttt{eI}_r$ are in sorted order, with vertices in $\texttt{eI}_\ell$ coming before $\texttt{eI}_r$. Algorithm 3.9 processes the right interval first, and then the left. If $\texttt{eI}_r\texttt{.count} > 0$, then $\texttt{eI}_r$ contains a vertex adjacent to v and we push $\texttt{eI}_r$ onto `eIStack` on Line 11. Since the stack was sorted before and we have only performed a single pop operation, Property 1 remains true. Since Property 2 held at the beginning of the iteration, and $\texttt{eI}_r$ holds vertices with indices larger than the vertices of $\texttt{eI}_\ell$ in $\texttt{V}_\texttt{v}$, edge intervals in `eIStack` remain sorted, and Property 2 is maintained. $\texttt{eI}_r\texttt{.count} > 0$, so Property 3 holds as well. If $\texttt{eI}_r\texttt{.count} = 0$, we do nothing, and Properties 1–4 still hold. We then move on to processing the left interval.

If $\texttt{eI}_\ell\texttt{.count} = \texttt{eI}_\ell\texttt{.V}.size() = 1$, then the single vertex in $\texttt{eI}_\ell$ forms an edge, and we add the vertex to to $\texttt{E}_v$ (Line 13). Note that Properties 1–4 hold after this operation by observing that indices of the vertices in $\texttt{eI}_\ell$ smaller than the indices of the vertices of

$\mathtt{eI}_r$ in $\mathtt{V_v}$ and then applying the same argument outlined above when $\mathtt{eI.V}.size()$ equals one. Moreover, note that if $\mathtt{eI}_\ell.\mathtt{count} = 0$, we do nothing, and Properties 1–4 still hold. Finally, if $\mathtt{eI}_\ell.\mathtt{count} > 1$ then the edge interval has at least one edge and more than one potential adjacent vertex in $\mathtt{eI}_\ell.\mathtt{V}$. In this case, we push $\mathtt{eI}_\ell$ onto `eIStack` on Line 15. Since $\mathtt{eI}_\ell.\mathtt{count} > 1$, Property 3 is maintained. Additionally, no vertices are added to $\mathtt{E}_v$ and all potential edges in the right and left intervals have been returned to `eIStack`, so both Property 1 and Property 4 are true. And since Property 2 was true before and the left interval comes before the right interval, it remains true. Thus, the invariant holds entering the $(i+1)^{\text{st}}$ iteration, as required.

Next, we show that when the loop ends, we return the correct answer. If the loop terminates, then `eIStack` is empty, which implies that $\mathtt{E_p}$ is also empty. Then, Property 4 of the loop invariant implies that all adjacent vertices to v are in $\mathtt{E}_v$, which is returned on Line 18.

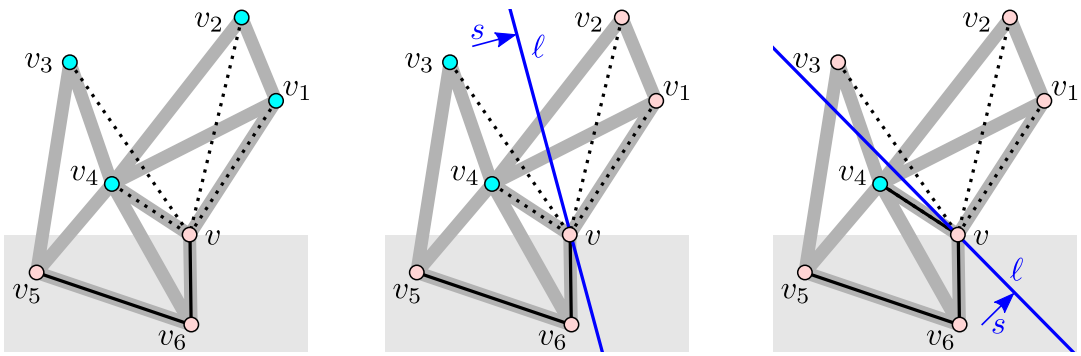
Finally, we analyze the time complexity of the algorithm and the number of diagrams it requires. The stack `eIStack` never is larger than $\deg(v)$ since it only stores edge intervals with positive edge count by Property 3 of our invariant. Then, we start with an edge interval with edge count $O(n_0)$ and we decompose the interval into at most $\deg(v)$ edge intervals with exactly one vertex each. Decomposing into an edge interval with a single edge and vertex requires $O(\log n_0)$ calls to Algorithm 3.8 by Property 2 of Lemma 13. Thus, the total number of iterations is $O(\deg(v)\log n_0)$, and the algorithm terminates. All operations in the loop take constant time except the call to Algorithm 3.8, which runs in $O(\Pi + n_0^2)$ time, so the total complexity is $O(\deg(v)\log n_0(\Pi + n_0^2))$. Furthermore, each call to Algorithm 3.8 requires two persistence diagrams, so the total number of persistence diagrams generated is $O(\deg(v)\log n_0)$. $\square$

Algorithm 3.9: $\texttt{FindUpEdges}(v, \texttt{E}_v, V_v)$

Input: vertex v, list of edges $(v, v') \in \mathbb{K}_1$ with $v' \cdot e_2 < v \cdot e_2$ denoted $\texttt{E}_v$ sorted cw around $\pi(v)$, V_v containing vertices ordered cw around $\pi(v)$

Output: $\texttt{E}_v$, the set of edges $(v, v') \in \mathbb{K}_1$

```
V↑ ← V_v restricted to vertices above L(e_2, v)
eI ← I_e(count = ComputeIndegree(−e_2, v, 1, ), V = V↑)
eIStack ← an empty stack of edge region objects
if eI.count > 0 then
    eIStack.push(eI)
while ∃eI ∈ eIStack do
    eI ← eIStack.pop()
    if eI.V.size() > 1 then
        eI_ℓ, eI_r ← SplitInterval(v, eI, E_v, V_v)
        if eI_r.count > 0 then
            eIStack.push(eI_r)
        if eI_ℓ.count = eI_ℓ.V.size() = 1 then
            append eI_ℓ.V[1] to E_v
        else if eI_ℓ.count > 0 then
            eIStack.push(eI_ℓ)
    else
        append eI.V[1] to E_v
return E_v
```



(a) At vertex v. (b) Split edge interval. (c) Split again.

Figure 3.3: Demonstration of two calls to Algorithm 3.8 for splitting edge intervals at a vertex v within Algorithm 3.9. (a) For our first call, $\mathtt{E}_v = \{v_6\}$, $\mathtt{eI.count} = 2$, and $\mathtt{eI.V} = (v_4, v_3, v_2, v_1)$. In other words, we know that $[v_6, v] \in \mathbb{K}_1$ and that two of the four vertices above v are adjacent to v. (b) Within Algorithm 3.8, we choose the direction s such that exactly half of the vertices in $\mathtt{eI}$ are below v. We create an edge interval $\mathtt{eI}_r$ corresponding to the vertex set above ℓ and outside the shaded region, and (in Algorithm 3.9) push that onto a stack to be processed later. We focus on the interval $\mathtt{eI}_\ell$ that contains the vertices below ℓ and not in the shaded box, $\mathtt{eI.V} = (v_4, v_3)$. Since two edges contribute to v's indegree in direction s and one of them is the edge $[v_6, v]$, we have $\mathtt{eI.count} = 2 - 1 = 1$ (c). Next, we find a new direction s that splits $\mathtt{eI}_\ell\mathtt{.V}$ into two sets of size one. We push the one above ℓ onto our stack, and notice that the wedge below ℓ and outside the shaded region has exactly one vertex ($\mathtt{eI.V} = \{v_4\}$) and that vertex is incident to v since $\mathtt{eI.count} = 2 - 1 = 1$.

See Figure 3.3 for an example of part of the execution of Algorithm 3.9. We now give an efficient algorithm for edge reconstruction.

Theorem 7 (Edge Reconstruction)**.** *Let $\mathbb{K} \subset \mathbb{R}^d$ be a simplicial complex. Given the locations of $\mathbb{K}_0$, Algorithm 3.10 reconstructs $\mathbb{K}_1$ using $O(n_1 \log n_0)$ augmented persistence diagrams in $O(n_1 \log n_0(\Pi + n_0^2))$ time.*

Proof. First, we prove that the Algorithm 3.10 finds all edges in $\mathbb{K}$. In Line 2, we sort the vertices by e_2-values. For $0 \leq j < n_0$, let v_j be the jth vertex in $\mathtt{V}_{\text{heights}}$. Our loop invariant is that, after iteration j, we have identified all edges that have a vertex on or below v_j with respect to the e_2 direction. We initialize our edge set to be empty. Next, consider vertex

v_j in iteration j and assume that we know all edges with a vertex below v_j with respect to direction e_2. To update the invariant, we must identify all edges above v_j that have v_j as an endpoint, which is returned by $\texttt{FindUpEdges}(v_j, \texttt{E}_v, V_v)$ (Algorithm 3.9) on Line 7. Finally, the algorithm terminates because $\texttt{V}_{\text{heights}}$ is finite and Algorithm 3.9 terminates during each iteration by Lemma 14.

Next, we analyze the running time of Algorithm 3.10. In Line 2, we sort the vertices by their e_2-values in $O(n_0 \log n_0)$ time. In Line 3, for each $v \in \mathbb{K}_0$, we radially order the vertices of $\mathbb{K}_0 \setminus \{v\}$ around v in $O(n_0^2)$ time by Lemmas 1 and 2 of [82]. In Lines 4–7, we call Algorithm 3.9 once for each vertex. For each $v \in \mathbb{K}_0$, each call takes $O(\deg(v) \log n_0(\Pi + n_0^2))$ time and summing over all vertices takes $O(n_0 \max_{v \in \mathbb{K}_0} \deg(v) \log n_0(\Pi + n_0^2))$. However, since no edge is checked at two different vertices, we can amortize the cost to be $O(n_1 \log n_0(\Pi + n_0^2))$ time. Finally, we analyze the number of diagrams used by Algorithm 3.10. Lines 2–3 compute no new diagrams. In Lines 4–7, similar to the time analaysis, for each $v \in \mathbb{K}_0$, Algorithm 3.9 uses $O(\deg(v) \log n_0)$ diagrams and summing over all vertices generates $O(n_1 \log n_0)$ diagrams. □

Next, we explicitly describe the results for embedded graphs in $\mathbb{R}^d$. Recall, that the number of diagrams used for edge reconstruction on embedded graphs in [12] was $n_0^2 - n_0$.

Corollary 5. *Let $G = (V, E)$ be an embedded graph in $\mathbb{R}^d$. The number of diagrams used to reconstruct E is $O(d + n_1 \log n_0)$.*

Furthermore, if we limit the scope to plane graphs, then we are able to reconstruct edges with a number of diagrams that is less than exponential in the ambient dimension.

Corollary 6. *Let $G = (V, E)$ be a plane graph in $\mathbb{R}^2$. The number of diagrams used to reconstruct E is $O(d + n_0 \log n_0)$.*

Algorithm 3.10: `FindEdges`$(\mathbb{K}_0)$

Input: $\mathbb{K}_0$, a list of all vertices in $\mathbb{K}$

Output: $\mathbb{K}_1$, a list of all edges in $\mathbb{K}$

1: $\mathbb{K}_1 \leftarrow \{\}$

2: $\mathtt{V}_{\text{heights}} \leftarrow$ vertices ordered by heights of hyperplanes in $\mathbb{L}(e_2, \mathbb{K}_0)$

3: $V_{sort} \leftarrow$ all vertices sorted cw radially around all other vertices

4: **for** v in $\mathtt{V}_{\text{heights}}$ **do**

5: $\quad V_v \leftarrow$ vertices from V_{sort} relative to v

6: $\quad \mathtt{E}_v \leftarrow v'$ for $(v, v') \in \mathbb{K}_1$ sorted radially cw using Line 3

7: $\quad$ Add edges from `FindUpEdges`$(v, \mathtt{E}_v, V_v)$ to $\mathbb{K}_1$

8: **return** $\mathbb{K}_1$

The proof falls out of Corollary 3, Theorem 6, and Theorem 7. Finally, Assumption 1 ensures that all points are in general position in the e_1 and e_2 plane, but does not ensure that the immersion of the graph in the e_1 and e_2 plane results in a plane graph.

Corollary 7. *Let $G = (V, E)$ be a graph immersed in the plane spanned by e_1 and e_2. The number of diagrams used to reconstruct E is $O(d + n_1 \log n_0)$.*

3.4.1 Contributions to Section 3.4

In Section 3.4, we have presented an algorithm for efficiently reconstructing edges for embedded graphs. Initially, the ideas for this were formulated by Samuel Micka, Brittany Terese Fasy, David L. Millman, Lucia Williams, and Anna Schenfisch in the Fall of 2018. However, we did not meet our paper submission deadline and no further progress was made until Fall 2019. When we returned to the work, Samuel Micka made substantial efforts to correct errors, rewrite the statements and proofs, and rewrite the algorithms to make them more concise and correct. Lucia Williams was critical in reviewing the content

that Samuel Micka had added, correcting and simplifying several statements. After the submission of the work in the Fall of 2019 [56], Samuel Micka came back through the section, making several more updates for clarity and readability. Brittany Terese Fasy put forward substantial efforts to make the section more readable with Figure 3.3 and some additional explanations.

3.5 Reconstructing k-Simplices

Combining the results from the previous subsections, we arrive at an algorithm to fully reconstruct an embedded simplicial complex that was originally introduced in our work [56]. Note that we utilize Algorithm 3.4 to reconstruct $\mathbb{K}_0$ but point out that Algorithm 3.5 could be substituted to utilize fewer diagrams at the cost of time complexity.

Algorithm 3.11: `ReconstructComplex`()

Input: None (but makes calls to global `Oracle` for unknown simplicial complex)

Output: simplicial complex $\mathbb{K}$

```
1: K_0 ← FindVerticesTIME()
2: K_1 ← FindEdges(K_0)
3: K̄_1 ← K_0 ∪ K_1
4: for i ∈ {2, 3, ..., κ} do
5:     for σ ∈ K̄_{i-1} do
6:         for v ∈ K_0 \ verts(σ) do
7:             if CheckSimplex(σ, v) then
8:                 K̄_i ← K̄_i ∪ (σ ∪ {v})
9: return K̄_κ
```

Theorem 8 (Simplicial Complex Reconstruction)**.** *Let $\mathbb{K}$ be a simplicial complex in $\mathbb{R}^d$, κ be the dimension of the highest-dimensional simplex in $\mathbb{K}$, n_0 be the number of zero-simplices, and let $\hat{n} = \max_i n_i$. If $\mathbb{K}$ meets the assumptions made in Assumption 1, then Algorithm 3.11 reconstructs $\mathbb{K}$ in $O(\kappa\hat{n}n_0 2^\kappa(n_0(\hat{n} + d^5 + \log n_0) + \Pi + 2^{\kappa-1}))$ time using $O(\kappa\hat{n}n_0 2^\kappa)$ APDs.*

Proof. Algorithm 3.11 iterates through each $(k-1)$-simplex σ and checks whether σ forms a k-simplex with each v. By Theorem 3, Algorithm 3.3 (`CheckSimplex`(σ, v)) determines

if the simplex defined by σ and v is present in K. Since we pass every potential k-simplex to `CheckSimplex`(σ, v), the algorithm finds all k-simplices.

Next, we analyze the runtime and number of diagrams. There are three nested loops. The first loop on Lines 4–8 performs at most κ iterations. The second loop on Lines 5–8 performs at most $\hat{n}$ iterations. The third loop on Lines 6–8 performs at most n_0 iterations. Inside the inner loop, Line 7 calls `CheckSimplex`(σ, v), which for dimension k, takes $O(2^k(n_0(n_{k-1} + d^5 + \log n_0) + \Pi + 2^{k-1}))$ time and uses $2^{k+1} - 2$ diagrams by Theorem 3. Thus, the algorithm takes $O(\kappa \hat{n} n_0 2^\kappa (n_0(\hat{n} + d^5 + \log n_0) + \Pi + 2^{\kappa-1}))$ time and uses $O(\kappa \hat{n} n_0 2^\kappa)$ diagrams. □

3.5.1 Contributions to Section 3.5

The content in Section 3.5 (originally appearing in [56]) was initially written by Samuel Micka when he was working to combine the predicates developed in Section 3.2. However, as Section 3.2 was updated, Section 3.5 also experienced changes. Lucia Williams carefully read and critiqued the section. Finally, Samuel Micka and David L. Millman came back to calculate time and diagram complexities when all of the predicates in Section 3.2 were finished.

3.6 Challenges Using Non-Augmented Descriptors

With the ability to reconstruct simplicial complexes using APDs, we ask: are similar constructions possible without augmentation? Or, using other topological descriptors? In this section, we consider several challenges that arise when reconstructing $\mathbb{K}_0$ when using descriptors without augmentation, i.e., storing information about the height, relative to a particular direction, of all vertices in the filtration. We already know that simplicial complexes in $\mathbb{R}^2$ cannot be reconstructed, without additional general position assumptions, using the ECC by Lemma 2. As such, we consider the problem of reconstructing plane graphs and focus on reconstructing the vertices. By focusing on plane graphs adhering to Assumption 1, every vertex will be witnessed by some set of directions chosen from $\mathbb{S}^{d-1}$. We note that the results described here were published in [55].

3.6.1 Towards Vertex Reconstruction

We are interested in reconstructing a plane graph from ECCs from a constant number of directions. While three directions was sufficient for reconstructing vertices using APDs, ECCs contain less topological and less geometric information under many circumstances. However, we observe the existence of a linear number of directions that allows to fully reconstruct the vertices of a plane graph:

Proposition 1 (ECC Existence)**.** *Given a plane graph $\mathbb{K} = (V, E)$ with $|V| = n_0$, there exist $3n_0$ directions, and corresponding ECCs, that can be used to reconstruct all vertices in V.*

Proof. Let $v \in V$ be a vertex in $\mathbb{K}$. First, we show that each vertex is witnessed from an infinite number of directions $\mathbb{S}^1$. If $deg(v) = 0$, v is witnessed from any direction for which it lies on a unique witness line (So, for all but $|V| - 1$ directions). If $deg(v) = 1$ with edge (v, v') for some $v' \in V$, then v is witnessed from an the infinite set of directions

from which v' appears after v in the lower-star filtration, and v lies on a unique witness line. If $deg(v) > 1$ with edges (v, v') and (v, v'') for $v', v'' \in V$, then v is witnessed from any direction from which v' and v'' appear before v in the filtration and v lies on a unique witness line. Thus, each vertex is witnessed from an infinite number of directions.

Let $\mathbb{I}_v$ be the set of directions that witness v. We can choose any three directions from $\mathbb{I}_v$ and generate a unique three-way intersection at v. Now, we need to show that a set of directions exist for each of the n_0 vertices such that no three-way intersections exist at locations where a vertex is not located. In order to do this, we give the vertices some arbitrary ordering $v_1, v_2, \ldots, v_{n_0}$. Then, select vertices in ascending order. For the first, any three directions in $\mathbb{I}_{v_1}$ will give a three-way intersection of witness lines at v_1 and, because $\mathbb{I}_{v_1}$ contains infinite choices for directions and there are only a finite number of vertices, these directions can be chosen to not create a three-way intersection at any location that does not correspond to a vertex. Specifically, the first two distinct directions will not create any three-way intersections. Moreover the third direction can be chosen from $\mathbb{I}_{v_1}$ so that all existing intersections (from the first two directions) are at unique heights.

For each successive vertex v_i, there exist up to a total of $3in_0$ witness lines. More importantly, the number of witness line intersections is finite. Thus, there exist three directions in $\mathbb{I}_{v_i}$ such that none of the witness lines created by these directions intersect existing intersections that do not correspond to vertex locations. For example, like in the case with the third direction for v_1, you can choose a direction in $\mathbb{I}_{v_i}$ for which each existing intersection point is at a unique height. Then, since only vertices can create new witness lines, the lines added in this direction will not intersect any existing intersection points except for those at vertices. Since the x- and y-coordinates of a vertex can be determined using a three-way line intersection, we can see that there exist a set of $3n_0$ directions which generates exactly n_0 three-way (or more) intersections of witness lines, in correspondence with the locations of all n_0 vertices. $\square$

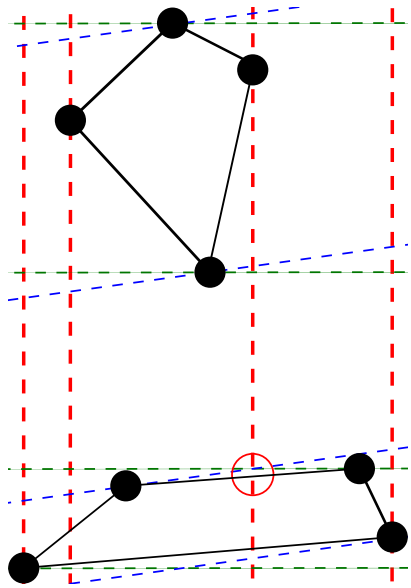


Figure 3.4: Scenario where a degree two vertex not witnessed by the directions e_1 and e_2 can create a three-way line intersection where a vertex does not exist when a third direction is chosen. This intersection is highlighted by a red circle in the image.

Remark 1. *We note that a similar observation for vertex reconstruction for simplicial complexes in $\mathbb{R}^d$ was made independently by Curry et al., where the authors provide a method for determining vertex locations using assumptions about the geometry of complex [40].*

Then, attempting to use one of our techniques for vertex reconstruction (such as [11]) seems promising for plane graph reconstruction using ECCs, i.e., we can define a correspondence between three-way witness line intersections (from carefully chosen directions) and vertices. However, Proposition 1 only proves the existence of such a set of directions and does not provide an algorithm if the vertex locations are not known. In fact, certain types of vertices introduce difficulties when developing a deterministic approach for reconstruction with the ECC. For example, consider Figure 3.4. A degree two vertex is not witnessed by any of the witness lines from the cardinal directions e_1, $-e_1$, e_2, and $-e_2$. As an alternative, we might ignore degree two vertices and attempt to

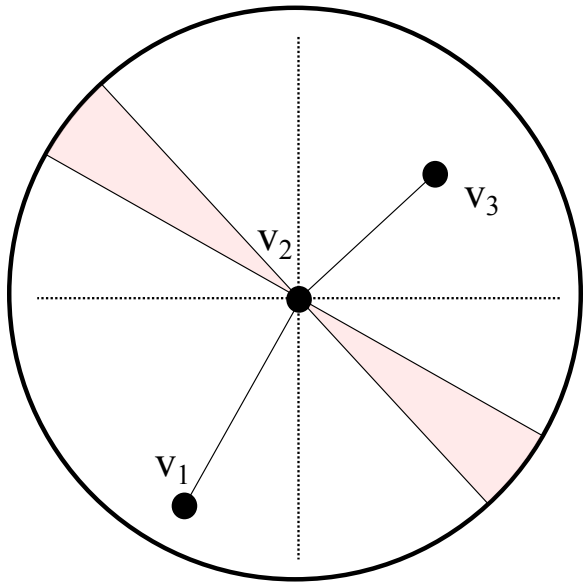


Figure 3.5: Embedding of a plane graph where vertices v_1, v_2, and v_3 are nearly collinear. As the vertices approach collinear the region on $\mathbb{S}^1$ containing directions which will witness v_2 grows arbitrarily small.

generate a correspondence between three-way intersections of witness lines and non-degree two vertices. However, if we use the technique described in Theorem 4 to choose such a direction, that direction may result in a witness line that causes a three-way intersection not corresponding to a vertex as in Figure 3.4. In fact, when degree two vertices are introduced to the plane graph, several problems arise. We discuss these problems in detail in Section 3.6.2.

