Applications of Persistent Homology to Time Varying Systems

by

Elizabeth Munch

Department of Mathematics
Duke University

Date: ______________________
Approved:

__________________________
John Harer, Supervisor

__________________________
Ezra Miller

__________________________
Kamesh Munagala

__________________________
William Pardon

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Mathematics in the Graduate School of Duke University 2013
Abstract
Applications of Persistent Homology to Time Varying Systems
by
Elizabeth Munch
Department of Mathematics
Duke University

Date: ____________________
Approved:

________________________
John Harer, Supervisor

________________________
Ezra Miller

________________________
Kamesh Munagala

________________________
William Pardon

An abstract of a dissertation submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in the Department of Mathematics
in the Graduate School of Duke University
2013
Abstract

This dissertation extends the theory of persistent homology to time varying systems. Most of the previous work has been dedicated to using this powerful tool in topological data analysis to study static point clouds. In particular, given a point cloud, we can construct its persistence diagram. Since the diagram varies continuously as the point cloud varies continuously, we study the space of time varying persistence diagrams, called vineyards when they were introduced by Cohen-Steiner, Edelsbrunner, and Morozov.

We will first show that with a good choice of metric, these vineyards are stable for small perturbations of their associated point clouds. We will also define a new mean for a set of persistence diagrams based on the work of Mileyko et al. which, unlike the previously defined mean, is continuous for geodesic vineyards.

Next, we study the sensor network problem posed by Ghrist and de Silva, and their application of persistent homology to understand when a set of sensors covers a given region. Giving each of these sensors a probability of failure over time, we show that an exact computation of the probability of failure of the whole system is NP-hard, but give an algorithm which can predict failure in the case of a monitored system.

Finally, we apply these methods to an automated system which can cluster agents moving in aerial images by their behaviors. We build a data structure for storing and querying the information in real-time, and define behavior vectors which quantify
behaviors of interest. This clustering by behavior can be used to find groups of interest, for which we can also quantify behaviors in order to determine whether the group is working together to achieve a common goal, and we speculate that this work can be extended to improving tracking algorithms as well as behavioral predictors.
To my parents, for their unwavering faith in me.
# Contents

Abstract iv  
List of Figures x  
Acknowledgements xii  

1 Introduction 1  
   1.1 Persistent Homology .............................................. 2  
   1.2 Time-varying systems, Vineyards, and Statistics .............. 4  
   1.3 Outline ........................................................... 6  

2 Basics and Background 7  
   2.1 Persistent Homology .............................................. 7  
      2.1.1 General Case ................................................... 7  
      2.1.2 Persistence Diagrams ........................................ 9  
   2.2 Point clouds and Simplicial Complexes ....................... 10  
      2.2.1 Simplicial Persistent Homology ......................... 10  
      2.2.2 Point clouds and simplicial complexes .................. 11  
   2.3 Algorithms for Computing Persistence ....................... 14  
      2.3.1 The Persistence Algorithm .................................... 14  
      2.3.2 UnionFind and 0-Dimensional Homology .................... 15  

3 Time Varying Point Clouds and Vineyards 18  
   3.1 Standard Metric Definitions ..................................... 19
3.2 Stability .................................................. 21
  3.2.1 Bottleneck Stability for Persistence Diagrams ............... 21
  3.2.2 Bottleneck Stability for Vineyards .......................... 24
3.3 Probability on Persistence Diagrams ............................ 29
  3.3.1 The Space \((D_p, W_p(L_\infty))\) .......................... 29
  3.3.2 Fréchet Means ......................................... 31
  3.3.3 Algorithm for Persistence Diagram Mean ...................... 33
3.4 Probability on Vineyards ..................................... 39
  3.4.1 Issues with extensions to Vineyards ......................... 40
  3.4.2 The Mean as a Distribution ................................ 41
3.5 Conclusions and Future Work .................................. 60