3.6.2 Degree Two Challenges

Degree two vertices introduce several complications in finding witness directions, because degree two vertices can have an arbitrarily small region on $\mathbb{S}^1$ from which they can be witnessed. For example, in Figure 3.5 the vertices v_1, v_2, and v_3 are nearly collinear. In order to witness v_2, we must choose directions from within the red region, where a decrease or increase in the ECC will be observed. However, these these regions becomes arbitrarily small as v_1, v_2 and v_3 approach collinearity.

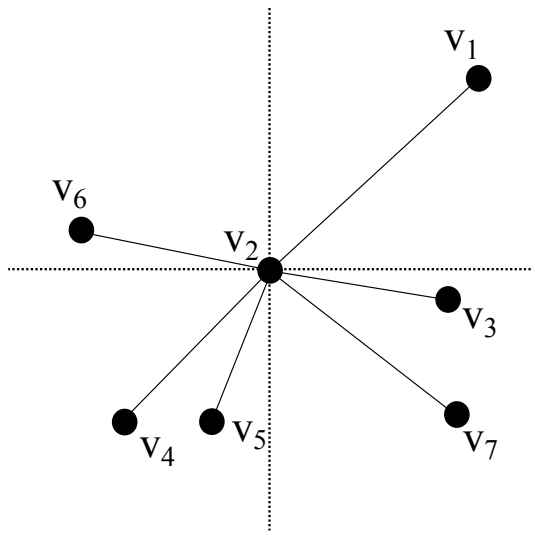


Figure 3.6: Different scenarios of edge embeddings for degree two vertices. We consider v_2 a degree two vertex when considering, exclusively, the sets of edges $\{(v_1, v_2), (v_3, v_2)\}$, $\{(v_4, v_2), (v_5, v_2)\}$, or $\{(v_6, v_2), (v_7, v_2)\}$. These three sets of edges highlight situations in which v_2 can be witnessed in different ways.

Remark 2. *We discuss this phenomenon seen in Figure 3.5 further in Section 3.7, but note that these challenges were simultaneously and independently investigated by [40]. The local geometry ("flatness") of simplices is related to the δ-observability discussed in [40, Section 7].*

As mentioned earlier, degree two vertices can also introduce additional ambiguities when witnessing non-degree two vertices. Recall the example found in Figure 3.4 and the discussion in Section 3.6.1.

Despite these difficulties, several situations exist in which degree two vertices can be witnessed. The following propositions summarize these scenarios. For clarity, we discuss quadrants as though v_2 is located at the origin. However, note that the following propositions also apply to arrangements with similar orientations and angles.

Proposition 2 (Same Quadrant)**.** *If v_1 and v_3 lie in the same quadrant, such as the vertices*

v_4 and v_5 in Figure 3.6, then v_2 will be witnessed in ECCs from every one of the cardinal directions.

Proof. If v_1 and v_3 lie in the same quadrant, then v_1 and v_3 will appear before v_2 from exactly one of the two x-axis parallel directions directions $(-1, 0)$ or $(1, 0)$ and before v_2 in exactly one of the y-axis parallel directions $(0, -1)$ or $(0, 1)$. Let $s_1 \in \{(0, 1), (0, -1)\}$ and $s_2 \in \{(1, 0), (-1, 0)\}$ be the directions that witness v_1 and v_3 before v_2. χ_{s_1} and χ_{s_2} will witness v_2 by seeing a decrease in the Euler Characteristic at the time that v_2 is first included in the filtration. Then, $-s_1$ and $-s_2$ will witness v_2 before v_1 or v_2. Since no other edges with v_2 as an endpoint exist, there will be an increase in χ_{-s_1} and χ_{-s_2} at the time that v_2 is first included in the filtration. Then, v_2 is witnessed from every cardinal direction, as required. □

Proposition 3 (Neighboring Quadrants)**.** *If v_1 and v_3 lie in neighboring quadrants, such as vertices v_1 and v_3 in Figure 3.6, then v_2 will be witnessed in ECCs from exactly two of the four cardinal directions.*

Proof. Recall that no two vertices share x- or y-coordinates, then any witness line from a cardinal direction will be unique. Let s be the cardinal direction for which $v_1 \cdot s < v_2 \cdot s$ and $v_3 \cdot s > v_2 \cdot s$ and $-s$ the cardinal direction chosen such that $v_3 \cdot s < v_2 \cdot s$ and $v_1 \cdot s > v_2 \cdot s$. Then, there is no change in Euler Characteristic at v_2 from either s or $-s$, since v_2 is added at the same time as (v_1, v_2) or (v_2, v_3), respectively. Now, let w and $-w$ be the remaining two cardinal directions, where w is the direction from which we include v_2 before v_1 or v_3. Direction w witnesses v_2 because no edges are included at height v_2 from that direction. Direction $-w$ witnesses v_2 because both (v_1, v_2) and (v_2, v_3) are added along with v_2. Thus, v_2 is witnessed from exactly two of the four cardinal directions. □

Proposition 4 (Degree Two Bounded Angle)**.** *If $angle((v_1, v_2), (v_2, v_3)) < \frac{\pi}{2}$ then v_2 will be witnessed in ECCs from at least two of the four cardinal directions.*

Proof. If $angle((v_1, v_2), (v_2, v_3)) < \frac{\pi}{2}$, then v_1 and v_3 must lie in neighboring quadrants or the same quadrant, since neither can lie on the boundary of a quadrant by assumption. If they are in the same quadrant, Proposition 2 tells us that they must be seen from all four cardinal directions. If they are in neighboring quadrants, Proposition 3 tells us that we can witness v_2 with ECCs from exactly two of the four cardinal directions. □

The above propositions show scenarios for which degree two vertices can be witnessed using cardinal directions. However, degree two vertices pose particular problems when the edges lie in non-neighboring quadrants, such as the edges (v_6, v_2) and (v_7, v_2) in Figure 3.6 or (v_1, v_2) and (v_2, v_3) in Figure 3.5. Then, when a plane graph $\mathbb{K}$ does not contain any degree two vertices, a constant number of ECCs can be used to determine the embeddings of the vertices.

3.6.3 A Special Case

If a plane graph contains no degree two vertices, then reconstructing the vertices poses fewer challenges. Let $\mathbb{K}$ denote a plane graph with vertex and edge sets V and E respectively. Recall from [11] that three way filtration line intersections from carefully chosen directions correspond to a vertex location for plane graphs using persistence diagrams. We show that this result still holds for reconstructing plane graphs using ECCs, if they do not contain degree two vertices.

First, we provide a lemma that yields insight into how non-degree two vertices are witnessed.

Lemma 15 (Linear Witness Lines)**.** *Let $\mathbb{K}$ be a plane graph in $\mathbb{R}^2$ with vertices V such that for all $v \in V$, $deg(v) \neq 2$ and denote $|V| = n_0$. Let ℓ be a line in $\mathbb{R}^2$ such that any line parallel to ℓ intersects at most one vertex in V. Let $s \in \mathbb{S}^1$ be chosen perpendicular to ℓ. Then,*

$$|\mathbb{W}(s, V) \cup \mathbb{W}(-s, V)| = n_0$$

Proof. We show that each vertex is seen by at least one of s or $-s$. Let $v \in V$ be a vertex with $deg(v) = 0$. Then, v will correspond to $\ell(s, v)$ for any arbitrary direction $s \in \mathbb{S}^1$ because χ_s will always increase by at least one at time $s \cdot v$. As such, v will be witnessed by both s and $-s$.

Let $v \in V$ be a vertex with $deg(v) = 1$ and $(v, v') \in E$ for some $v' \in V$. Then, if $s \in \mathbb{S}^1$ is chosen such that $s \cdot v' < s \cdot v$, v will not result in a change in χ_s. However, s was chosen such that no two vertices will be witnessed at the same time. As a result, no edge in E can be parallel to ℓ. Then, if $s \cdot v' < s \cdot v$ then $-s \cdot v' \geq -s \cdot v$ and an increase in χ_{-s} is seen at time $-s \cdot v$. This implies that v is witnessed by s or $-s$ but not both.

Finally, if $v \in V$ is a vertex with $deg(v) > 2$, then we must consider two cases. If, for $s \in \mathbb{S}^1$, there exists exactly one edge $(v, v') \in E$ such that $s \cdot v' < s \cdot v$, then there must exist at least two additional edges that will result in a decrease in χ_{-s} at time $-s \cdot v$. As such, v will be witnessed by at least one of the ECCs resulting from s or $-s$. On the other hand, if, for $s \in \mathbb{S}^1$, there exists either zero edges or more than one edge that appear before v in the height filtration from s, then χ_s will either increase (in the case where no edges appear before v) or decrease (in the case where two or more edges appear before v). Then, all non-degree two vertices result in a change in χ_s or χ_{-s} and, as such, $|\mathbb{W}(s, V) \cup \mathbb{W}(-s, V)| = n_0$, as required. $\square$

By generalizing the results of Lemma 15, we introduce the the following Lemma to generate n_0^2 potential vertex locations in $\mathbb{R}^2$, where n_0 is the number of vertices.

Lemma 16 (Witness Line Intersections)**.** *Let $\mathbb{K}$ be a plane graph in $\mathbb{R}^2$ with vertices V. Recall the cardinal directions $(0, 1), (1, 0), (0, -1), (-1, 0) \in \mathbb{S}^1$. If for all $v \in V$,*

$deg(v) \neq 2$ *then*

$$|\mathbb{W}((0,1),V) \cup \mathbb{W}((0,-1),V)| = n_0, \textit{ and}$$
$$|\mathbb{W}((1,0),V) \cup \mathbb{W}((-1,0),V)| = n_0.$$

Proof. By Lemma 15, if s is chosen such that no two vertices are intersected by a line perpendicular to s, then $\mathbb{W}(s,V) \cup \mathbb{W}(-s,V)$ will result in n_0 filtration lines. Recall that, by Assumption 1, no two vertices in $\mathbb{K}$ share an x- or y-coordinate. Then, by Lemma 15, $|\mathbb{W}((0,1),V) \cup \mathbb{W}((0,-1),V)| = n_0$ and $|\mathbb{W}((1,0),V) \cup \mathbb{W}((-1,0),V)| = n_0$, as required. □

Utilizing these n_0 horizontal and n_0 vertical witness lines, we are able to pick two additional directions to generate three-way filtration line intersections using a technique similar to the one described in Theorem 4. Then, the following theorem holds as well.

Theorem 9 (ECC Vertex Reconstruction)**.** *Let $\mathbb{K} = \langle V, E\rangle$ be a plane graph with vertices V and $|V| = n_0$. If for all $v \in V$, $deg(v) \neq 2$, then the locations of all vertices can be determined using six ECCs in $O(n_0 \log n_0)$ time.*

Proof. Using Lemma 16 we construct n_0 horizontal and n_0 vertical lines corresponding to vertices using four ECCs and we denote them L_H and L_V respectively. Then, we must identify an additional two directions which will, together, generate an additional n_0 unique witness lines and exactly n_0 three-way filtration line intersections. We choose these final directions $s_3 \in \mathbb{S}^1$ and $-s_3$ using the method described in Theorem 4. We observe that, by Lemma 8, no two vertices will be intersected by any single line perpendicular to s_3. Then, since each vertex will be witnessed by at least one of the ECCs from $\mathbb{W}(s_3,V)$ or $\mathbb{W}(-s_3,V)$ by Lemma 15, these two directions will yield n_0 distinct filtration lines each of

which will intersect exactly one two-way intersection between lines of L_H and L_V. Then, Lemma 9 implies that these three-way intersections are the locations of the n_0 vertices in V. The $O(n_0 \log n_0)$ running time follows from the proof of Theorem 4. □

We note that, due to the challenges of reconstructing $\mathbb{K}_0$, we are not particularly optimistic about reconstructing $\mathbb{K}$ with ECCs using similar approaches to the ones described in Section 3.5. Next, in Section 3.7, we explore lower-bounds on the number of various topological descriptors for reconstructing simplicial complexes.

3.6.4 Contributions to Section 3.6

In Section 3.6, we have presented several scenarios in which problems arise when performing reconstruction with non-augmented descriptors. These results were published in [55] and the authors worked on all aspects of the work at one point or another. As such, we will attempt to highlight the contributions of Samuel Micka:

- Proposition 1 was initially developed by Samuel Micka, Lucia Williams, and Anna Schenfisch. We also note that observations about "flatness" of simplices were simultaneously developed in [40].

- The observation summarized in Figure 3.4 is a simplified version of an observation made during discussions between David L. Millman and Samuel Micka.

- Proposition 1 led to observations about the difficulties of reconstructing degree two vertices. Samuel Micka developed Proposition 2, Proposition 3, and Proposition 4 to offer insight into the specific problems related to degree two vertices. Several discussions and critiques were carried out pertaining to these propositions amongst the authors.

- The lemmas and theorems in Section 3.6.3 were developed by Samuel Micka, Lucia Williams, and Anna Schenfisch when it was realized that, without degree two vertices, reconstruction was significantly simplified.

3.7 Lower-bounds on Reconstruction with Various Descriptors

In Section 3.6, we focused on degree two vertices and reconstructing plane graphs. While we are demonstrating problems related to reconstructing simplicial complexes, similar observations have been made when using other descriptors for representing simplicial complexes as well. In general, this problem of "witnessing" vertices generalizes to higher dimensions, as observed for simplicial complexes by Curry et al. and for cubical complexes by Betthauser [14, 40]. Moreover, Betthauser proves lower-bounds on the number of directions necessary for reconstructing cubical complexes using PDs generated from height filtrations (see [14, Lemma 4.19]). However, we would like to investigate similar questions in the space of simplicial complexes. Using ideas related to witnessing simplices in simplicial complexes, Curry et al. were able to find a set of directions, along with a decomposition of the unit sphere, sufficient for uniquely representing simplicial complexes in $\mathbb{R}^d$ that satisfy certain geometric assumptions. This sufficiency result provides an upper-bound for the number of PDs and ECCs used for uniquely representing a simplicial complex in $\mathbb{R}^d$. In this section, we explore definitions from [40], explaining the sufficiency results. Then, we develop lower-bounds on the number of directions necessary for reconstruction using the ECC (as well as an augmented version of the ECC), marrying observations in reconstruction and representation. Furthermore, we demonstrate that reconstructing particular simplicial complexes requires strictly fewer APDs than ECCs. We note that the results described here initially appeared in [80] as preliminary results but the full results will appear in [54].

First, we move away from the term *witness* and adapt the notion of *Euler observable*

from Curry et al..

Definition 17 (Euler Observable: Def. 7.2 [40])**.** *If $\mathbb{K} \subset \mathbb{R}^d$ is a simplicial complex and $v \in \mathbb{K}$ a vertex then v is* Euler observable *from a direction $s \in \mathbb{S}^{d-1}$ if χ_s changes at height $s \cdot v$. For a set of directions $D \subset \mathbb{S}^{d-1}$, v is* observed *if it is Euler observable from any $s \in D$.*

Curry et al. also introduce the useful notion of δ-observability.

Definition 18 (δ-observability: Def. 7.3 [40])**.** *If $\mathbb{K} \subset \mathbb{R}^d$ is a simplicial complex and $v \in \mathbb{K}$ a vertex then v is* δ-observable *if there exists a ball of directions B centered on $\mathbb{S}^{d-1}$ of radius δ such that χ_s changes at height $s \cdot v$ for all $s \in B$.*

Intuitively, this value of δ relates the "flatness" of a simplex and the region of $\mathbb{S}^{d-1}$ for which that simplex is observed. This idea of flatness was explored briefly in Section 3.6.2 and we use this definition to formally relate the geometry of the simplicial complex and the regions for which we can sample to observe each vertex. Curry et al. use this observation to define a δ-net which we limit to $\mathbb{S}^{d-1}$ and adapt here.

Definition 19 (δ-net: Def. 7.6 [40])**.** *Given $\mathbb{S}^{d-1}$, a δ-net on $\mathbb{S}^{d-1}$ is a subset of points for which balls of radius δ centered at the points cover $\mathbb{S}^{d-1}$.*

Combining definitions, Curry et al. proved, independently of our work in Section 3.6, that if all of the vertices of a simplicial complex $\mathbb{K} \subset \mathbb{R}^d$ are δ-observable then a set of ECCs or PDs generated from from enough directions in each of the balls of a δ-net on $\mathbb{S}^{d-1}$ suffices to determine the vertices [40, Proposition 7.11 and Theorem 7.14]. With the vertex locations, a stratification of the sphere is induced by the vertex locations, where a sample from each stratum generates a sufficient set of parameterized descriptors to represent the simplicial complex. [40, Theorem 7.14] is the first result proving a finite upper-bound on the number of directions sufficient for representing a simplicial complex in $\mathbb{R}^d$.

Our results complement the geometry-based upper-bounds of [40] by exploring lower-bounds in terms of the number simplices, showing that when using ECCs (or an augmented version of the ECC), strictly fewer APDs reconstruct particular simplicial complexes. These lower-bounds also demonstrate examples where *at least* a linear number of ECCs (and augmented versions of the ECC) are necessary to determine the shape. First, we piece together the regions on the sphere for which a vertex is observable by an ECC.

Definition 20 (Observing Region)**.** *If $\mathbb{K} \subset \mathbb{R}^d$ is a simplicial complex and $v \in \mathbb{K}$ a vertex, then the* observing region *of v is $O(v) \subset \mathbb{S}^{d-1}$ such that for every $s \in O(v)$, v is observed from s and for any $s' \in \mathbb{S}^{d-1} \setminus O(v)$, v is not observed from s'.*

Additionally, we note that reconstructing a simplicial complex requires that each vertex is observed.

Observation 1. *Let $\mathbb{K} \subset \mathbb{R}^d$ be an unknown simplicial complex and $v \in \mathbb{K}$ be a vertex with observing region $O(v) \subset \mathbb{S}^{d-1}$. The vertex v can only be reconstructed using the ECC if a direction $s \in O(v)$ is chosen.*

To avoid problems where vertices are not observed from any direction, as in Lemma 2, we focus our attention on plane graphs in $\mathbb{R}^2$ satisfying Assumption 1 when proving lower-bounds with the ECC. For a specific simplex, the observing region can be arbitrarily small and, as a result, some vertices can only be reconstructed from very limited sets of directions. As an example, the observing region for degree two vertices in plane graphs can be arbitrarily small as they approach collinearity with their adjacent vertices. We formalize this idea with the following lemma and note that this lemma is similar to an observation made about observing a vertex in a two-simplex in [40, Example 7.4] but generalized to degree two zero-simplices.

Lemma 17 (Observing Region)**.** *Let $\mathbb{K} = (V, E) \subset \mathbb{R}^2$ be a plane graph. Let $v \in V$ be a degree two vertex with adjacent vertices v_ℓ and v_r. Let*

$$R_1 = \{s \in \mathbb{S}^{d-1} \mid s \cdot v_\ell < s \cdot v, s \cdot v_r < s \cdot v\},$$

and

$$R_2 = \{s \in \mathbb{S}^{d-1} \mid s \cdot v_\ell > s \cdot v, s \cdot v_r > s \cdot v\}.$$

Then, the observing region for v is exactly $O(v) = R_1 \cup R_2$.

Proof. Assume, for contradiction, that we were able to observe v using a direction $s \in \mathbb{S}^1$ outside of $R_1 \cup R_2$. If χ_s decreases at $s \cdot v$, then $s \cdot v_\ell < s \cdot v$ and $s \cdot v_r < s \cdot v$. However, this only occurs when $s \in R_1$. Then, χ_s must increase at $s \cdot v$. Since v has degree two, that means that $s \cdot v_\ell > s \cdot v$ and $s \cdot v_r > s \cdot v$. However, $s \cdot v_\ell > s \cdot v$ and $s \cdot v_r > s \cdot v$ when $s \in R_2$. Then, s must lie in either R_1 or R_2 on $\mathbb{S}^1$, a contradiction. □

Next, we describe the construction of a plane graph for which strictly more ECCs are necessary for reconstruction than APDs, see Figure 3.7 for an example.

Construction 1 (Lower-bound plane graph)**.** *We construct $G = (V, E) \subset \mathbb{R}^2$, with V_x denoting the vertices V sorted by x-coordinate and V_y denoting the vertices V sorted by y-coordinate, as a plane graph satisfying the following properties.*

1. *$n_0 > 1$ and odd.*
2. *G satisfies Assumption 1.*
3. *For all $v \in V$, the index of v is the same in V_x and V_y.*
4. *If v_i is the ith coordinate, then for all $1 < i < n_0$ then $(v_i, v_{i+1}) \in E$ and $(v_{i-1}, v_i) \in E$. The edge set E contains no other edges.*

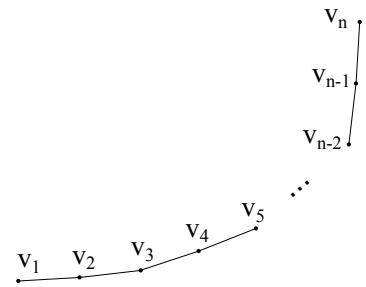


Figure 3.7: Graph G meeting the constraints in Construction 1 which requires $\Omega(n)$ directions to observe every vertex using the ECC but only $\Theta(1)$ directions when using the APD (see Theorem 10).

5. *If $v_i, v_j \in V_x$ with i, j even and $i \neq j$, then the observing regions for v_i and v_j are disjoint.*

Figure 3.7 gives an example of a plane graph G meeting the requirements of Construction 1. Then, using any plane graph G that meets the constraints in Construction 1, we prove lower-bounds on the number of ECCs necessary for reconstruction.

Theorem 10. *Let $G = (V, E) \subset \mathbb{R}^2$ be any plane graph meeting the constraints in Construction 1 with $n_0 = |V|$. G can be reconstructed with $\Theta(1)$ APDs but requires $\Omega(n_0)$ ECCs.*

Proof. The vertices in G meet the general position assumptions for [11, Theorem 5], by Construction 1 Property 2, which provides a method for finding three APDs for determining vertex locations. For the edges in G, we generate constraints from the directions e_1 and $-e_1$. We note the diagram resulting from the height filtration from e_1 was already generated when reconstructing vertex locations. Thus, we generate one additional diagram from $-e_1$.

Then, we prove inductively that if we know V, we can reconstruct E using four diagrams for any value of n_0. Since n_0 must be odd and strictly larger than one by Property 1, we prove our base case $n_0 = 3$. $\overline{D}_0(e_1)$ gives us that the edge $(v_1, v_2) \in E$

since the birth time of v_2 is equal to its death time and the only vertex with height less than v_2 in direction e_1 is v_1. $\overline{D}_0(-e_1)$ gives us that $(v_2, v_3) \in E$ for the same reason. Then, since $\overline{D}_1(e_1)$ contains no cycles by Construction 1 Property 4, there must be no other edges and E is reconstructed.

Assume that we can reconstruct the edges of a graph G with $n_0 \geq 3$ vertices. Then, we prove that we can reconstruct G with $n_0 + 2$ vertices. Let G have $n_0 + 2$ vertices and satisfy the requirements of Construction 1, $\overline{D}_0(e_1)$ gives us that the edge $(v_1, v_2) \in E$ since the birth time of v_2 is equal to the death time and the only vertex with height less than v_2 in direction e_1 is v_1 by Property 3. Furthermore, $\overline{D}_0(e_1)$ gives us that vertex v_{n_0+2} has degree one since only one death event occurs at height $e_1 \cdot v_{n_0+2}$ and no birth events occur in $\overline{D}_1(e_1)$. $\overline{D}_0(-e_1)$ gives us that $(v_{n_0+1}, v_{n_0+2}) \in E$ for the same reason that $(v_1, v_2) \in E$. Furthermore, $\overline{D}_0(-e_1)$ gives us that vertex v_1 has degree one since only one death event occurs at height $-e_1 \cdot v_1$ and no birth events occur in $\overline{D}_1(-e_1)$ at $-e_1 \cdot v_1$. Then, since both v_1 and v_{n_0+2} have degree one and we have identified both edges incident to those vertices, we no longer need to consider any edges which could involve v_1 or v_{n_0+2}. Then, our remaining vertex set $V \setminus \{v_1, v_{n_0+2}\}$ has size n_0, which implies that we can reconstruct the edges $E \setminus \{(v_1, v_2), (v_{n_0+1}, v_{n_0+2})\}$ by our inductive assumption.

Finally, we prove that $\Omega(n_0)$ ECCs are necessary for reconstructing G. Each $v_i \in \mathbb{K}_0$ for $1 \leq i \leq n_0$ with even i is degree two. Then, by Lemma 17 and Observation 1, each v_i requires a direction $s \in \mathbb{S}^{d-1}$ that lies in the observing region for v_i to be determined. Furthermore, each v_i has an observing region disjoint from any other v_j, for $1 \leq j \leq n_0$, where j is even and $i \neq j$, by Construction 1 Property 5. Then, there are $\Theta(n_0)$ disjoint observing region for degree two vertices and $\Theta(n_0)$ directions must be used to determine each vertex by Observation 1. Thus, we conclude that $\Omega(n_0)$ ECCs are necessary for reconstructing G. $\square$

Theorem 10 provides the first evidence that fewer APDs are required for certain

simplicial complexes than ECCs. We can also consider the augmented version of the ECC as well. Recall from Definition 8 that the AECC is an ECC augmented with intervals at the height of each vertex in direction s. For $s \in \mathbb{S}^{d-1}$, we denote the AECC by $\bar{\chi}_s(\mathbb{K})$. This augmentation overcomes problems associated with missing degree two vertices and reconstruction of the zero-skeleton of a simplicial complex is greatly simplified. However, we show in Theorem 11 that strictly more AECCs are necessary for reconstruction in certain situations than APDs. First, we make an observation about the requirements for reconstructing two-simplices using the AECC.

Lemma 18. *Let $\mathbb{K} \subset \mathbb{R}^2$ be a simplicial complex consisting of at least one two-simplex $\sigma \subset \mathbb{K}$ disjoint from the rest of $\mathbb{K}$. Let the vertex set V of $\mathbb{K}$ be given as input. Let $v_1, v_2, v_3 \in \mathrm{verts}(\sigma)$ be the vertices of σ. If $\angle v_1 v_2 v_3$ is the largest angle in the two-simplex, then identifying σ as a two simplex requires at least one AECC generated from a direction $s \in \mathbb{S}^1$ with n_0 filtration hyperplanes such that*

$$s \cdot v_1 < s \cdot v_2 \textit{ and } s \cdot v_3 < s \cdot v_2.$$

Proof. Assume that we have determined that $(v_1, v_2) \in \mathbb{K}_1$, $(v_2, v_3) \in \mathbb{K}_1$, and that (v_1, v_2, v_3) is a connected component disjoint from the rest of $\mathbb{K}$. Then, let $s \in \mathbb{S}^1$. We simplify our notation by writing s satisfies $[v_1, v_2, v_3]$ to mean that $s \cdot v_1 < s \cdot v_2 < s \cdot v_3$. Then, let $s \in \mathbb{S}^1$ be any direction not satisfying $[v_1, v_3, v_2]$ or $[v_3, v_1, v_2]$. We have four possible scenarios, all of which follow the same argument. As such, without loss of generality, we consider $[v_1, v_2, v_3]$ and apply the same argument to $[v_3, v_2, v_1]$, $[v_2, v_1, v_3]$, $[v_2, v_3, v_1]$. If $s \cdot v_1 < s \cdot v_2$ and $s \cdot v_2 < s \cdot v_3$ then we know that the AECC does not change at height $s \cdot v_2$ because $(v_1, v_2) \in \mathbb{K}_1$. However, when the AECC does not change at height $s \cdot v_3$, we can attribute this to $(v_2, v_3) \in \mathbb{K}_1$ or, since we do not know if $(v_1, v_3) \in \mathbb{S}^1$, it could be that (v_1, v_3) is also an element of $\mathbb{K}_1$ and $(v_1, v_2, v_3) \in \mathbb{K}_2$. Furthermore, notice

that by assuming we have determined $(v_1, v_2) \in \mathbb{K}_1$, $(v_2, v_3) \in \mathbb{K}_1$, and that (v_1, v_2, v_3) is a connected component disjoint from the rest of $\mathbb{K}$, we have only reduced the number of constraints that we must consider. Thus, if we can show that we must still sample a direction satisfying $[v_1, v_3, v_2]$ or $[v_3, v_1, v_2]$, even with this assumption, then the claim holds.

We know that σ is disjoint and none of the four scenarios described above (directions satisfying $[v_1, v_2, v_3]$, $[v_3, v_2, v_1]$, $[v_2, v_1, v_3]$, or $[v_2, v_3, v_1]$) determine if $(v_1, v_3) \in \mathbb{K}_1$ or if $(v_1, v_2, v_3) \in \mathbb{K}_2$. However, we know that if we can determine $(v_1, v_3) \in \mathbb{K}_1$ or $(v_1, v_2, v_3) \in \mathbb{K}_2$, the other is also implied. Then, the only remaining directions are $s \in \mathbb{S}^1$ satisfying $[v_1, v_3, v_2]$ or $[v_3, v_1, v_2]$. We can determine if $(v_1, v_3) \in \mathbb{K}_1$ using $s \in \mathbb{S}^1$ satisfying $[v_1, v_3, v_2]$ (or $[v_3, v_1, v_2]$) since the AECC will not change at $s \cdot v_3$ (or $s \cdot v_1$) implying the edge (v_1, v_3) must exist since (v_1, v_2, v_3) is disjoint. Moreover, $(v_1, v_3) \in \mathbb{K}_1$ implies that $(v_1, v_2, v_3) \in \mathbb{K}_2$. Since we could not determine if $(v_1, v_2, v_3) \in \mathbb{K}_2$ using any directions not satisfying $[v_1, v_3, v_2]$ or $[v_3, v_1, v_2]$, we need to sample a direction $s \in \mathbb{S}^1$ such that $s \cdot v_1 < s \cdot v_2$ and $s \cdot v_3 < s \cdot v_2$, as required. $\square$

Lemma 18 is similar to [40, Example 7.4] but generalizes the idea of observing a zero-simplex with an ECC to identifying a two-simplex with an AECC. Next, we notice that, for a two-simplex σ composed of vertices v_1, v_2, and v_3, with $\angle v_1 v_2 v_3$ as the largest angle, identifying σ is similar to observing v_2 as a degree two-vertex. In fact, one of the sides of the observing region for v_2 is exactly the region we can sample directions from in Lemma 18 that will observe the two-simplex. As such, when we speak about reconstructing a two-simplex, we refer to the observing region for a two-simplex σ, as the set of directions $s \in \mathbb{S}^1$ satisfying $s \cdot v_1 < s \cdot v_2$ and $s \cdot v_3 < s \cdot v_2$.

Construction 2 (Lower-bound simplicial complex)**.** *We construct $\mathbb{K} \subset \mathbb{R}^2$ as a simplicial complex, where V_x denotes the vertices of V sorted by x-coordinate and V_y denotes the vertices of V sorted by y-coordinate, with the following properties:*

1. *$\mathbb{K}$ satisfies Assumption 1.*

2. *$\mathbb{K}$ is has $n_2 = \frac{n_0}{3}$ disjoint two-simplices and no additional simplices.*

3. *Any line parallel to e_1 intersects at most one two-simplex and any line parallel to e_2 intersects at most one two-simplex.*

4. *For all $v \in V$ the index of v is the same in V_x and V_y.*

5. *The observing regions for any two two-simplices in $\mathbb{K}$ are disjoint.*

We can see an example of a simplicial complex satisfying Construction 2 in Figure 3.8. Next, we prove that simplicial complexes satisfying this construction require strictly more AECCs for reconstruction than APD s.

Theorem 11. *Let $T = (V, E) \subset \mathbb{R}^2$ be the simplicial complex described in Construction 2 with $n_0 = |V|$. If V is known, then T can be reconstructed with $\Theta(1)$ APDs but requires $\Omega(n_0)$ AECCs.*

Proof. First, we prove that we can reconstruct T with a constant number of APDs. We prove this claim inductively on n_0. For the base case, consider $n_0 = 3$. Let us choose e_1 as a single direction and use the respective zeroth and first APDs to generate constraints. Since there is only a single component with an infinite death time in the zeroth APDs by Property 2 of Construction 2, we infer that the three known vertices (v_1, v_2, v_3) are a single component. Furthermore, since there is a single on-diagonal point in the first APD from direction s, we are able to determine that a cycle was born and immediately killed by a two-simplex. As such, the complex must be a single two-simplex consisting of vertices v_1, v_2, and v_3.

Assume that we can determine the simplicial complex for some value of $n_0 \geq 3$ that is divisible by three. Next, we show that we can determine the simplicial complex for

$n_0 + 3$. We choose directions e_1, $-e_1$, and e_2 and generate zeroth and first APD s in these directions. In directions e_1 and e_2 we determine that v_1, v_2, and v_3 are a single connected component and a two-simplex because, by Property 3 and Property 4 of Construction 2, $e_1 \cdot v_3$ and $e_2 \cdot v_3$ are smaller than the height value for $e_1 \cdot v_i$ or $e_2 \cdot v_i$ for any $3 < i \leq n_0 + 3$. Furthermore, by $-e_1$, we know that v_1, v_2, and v_3 are disconnected from the rest of the complex by Property 4 of Construction 2. Thus, we have n_0 vertices remaining in the unknown complex which we can reconstruct by the inductive hypothesis.

Next, we prove the $\Omega(n_0)$ AECCs are necessary to reconstruct T. We see that each two-simplex is disjoint from any other two-simplex and we know vertex locations, so by Lemma 18, each two-simplex requires a direction sampled from a region in its observing region to be detected. Then, since Property 5 of Construction 2 requires that each two-simplex has a disjoint observing region, we need at least $\Omega(n_0)$ AECCs to determine the two-simplices of T, as required. □

Theorem 11 implies that there exist scenarios where the APD is preferable for reconstruction over the ECC, even when augmented. Furthermore, Theorem 11 requires that the vertex locations are known apriori, but we note that this is not necessary for the APD since the vertex locations can be reconstructed using three diagrams by [11, Theorem 5]. Thus, the entire complex can be reconstructed using a constraint based approach and only four APDs.

Corollary 8. *If $T = (V, E) \subset \mathbb{R}^2$ is a simplicial complex satisfying the constraints of Construction 2, then T can be reconstructed with four APDs.*

3.7.1 Contributions to Section 3.7

In Section 3.7, we formalized several of the ideas from Section 3.6 using terminology adopted from [40] and adapted to prove lower bounds on the number of descriptors

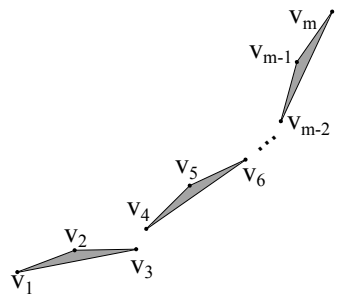


Figure 3.8: Example of a simplicial complex satisfying Construction 2. $\Omega(n_0)$ directions are necessary to reconstruct the simplicial complex using the AECC. Only $\Theta(1)$ directions are necessary for the APD to reconstruct the simplicial complex.

necessary to reconstruct certain classes of simplicial complexes. The work in this section was initially submitted by Samuel Micka and David L. Millman as a Young Researchers Forum extended abstract [80]. However, given the two-page limit, formal proofs were not included but informal proofs had been completed prior to submission. As a result, Samuel Micka adapted the notation of [40], updated the statements, developed constructions, and proved the lower bounds claimed in [80] formally. The work is to appear in [54].

3.8 Experiments on Challenges in Reconstruction

In Section 3.7, we identified situations where the observing region of particular vertices can be arbitrarily small in a theoretical setting. Recall that these observing regions are important for reconstruction and representation alike. Here, we investigate the distribution and size of observing regions, and other regions on the sphere useful for reconstruction, in real (contours) and synthetic (random point cloud) data sets as well as the effectiveness of uniform sampling techniques. We note that the work in this section is to appear in [54]. With our experiments, we aim to identify the strength of the relationship between our independent and dependent variables. To measure the goodness of fit of our models, we use the *mean squared error* (MSE).

First, we observe that the directions chosen from regions on $\mathbb{S}^{d-1}$ for which vertex orderings remain consistent in $\mathbb{R}^d$ form a stratification of $\mathbb{S}^{d-1}$. Here, each stratum corresponds to an ordering for a specific (but not necessarily unique) set of vertices. Curry et al. proved that it is possible to infer all PDs (or ECCs) in a strata if one direction from the strata is chosen [40]. Thus, sampling from each strata is sufficient for recovering the ECT or PHT. As such, investigating the strata sizes in data is important to determine the effectiveness of techniques like uniform sampling directions for generating sets of PDs or ECCs with height filtrations for shape representation. Intuitively, missing a stratum during sampling could lead to lost information about the shape. Moreover, missing multiple strata could result in missing an observing region and, as a result, failing to capture information about entire simplices. For the sake of this work, we focus on a stratification in $\mathbb{R}^2$ but note that this generalizes naturally to higher dimensions. To identify strata boundaries for a shape with n_0 vertices, we consider a stratification induced by directions for which vertices change order in the filtration. We refer to this as the *orthogonal stratification*.

Definition 21. *Let $\mathbb{K} \subset \mathbb{R}^2$ be a simplicial complex satisfying Assumption 1. Let E' be equal to all $\binom{n_0}{2}$ possible edges. Then, the* orthogonal stratification *is*

$$\{\ell^\perp \textit{ intersecting the origin} \mid \ell \textit{ intersects } x, y \textit{ for } (x, y) \in E'\} \cap \mathbb{S}^1.$$

Remark 3. *We note that this stratification is the two-dimensional version of the hyperplane division and resulting stratification described in [40, Definition 5.13].*

These orthogonal lines define strata boundaries for $2\binom{n_0}{2}$ strata on $\mathbb{S}^1$ where height filtrations on the simplicial complex have a new vertex ordering. Certain strata boundaries also contain (possibly overlapping) regions, where particular vertices are observed as critical points in the height filtration, i.e, the observing regions.

3.8.1 Data

In our experiments, we employed three types of data: random point clouds, the MPEG7 dataset [92], and a subset of the EMNIST data set (an extension of the MNIST data set) [36]. [2]

For the random (synthetic) data, we generated 100 point clouds for each value of k in $\{3, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, where k represents the number of vertices. The vertices had uniformly random coordinates generated from the `random` python package in a 10.0×10.0 box in the plane for a total of 1200 point clouds.

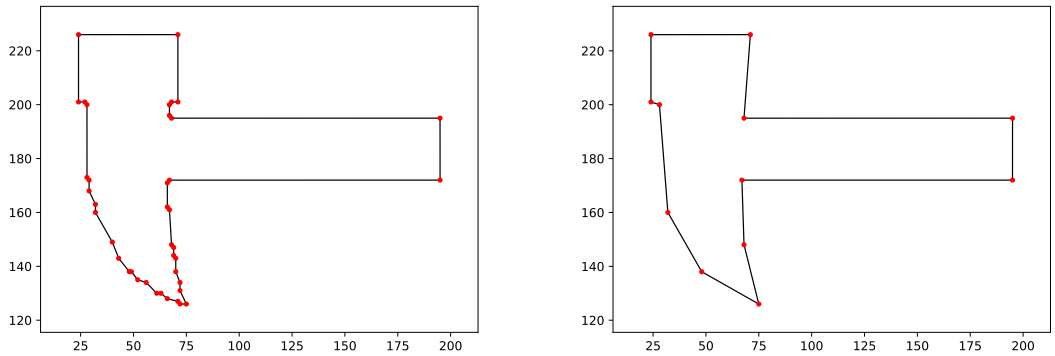
Initially, for the MPEG7 dataset, we started with 1400 images and, for the EMNIST dataset, we chose the first 100 samples from each of the 62 classes. For both MPEG7 and EMNIST data we followed similar data cleaning procedures to generate contours, so we describe them together. Contours are particularly appealing for our experiments because of the presence of degree-two vertices and available algorithms for generating contours from images. We computed the contours for MPEG7 and EMNIST images using python 2.7's `cv2` library's `threshold` and `findContours` functions, with the threshold set to zero for MPEG7 (using the BINARY thresholding method). The MPEG7 dataset is already binary and the EMNIST is grey scale. As such, we sampled the first image from each class of the EMNIST data set and used OTSU's thresholding to determine an appropriate threshold. We averaged the thresholds chosen for each of the 62 images for a final threshold value for the EMNIST images of 102.951 (standard deviation of 7.631). We chose to fix a constant threshold to ensure that the contours are generated similarly for each image.

To reduce the number of vertices and simplify the contours, we utilized a simple approximation method (eliminating degenerate vertices). We further simplified each contour by creating an approximate curve and eliminating all points within a certain ϵ of

[2] Data can be found at: `https://github.com/lionux/stratification-data` commit number 40a8c2d

the approximate curve. With $arcLength$ representing the length of the perimeter of each contour, we set our $\epsilon = .001 * arcLength$ (and later $\epsilon = .005 * arcLength$ to see the effects of coarser contours) of each contour using the `cv2 approxPolyDP` function [18] (i.e., the Douglas-Peucker algorithm [45]). We kept the contour with the longest perimeter for each image and stored the contour as a geometric networkx graph [60] consisting of vertices and undirected edges defining the contour. The contour approximation has different affects on the EMNIST and MPEG7 data sets since the perimeters in MPEG7 are substantially longer and the vertex sets much larger, but the primary goal of the approximations was to reduce the input to a manageable size for our experiments. We note that these approximate contours have several "nearly collinear" points removed, which affects the experiments. Specifically, we expect that this approximation eliminates several especially small strata in the stratification. As such, we test both $\epsilon = .001 * arcLength$ and $\epsilon = .005 * arcLength$ to verify that trends in the data are consistent. We note that the change in ϵ may not have an affect on some contours, but our goal is to verify trends in the data set as a whole and not necessarily for specific samples. For clarity, we denote the EMNIST data set (and MPEG7) with $\epsilon = .001 * arcLength$ as EMNIST$_{.001}$ (and MPEG7$_{.001}$) and the EMNIST data set (and MPEG7) with $\epsilon = .005 * arcLength$ as EMNIST$_{.005}$ (and MPEG7$_{.005}$).

To ensure that no three points were collinear and that no two vertices shared an x- or y-coordinate, we performed perturbations on the contour points in the EMNIST and MPEG7 data sets using a random uniform distribution with values chosen from the interval $[-.01, .01)$ from the python `numpy` library's `random.uniform` function [89]. A random value was added to each coordinate of each vertex. Small perturbations keep the data as close to the original as possible while ensuring that we can adhere to general position assumptions. However, for data, such as the EMNIST data set, where the coordinates are all integer and in a 28 by 28 grid, these small perturbations can create very small stratum. We note that such small perturbations may have an affect on our experiments, but choose



(a) .001 approx

(b) .005 approx

Figure 3.9: From the MPEG7 (and EMNIST) data we extract the contours using a binary thresholding method and a simple approximation. Then, we keep the longest contour. In our experiments, we use their perimeters ($arcLength$) to further simplify the approximation using the Douglas-Peucker algorithm [45] as implemented in `cv2` `approxPolyDP` with $\epsilon = .001 * arcLength$ (see left image) and $\epsilon = .005 * arcLength$ (see right image). As ϵ increases, fewer vertices are included in the approximation.

to keep them small for consistency and to differentiate between random data, real data with small integer coordinates (EMNIST), and real data with larger integer coordinates (MPEG7). Sometimes when the curves are approximated, vertices are removed and the graph self-intersects, as such we check each resulting contour to ensure that the points define a simple polygon. If the contour fails to be a simple polygon, has points sharing coordinates, or has three points that are collinear the sample is not considered in our experiments. We also removed one MPEG7 image for having inverted colors. After generating and cleaning the data, MPEG7$_{.001}$ had 1387 graphs, MNIST$_{.001}$ had 5563 graphs, MPEG7$_{.005}$ had 1364 graphs, and MNIST$_{.005}$ had 5751 graphs. All decimal values in this section are rounded at three decimal positions. We include an example of the different contour approximations in Figure 3.9.

3.8.2 Smallest Stratum Size Distribution

In this experiment, we investigate the size of the smallest stratum m using the stratification technique described in Definition 21 as the number of vertices n_0 increases. We choose to log-transform the data to improve the fit of the linear models. Each point in the plots has independent variable (x-value) $\log(n_0)$ and response variable (y-value) $\log(m)$, where m denotes the smallest stratum size, in radians, for that graph.

First, we consider the random data set. In this data set, we have 100 examples for each number of vertices and we consider fitting a linear model with $\log(m)$ as the response and $\log(n_0)$ as the independent variable. A plot showing the resulting data and curve plotted with n_0 on the x-axis and $\log(m)$ on the y-axis is shown in Figure 3.10(c). The line $y = c_0 + c_1 n_0$ had coefficients $c_0 = 3.175$ and $c_1 = -4.281$. Since a log-log transformation was used on the data, we interpret the coefficient c_1 as an exponent in the original function. The MSE of the experiment was 1.492 and the adjusted R-squared value was 0.939, implying a strong negative correlation between n_0 and the natural logarithm of the minimum stratum size. We infer that the size of the $\log$ of the smallest stratum in the random data set appears small, very quickly, as a function of the $\log$ of the number of vertices. And in the original data set, we infer that the size of the smallest stratum decreases as n_0 increases and more drastically for smaller values of n_0.

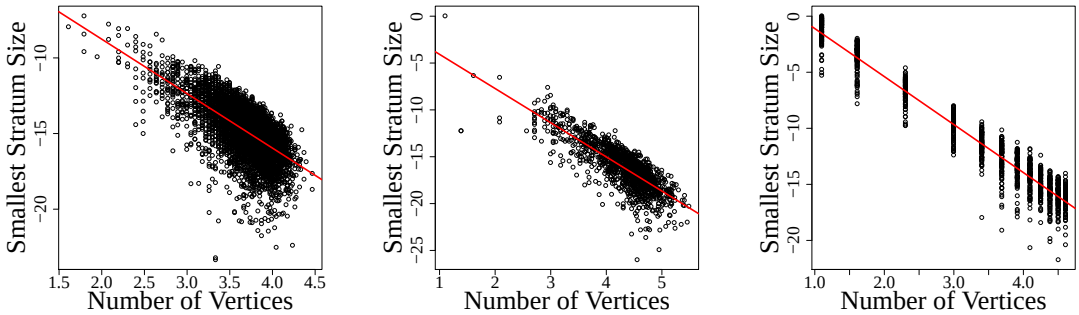
For EMNIST$_{.001}$, we fit $y = c_0 + c_1 n_0$ with response variable $\log(m)$ and independent variable $\log(n_0)$. The resulting curve had coefficients $c_0 = -1.563$ and $c_1 = -3.600$ and can be seen fitted to the data in Figure 3.10(a). The MSE of the experiment was 1.694 and the adjusted R-squared value was 0.432, implying a moderate negative correlation between $\log(n_0)$ and the natural logarithm of the minimum stratum size. As such, as the number of vertices increases, we can expect to see smaller stratum appearing in the stratification with more drastic changes being seen for smaller values of n_0. We note that since the EMNIST data set has small integer coordinate values, the perturbations chosen may have an affect on

the size of the resulting stratum. Our goal is to demonstrate a negative correlation between the variables and not necessarily to focus on the size of the stratum themselves. However, an interesting research problem, left to future work, is to investigate various perturbation techniques and their affects on stratum size.

Since the contour approximations remove vertices, it is natural to ask whether a coarser approximation will yield a different correlation between m and n_0. To test this, we consider the data set EMNIST$_{.005}$. The resulting model had coefficients $c_0 = -1.598$ and $c_1 = -3.544$ and can be seen fitted to the data in Figure 3.11(a). The MSE of the experiment was 1.801 and the adjusted R-squared value was 0.210, implying a weak negative correlation between $\log(n_0)$ and the natural logarithm of the minimum stratum size. The R-squared value still implies a weak negative correlation on EMNIST$_{.005}$ and we infer from the model that coarsening the approximation does not completely alleviate the negative association between the variables, suggesting that the negative correlation (albeit weaker) is still present in coarser data.

For the MPEG7$_{.001}$ data, we fit our model using $\log(n_0)$ as the independent variable and $\log(m)$ as the response, plotting the results in Figure 3.10(b). The resulting model had coefficients $c_0 = -0.426$ and $c_1 = -3.648$, an MSE of 2.032, and adjusted R-squared value of 0.663, implying a moderate to strong negative correlation between $\log(n_0)$ and $\log(m)$. Moreover, we infer that as n_0 increases, we can expect m to shrink as well, with more drastic changes for smaller values of n_0. With larger integer coordinates in the MPEG7 data sets, we expect the perturbation size to affect the data less than the EMNIST data. But, we still consider experiments with a coarser representation of the contours in our data. So, we fit the MPEG7$_{.005}$ data, with a linear model using $\log(n_0)$ as the independent variable and $\log(m)$ as the response, plotting the results in Figure 3.11(b). The resulting line had coefficients $c_0 = -0.185$ and $c_1 = -3.535$, an MSE of 2.375, and adjusted R-squared value of 0.435, implying an moderate to weak negative correlation between $\log(n_0)$ and

$\log(m)$. Thus, like with the EMNIST data, we find a negative correlation between n_0 and m and still find a negative correlation in the coarsened data.

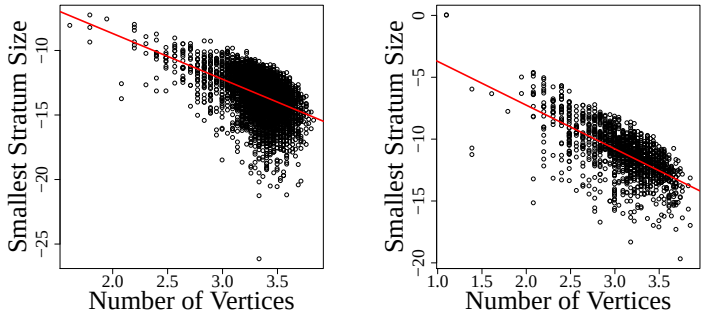


(a) EMNIST$_{.001}$ (b) MPEG7$_{.001}$ (c) Random

Figure 3.10: Log-log plot of the smallest stratum size versus the number of vertices for Figure 3.10(a) EMNIST$_{.001}$, Figure 3.10(b) MPEG7$_{.001}$, and Figure 3.10(c) random data sets with a $.001$ epsilon contour approximation. As detailed in Section 3.8.1, all three data sets are graphs embedded in $\mathbb{R}^2$. We infer from all three experiments, and the log scales, that there is a negative correlation between the number of vertices and the smallest stratum size.

3.8.3 Uniform Random Sampling

In Section 3.8.2 we saw that the size of strata induced by the exhaustive stratification gets smaller as the number of vertices increases. However, due to the complexity of determining directions for sampling a sufficient number of descriptors, some approaches have utilized sampling evenly spaced directions for height filtrations [39, 100]. Then, we ask, how effective are evenly spaced sampling techniques for sampling from each stratum? Sampling from each stratum is important for satisfying the sufficiency conditions for representation defined in [40, Theorem 7.14]. In this experiment, we sample $2^{14} = 16384$ samples uniformly spaced directions on $\mathbb{S}^1$. This number of samples is significantly larger than the number used in practice, where 72 directions were evenly sampled over $[0, 2\pi]$ in [39] and 64 directions were used in [100]. Our goal is to track the number of strata we



(a) EMNIST$_{.005}$ (b) MPEG7$_{.005}$

Figure 3.11: Log-log plot of the smallest stratum size versus the number of vertices for Figure 3.11(a) EMNIST$_{.005}$ and Figure 3.11(b) MPEG7$_{.005}$ contours data sets. As detailed in Section 3.8.1, both data sets embedded in $\mathbb{R}^2$. We infer from the two experiments, and the log scales, that there is a weaker, but still negative correlation between the number of vertices and the smallest stratum size, despite coarser contours.

miss (from an orthogonal stratification) using a uniform sampling approach as the number of vertices increases. For a point cloud with n_0 vertices, we consider the total number of strata from the orthogonal stratification approach from Definition 21 sampled using the uniform sampling approach (referred to as *hits*) over the total number of strata $N = 2\binom{n_0}{2}$. To ensure that an adequate number of samples were taken to hit each strata, we only include shapes from our data sets with fewer than $2\binom{n_0}{2} = 5000$ strata.

For the random data, our data set was reduced to 900 samples. We fit our model using n_0 as the independent variable and $\frac{hits}{N}$ as the response, plotting the results in Figure 3.12(c). Then, we fit a degree two polynomial to the data to correct for an uneven distribution in the residuals seen when fitting a line to the data. The resulting model has coefficients $c_0 = 0.956$, $c_1 = -1.347$, and $c_3 = -0.315$ and an MSE of $3.733711e-05$. Furthermore, the adjusted R-squared error was 0.983, implying a very strong negative correlation between n_0 and $\frac{hits}{N}$. Thus, we infer that using uniform random sampling to hit each strata in the random data set quickly loses effectiveness as the number of vertices increases.

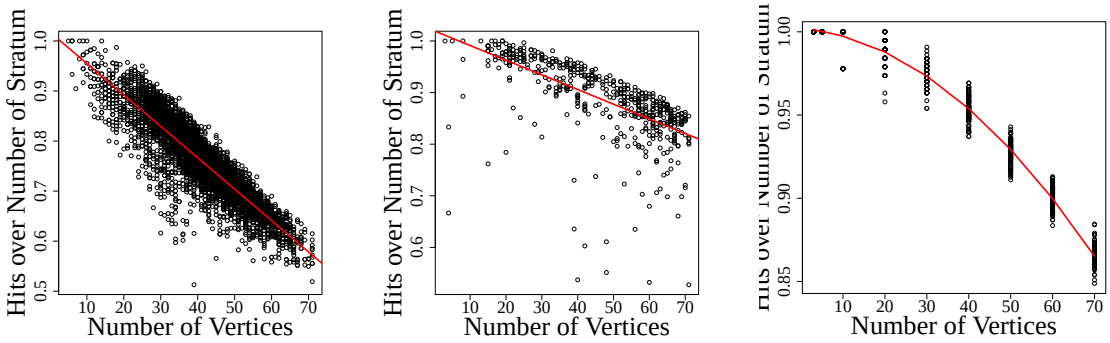
For the EMNIST$_{.001}$ data we considered 5543 samples, we fit our model using n_0 as the independent variable and $\frac{hits}{N}$ as the response, plotting the results in Figure 3.12(a). The resulting model had coefficients $c_0 = 1.018e + 00$, $c_1 = -6.275e - 03$, an MSE of 0.001, and R-squared value of 0.806, implying a moderately strong negative correlation between n_0 and $\frac{hits}{N}$. Thus, we infer that using uniform random sampling to hit each strata in the EMNIST data set quickly loses effectiveness as the number of vertices increases. Once again, since the contour approximations remove vertices, it is natural to ask whether a coarser approximation yields a different correlation between m and n_0? To test this, we fit EMNIST$_{.005}$ with 5751 samples using n_0 as the independent variable and $\frac{hits}{N}$ as the response, plotting the results in Figure 3.13(a). The model had coefficients $c_0 = 1.035e + 00$, $c_1 = -6.329e - 03$, an MSE of 0.001, and R-squared value of 0.469, implying a moderate negative correlation between n_0 and $\frac{hits}{N}$. The correlation grows weaker, but is still negative, so the coarser approximation does not drastically affect our interpretation of the results. As such, we suggest that uniform sampling will quickly lose effectiveness as the number of vertices increases in the EMNIST data set. However, we also point out that alternative perturbation techniques may yield different results. We imagine that larger perturbations (and larger strata) may improve sampling accuracy.

For the MPEG7$_{.001}$ data, our data set was reduced to 521 samples, we fit our model using n_0 as the independent variable and $\frac{hits}{N}$ as the response, plotting the results in Figure 3.12(b). The resulting model had coefficients $c_0 = 1.020$, $c_1 = -0.003$, an MSE of 0.003, and R-squared value of 0.410, implying a moderate negative correlation between n_0 and $\frac{hits}{N}$. We can infer that using uniform random sampling to hit each stratum in the MPEG7 data set loses effectiveness as the number of vertices increases. We test this, again, against MPEG7$_{.005}$ on 1364 samples by fitting a model using n_0 as the independent variable and $\frac{hits}{N}$ as the response, plotting the results in Figure 3.13(b). The resulting model had coefficients $c_0 = 1.006$, $c_1 = -0.002$, an MSE of 0.001, and R-squared value of 0.132,

implying a very weak negative correlation between n_0 and $\frac{hits}{N}$. However, we observe that there are several outliers in the data, weakening the correlations in both models. While these outliers may very well be a consequence of using real world image data, we are interested in seeing how the models are affected by removing these points. When we update the models to weight influential observations with a Cook's distance of more than $\frac{4}{521}$ for MPEG7$_{.001}$ and $\frac{4}{1364}$ for MPEG7$_{.005}$ to zero, we see the R-squared value increase to 0.706 ($c_0 = 1.034e + 00$, $c_1 = -2.943e - 03$, and MSE of 0.001) for the MPEG7$_{.001}$ data and 0.427 ($c_0 = 1.011e + 00$, $c_1 = -1.665e - 03$, and MSE of 0.0002) for the MPEG7$_{.005}$ data. The updated models consider 25 fewer samples from MPEG7$_{.001}$ and 33 fewer samples from MPEG7$_{.005}$. The influential point threshold is chosen based on the frequently recommended value of four over the number of observations (noted on [17, Page 266]). Thus, if we remove the influential observations, we can see a moderate to strong negative correlation between the number of vertices and the number of strata missed using evenly spaced samples.

3.8.4 Value of δ

Curry et al. use a method for determining vertex locations based on δ-observability (similar to Definition 18) [40, Definition 7.3]. The value of δ is related to a lower-bound on the "flatness" of any simplex in the complex. Similarly, for reconstruction, we saw several instances with degree two vertices where we were not able to reconstruct their locations without sampling from somewhere inside of the observing region for that vertex. In the case of graphs, the value of δ, and the observing region in the case of degree two vertices, is closely linked to the angle between any three adjacent vertices. In the MPEG7 and EMNIST data sets, we store contours approximating the shapes at different resolutions. Thus, to examine how the "flatness" of simplices varies in real-world data, we consider the largest angle θ (less than π) between any three adjacent vertices in each

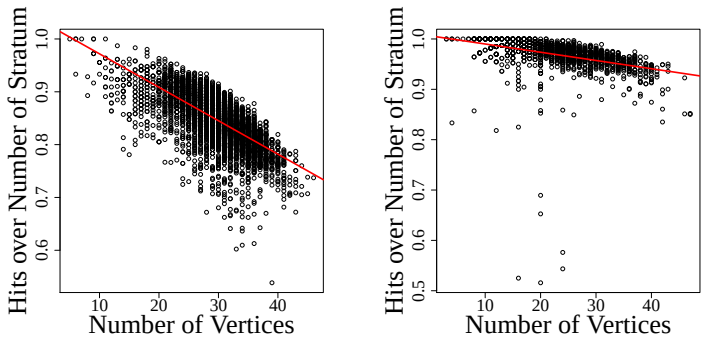


(a) EMNIST$_{.001}$ (b) MPEG7$_{.001}$ (c) Random

Figure 3.12: Plot of the ratio of hit stratum over the total number of stratum versus the number of vertices for Figure 3.12(a) EMNIST$_{.001}$, Figure 3.12(b) MPEG7$_{.001}$, and Figure 3.12(c) random graphs data sets using 16384 uniformly distributed directions from $\mathbb{S}^1$. For the random data set, we fit a degree-two polynomial to the independent variable n_0 to compensate for a parabolic distribution in the residual plot. As detailed in Section 3.8.1, all three data sets are graphs embedded in $\mathbb{R}^2$. We infer that using uniform random sampling to hit each stratum in the data sets loses effectiveness as the number of vertices increases.

shape and set $\delta = \pi - \theta$. Here, we look at the smallest δ value for each graph in the the MPEG7$_{.001}$, MPEG7$_{.005}$, EMNIST$_{.001}$, and EMNIST$_{.005}$ data sets. Intuitively, we would expect the value of the smallest δ to increase going from finer approximations ($.001$) to coarser approximations ($.005$), but results in the EMNIST data set, surprisingly, do not follow this trend.

For the EMNIST and MPEG7 data we consider the smallest δ (computed as described above) for each graph in both the $.001$ approximation and $.005$ approximation. We plot these values as histograms in Figure 3.14. On the left side of Figure 3.14, we have EMNIST$_{.005}$ at Figure 3.14(a) and MPEG7$_{.005}$ at Figure 3.14(c). The EMNIST$_{.005}$ data has a multimodal distribution, possibly due to the integer coordinates and small coordinate perturbations. The smallest value of δ in the EMNIST$_{.005}$ data set was 0.141 radians while the largest minimum δ was 0.784. The smallest value for δ in the MPEG7$_{.005}$ data set was 0.042 and the largest minimum δ was 2.323. A significant amount of variance is present in

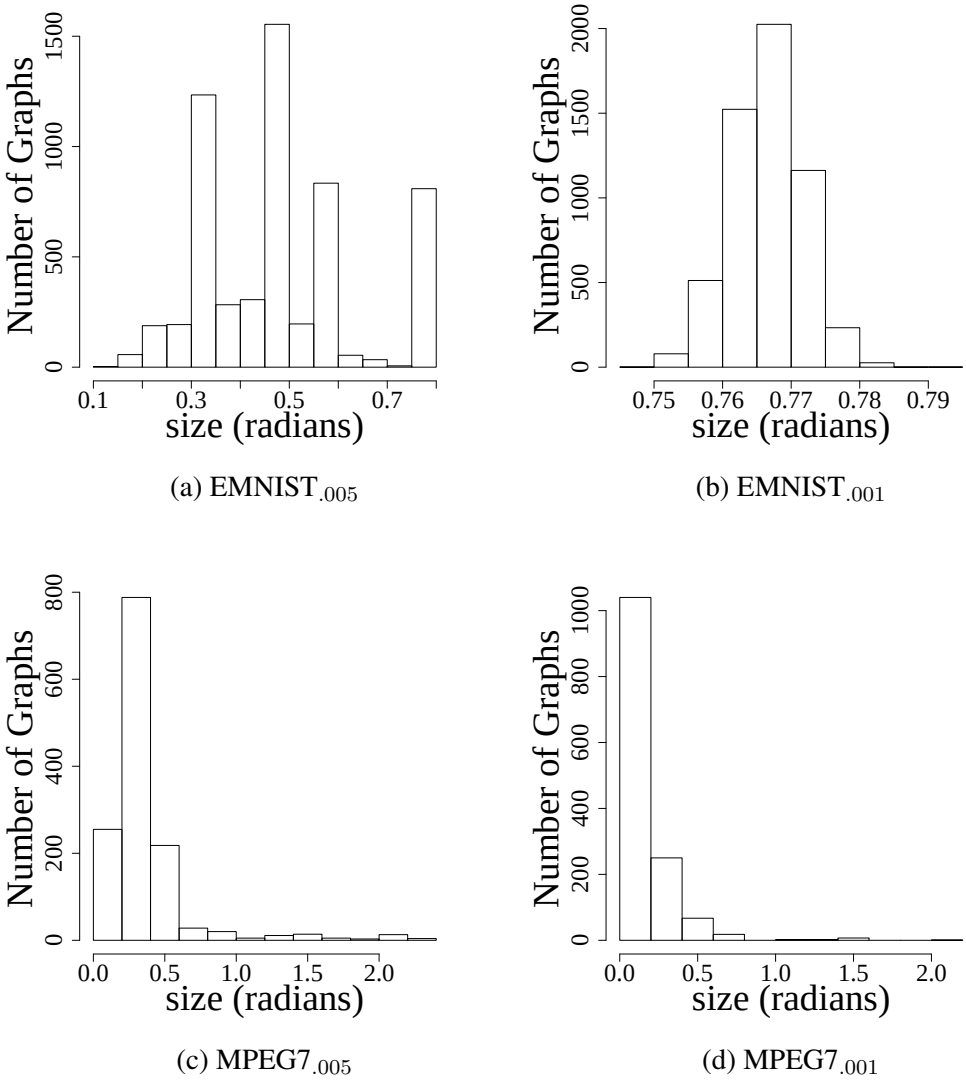


(a) EMNIST$_{.005}$ (b) MPEG7$_{.005}$

Figure 3.13: Plot of the ratio of hit stratum over the total number of stratum versus the number of vertices for Figure 3.13(a) EMNIST$_{.005}$ and Figure 3.13(b) MPEG7$_{.005}$ graphs data sets using 16384 uniformly distributed directions from $\mathbb{S}^1$. As detailed in Section 3.8.1, both sets are contours embedded in $\mathbb{R}^2$. We infer that using uniform sampling to hit each stratum in the data sets loses effectiveness as the number of vertices increases.

the MPEG7 data, in fact, the mean minimum δ size for the MPEG7$_{.005}$ data set was 0.369 and the third quartile was only 0.389, implying substantial outliers and a heavy right skew.

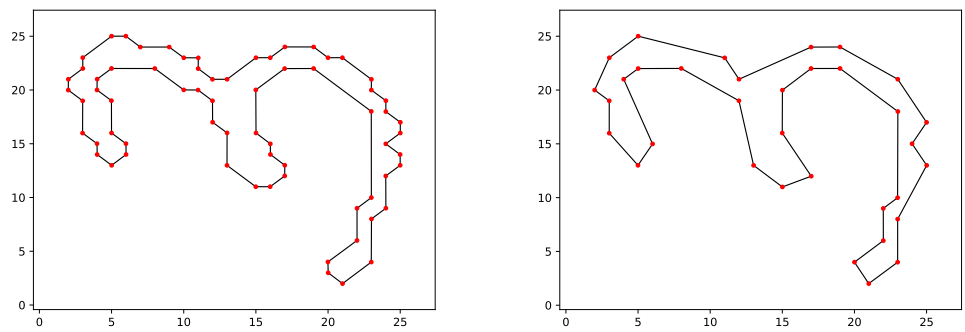
Next, we consider the $.001$ approximations on the right side of Figure 3.14. We plot EMNIST$_{.001}$ at Figure 3.14(b) and MPEG7$_{.001}$ at Figure 3.14(d). Since the $.001$ approximations offer a finer contour approximation than the $.005$ data sets, we initially expected to see the value of the smallest δ for each sample decrease. For the MPEG7$_{.001}$ data set, this is the case. The smallest δ value in MPEG7$_{.001}$ was 0.022, the average minimum δ was 0.184, the third quartile was 0.200, and the largest minimum δ was 2.087. While there are still outliers in the data, skewing the distribution right, we see that the minimum value, the mean, and quartiles all decrease. This is consistent with our expectations. However, when we consider the EMNIST$_{.001}$ data set, we find differing results. Specifically, we see the values of δ in EMNIST$_{.001}$ generally *increase* compared to the EMNIST$_{.005}$ data set. In fact, the smallest minimum δ value in EMNIST$_{.001}$ was 0.750 radians, an increase from 0.141 radians in EMNIST$_{.005}$. We suspect that this is



(a) EMNIST$_{.005}$ (b) EMNIST$_{.001}$

(c) MPEG7$_{.005}$ (d) MPEG7$_{.001}$

Figure 3.14: Histograms showing the distribution of the minimum δ for each sample (a variable lower-bounding how "flat" the local geometry of a simplex can be) on the EMNIST and MPEG7 data sets. We see EMNIST$_{.005}$ in Figure 3.14(a), EMNIST$_{.001}$ in Figure 3.14(b), MPEG7$_{.005}$ in Figure 3.14(c), and MPEG7$_{.001}$ in Figure 3.14(d). We expect to see δ decrease at the $.001$ approximation verses the $.005$ approximation. This is consistent with the data for MPEG7, but not EMNIST. We infer that coarsening the contour approximation is not necessarily going to allow us to increase our assumed lower-bound on the "flatness" of simplicies.



(a) .001 approx

(b) .005 approx

Figure 3.15: Example of minimum δ increasing from 0.383 in the EMNIST$_{.005}$ data set to 0.762 in the EMNIST$_{.001}$ data set. Here, we see that vertices in the EMNIST$_{.001}$ sample with larger values of δ are removed in the EMNIST$_{.005}$ to simplify the contours. We speculate that examples, like the one included here, are the cause of the larger δ values in EMNIST$_{.001}$ versus the EMNIST$_{.005}$ data set. Moreover, we can also infer that attempting to increase the assumed lower-bound on the "flatness" of simplices (i.e., δ) by coarsening the contour approximations is not effective on all data.

due to observations highlighted in Figure 3.15, where we see vertices in EMNIST$_{.001}$ being removed that induced larger minimum δ values. This is likely due to the coarse resolution of the images (i.e., 28 by 28 pixels). In general, this inconsistency implies that coarsening the contour approximations may not be an effective method for increasing the assumed lower-bound on simplex "flatness", especially on cubical data. Thus, assumptions about δ should be made based on specific data and may differ significantly, even between approximations of the same shapes. As such, substantial parameter tuning should be performed to determine an appropriate value for δ.

3.8.5 Contributions to Section 3.8

The work in Section 3.8 started in Summer 2019 with Samuel Micka and undergraduate Nicholas Dzomba under the advisement of Brittany Terese Fasy and David L. Millman.

At the end of the Summer, Samuel Micka took over running experiments and finished implementing several components of the framework. Samuel Micka generated the graphs, simplified the contours of real-world data, ran the experiments described in this section, and performed the analysis and interpretation of the results. The work described in this section is to appear in [54]. Samuel Micka is also very grateful for his discussions with Sally Slipher, Jordan Schupbach, and Stacey Hancock from the statistics department at Montana State University.

3.9 The Cost of Missing Simplices

In Section 3.8, we illustrated that the value of δ is highly dependent on the data and cleaning techniques. Moreover, in plane graphs, the smallest value of δ is linked to the size of the smallest observing region of degree two vertex. This result, along with the observations from Section 3.7, strengthens the argument for using APDs for reconstructing entire simplicial complexes. However, in some situations, reconstructing the entire complex may not be necessary, i.e., an approximate solution may suffice. Approximate solutions will trade efficiency for some error in the reconstructed shape. However, the amount of error remains an open question. In this section, we offer the first results exploring bounds on the distance between a simplicial complex $\mathbb{K}$ and a reconstructed simplicial complex $\mathbb{K}'$ when all observing regions are not sampled. We note that this work is to appear in [54].

Here, we provide a construction wherein a vertex is missed in reconstruction and consider the Hausdorff distance between the reconstructed complex and the original complex. We assume we have a graph $G = (V, E)$ embedded in $\mathbb{R}^2$ and a reconstructed version of G denoted $G' = (V', E')$, also embedded in $\mathbb{R}^2$. We note that $V' \subseteq V$ and

$E' \subseteq V' \times V'$. Then, we define the Hausdorff distance between the two graphs as

$$d_H(G, G') = \max(\sup_{x \in G} \inf_{x' \in G'} d(x, x'), \sup_{x' \in G'} \inf_{x \in G} d(x', x)).$$

In words, this is the largest minimum distance that a point (either a vertex or on an edge) in G must travel to the nearest point in G', or vice versa.

Construction 3 (Lower-bound plane graph)**.** *Let $G = (V, E) \subset \mathbb{R}^2$ be a plane graph with the following properties:*

1. *G satisfies Assumption 1,*
2. *$V = \{v, v_r, v_\ell\}$,*
3. *$E = \{(v, v_r), (v, v_\ell)\}$,*
4. *and v is equidistant from v_r and v_ℓ.*

Then, we consider the Hausdorff distance between the original graph and the reconstructed graph after missing the observing region for a single vertex.

Theorem 12. *Let $G = (V, E)$ be a plane graph satisfying the properties in Construction 3 with vertices $v, v_r, v_\ell \in V$ as defined above. Let $G' = (V', E')$ be a reconstruction of G with $v_\ell, v_r \in V'$ after choosing a finite number of directions to generate ECCs from $\mathbb{S}^1$. Let $\theta = \min(\angle v_\ell v v_r, \angle v_r v v_\ell)$. Let ℓ be the line intersecting v_r and v_ℓ and p^ℓ be the nearest point on ℓ to v. Then,*

1. *If no direction is chosen from the observing region of v and $(v_r, v_\ell) \notin E'$, then*

$$d_H(G, G') = \frac{d(v_\ell, v_r)}{2 \sin\left(\frac{\theta}{2}\right)}$$

.

2. *If no direction is chosen from the observing region of v and $(v_r, v_\ell) \in E'$, then*

$$d_H(G, G') = \frac{d(v_\ell, v_r)}{2\tan\left(\frac{\theta}{2}\right)}$$

.

Proof. If a direction is never chosen in the observing region for v, then v is never observed. Thus, v is not reconstructed and if G' has an edge $(v_\ell, v_r) \in E'$ then $d_H(v, G') = d_H(G, G') = \frac{d(v_\ell, v_r)}{2\tan\left(\frac{\theta}{2}\right)}$ which is the distance of the projection of v to p^ℓ. Otherwise, if $(v_\ell, v_r) \notin E'$ then we recall that $d(v, v_r) = d(v, v_\ell)$ by Property 4. As a result, $d_H(v, G') = d_H(G, G') = \frac{d(v_\ell, v_r)}{2\sin\left(\frac{\theta}{2}\right)}$, which is the distance from v to either v_ℓ or v_r. □

While examples satisfying Construction 3 are limited, we can generalize the idea to graphs containing specific types of degree two vertices.

Construction 4 (Lower-bound plane graph)**.** *Let $G = (V, E) \subset \mathbb{R}^2$ be a plane graph with the following properties:*

1. *G satisfies Assumption 1,*
2. *G contains at least one degree two vertex $v \in V$ with adjacent vertices $v_r, v_\ell \in V$,*
3. *$d(v, v_r), d(v, v_\ell) < d(v, v')$ for any other $v' \in V$ for which $v' \neq v, v_r, v_\ell$,*
4. *and v is the only vertex in V on one side of the line intersecting v_r and v_ℓ.*

Theorem 13. *Let $G = (V, E)$ be a plane graph satisfying the properties in Construction 4 with vertices $v, v_r, v_\ell \in V$ as defined above. Let $G' = (V', E')$ be a reconstruction of G after choosing a finite number of directions to generate ECCs from $\mathbb{S}^1$. Let ℓ be the line intersecting v_r and v_ℓ. Let $\ell^\perp$ be the line orthogonal to ℓ intersecting v, and $v_{int} = \ell \cap \ell^\perp$. If no directions are sampled from the observing region of v, and $(v_r, v_\ell) \in E'$ then $d(v, v_{int}) \leq d_H(G, G')$.*

Proof. First, we note that by Property 4, $d(v, v_e) < d(v, v_{e'})$ where v_e is a point on the edge (v_r, v_ℓ) and $v_{e'}$ is a point on any edge $e' \in E'$, $e' \neq (v_r, v_\ell)$. Then, if no directions are sampled from the observing region of v and $(v_r, v_\ell) \in E'$, we recall that v_r and v_ℓ are the nearest neighbors to v by Property 3 of Construction 4. Since $(v_r, v_\ell) \in E'$ and the nearest point on (v_r, v_ℓ) to v is v_{int}, the three nearest points to v in G' are v_{int}, v_r, or v_ℓ. However, we observe that $d(v, v_{int}) < d(v, v_r), d(v, v_\ell)$ by the triangle inequality. Thus, $d_H(v, G') = d(v, v_{int}) \leq d_H(G, G')$, as required.

Moreover, if no directions are sampled from the observing region of v and $(v_r, v_\ell) \notin E'$ then the nearest point to v in G' is lower-bounded by $d(v, v_{int})$ by Property 3 and Property 4 of Construction 4. Then, $d(v, v_{int}) \leq d_H(v, G') \leq d_H(G, G')$, as required. □

3.9.1 Contributions to Section 3.9

In Section 3.9 we have discussed the cost of missing vertices in particular classes of simplicial complexes in terms of the distance between the original graph and the reconstructed graph. The results in this section were motivated by discussions between Samuel Micka, Brittany Terese Fasy, and David L. Millman and will appear in [54]. Samuel Micka wrote the statements and proofs found above, but both Samuel Micka and Brittany Terese Fasy conjecture that it may be possible to find examples that make the distance between the graphs arbitrarily large.

3.10 Discussion

In this chapter, we have made progress towards addressing Research Question 1 using APDs and Research Question 2 describing our work from [11, 12, 54–56, 80]. Specifically, Section 3.2, Section 3.3, Section 3.4, and Section 3.5 addressed open problems related to Research Question 1. These algorithms provide deterministic methods of generating APDs for the purposes of shape reconstruction. These approaches complement the

work of [39, 40, 58], which focus on shape representation of simplicial complexes, the work of [14], which investigates problems related to representation and reconstruction of cubical complexes, and extends our work in [11, 12] by offering methods for generating a parameterized set of APDs that can be used to reconstruct the embedding of a shape. While APDs require more space to store than their non-augmented counterparts, this trade-off may be viewed positively in situations where reconstruction is a desired task.

Naturally, the set of APDs that are generated from the algorithms in Sections 3.2-3.5 are sufficient for representing the original simplicial complex. However, an interesting question that arises is whether or not the corresponding PDs (without the on-diagonal points used in the APDs) are sufficient representations for the shape, or not? The conditions for sufficiency highlighted in [40, Theorem 7.14] use a sample from each stratum from the hyperplane decomposition of $\mathbb{S}^{d-1}$ (see [40, Definition 5.13]). Our approach does not guarantee that we generate a diagram from each of these stratum. However, we conjecture that the stratum we sample from are sufficient and that diagrams from stratum we miss can be inferred using the information we already have. Specifically, we conjecture that the strata we miss contain no new information about specific simplices because we knowingly miss these strata based on prior knowledge. For example, we do not need to sample a stratum to test for a two-simplex if we know that there are not enough one-simplices to bound the face. We continue this discussion in future work at the end of this dissertation (see Section 5.1).

In the rest of the chapter, we investigated the feasibility of applying these techniques with other types of descriptors and explained several challenges in Sections 3.6-3.9. These results offer insights into the difficulties related to Research Question 2 and encompass results from [54,55,80]. Moreover, the results (particularly those in Section 3.7) strengthen the argument for exhaustive sampling approaches for shape representation with descriptors like the ECC, such as the methods described in [40]. Specifically, the exhaustive sampling

approach in [40] determines the vertex set which is used to identify the stratum that sufficiently determine the shape, and methods for improving this method, without strict assumptions on the data or modifications to the descriptors, are not clear. Finally, to the best of our knowledge, we provide the first lower-bounds on the distance between a complex and the reconstructed complex resulting from not sampling from particular observing regions in Section 3.9.

CHAPTER FOUR

SEARCHING IN THE SPACE OF PERSISTENCE DIAGRAMS

The results in this chapter come from our work [51]. With research utilizing the persistence diagram as a shape descriptor becoming more popular [2, 26, 43, 64, 76, 100], solutions for searching large sets of PDs are becoming increasingly important. The problem of *nearest neighbor search* was first posed in 1969 and is one of the most common types of searching problems [84]. The nearest neighbor search problem asks: given a set of objects S and a query point q, can we compute a data structure to efficiently return the nearest object to q from the set S? In earlier formulations, the problem focused on points in Euclidean space and low-dimensional domains. For this setting, the problem is well-solved by first computing a search structure on the Voronoi diagram of the data points and then performing point location queries for query points [47]. However, the complexity increases exponentially as the dimension increases (known as the "curse of dimensionality") [34]. To overcome these challenges, researchers resort to approximate nearest neighbor search [8]. For points in $\mathbb{R}^d$, the problem of searching for approximate nearest neighbors has also been well studied [28, 61, 77].

The problem of nearest neighbor search also expands beyond points in Euclidean space. In 2002, Indyk considered the data to be a set of n polygonal curves (each with at most m vertices) and the distance between two curves being the discrete Fréchet distance. In particular, a data structure of exponential size was built so that an approximate nearest neighbor query (with a factor $O(\log m + \log\log n)$) can be done in $O(m^{O(1)} \log n)$ time [66]. More recently, Driemel and Silvestri used locality-sensitive hashing to answer near neighbor queries on polygonal curves (within a constant factor) in $O(2^{4md} m \log n)$ time using $O(2^{4md} n \log n + nm)$ space (this bound is practical only for some $m =$

$O(\log n)$) [46].

In this chapter, we consider the problem of near-neighbor searching in the space of persistence diagrams. We emphasize that, while Chapter 3 utilized *augmented* persistence diagrams, in this chapter we focus our attention on *persistence diagrams* that are not augmented. Specifically, we work towards addressing Research Question 3 and, once again, note that the results and much of the discussion presented here are from [51]. Finding the exact nearest neighbor for a query diagram Q in a set of n diagrams (where any diagram has at most m points) can be done in $O(nm^{1.5} \log m)$ time using the algorithm described in by Kerber et al. [68]. However, as n increases, the computation time can become prohibitive. To address the expense, preliminary work by Kerber and Nigmetov investigated a cover tree representation of the space of persistence diagrams, but do not offer worst-case performance guarantees on searching due to the infinite doubling dimension of persistence diagrams under the bottleneck distance [69]. To reduce the complexity of comparing a query diagram to a set of diagrams, Di Fabio and Ferri represented persistence diagrams as complex polynomials and compared the persistence diagrams using complex vectors storing coefficients for the polynomials [44]. However, the results are experimental and offer no performance guarantees on the distance between two diagrams deemed to be close to one another by comparing the complex vectors [44]. Wang et al. extended the idea of converting persistence diagrams to complex vectors and prove stability results of the vector representation [101]. Specifically, for two diagrams P and Q, the distance between the resulting vectors v_P and v_Q has the following upper-bound:

$$||v_P - v_Q||_1 \leq \sqrt{2}(1 + \frac{\sqrt{\pi}}{\sigma})d_{W,1}(P, Q),$$

where $d_{W,1}$ denotes the 1-Wasserstein distance and σ is a variance parameter [101, Theorem 1]. However, without a lower-bound on the distance, the result does not naturally extend to

guaranteeing the distance between a returned diagram and the nearest neighbor. Mumey, simultaneously, investigated determining approximate nearest neighbors in the space of point sets [86], but in persistence diagrams, the inclusion of points on the diagonal with infinite multiplicity introduces additional complexities.

The remainder of this chapter is organized as follows: we provide preliminaries and necessary background in Section 4.1, develop a hashing scheme for approximating the location of points in PDs in Section 4.2, provide approximation bounds on a data structure and search technique for both the nearest neighbor and k-nearest neighbors using the hashed diagrams in Section 4.3, discuss the exponential time complexity in Section 4.4, and discuss the results in Section 4.5.

4.1 Preliminaries

In this section, we recall the definitions for persistence diagrams, bottleneck distance and additional concepts used throughout this paper.

4.1.1 Bottleneck Distance

In this chapter, we are working in the *space of persistence diagrams* under the bottleneck distance (see Definition 6). Recall that persistent homology tracks the *birth* and *death* of the topological features (i.e., the connected components, tunnels, and higher-dimensional 'holes') at multiple scales. Furthermore, a *persistence diagram* summarizes this information by representing the birth and death times (b and d, respectively) of homology generators as points (b, d) in the extended plane (see Definition 3). These birth and death times correspond to the appearances and merging of topological features in a filtered topological space. We simplify this representation by laying a grid over the diagram (see Section 4.1.2) and snapping points in the diagram to points in the grid using the L_∞ distance. Let D denote the diagonal (the line $y = x$) with points on D represented with

infinite multiplicity. Notice that points with small persistence are close to the diagonal and points with large persistence are far from the diagonal; in particular, the point (b, d) has distance $\frac{1}{2}\|d - b\|_\infty$ from D. In what follows, we set $M, m > 0$ and consider diagrams with at most m off-diagonal points such that each off-diagonal point with finite coordinates (a, b) satisfies $|a| \leq M$ and $|b| \leq M$ (we assume all points have finite birth times). We call such persistence diagrams (M, m)-bounded; see Figure 4.1.

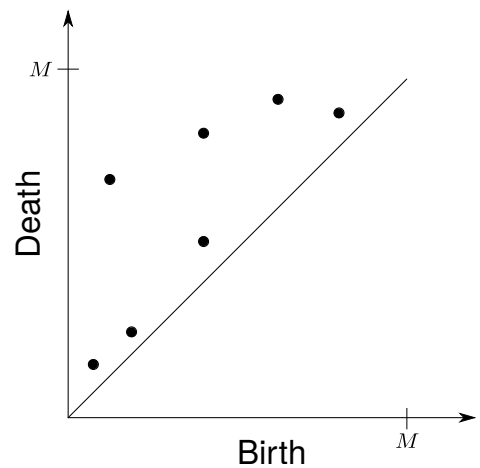


Figure 4.1: An example of an $(M, 7)$-bounded persistence diagram. Each point $p = (b, d) \in P \backslash D$ represents a topological feature–in particular, a homology generator. A point that is close to the diagonal (i.e., has small persistence), cannot be easily distinguished from *topological noise*. Standard persistence has points above the line $y = x$; however, extended persistence allows points above or below the diagonal.

Given persistence diagrams P and Q, recall from Definition 6 that the *bottleneck distance* between them is:

$$d_B(P, Q) = \inf_{\phi} \sup_{p \in P} \|p - \phi(p)\|_\infty,$$

where the infimum is taken over all bijections $\phi : P \to Q$.

Next, we prove a property about the space $\mathcal{D}_M^m$ with the bottleneck metric known as the *doubling dimension*. Following Clarkson's definition of the doubling space [35]:

Definition 22 (Doubling Dimension)**.** *Let X be a metric space with metric d. The space X is said to be* doubling *if there exists a constant integer $C > 0$ such that for every $x \in X$ and any $r > 0$ the open ball $B(x, r)$ can be covered by at most 2^C balls of radius $\frac{r}{2}$. The* doubling constant *of X is 2^C and the* doubling dimension *of X is C.*

Then, the following theorem highlights the problem with employing search approaches that depend on decomposing the search space, such as cover trees. We note that similar observations were made about persistence diagrams (not (M, m)-bounded) under the bottleneck distance in [69, 70].

Theorem 14. *The space of (M, m)-bounded persistence diagrams under the bottleneck distance has infinite doubling dimension.*

Proof. Assume, for contradiction, that there exists a constant C such that for any positive radius r and all $P \in \mathcal{D}_M^m$ that $B(P, r)$ can be covered by 2^C balls of radius $\frac{r}{2}$. Let $r = \frac{M}{2^C m}$ and let $P \in \mathcal{D}_M^m$ have exactly one point $\frac{r}{2}$ from the diagonal. Let $\mathbb{B}$ be the region greater than or equal to $\frac{r}{2}$ but strictly less than r from the diagonal. Note that $\mathbb{B}$ can be decomposed into $\frac{2M}{r}$ disjoint boxes with edge length $\frac{r}{2}$, we denote this decomposition as $\mathbb{B}'$. Let $\mathbb{D}$ be the set of diagrams with m points, where each of the m points lie at the center of a different box in $\mathbb{B}'$. Each $Q \in \mathbb{D}$ has m points lying at m of $\frac{2M}{r} = 2^{C+1}m$ possible locations. For any $Q, Q' \in \mathbb{D}$, where $Q \neq Q'$ we observe that $d_B(P, Q) < r$, $d_B(P, Q') < r$, and $d_B(Q, Q') \geq \frac{r}{2}$. In other words, we must cover every diagram in $\mathbb{D}$ but each requires a separate open ball of radius $\frac{r}{2}$. However,

$$2^C < 2^{C+1} \leq \binom{2^{C+1}m}{m} = \binom{\frac{2M}{r}}{m} = |\mathbb{D}|,$$

a contradiction to the claim that $B(P, r)$ can be covered by 2^C balls of radius $\frac{r}{2}$. $\square$

4.1.2 Uniform Grid

A M bounded *uniform grid* in $\mathbb{R}^2$ is a Cartesian grid with η^2 grid cells all contained in the box $[-M, M]^2$ and is denoted $\mathcal{L}_{M,\eta}$. We include $-M$ in the bounding box because the coordinates of points in the persistence diagrams are not necessarily positive. Grid cells are defined by four grid edges and four grid points. The *grid parameter* is the cell width of each grid cell and is denoted $\delta = 2M/\eta$. Since we are working with persistence diagrams, which may have points at ∞, we include two additional one-dimensional grids to handle points with a single infinite coordinate and one additional zero-dimensional grid cell to handle points with two infinite coordinates. The *complexity* of the grid is the number of grid cells: $|\mathcal{L}_{M,\eta}| = \eta^2 + 2\eta + 1$. For simplicity of exposition, we assume that no input points lie on either grid edges or equidistant from any two grid points. Thus, nearest grid points in the grid are unique, and every point has exactly four grid points defining the grid cell containing it.

4.1.3 Contributions to Section 4.1

The content of Section 4.1 is primarily definitions. However, an important theorem was proven to motivate the results in the following sections. Theorem 14 was discussed by Samuel Micka, David L. Millman, and Brittany Terese Fasy at length after talks with Michael Kerber and comments included in recent work [69, 70]. The proof of the theorem was ultimately written by Samuel Micka for the space of (M, m)-bounded persistence under the bottleneck distance.

4.2 Generating and Searching Persistence Keys

In this section, we define a key function that maps a persistence diagram in $\mathcal{D}_M^m$ to a vector in $\mathbb{Z}_{\geq 0}^a$, where the exponent a depends on the complexity of the grid. Hence, as a increases, the keys become more discerning. We order the diagrams using the dictionary

order on $\mathbb{Z}^a_{\geq 0}$, and store the keys in a multilevel data structure that supports binary search. We note here that the hierarchical grid is adapted from approaches to locality-sensitive hashing [61,63,67]. More recent general results on locality-sensitive can be found in [4–7, 33].

Let $M, m > 0$ and $\eta \in \mathbb{Z}_+$. Let $P \in \mathcal{D}^m_M$ be a persistence diagram. We consider the grid $\mathcal{L}_{M,\eta}$. We then snap each off-diagonal point $p \in P \backslash D$ to a grid point $\rho_i \in \mathcal{L}_{M,\eta}$ and count the multiplicity π_i for each grid point. The number of grid points from two-dimensional grid cells is $(\eta + 1)^2$, the number of grid points from one-dimensional grid cells is $2(\eta + 1)$, and there is a single zero dimensional grid cell with one point. We note that while our key function was inspired by the hash function of [46] that ignored multiplicities, we must count the multiplicity of duplicated grid points.

Recall from Section 4.1.2 that the grid parameter is $\delta = 2M/\eta$. We define our key function $\kappa \colon \mathcal{D}^m_M \times \mathbb{G} \to \mathbb{Z}^a$ by:

$$\kappa(P, \mathcal{L}_{M,\eta}) = \sum_{p \in P \backslash D} e_{nn(p)},$$

where $\mathbb{G}$ denotes the set of all uniform grids, $nn(p)$ maps each off-diagonal $p \in P$ to the index of the nearest grid point and e_i is the ith standard basis vector in $\mathbb{Z}^a$, where $a = (\eta + 1)^2 + 2(\eta + 1) + 1$; see Figure 4.2 for an example. For simplicity of the proofs to follow and so κ is well-defined, we assume that no persistence point lies on a grid edge of $\mathcal{L}_{M,\eta}$ or equidistant to any two grid points. Of course, since many coordinates of $\kappa(\cdot, \cdot)$ are zero, we store it using a sparse vector representation; moreover, for the empty diagram D (i.e., with no point but the line $y = x$), we notice that $\kappa(D, \cdot) = 0$.

Remark 4. *This vector could also correspond to a product of prime numbers, where $\{\sigma_j\}$ is an ordered set of a prime numbers. Then, we have a unique integer $\prod_j \sigma_j^{v_j}$ for each vector $v = (v_1, v_2, \ldots, v_a) \in \mathbb{Z}^a$. Doing so would put us in the more conventional setting,*

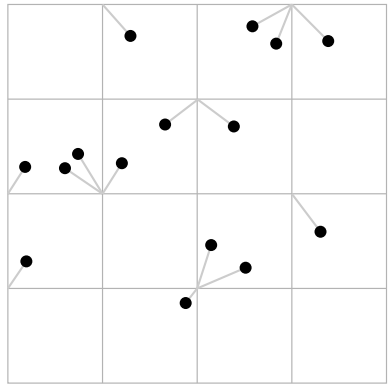


Figure 4.2: The key function snap-rounding each point of the input set to the nearest grid point. Note that the actual rounding produced by the key function is denoted as a line from each point to the corresponding grid point that it is rounded to.

where we have indices into a hash table instead of keys. However, we would then either need to have a pre-generated list of a primes (which adds to our storage space) or must account for computing the primes (which adds time complexity).

Suppose we snap-round P before applying κ; a natural choice for rounding each $p \in P$ would be to one of the (at most) four grid points defining the grid cell containing p. Formally:

Definition 23 (Snap Sets and Canonical Ordering)**.** *Given a grid $\mathcal{L}$ with grid parameter δ, let* $\textsc{Snap}(P, \mathcal{L})$ *denote the set of all possible snap-roundings of P obtained by allowing each $p \in P$ to snap-round to one of the grid points within L_∞ distance δ of p, i.e., one of the grid points bounding the cell containing p. For example, points with finite coordinates lie within δ of four grid points, points with one infinite coordinate lie within δ of two grid points, and points with two infinite coordinates lie within δ of one grid point. Let* $\textsc{DelSnap}(P, \mathcal{L})$ *denote the set of all snap-roundings of P obtained by additionally allowing $p \in P$ distance less than or equal to δ from the diagonal to be optionally deleted; see Figure 4.3 for an example of points that are eligible for removal.*

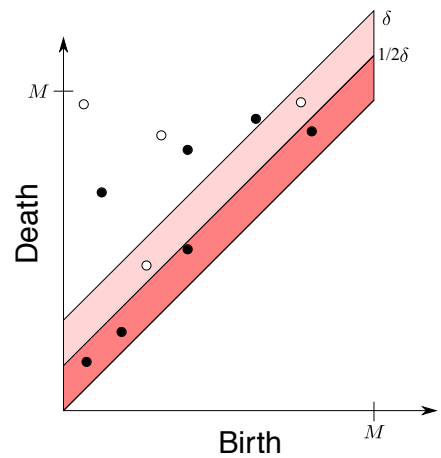


Figure 4.3: An example demonstrating the thresholds where points near the diagonal may be deleted. We consider diagrams $P, Q \in \mathcal{D}_M^m$ denoted with black and white points, respectively. The two thresholds are shown as bands above the diagonal denoting distances $\frac{1}{2}\delta$ and δ from the diagonal. Removing all black points within the first threshold will produce the diagram $\widetilde{Q}$ from Q. Removing some subset of white points within the second threshold may produce P_* from P.

For the remainder of this chapter, without loss of generality, we only consider points with four snap-roundings since points with an infinite coordinate value (or two infinite coordinate values) follow the same argument as points with a finite death time that are not too "near the diagonal", but with fewer snap-roundings. Then, each point $p \in P$, not lying on a grid edge, can snap to the four grid points defining the grid cell containing p. As such, we bound the number of snap-roundings for a fixed diagram:

Lemma 19 (Enumerating Keys)**.** *If $P \in \mathcal{D}_M^m$ and if $\mathcal{L}$ is a uniform grid centered on $[-M, M]^2$, then the number of keys in* SNAP$(P, \mathcal{L})$ *and* DELSNAP$(P, \mathcal{L})$ *have the following upper bounds:*

- SNAP$(P, \mathcal{L})$ *has size* $O(2^{2m}) = O(4^m)$.
- DELSNAP$(P, \mathcal{L})$ *has size* $O((2^2 + 1)^m) = O(5^m)$.

We are now almost ready to prove Theorem 15, which shows that a query diagram Q collides with a snapping of diagram P if and only if P and Q are close diagrams. (Note that this 'if and only if' statement uses asymmetric notions of *close*). We first prove a simplified version in Lemma 20, where we consider the perfect matching problem in the extended plane $\overline{\mathbb{R}}^2$. In this case, the proof is made easier as the two diagrams necessarily have the same number of points (as otherwise, a perfect matching is not possible). Then, to prove Theorem 15, we delete points that are less than or equal to distance $\frac{1}{2}\delta$ of D in the query diagram Q and observe that some of these points could have been matched with off-diagonal points in P with persistence up to, and including, $\frac{3}{2}\delta$.

Lemma 20 (Collision without Diagonal Interference)**.** *Let $P, Q \subset \overline{\mathbb{R}}^2$ be finite (M, m)-bounded point clouds. Let $\eta \in \mathbb{Z}_+$ and consider the grid $\mathcal{L} = \mathcal{L}_{M,2\eta}$. Let δ denote the grid parameter. Then,*

1. *if $\exists P_* \in \textsc{Snap}(P, \mathcal{L})$ such that $\kappa(P_*, \mathcal{L}) = \kappa(Q, \mathcal{L})$, then $d_B(P, Q) \leq \frac{3}{2}\delta$;*
2. *if $d_B(P, Q) \leq \frac{1}{2}\delta$, then $\exists P_* \in \textsc{Snap}(P, \mathcal{L})$ such that $\kappa(P_*, \mathcal{L}) = \kappa(Q, \mathcal{L})$.*

Proof. Recall from Section 4.1.2 that $\delta = \frac{2M}{\eta}$ and $|\mathcal{L}_{M,2\eta}| = (2\eta)^2 + 2\eta$.

For the first part, let $P_* \in \textsc{Snap}(P, \mathcal{L})$ such that $\kappa(P_*, \mathcal{L}) = \kappa(Q, \mathcal{L})$. Then, we construct a matching $\mathcal{M}$ between P and Q iteratively by peeling off pairs $(p, q) \in P \times Q$ that are mapped to the same grid point ℓ. By Lemma 19, we know that p was snap-rounded to one of the four grid points within distance δ of p. Additionally, we know that q was snap-rounded to the closest grid point. Hence, we have $d_\infty(p, \ell) \leq \delta$ and $d_\infty(q, \ell) \leq \frac{1}{2}\delta$. Thus, by the triangle inequality, we have $d_\infty(p, q) \leq \delta + \frac{1}{2}\delta = \frac{3}{2}\delta$; an example of this situation is shown in Figure 4.4(a). Therefore, if $\kappa(P_*, \mathcal{L}) = \kappa(Q, \mathcal{L})$, we have $d_B(P, Q) \leq \frac{3}{2}\delta$.

To prove the second part, we assume that $d_B(P, Q) \leq \frac{1}{2}\delta$. Recall that P and Q have the same number of points and let $\mathcal{M} \subset P \times Q$ be a perfect matching that realizes

(a) Scenario where in $d_\infty(p,q)$ a P_* exists that snaps p and q to ℓ. Notice that $d_\infty(p,\ell) \leq \delta$ and $d_\infty(q,\ell) \leq \frac{1}{2}\delta$.

(b) Scenario where $d_\infty(p,q)$ is just over $\frac{1}{2}\delta$ but p and q do not snap to the same grid point. Specifically, $d_\infty(p,\ell) \leq \delta$ and $d_\infty(q,\ell) > \frac{1}{2}\delta$.

Figure 4.4: Illustration for the proof of Lemma 20 demonstrating the two boundary-case scenarios, where two points are near one another with no snap-rounding, and where two points are far from one another but snap-round to the same grid point.

the bottleneck distance. For each pair $(p,q) \in \mathcal{M}$, let ℓ_q be the grid point to which q snaps in $\kappa(q,\mathcal{L})$, as illustrated in Figure 4.4(b). Then, $d_\infty(q,\ell_q) < \frac{1}{2}\delta$ because it is the nearest point to q. Since $d_B(P,Q) \leq \frac{1}{2}\delta$, we know that $d_\infty(p,q) \leq \frac{1}{2}\delta$ and $d_\infty(p,\ell_q) \leq d_\infty(p,q) + d_\infty(q,\ell_q) < \delta$ by the triangle inequality which implies that p and ℓ_q lie in the same grid cell. Thus, a snap-rounding of P exists such that each p is snapped to ℓ_q; we denote this snap-rounding P_*. Since $|\mathcal{M}| = |P| = |Q|$ and each pair in $\mathcal{M}$ share a grid point in $\kappa(Q,\mathcal{L})$ and $\kappa(P_*,\mathcal{L})$, we conclude that $\kappa(Q,\mathcal{L}) = \kappa(P_*,\mathcal{L})$. $\square$

The above lemma is restricted to matchings of points in $\overline{\mathbb{R}}^2$. Next, we generalize Lemma 20, by allowing matchings to the diagonal. This is the central theorem of this paper.

Theorem 15 (Collisions between Diagrams)**.** *Let $P, Q \in \mathcal{D}_M^m$. Let $\eta \in \mathbb{Z}_+$ and consider the grid $\mathcal{L} = \mathcal{L}_{M,2\eta}$. Let δ denote the grid parameter, and let $\widetilde{Q}$ be the diagram obtained from Q by removing all points less than or equal to distance $\frac{1}{2}\delta$ from the diagonal. Then:*

1. *If $\exists P_* \in$ DELSNAP$(P,\mathcal{L})$ such that $\kappa(P_*,\mathcal{L}) = \kappa(\widetilde{Q},\mathcal{L})$, then $d_B(P,Q) \leq \frac{3}{2}\delta$.*

2. *If $d_B(P,Q) \leq \frac{1}{2}\delta$, then $\exists P_* \in$ DELSNAP$(P,\mathcal{L})$ such that $\kappa(P_*,\mathcal{L}) = \kappa(\widetilde{Q},\mathcal{L})$.*

Proof. Again, recall from Section 4.1.2 that $\delta = \frac{2M}{\eta}$ and $|\mathcal{L}_{M,2\eta}| = (2\eta)^2 + 2\eta$.

We start with the first part. Let $P_* \in \textsc{DelSnap}(P, \mathcal{L})$ such that $\kappa(P_*, \mathcal{L}) = \kappa(\widetilde{Q}, \mathcal{L})$. Then, each off-diagonal persistence point $p \in P$ is either snap-rounded to one of its neighbors within distance δ or deleted in order to obtain P_*. Then, we construct a matching $\mathcal{M}$ between P_* and $\widetilde{Q}$ from Lemma 20 Part 1 with bottleneck cost at most $\frac{3}{2}\delta$; next, we add to the matching $\mathcal{M}$ in order to extend the matching to a perfect matching between P and Q by considering unmatched points of Q followed by unmatched points of P. Notice that all deleted points of Q are within $\frac{1}{2}\delta$ of the diagonal by construction. Furthermore, all points in $P \backslash P_*$ must have been deleted by κ, and hence are within δ of the diagonal. We add all of these pairs (i.e., between $P \backslash P_*$ and the diagonal, and between $Q \backslash \widetilde{Q}$ and the diagonal) to $\mathcal{M}$, thus obtaining a perfect matching between P and Q with bottleneck cost at most $\frac{3}{2}\delta$.

We now prove the second part. Assume that $d_B(P, Q) \leq \frac{1}{2}\delta$. Let $\mathcal{M} \subset P \times Q$ be a matching that realizes the bottleneck distance between P and Q. We use this matching to construct a diagram $P_* \in \textsc{DelSnap}(P, \mathcal{L})$ by choosing which points of P to snap-round to grid points and which points to delete; see Figure 4.3 for an example of the points that are eligible for removal during the snap-rounding. For each $(p, q) \in \mathcal{M}$ with $q \in \widetilde{Q}$, we know that $p \notin D$ (recall that D denotes the diagonal) since $d_B(P, Q) \leq \frac{1}{2}\delta$. Letting ℓ_q be the closest grid point to q (just as we did in Lemma 20), we snap-round p to ℓ_q in P_*. Next, we consider all $(p, q) \in \mathcal{M}$ with $q \notin \widetilde{Q}$ (i.e., $d_\infty(q, D) \leq \frac{1}{2}\delta$), where q is snapped to the diagonal. We must show that we can also delete p when constructing $P_* \in \textsc{DelSnap}(P, \mathcal{L})$, we know that

$$d_\infty(p, D) \leq d_\infty(p, q) + d_\infty(q, D) \leq d_B(P, Q) + \frac{1}{2}\delta \leq \delta,$$

and so we choose to delete p since it is within δ of D (q is deleted by the definition

of $\widetilde{Q}$). Then, we have matched all points in $\widetilde{Q}$ with points in P and all points that were deleted from Q when obtaining $\widetilde{Q}$ had their corresponding points in P deleted when constructing P_*. Therefore, we have constructed a diagram $P_* \in \textsc{DelSnap}(P, \mathcal{L})$ such that $\kappa(\widetilde{Q}, \mathcal{L}) = \kappa(P_*, \mathcal{L})$. $\square$

Theorem 15 implies that diagrams with a small bottleneck distance relative to the chosen δ value will have matching keys generated by the key function while diagrams with a large bottleneck distance, relative to δ, will not. Next, using Theorem 15, we discuss a multi-level data structure that, for some query diagram Q, supports searching for approximate nearest neighbors in $\mathcal{D}_M^m$.

4.2.1 Contributions to Section 4.2

In Section 4.2, we provided a key function for (M, m)-bounded persistence diagrams and proved certain properties about the relationship between the bottleneck distance between two diagrams and their corresponding keys. Discussions pertaining to this work were initially motivated by [46] when Binhai Zhu, Brittany Terese Fasy, and David L. Millman wondered if it were possible to apply similar techniques to persistence diagrams. Samuel Micka, Binhai Zhu, Brittany Terese Fasy, David L. Millman, Xiaozhou He, and Zhihui Liu developed much of the initial theory found in this section during Fall 2017. Most of the writing was done by Samuel Micka, Brittany Terese Fasy, and David L. Millman with frequent feedback and input from Binhai Zhu, Xiaozhou He, and Zhihui Liu. We also thank Brendan Mumey for his discussions and input on the snapping methods for point clouds (see [86]). Samuel Micka played a critical role in the proofs of Lemma 20, Theorem 15, and refinements to the grid after anonymous reviewer comments.

4.3 Approximate Nearest Neighbors

In Theorem 15, we saw that for a query diagram Q and grid parameter δ, the diagram Q shares a key with some diagram P 'if and only if' they are close, with respect to the chosen scale δ. To find the near-neighbor, we must select a δ with the correct relationship to $d_B(P, Q)$. The relationship presents two problems. First, how do we determine the correct value for δ? Second, a single δ value would rarely be sufficient for all queries.

In this section, we build a multi-level data structure to support approximate nearest neighbor queries in the space of persistence diagrams. Each level of the data structure corresponds to a grid with a different resolution. In the previous section, we needed a flexible notion for SNAP and DELSNAP, but in this section, the data structure level and grid are dependent. So, we simplify notation. Recall that, as our persistence diagrams are all (M, m)-bounded (i.e., in $\mathcal{D}_M^m$), all points lie in $[-M, M]^2$. For $i \in \mathbb{Z}_{\geq 0}$, we define

- $\mathcal{L}_i := \mathcal{L}_{M,2^{i+1}}$
- $\delta_i := 2M/2^i$, that is, the grid parameter for $\mathcal{L}_i$
- $\kappa_i(P) := \kappa(P, \mathcal{L}_i)$
- $\text{SNAP}_i(P) := \text{SNAP}(P, \mathcal{L}_i)$
- $\text{DELSNAP}_i(P) := \text{DELSNAP}(P, \mathcal{L}_i)$

Definition 24 (Data Structure)**.** *Let $\Gamma \subset \mathcal{D}_M^m$ be finite, let $c > \frac{3}{2}$, and let ϵ be the minimum bottleneck distance between any two diagrams in Γ. Let $\tau = \lceil \log((2M)/(c\epsilon)) \rceil$, then for each integer $i \in \{0, \ldots, \tau\}$, we define $\Delta_i = \Delta_i(\Gamma)$ to be the data structure that stores the sorted list of keys $\{\kappa_i(P_t)\}_{i,t,P}$, for each $P_t \in \text{DELSNAP}_i(P)$ and $P \in \Gamma$. With each key, we store a list of distinct persistence diagrams from Γ which have a snap-rounding to that key, and the number of distinct diagrams from Γ which snap to the key. We note that a diagram*

with a given key can be found in time logarithmic in the number of distinct keys at that level. We denote the array of the multi-level data structure as $\mathbb{D}_\tau = \mathbb{D}_\tau(\Gamma) := \{\Delta_i\}_{i=0}^{\tau}$. *We can access a given level in constant time.*

In the definition above, the choice of c and ϵ provides a point at which the diagrams with the smallest bottleneck distance stop colliding and we can stop considering smaller values of δ. In particular, we choose $c > \frac{3}{2}$, because the contrapositive of Theorem 15 Part 1 implies that if $d_B(P, Q) > \frac{3}{2}\delta_i$, then $\kappa_i(P_*) \neq \kappa_i(\widetilde{Q})$ for any $P_* \in \textsc{DelSnap}_i(P)$. Thus, we can guarantee that Q will share a key with a representative of P, for each Q close enough to P.

Remark 5. *For each level, each diagram has* $O(5^m)$ *snap-roundings and keys that can each be generated in* $O(m)$ *time. Comparing two keys to determine their relative order requires* $O(m)$ *time.*

For each diagram at each level, we can determine the set of unique keys in $O(m5^m)$ *by sorting the* $O(5^m)$ *keys and removing duplicates. Finding the unique keys for* n *diagrams takes* $O(nm5^m)$. *Sorting the keys at a given level for* n *diagrams takes* $O(m(n5^m \log n5^m)) = O(m(n5^m \log n + n5^m \log 5^m)) = O(m(n5^m \log n + n5^m m)) = O(mn5^m(\log n + m))$ *time. Creating a list of diagrams for each unique key at a given level requires* $O(n5^m)$ *time but this operation is asymptotically smaller than the complexity of sorting the keys. Then, generating the data structure* $\mathbb{D}_\tau$ *with* τ *levels takes* $O(\tau(mn5^m(\log n + m))$ *time.*

Next, we consider some properties of $\mathbb{D}_\tau$, specifically, that collisions on a level of the data structure with a fine resolution imply collisions between the same diagrams on levels with coarser resolutions. To simplify notation, for $Q \in \mathcal{D}_M^m$ and $i \in \mathbb{Z}_{\geq 0}$, we let $\widetilde{Q}_i$ be the diagram obtained from Q by removing all points less than or equal to distance $\frac{1}{2}\delta_i$ of the diagonal.

Lemma 21 (Hierarchical Collision)**.** *Let $\Gamma \subset \mathcal{D}_M^m$ be finite. Let $Q \in \mathcal{D}_M^m$ and $P \in \Gamma$. Let $j \in \mathbb{Z}_{\geq 0}$ and let $\widetilde{Q}_j$ and $\widetilde{Q}_i$ be the diagrams obtained from Q by removing all points less than or equal to distance $\frac{1}{2}\delta_j$ (resp., $\frac{1}{2}\delta_i$) of the diagonal. Suppose there exists $P_j \in$ DELSNAP$_j(P)$ such that $\kappa_j(P_j) = \kappa_j(\widetilde{Q}_j)$ (i.e., P and Q collide in level Δ_j), then for any $i < j$, there exists $P_i \in$ DELSNAP$_i(P)$ such that $\kappa_i(P_i) = \kappa_i(\widetilde{Q}_i)$.*

Proof. It suffices to prove that if P and Q collide in Δ_j then P and Q collide in Δ_{j-1}. From Theorem 15 Part 1, since $P_j \in$ DELSNAP$_j(P)$ and $\kappa_j(P_j) = \kappa_j(\widetilde{Q}_j)$, a matching $\mathcal{M}$ exists between P and Q such that $\forall (p,q) \in \mathcal{M}$, $d_\infty(p,q) \leq \frac{3}{2}\delta_j$. We use this matching to find $P_{j-1} \in$ DELSNAP$_{j-1}(P)$ such that $\kappa_{j-1}(P_{j-1}) = \kappa_{j-1}(\widetilde{Q}_{j-1})$, which happens when there exists a matching $\mathcal{M}' \subset P \times Q$ such that for each $(p', q') \in \mathcal{M}'$, either $d_\infty(p', \ell'_q) < \delta_{j-1}$, where ℓ'_q is the closest grid point to q', or $\max\{d(p', D), 2d(q', D)\} \leq \delta_{j-1}$. Then, we show that we can construct $\mathcal{M}'$. We begin with $\mathcal{M}' = \emptyset$, and for each $(p,q) \in \mathcal{M}$ we construct one, or more, edges and add them to $\mathcal{M}'$. To construct each edge, we consider three cases: CASE 1, where $q \in D$; CASE 2, where $p \in D$; and CASE 3, where neither p nor q are in D, which has two subcases.

CASE 1 *($q \in D$):* Since $(p,q) \in \mathcal{M}$ and $q \in D$, then p is within δ_j of D. Since $\delta_j = \frac{1}{2}\delta_{j-1}$, we know that p lies at most $\frac{1}{2}\delta_{j-1}$ from D and can be matched with D in a matching corresponding to a collision at Δ_{j-1} as well. Thus, we add (p,q) to $\mathcal{M}'$.

CASE 2 *($p \in D$)*: Since $(p,q) \in \mathcal{M}$ and $p \in D$, then q is within $\frac{1}{2}\delta_j$ of D. Then, q is at most $\frac{1}{4}\delta_{j-1}$ from D, and it can be matched with $p \in D$ in a matching corresponding to a collision at Δ_{j-1} as well. Thus, we add (p,q) to $\mathcal{M}'$.

CASE 3 *(neither p nor q are in D in Δ_j):* By construction, in level Δ_j, points p and q snap to the same grid point ℓ_q such that $d_\infty(q, \ell_q) \leq \frac{1}{2}\delta_j$ and $d_\infty(p, \ell_q) \leq \delta_j$, since q is snapped to the nearest grid point and p is snapped to one of the four nearest grid points.

SUBCASE 3*a* *($d_\infty(q, D) \leq \frac{1}{2}\delta_{j-1}$).* We add $(\pi_D(q), q)$ to $\mathcal{M}'$, since q must be snap-rounded to D in Δ_{j-1}. In order to match p in $\mathcal{M}'$, we show that p also has a snap-rounding

to D in level Δ_{j-1}. Since $d_\infty(q, D) \leq \frac{1}{2}\delta_{j-1}$, we know that either $d(\ell_q, D) = \delta_j = \frac{1}{2}\delta_{j-1}$ or $d(\ell_q, D) = \frac{1}{2}\delta_j = \frac{1}{4}\delta_{j-1}$. Since $d_\infty(p, \ell_q) \leq \delta_j = \frac{1}{2}\delta_{j-1}$, the triangle inequality implies that $d(p, D) \leq \delta_{j-1}$. Therefore, p can be deleted in the snap-rounding in Δ_{j-1}, so we add $(p, \pi_D(p))$ to $\mathcal{M}'$.

SUBCASE 3*b* *($d_\infty(q, D) > \frac{1}{2}\delta_{j-1}$).* Let ℓ'_q be the grid point in level Δ_{j-1} to which q snap-rounds. Since q is more than $\frac{1}{2}\delta_{j-1}$ from D, $d_\infty(\ell_q, \ell'_q) \leq \frac{1}{2}\delta_{j-1}$, which implies that $d_\infty(p, \ell'_q) \leq \delta_j + \frac{1}{2}\delta_{j-1} \leq \delta_{j-1}$ by the triangle inequality. Therefore, we can snap-round p to ℓ'_q in Δ_{j-1}. So, we add (p, q) to $\mathcal{M}'$.

Finally, we find $P_{j-1} \in \text{DELSNAP}_{j-1}(P)$ as follows: for every $(p, q) \in \mathcal{M}'$ such that p is off-diagonal, we either (1) delete p if q is on D, or (2) snap-round p to the grid point in Δ_{j-1} nearest to q. Therefore, we conclude that if P and Q collide at Δ_j, then P and Q also collide at Δ_{j-1}. □

To find a near neighbor to Q in Γ, we determine the last level such that Q collides with an existing key in the data structure. However, first we must consider where the nearest neighbor lies relative to this level.

Lemma 22 (Nearest Neighbor Bin)**.** *Let Γ, Q, and $\widetilde{Q}_i$ be as defined in Lemma 21. Let $\widetilde{Q}_{i-2}$ be obtained from Q by removing all points within $\frac{1}{2}\delta_{i-2}$ of the diagonal. Let $P^{nn} \in \Gamma$ be the nearest neighbor of Q in Γ, with respect to the bottleneck distance between diagrams. Let i be the largest index such that $\kappa_i(\widetilde{Q}_i)$ has a collision in Δ_i. Then, there exists a snap-rounding $P^{nn}_{i-2} \in \text{DELSNAP}_{i-2}(P^{nn})$ such that $\kappa_{i-2}(P^{nn}_{i-2}) = \kappa_{i-2}(\widetilde{Q}_{i-2})$.*

Proof. Assume that P^{nn} does not have a collision with $\kappa_{i-2}(\widetilde{Q}_{i-2})$. By the contrapositive of Theorem 15 Part 2, $2\delta_i = \frac{1}{2}\delta_{i-2} < d_B(P^{nn}, Q)$. As Q collides with P in Δ_i, by Theorem 15 Part 1, $d_B(P, Q) \leq \frac{3}{2}\delta_i$. Combining the inequalities, we get $d_B(P, Q) \leq \frac{3}{2}\delta_i < 2\delta_i < d_B(P^{nn}, Q)$, which implies that P^{nn} is not the nearest neighbor, a contradiction. □

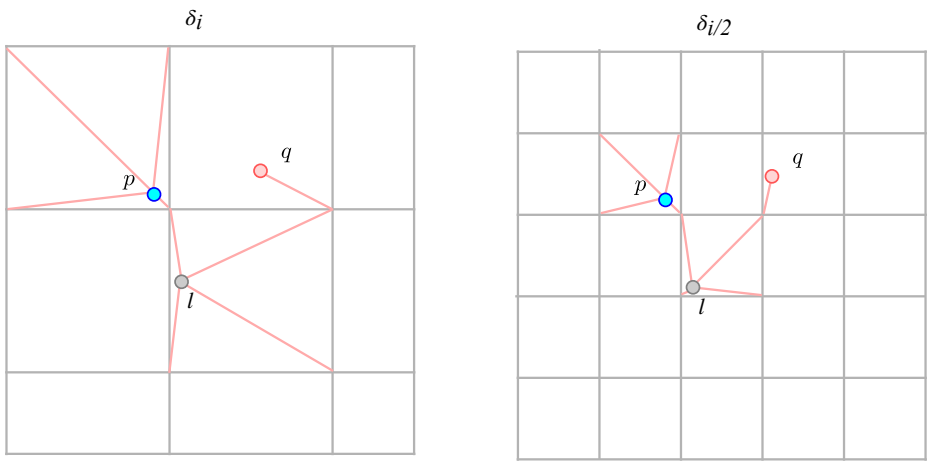


Figure 4.5: Situation in which the nearest neighbor of diagram Q is not in the lowest bin that Q has a collision in. Let query diagram Q be composed of the single point q and let $P, L \in \mathcal{D}_M^m$, where P has one point p and L has one point l. We see that $d_B(P, Q) = \frac{\delta_i}{2} + \epsilon_1$ for some small constant ϵ_1 and $d_B(L, Q) = \frac{3\delta_i}{4} - \epsilon_2$ for some small constant ϵ_2. However, only L and Q collide at level $\frac{\delta_i}{2}$, even though $d_B(P, Q) < d_B(L, Q)$.

A result of Lemma 22 is that if i is the largest index such that Q has a collision in Δ_i, then we can construct examples in which Q does not collide with the nearest neighbor in Δ_i, see diagrams in Figure 4.5. Next, we show that any diagram colliding with Q in Δ_i is an approximate nearest neighbor.

Lemma 23 (Nearest Neighbor Approximation)**.** *Let Γ, Q, and $\widetilde{Q}_i$ be as defined in Lemma 21. Let i be the largest index such that $\kappa_i(\widetilde{Q}_i) \in \Delta_i$. Let $P^{nn} \in \Gamma$ be the nearest neighbor of Q in terms of bottleneck distance. The bottleneck distance between Q and every diagram of Γ with a key $\kappa_i(\widetilde{Q}_i)$ in Δ_i is a six-approximation of $d_B(P^{nn}, Q)$.*

Proof. Let $P \in \Gamma$ and $P_i \in \textsc{DelSnap}_i(P)$ such that $\kappa_i(P_i) = \kappa_i(\widetilde{Q}_i)$. In other words, P_i is a snap-rounding of P that collides with Q at level i. By Lemma 22, P^{nn} has a snap-rounding in Δ_{i-2} colliding with Q. And, by our assumption, as Q has no collisions in level $i+1$, P^{nn} may have its last collision with Q in Δ_{i-2}, Δ_{i-1}, or Δ_i. To bound the bottleneck distance, we must only consider the worst-case scenario, where $d_B(P^{nn}, Q)$ is

as small as possible and $d_B(P, Q)$ is as large as possible. As P and Q collide in level i, by Theorem 15 Part 1, $d_B(P, Q) \leq \frac{3}{2}\delta_i$. And, by the contrapositive of Theorem 15 Part 2, if the last collision of P^{nn} and Q is: in Δ_{i-2}, then $\delta_i = \frac{1}{2}\delta_{i-1} < d_B(P^{nn}, Q)$; in Δ_{i-1}, then $\frac{1}{2}\delta_i < d_B(P^{nn}, Q)$; or in Δ_i, then $\frac{1}{4}\delta_i = \frac{1}{2}\delta_{i+1} < d_B(P^{nn}, Q)$. Therefore,

$$\frac{1}{2}\delta_{i+1} < d_B(P^{nn}, Q) \leq d_B(P, Q) \leq \frac{3}{2}\delta_i = 6\left(\frac{1}{2}\delta_{i+1}\right) < 6\big(d_B(P^{nn}, Q)\big),$$

which implies that every diagram P with a key in Δ_i colliding with $\kappa_i(\widetilde{Q})$ is a six-approximation of the nearest neighbor of Q in terms of bottleneck distance. □

The previous discussion implies that we can find an approximate nearest neighbor by identifying the bin in the lowest level with a collision and picking any diagram in that bin. Moreover, it tells us that we can find the true nearest neighbor by linearly searching for it though all diagrams with a collision two levels up. Next, we prove that we can query for an approximate kth nearest neighbor for $k > 1$. First, we establish bounds on the location of kth nearest neighbor, generalizing results from Lemma 22.

Lemma 24 (kth-NN Location Upper Bound)**.** *Let Γ, Q, and $\widetilde{Q}_i$ be as defined in Lemma 21. Let $P^k \in \Gamma$ be the kth nearest neighbor of Q in Γ, with respect to the bottleneck distance between diagrams. Let i be the largest index such that the number of distinct diagrams with snap-roundings and keys equal to $\kappa_i(\widetilde{Q}_i)$ in Δ_i is at least k. Then, there exists a snap-rounding $P^k_{i-2} \in \textsc{DelSnap}_{i-2}(P^k)$ such that $\kappa_{i-2}(P^k_{i-2}) = \kappa_{i-2}(\widetilde{Q}_{i-2})$.*

Proof. Assume, for contradiction, that P^k does not have a collision with Q in Δ_{i-2}. By the contrapositive of Theorem 15 Part 2, $2\delta_i = \frac{1}{2}\delta_{i-2} < d_B(P^k, Q)$. Furthermore, by Theorem 15 Part 1, if a diagram $P \in \Gamma$ collides with Q in Δ_i, then $d_B(P, Q) \leq \frac{3}{2}\delta_i$. Since there are at least k collisions with Q in Δ_i, there must be k diagrams with bottleneck distance less than or equal to $\frac{3}{2}\delta_i < 2\delta_i < d_B(P^k, Q)$. The previous statement, however,

is a contradiction to the claim that P^k is the kth nearest neighbor of Q with respect to bottleneck distance. $\square$

Then, we also bound the number of levels with a finer grid resolution that the kth-nearest neighbor can collide with a snap-rounding of the query diagram.

Lemma 25 (kth-NN Location Lower Bound)**.** *Let Γ, Q, and $\widetilde{Q}_i$ be as defined in Lemma 21. Let $P^k \in \Gamma$ be the kth nearest neighbor of Q in Γ, with respect to the bottleneck distance between diagrams. Let i be the largest index such that the number of distinct diagrams with snap-roundings and keys equal to $\kappa_i(\widetilde{Q}_i)$ in Δ_i is at least k. Then, P^k does not have a snap-rounding and key colliding with $\widetilde{Q}$ in any Δ_j such that $j > i + 2$.*

Proof. If P^k has a snap-rounding that collides with Q in any Δ_j for $j > i + 2$ then it will also have a snap-rounding colliding with Q in Δ_{i+3} by Lemma 21. Therefore, it suffices to show that P^k does not collide with Q in Δ_{i+3}. Then, suppose, by contradiction, that P^k has a snap-rounding that collides with Q in Δ_{i+3}. Then, by Theorem 15 Part 1, we know that $d_B(P^k, Q) \leq \frac{3}{2}\delta_{i+3} = \frac{3}{8}\delta_{i+1} < \frac{1}{2}\delta_{i+1}$, which implies that at least k diagrams in Γ have distance at most $\frac{1}{2}\delta_{i+1}$ from Q. Furthermore, by Theorem 15 Part 2, we know that all $P \in \Gamma$ such that $d_B(P, Q) \leq \frac{1}{2}\delta_{i+1}$ collide with Q in Δ_{i+1}. Hence, Q has at least k collisions in Δ_{i+1}, which contradicts our choice of i. $\square$

Using Lemma 24 and Lemma 25, we bound levels for which P^k can collide with $\widetilde{Q}$.

Corollary 9 (kth-NN Location)**.** *Let Γ, Q, and $\widetilde{Q}_i$ be as defined in Lemma 21. Let $P^k \in \Gamma$ be the kth nearest neighbor of Q, with respect to the bottleneck distance between diagrams. Let i be the largest index such that the number of distinct diagrams with snap-roundings and keys equal to $\kappa_i(\widetilde{Q}_i)$ at Δ_i is at least k. Let j be the largest level in $\mathbb{D}_\tau$ such that there is a snap rounding and key of P^k colliding with Q. Then, $i - 2 \leq j \leq i + 2$, i.e., P^k must have a snap-rounding in, at most, Δ_{i+2} and in, at least, Δ_{i-2}.*

To find the kth nearest neighbor to Q in Γ, we determine the last level of $\mathbb{D}_\tau$ where Q has at least k collisions. The proof is a modification of the proof of Lemma 23.

Lemma 26 (kth-Nearest Neighbor Approximation)**.** *Let $\Gamma, Q,$ and $\widetilde{Q}_i$ be as defined in Lemma 21. Let k be a positive integer greater than one. Let $P^k \in \Gamma$ be the kth nearest neighbor of Q, with respect to the bottleneck distance between diagrams. Let i be the largest index such that the number of distinct diagrams with snap-roundings and keys equal to $\kappa_i(\widetilde{Q}_i)$ at Δ_i is at least k. The bottleneck distance between Q and every diagram of Γ with a key $\kappa_i(\widetilde{Q}_i)$ in Δ_i is a 24-approximation of the $d_B(P^k, Q)$.*

Proof. Let $P \in \Gamma$ and $P_i \in \textsc{DelSnap}_i(P)$ such that $\kappa_i(P_i) = \kappa_i(\widetilde{Q}_i)$. That is, P is one of the k distinct diagrams that collide with Q at level i. By Corollary 9, P^k has a collision with Q in Δ_{i-2}, Δ_{i-1}, Δ_i, Δ_{i+1}, or Δ_{i+2}. To bound the bottleneck distance, we must only consider the worst-case scenario, in which $d_B(P^k, Q)$ is as small as possible and $d_B(P, Q)$ is as large as possible. As P and Q collide in level i, by Theorem 15 Part 1, $d_B(P, Q) \leq \frac{3}{2}\delta_i$. And, by the contrapositive of Theorem 15 Part 2, for $\alpha \in \{-2, -1, \ldots, 2\}$, if the last collision of P^k and Q is in $\Delta_{i+\alpha}$, then $2^{-2-\alpha}\delta_i < d_B(P^k, Q)$. The distance is smallest when $\alpha = 2$, that is, $\frac{1}{16}\delta_i < d_B(P^k, Q)$. Therefore,

$$d_B(P, Q) \leq \frac{3}{2}\delta_i = 24\left(\frac{1}{16}\delta_i\right) < 24\big(d_B(P^k, Q)\big),$$

which implies that every diagram P with a key in Δ_i colliding with $\kappa_i(\widetilde{Q})$ is a 24-approximation of the kth nearest neighbor of Q in terms of bottleneck distance. $\square$

Remark 6. *The approximation factor is controlled by the last level where P^k and Q collide. So, while we do not propose an efficient test for identifying the last level, we observe that in some cases, the approximation factor is much tighter. For example, if P^k last collides with Q in Δ_{i-2}, then for all $P \in \Gamma$ that collide with Q in Δ_i, $d_B(P, Q) \leq \frac{3}{2}d_B(P^k, Q)$.*

We now identify approximate nearest neighbors for a query diagram.

Theorem 16 (Approximate Nearest Neighbor Query)**.** *Let Γ, Q, and $\widetilde{Q}_i$ be as defined in Lemma 21. Let $n = |\Gamma|$ and let $\mathbb{D}_\tau$ be the multi-level structure described in Definition 24 with τ levels. Then, the data structure $\mathbb{D}_\tau$ is of size $O(n5^m\tau)$ and supports finding a six-approximation of the nearest neighbor of Q in Γ in $O((m \log n + m^2) \log \tau)$ time.*

Proof. We begin by describing how we can use the previous lemmas to organize and search $\mathbb{D}_\tau$. Lemma 21 tells us that for $i < j$, $\kappa_j(\widetilde{Q}_i) \in \Delta_j$ then $\kappa_i(\widetilde{Q}_i) \in \Delta_i$ which implies that $\mathbb{D}_\tau$ can be ordered by $i \in \{0, \ldots, \tau\}$. Using the ordering, we can perform a binary search to find the largest i such that $\kappa_i(\widetilde{Q}_i) \in \Delta_i$. Every diagram with a key in Δ_i colliding with $\kappa_i(\widetilde{Q}_i)$ is a six-approximation of the nearest neighbor of Q by Lemma 23.

We begin by analyzing the space of $\mathbb{D}_\tau$. The structure contains τ levels. For each level, we store, in increasing order, the $O(5^m)$ snap-roundings for each of the n persistence diagrams. So, the total space of $\mathbb{D}_\tau$ is $O(n5^m\tau)$.

Next, we consider the complexity of finding a six-approximation of the nearest neighbor for Q. Searching for the largest i such that $\kappa_i(\widetilde{Q}_i) \in \Delta_i$ requires a binary search through $\mathbb{D}_\tau$ and another binary search through each Δ_i that is encountered to search for a collision.

The time for each search in Δ_i is analyzed using three observations. First, generating a key for Q, i.e., $\kappa_i(\widetilde{Q}_i)$ takes $O(m)$ time. Second, comparing the keys of two diagrams takes $O(m)$ time. Third, let $n = |\Gamma|$, since each diagram has $O(5^m)$ hashes, each Δ_i has at most $O(n5^m)$ keys. Using these three observations, we get that searching Δ_i takes $O(m + m\log(n5^m))$. The search time can be simplified. Since, $m\log(n5^m) = m\log(n) + m^2\log(5)$, the search at each Δ_i is $O(m + m\log(n) + m^2) = O(m\log(n) + m^2)$ time. Since we search $O(\log(\tau))$ levels of $\mathbb{D}_\tau$, and the search at each level is $O(m\log(n) + m^2)$, the total query time is $O((m \log n + m^2) \log \tau)$. $\square$

Remark 7. *We note that exponential search could replace binary search for both finding the last Δ_i where $\kappa_i(\widetilde{Q}_i)$ collides with another key as well as on each $\Delta_i \in \mathbb{D}_\tau$. If i is the largest Δ_i such that the snap-rounding of $\widetilde{Q}_i$ collides with another key, and γ is the index of the key in Δ_i that collided with $\kappa_i(\widetilde{Q}_i)$ then the query time becomes $O(\log i(m \log(\gamma))$.*

Finally, we prove that this data structure can provide responses to queries requesting the k-nearest neighbors. Specifically, the k-nearest neighbors returned are a 24-approximation of the kth nearest neighbor.

Corollary 10 (k-Nearest Neighbor Query)**.** *Let Γ, Q, and $\widetilde{Q}_i$ be as defined in Lemma 21. Let $n = |\Gamma|$ and let $\mathbb{D}_\tau$ be the multi-level structure described in Definition 24 with τ levels. There exists a data structure of size $O(n5^m\tau)$ that supports finding k diagrams that are each, in the worst case, a 24-approximation of the kth nearest neighbor of Q from Γ in $O((m \log n + m^2 + km) \log \tau)$ time.*

Proof. We begin by searching to find the largest i such that: $\kappa_i(\widetilde{Q}_i) \in \Delta_i$, and the list of diagrams at key $\kappa_i(\widetilde{Q}_i)$ is of at least length k. We can still utilize binary search to traverse the levels of $\mathbb{D}_\tau$. At each level, however, once a matching key is found, we must determine if there are k unique neighboring keys with the same value. Recall that at each key, we stored the count of the number of unique diagrams. Thus, we can determine the number of unique diagrams hashed to a particular key in constant time. Then, searching each level Δ_i takes time $O(m \log(n) + m^2)$ from Theorem 16. Once the largest i with k colliding diagrams is found, we return any k diagrams from the list at $\kappa_i(\widetilde{Q}_i)$. Any of these colliding diagrams will be a 24-approximation of the kth nearest neighbor of Q by Lemma 26. Finding k diagrams from the list at $\kappa_i(\widetilde{Q}_i)$ takes time $O(k)$ time, so the total time complexity for searching and returning k diagrams at a particular level is $O(m \log(n) + m^2 + k)$, making the overall time complexity for searching $O((m \log n + m^2 + km) \log \tau)$. No modifications need to be made to $\mathbb{D}_\tau$ to support these

queries so the space complexity remains the same from Theorem 16. □

4.3.1 Contributions to Section 4.3

In Section 4.3, we develop the data structure and searching methods used to return approximate nearest neighbors and the k approximate nearest neighbors to a query persistence diagram. Since the formulation in Fall 2017, k-nearest neighbors results were added over the year of 2018. Results continued to be refined until as recently as Fall 2019. We summarize the specific contributions of Samuel Micka in the following list.

- Samuel Micka, David L. Millman, Brittany Terese Fasy, and Binhai Zhu developed the data structure in Definition 24 with input and feedback from Xiaozhou He, and Zhihui Liu.
- Lemma 21 and Lemma 22 have seen several revisions. The initial proofs were written by Samuel Micka, Brittany Terese Fasy, and David L. Millman and many revisions have been made over the years by Samuel Micka.
- The example demonstrating that the nearest neighbor does not necessarily lie in the last bin with a collision was an observation made by Samuel Micka (see Figure 4.5).
- Lemma 23 has been revised several times by Samuel Micka, David L. Millman, and Brittany Terese Fasy with some errors being corrected over the years. We thank Brendan Mumey for his discussions related to this lemma in particular. Observations made by Brendan Mumey in [86] were particularly helpful in uncovering an error in our own work.
- The lemmas and theorems related to the k-nearest neighbors were all written initially by Samuel Micka. Further discussions between Samuel Micka, David L. Millman, Brittany Terese Fasy, and Binhai Zhu led to additional revisions being made by Samuel Micka.

- Theorem 16 and Corollary 10 were written and revised over time by Samuel Micka, David L. Millman, Brittany Terese Fasy, and Binhai Zhu with additional input and feedback from Xiaozhou He, and Zhihui Liu.

4.4 Discussion on Space Complexity

While searching $\mathbb{D}_\tau$ is logarithmic in the number of diagrams, the data structure becomes very large when the diagrams have even a moderate number of points. For example, with $m = 15$, we may have over 2^{34} keys at a given level. In this section, we discuss approaches intended to mitigate the size of the data structure and explain why many become unrealistic in practice.

One way to reduce the size of each level is to "flip" the key generation and querying. In particular, instead of generating many keys for each diagram in Γ, we compute one for each diagram. Then, for a query diagram Q, we compute and search $O(5^m)$ keys at each level. This may be practical for scenarios in which diagrams in Γ are large, but the query diagrams are small. Moreover, since searching for a key is independent of the other keys, searching in parallel follows naturally. Furthermore, this approach reduces the size of $\mathbb{D}_\tau$ from $O(n5^m\tau)$ to $O(nm\tau)$. However, this trades exponential space for exponential search time relative to the number of points in the diagram. Preliminary work on more complex snapping schemes have shown that we can reduce the size of the data structure. While the size is still exponential in m, some of the more promising schemes have been able to reduce the expected size to $O(2.6^m)$ and increase the approximation factor to a larger constant value. The next natural approach to pursue is probabilistic snap-rounding.

Consider generating a single key for each diagram in Γ by snap-rounding each diagram a single time at each grid resolution. If we randomly snap-round our query diagram a polynomial number of times, we may never collide with a diagram with small bottleneck distance since only one of the $O(5^m)$ snap-roundings may yield a match. Another approach

is defining a match to be when a certain ratio of points collide. However, we can run into situations where we return diagrams with exceptionally large bottleneck distance. For example, consider a class of diagrams with k off-diagonal points where for any two diagrams α and β in this class, $d_B(\alpha, \beta)$ is large, but if a subset of $k-1$ points from each are considered, the bottleneck distance is small. Then, if we query with a diagram Q in which $k-1$ points from Q have a small bottleneck distance to $k-1$ points of any diagram in the class we will return a match for every diagram. However, the actual bottleneck distance between the query diagram and any diagram in this class can be arbitrarily large.

Finally, while our space complexity is exponential in diagram size, our queries do not rely on probabilistic snap-roundings and the problems discussed above resulting from these approaches. We conclude this discussion by offering an analysis of the complexity of our data structure, in practice, relative to similar approaches used on polygonal curves from [46] and demonstrate a substantial decrease in size. While we are comparing approaches for different problem domains, our goal is to draw attention to the complexity related to the number of points in the polygonal curves and diagrams to emphasize the challenges related to reducing data structure size. Specifically, for a data structure, storing n curves in $\mathbb{R}^2$, each with complexity at most $m = 15$, the constant factor approximation from [46] requires $O(2^{120} n \log n + 15n)$ space; whereas, our approach to storing n diagrams with at most 15 off-diagonal points requires $O(n5^{15}\tau)$ space. However, we note that our approach depends on the distribution of points in the diagrams which dictates the size of τ while the approach found in [46] is independent of the distribution of curves.

4.4.1 Contributions to Section 4.4

In Section 4.4 we discuss the exponential space complexity of our data structure and compare the methodology to similar approaches in the literature. This discussion was motivated by anonymous reviewer comments which were followed up by discussions

between Binhai Zhu and Samuel Micka. Samuel Micka wrote the content of this section and received feedback from Binhai Zhu, Brittany Terese Fasy, and David L. Millman.

4.5 Discussion

In this chapter, we have investigated the problem of near-neighbor search in the space of PDs with our work from [51]. Specifically, we worked towards addressing Research Question 3 and developed a solution to querying for approximate nearest neighbors and the k-approximate nearest neighbors in the space of persistence diagrams. Our work was motivated by locality-sensitive hashing approaches [4–7, 33, 61, 63, 67] and searching in the space of polygonal curves [46, 66]. We provided performance guarantees on the diagrams returned, complementing experimental work on transforming persistence diagrams into polynomials with complex coefficients for the purposes of searching [44]. Furthermore, we provide a proof that the doubling dimension of (M, m)-bounded persistence diagrams under the bottleneck distance is infinite, offering concrete evidence of observations from [69, 70]. While the space complexity of our approach is exponential, we offer an extensive evaluation of the approach, comparing it to similar methods to put the scope into perspective.

CHAPTER FIVE

SUMMARY AND FUTURE WORK

In this dissertation, we have described significant contributions to the problem spaces of reconstructing shapes using topological descriptors and searching in the space of persistence diagrams. We described algorithms for reconstructing embedded graphs (see Section 3.3, Section 3.4, and [11, 12, 56]) and reconstructing simplicial complexes in $\mathbb{R}^d$ (see Section 3.2 and Section 3.5 and [56]). These algorithms offer novel approaches for determining parameterized sets of augmented persistence diagrams for shape representation. Moreover, these algorithms are the first steps towards developing deterministic approaches for minimizing the number of descriptors we need to represent shapes. We also discussed challenges encountered when utilizing non-augmented topological descriptors and implementing random sampling techniques for choosing directions to generate descriptors from (see Sections 3.6-3.9 and [54, 55, 80]). These results justify the use of augmented descriptors or exhaustive sampling approaches when using non-augmented descriptors for reconstruction and reveal several additional research problems.

In the second part of the dissertation, we explored efficient methods for identifying approximate nearest-neighbors in the space of persistence diagrams (see Chapter 4 and [51]). We improve the runtime for determining approximate nearest neighbors in large sets of diagrams by employing a snap-rounding approach that transforms each diagram into a set of vectors, each approximating the original diagram. Then, we compare vectors, instead of the diagrams, to search. Our approach provides the first approximation bounds, in terms of the bottleneck distance to the nearest neighbor, on searching for approximate nearest persistence diagrams. Moreover, we prove approximation bounds for diagrams we return for both the nearest, and k^{th} nearest, neighbors, providing a new approach searching for

neighbors in large databases of persistence diagrams.

5.1 Future Work

In the future, we would like to explore several avenues of potential work. First, while we may have the ability to reconstruct simplicial complexes in $\mathbb{R}^d$, the diagram and time complexities are exponential relative to the highest-dimensional simplex in the complex. When reconstructing $\mathbb{K}_1$ in Section 3.4, we were able to make our approach for reconstruction sub-quadratic and output sensitive. The speed-up is a result of utilizing information about edges that were already constructed to reduce the search time for new edges in the complex. As such, we will explore the possibility generalizing these ideas to higher dimensions and reducing the time and diagram complexity described in Section 3.5.

Recent work by Maria et al. defines an intrinsic transform on shapes [79]. This transform encodes information about the *intrinsic* geometry, allowing it to effectively ignore the embedding of the shape. The transform is particularly applicable in the domain of shape comparison by overcoming challenges associated with shape alignment. Furthermore, the authors provide stability results for the transform. An avenue for future research may be developing algorithms for computing sets of descriptors for this transform.

Code for reconstructing plane graphs in $\mathbb{R}^2$ using the approaches from our previous work [11] has been made publicly available [1]. However, this approach does not include the implementation of the general vertex reconstruction algorithms described in Section 3.3, the faster algorithm for edge reconstruction in Section 3.4, or the general algorithm for simplicial complex reconstruction described in Section 3.5. Implementing these algorithms would provide researchers using topological descriptors as input into other algorithms (such as [39, 64, 100]) a method for generating sets of APDs that reconstruct the shape.

[1]The code is available in a git repo hosted on GitHub: `https://github.com/compTAG/reconstruction`.

In our experiments, our real-world data came from the MPEG7 and EMNIST data sets. However, both data sets are composed of images. When converting from image (cubical) data to simplicial complexes, we used perturbations to meet our general position assumptions. However, preliminary investigations lead us to believe that these small perturbations may be contributing to results in stratum size and, as a result, uniform sampling. As such, different perturbation techniques warrant further investigation. More generally, investigating problems that arise when converting image data to simplicial complexes, such as choosing contour approximations, for use with these transforms is also interesting. We are also interested in running additional experiments on data sets stored as simplicial complexes since the cubical structure of image data may be lurking variable.

One important distinction that we touched on Section 3.10 was between representation and reconstruction. In this dissertation, specifically, in Section 3.5, we provide an algorithm for reconstructing simplicial complexes using augmented persistence diagrams. However, if the on-diagonal points are ignored, then we are left with a set of PDs parameterize by directions on the unit sphere. Thus, we ask, does the set of PDs, inferred from APDs generated during reconstruction, sufficiently represent the simplicial complex we reconstruct? To address this question, we introduce the following conjecture:

Conjecture 1. *Let $\mathbb{K}, \mathbb{K}' \subset \mathbb{R}^d$ be simplicial complexes. Let P, P' be the parameterized sets of PDs generated when reconstructing $\mathbb{K}$ and $\mathbb{K}'$ with Algorithm 3.11. Then, $P = P'$ if and only if $\mathbb{K} = \mathbb{K}'$.*

As future work, we plan to prove Conjecture 1 as a theorem. This result would be beneficial for problems related to shape comparison because we would offer an approach that generates persistence diagrams that can be used for reconstruction and also for representation. While we can use APDs for representation now, PDs are widely used and distances for comparing PDs are well studied.

In Section 3.7, we saw that reconstruction of particular classes of simplicial complexes was possible using a constant number of APDs. The reconstruction was done using a *constraints based approach* where certain directions yielded information that, when combined with previously known constraints on the shape, allowed us to reconstruct the complex. We are interested in exploring the idea of constraint based approaches in practice. Specifically, we would like to generate constraints based on a set of APDs and use these constraints in a constraint solving system such as `python-constraint`[2] or `Gecode`[3] to see if the constraints can be resolved.

In Section 3.9, we saw two constructions that provide lower-bounds on the cost of failing to reconstruct degree two vertices. However, we are not convinced that this lower-bound is tight, in fact, we conjecture that there are classes of simplices which can produce arbitrarily large distances between the original graph and a reconstructed graph missing specific simplices. This problem is accessible for undergraduates with little to no prior experience in TDA and, as such, will be investigated soon.

Chapter 4 provides the first solution to searching for approximate nearest neighbors in the space of PDs with performance guarantees. However, the solution has not yet been implemented. This framework will be useful for projects that generate large sets of PDs and need to be able to search for nearest neighbors quickly. The implementation of this data structure and searching algorithm are both goals for the near future. Furthermore, we are considering extending the approach of [101] to a two-dimensional "grid" (a complex vector quadratic in size) to see if our lower- and upper-bounds generalize into the space of complex vectors. This extension could yield a persistence diagram representation that has an inner-product space and enables approximate near-neighbor queries.

[2] `https://labix.org/python-constraint`
[3] `https://www.gecode.org/`

REFERENCES CITED

[1] H. Adams, A. Tausz, and M. Vejdemo-Johansson. Javaplex: A research software package for persistent (co) homology. In *International Congress on Mathematical Software*. Springer, 2014.

[2] M. Ahmed, B. T. Fasy, and C. Wenk. Local persistent homology based distance between maps. In *International Conference on Advances in Geographic Information Systems*, 2014.

[3] H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of Computational Geometry*, pages 121–153. Elsevier, 2000.

[4] E. Anagnostopoulos, I. Z. Emiris, and V. Fisikopoulos. Algorithms for deciding membership in polytopes of general dimension. *ArXiv preprint arXiv:1804.11295*, 2018.

[5] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal LSH for angular distance. In *Advances in Neural Information Processing Systems*, 2015.

[6] A. Andoni and I. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Symposium on Theory of Computing*. ACM, 2015.

[7] A. Andoni and I. Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. *ArXiv preprint arXiv:1507.04299*, 2015.

[8] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[9] D. Attali, A. Lieutier, and D. Salinas. Efficient data structure for representing and simplifying simplicial complexes in high dimensions. *International Journal of Computational Geometry & Applications*, 22(04):279–303, 2012.

[10] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. Phat–persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78:76–90, 2017.

[11] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Learning simplicial complexes from persistence diagrams. In *Canadian Conference on Computational Geometry*, 2018.

[12] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Reconstructing embedded graphs

from persistence diagrams. *Computational Geometry: Theory and Applications*, To appear. https://doi.org/10.1016/j.comgeo.2020.101658.

[13] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd edition, 2008.

[14] L. M. Betthauser. *Topological Reconstruction of Grayscale Images*. PhD thesis, University of Florida, 2018.

[15] S. Biasotti, L. De Floriani, B. Falcidieno, P. Frosini, D. Giorgi, C. Landi, L. Papaleo, and M. Spagnuolo. Describing shapes by geometrical-topological properties of real functions. *ACM Computing Surveys*, 40(4):1–87, 2008.

[16] O. Bobrowski, S. Mukherjee, and J. E. Taylor. Topological consistency via kernel estimation. *Bernoulli*, 23(1):288–328, 2017.

[17] K. A. Bollen and R. W. Jackman. Regression diagnostics: An expository treatment of outliers and influential cases. In J. Fox and J. S. Long, editors, *Modern Methods of Data Analysis*. SAGE Publications, 1990.

[18] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008.

[19] P. Bubenik. Statistical topological data analysis using persistence landscapes. *The Journal of Machine Learning Research*, 16(1):77–102, 2015.

[20] M. Buchet, F. Chazal, S. Y. Oudot, and D. R. Sheehy. Efficient and robust persistent homology for measures. *Computational Geometry*, 58:70–96, 2016.

[21] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Symposium on Discrete Algorithms*, 1994.

[22] A. Cardone, S. K. Gupta, and M. Karnik. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *Journal of Computing and Information Science in Engineering*, 3(2):109–118, 2003.

[23] G. Carlsson, G. Singh, and A. Zomorodian. Computing multidimensional persistence. In *International Symposium on Algorithms and Computation*. Springer, 2009.

[24] G. Carlsson and A. Zomorodian. The theory of multidimensional persistence. *Discrete & Computational Geometry*, 42(1):71–93, 2009.

[25] G. Carlsson, A. Zomorodian, A. Collins, and L. J. Guibas. Persistence barcodes for shapes. *International Journal of Shape Modeling*, 11(02):149–187, 2005.

[26] M. Carrière, S. Y. Oudot, and M. Ovsjanikov. Stable topological signatures for points on 3D shapes. 34(5):1–12, 2015.

[27] A. Cerri, B. D. Fabio, M. Ferri, P. Frosini, and C. Landi. Betti numbers in multidimensional persistent homology are stable functions. *Mathematical Methods in the Applied Sciences*, 36(12):1543–1557, 2013.

[28] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20(3):359–373, 1998.

[29] F. Chazal, W. Crawley-Boevey, and V. De Silva. The observable structure of persistence modules. *ArXiv preprint arXiv:1405.5644*, 2014.

[30] F. Chazal, V. De Silva, M. Glisse, and S. Oudot. The structure and stability of persistence modules. *ArXiv preprint arXiv:1207.3674*, 21, 2012.

[31] F. Chazal, V. De Silva, M. Glisse, and S. Oudot. *The Structure and Stability of Persistence Modules*. Springer, 2016.

[32] F. Chazal, B. T. Fasy, F. Lecci, A. Rinaldo, A. Singh, and L. A. Wasserman. On the bootstrap for persistence diagrams and landscapes. *Modeling and Analysis of Information Systems*, 20(6):95–105, 2013.

[33] T. Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2017.

[34] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Symposium on Computational Geometry*. ACM, 1994.

[35] K. L. Clarkson. Nearest-neighbor searching and metric space dimensions. In G. Shakhnarovich, T. Darrell, and P. Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, page 1559. MIT Press, 2006.

[36] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: an extension of MNIST to handwritten letters. *ArXiv preprint arXiv:1702.05373*, 2017.

[37] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[38] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Extending persistence using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, 2009.

[39] L. Crawford, A. Monod, A. X. Chen, S. Mukherjee, and R. Rabadán. Predicting clinical outcomes in glioblastoma: an application of topological and functional data analysis. *Journal of the American Statistical Association*, pages 1–12, 2019.

[40] J. Curry, S. Mukherjee, and K. Turner. How many directions determine a shape and other sufficiency results for two topological transforms. *ArXiv preprint arXiv:1805.09782*, 2018.

[41] L. De Floriani, D. Greenfieldboyce, and A. Hui. A data structure for non-manifold simplicial d-complexes. In *Symposium on Geometry processing*, 2004.

[42] L. De Floriani and A. Hui. Data structures for simplicial complexes: An analysis and a comparison. In *Symposium on Geometry Processing*, 2005.

[43] O. Delgado-Friedrichs, V. Robins, and A. Sheppard. Morse theory and persistent homology for topological analysis of 3D images of complex materials. In *International Conference on Image Processing*. IEEE, 2014.

[44] B. Di Fabio and M. Ferri. Comparing persistence diagrams through complex vectors. In *International Conference on Image Analysis and Processing*. Springer, 2015.

[45] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[46] A. Driemel and F. Silvestri. Locality-sensitive hashing of curves. In *Symposium on Computational Geometry*, 2017.

[47] H. Edelsbrunner. *Algorithms in Combinatorial Geometry: Monographs on Theoretical Computer Science (an EATCS Series, Volume 10)*. Springer-Verlag, Heidelberg, West Germany, 1987.

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

[49] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

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

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

[52] B. T. Fasy, J. Kim, F. Lecci, and C. Maria. Introduction to the R package TDA. *ArXiv preprint arXiv:1411.1830*, 2014.

[53] B. T. Fasy, F. Lecci, A. Rinaldo, L. Wasserman, S. Balakrishnan, and A. Singh. Confidence sets for persistence diagrams. *The Annals of Statistics*, 42(6):23012339, Dec 2014.

[54] B. T. Fasy, S. Micka, and D. L. Millman. Too many or too few? sampling bounds for topological descriptors. In progress, 2020.

[55] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. Challenges in reconstructing shapes from Euler characteristic curves. *Fall Workshop on Computational Geometry*, 2018.

[56] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. Persistence diagrams for efficient simplicial complex reconstruction. *ArXiv preprint arXiv:1912.12759*, 2019.

[57] M. Ferri and C. Landi. Representing size functions by complex polynomials. *Proc. Math. Met. in Pattern Recognition*, 9:16–19, 1999.

[58] R. Ghrist, R. Levanger, and H. Mai. Persistent homology and Euler integral transforms. *Journal of Applied and Computational Topology*, 2(1-2):55–60, 2018.

[59] C. Giusti, E. Pastalkova, C. Curto, and V. Itskov. Clique topology reveals intrinsic geometric structure in neural correlations. *Proceedings of the National Academy of Sciences*, 112(44):13455–13460, 2015.

[60] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab, Los Alamos, NM (United States), 2008.

[61] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[62] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[63] P. J. Heffernan and S. Schirra. Approximate decision algorithms for point set congruence. *Computational Geometry*, 4(3):137–156, 1994.

[64] C. Hofer, R. Kwitt, M. Niethammer, Y. Höller, E. Trinka, A. Uhl, et al. Constructing shape spaces from a topological perspective. In *International Conference on Information Processing in Medical Imaging*. Springer, 2017.

[65] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[66] P. Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Symposium on Computational Geometry*. ACM, 2002.

[67] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Symposium Theory of Computing*. ACM, 1998.

[68] M. Kerber, D. Morozov, and A. Nigmetov. Geometry helps to compare persistence diagrams. *Journal of Experimental Algorithmics*, 22:1–4, 2017.

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

[70] M. Kerber and A. Nigmetov. Metric spaces with expensive distances. *ArXiv preprint arXiv:1901.08805*, 2019.

[71] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quarterly*, 2(1):83–97, March 1955.

[72] G. Kusano, K. Fukumizu, and Y. Hiraoka. Kernel method for persistence diagrams via kernel embedding and weight factor. *ArXiv preprint arXiv:1706.03472*, 2017.

[73] R. Kwitt, S. Huber, M. Niethammer, W. Lin, and U. Bauer. Statistical topological data analysis-a kernel perspective. In *Advances in Neural Information Processing Systems*, 2015.

[74] P. Lawson, J. Schupbach, B. T. Fasy, and J. W. Sheppard. Persistent homology for the automatic classification of prostate cancer aggressiveness in histopathology images. In *Medical Imaging 2019: Digital Pathology*, volume 10956, page 109560G. International Society for Optics and Photonics, 2019.

[75] M. Lesnick. The theory of the interleaving distance on multidimensional persistence modules. *Foundations of Computational Mathematics*, 15(3):613–650, 2015.

[76] C. Li, M. Ovsjanikov, and F. Chazal. Persistence-based structural recognition. In *Conference on Computer Vision and Pattern Recognition*, 2014.

[77] T. Liu, A. W. Moore, K. Yang, and A. G. Gray. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems*, 2005.

[78] L. Lovász and M. D. Plummer. *Matching Theory*, volume 367. American Mathematical Society, 2009.

[79] C. Maria, S. Oudot, and E. Solomon. Intrinsic topological transforms via the distance kernel embedding. *ArXiv preprint arXiv:1912.02225*, 2019.

[80] S. Micka and D. L. Millman. First steps towards lower-bounding the number of topological descriptors for reconstruction, 2019. CG Week Young Researcher's Forum.

[81] Y. Mileyko, S. Mukherjee, and J. Harer. Probability measures on the space of persistence diagrams. *Inverse Problems*, 27(12):124007, 2011.

[82] D. L. Millman and V. Verma. A slow algorithm for computing the Gabriel graph with double precision. *Canadian Conference on Computational Geometry*, 2011.

[83] N. Milosavljević, D. Morozov, and P. Skraba. Zigzag persistent homology in matrix multiplication time. In *Symposium on Computational Geometry*, 2011.

[84] M. Minsky and S. Papert. Perceptrons. 1969.

[85] K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete & Computational Geometry*, 50(2):330–353, 2013.

[86] B. Mumey. Indexing point sets for approximate bottleneck distance queries. *ArXiv preprint arXiv:1810.09482*, 2018.

[87] J. R. Munkres. *Topology*. Prentice Hall, 2000.

[88] M. Nicolau, A. J. Levine, and G. Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, 108(17):7265–7270, 2011.

[89] T. E. Oliphant. *A Guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[90] S. Oudot and E. Solomon. Inverse problems in topological persistence. *ArXiv preprint arXiv:1810.10813*, 2018.

[91] A. Poulenard, P. Skraba, and M. Ovsjanikov. Topological function optimization for continuous shape matching. 37(5):13–25, 2018.

[92] R. Ralph. Mpeg-7 core experiment ce-shape-1 test set, accessed 2019. http://www.dabi.temple.edu/ shape/MPEG7/dataset.html.

[93] A. H. Rizvi, P. G. Camara, E. K. Kandror, T. J. Roberts, I. Schieren, T. Maniatis, and R. Rabadan. Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. *Nature Biotechnology*, 35(6):551, 2017.

[94] P. Schapira. Tomography of constructible functions. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*. Springer, 1995.

[95] M. Scolamiero, W. Chachólski, A. Lundman, R. Ramanujam, and S. Öberg. Multidimensional persistence and noise. *Foundations of Computational Mathematics*, 17(6):1367–1406, 2017.

[96] J. Shewchuk, T. K. Dey, and S.-W. Cheng. *Delaunay Mesh Generation*. Chapman and Hall/CRC, 2016.

[97] J. W. H. Tangelder and R. C. Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia Tools and Applications*, 39(3):441, Dec 2007.

[98] L. N. Trefethen and D. Bau III. *Numerical Linear Algebra*, volume 50. Society for Industrial and Applied Mathematics, 1997.

[99] K. Turner, Y. Mileyko, S. Mukherjee, and J. Harer. Fréchet means for distributions of persistence diagrams. *Discrete & Computational Geometry*, 52(1):44–70, 2014.

[100] K. Turner, S. Mukherjee, and D. M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 2014.

[101] Z. Wang, Q. Li, G. Li, and G. Xu. Polynomial representation for persistence diagram. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[102] C.-K. Yap. Symbolic treatment of geometric degeneracies. *Journal of Symbolic Computation*, 10(3-4):349–370, 1990.

[103] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.