4 Sensor Networks ........................................ 62
  4.1 The Coverage Criterion ...................................... 63
    4.1.1 The Standard Formulation of the dS-G Criterion ......... 63
    4.1.2 Alternative Formulations of the dS-G Criterion ......... 65
  4.2 Sensor Failure ............................................ 66
    4.2.1 Relative Homology Version ............................... 66
    4.2.2 Absolute Homology Version .............................. 67
    4.2.3 Cake or Death ........................................ 68
  4.3 Complexity Issues .......................................... 69
    4.3.1 Use of the 2-skeleton .................................. 69
    4.3.2 \#P-Hard .................................................. 70
  4.4 A Deterministic Algorithm ................................... 74
    4.4.1 The Hasse Diagram ...................................... 75
    4.4.2 Probability of Failure .................................. 76
4.4.3 Death Sets Algorithm .................................... 77
4.4.4 Complexity of Algorithm ................................. 81
4.5 A Dynamic Algorithm for a Monitored System .......... 83
  4.5.1 The new criterion and complexity ................... 86
4.6 Conclusions and Possible Extensions .................... 88

5 Behavioral Clustering ........................................... 90
  5.1 Dynamic Data Structure ................................... 92
    5.1.1 Mesh Storage .......................................... 92
    5.1.2 Track Storage .......................................... 98
  5.2 Single Track Behavior Vectors ............................ 98
    5.2.1 Stopping Time Vector .................................. 98
    5.2.2 Acceleration Vector ................................... 99
    5.2.3 Looping Vector ........................................ 99
    5.2.4 Dendrograms .......................................... 102
  5.3 Group Behaviors ............................................ 104
    5.3.1 Group Creation ........................................ 104
    5.3.2 Analyzing Group Behavior ............................ 105
  5.4 Future Work ................................................. 106

Bibliography ....................................................... 108

Biography .......................................................... 112

ix
# List of Figures

1.1 Noisy circular point cloud example ..................................... 2
1.2 Examples of sensor networks ............................................. 5
2.1 A visual representation of births and deaths of homology classes. . . 8
2.2 Rips and Čech complexes are not equivalent .......................... 13
3.1 The Hausdorff distance between two topological spaces and the Wasserstein distance between two persistence diagrams. ............... 20
3.2 A vineyard ........................................................................ 25
3.3 Computation of the Wasserstein Distance .............................. 37
3.4 An example of a matching for three persistence diagrams and the corresponding mean diagram. .............................................. 38
3.5 A counterexample to uniqueness of the Fréchet mean in $D_p$. .. 41
3.6 Two vineyards whose pointwise mean is not continuous. .......... 42
3.7 The method for drawing points. .......................................... 44
3.8 An example of corresponding matchings for a given draw. ......... 46
3.9 The pictures for the cases of Lemma 31 ............................... 50
4.1 Small $(\mathcal{R}, \mathcal{F})$ sensor network example ........................ 67
4.2 Reducing 1D Network Reliability to the sensor network problem. 73
4.3 Hasse diagram. ............................................................... 75
4.4 The matrix column representing the birth of a fundamental class. . 81
4.5 Counter-example to show that the assumption $H_2(\mathcal{R}) = 0$ is necessary for Theorem 44. ................................................. 85
5.1 A quadtree subdivision and its corresponding quadtree. . . . . . . . . 94
5.2 A quadtree subdivision and its corresponding balanced quadtree sub-
division. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95
5.3 The node returned by the NorthNeighbor algorithm, Algorithm 5.1. . 95
5.4 Similar tracks with very different looping vectors . . . . . . . . . . . . 100
5.5 Using persistence for loop vector robustness. . . . . . . . . . . . . . . 102
5.6 The dendrogram arising from looping vectors on a set of tracks. . . . 103
5.7 Group Loop Formation . . . . . . . . . . . . . . . . . . . . . . . . . . 106
Acknowledgements

I strongly believe that there is no way that I would have made it to this point without surrounding myself by absolutely fantastic people who have made the last 5 years completely splendid. If I have forgotten anyone, I am very sorry.

First and foremost, thank you to my wonderful family. My parents, Bill and Brenda, have always been my biggest supporters, no matter what hair-brained scheme I decided on for my future plans, and my brother Jim has been a wonderful co-conspirator in our grad school adventures. My aunt, Barbara, has provided a safe-haven in North Carolina to run away from Duke and has shared in many drives to and from DC. I can not thank them enough for their love and support.

My advisor, John Harer, is an exceptional role model and inspiration. His intense passion and collaborative spirit create an excellent environment in which to work. He has constructed around him a fantastic group of people who have led to creative collaborations, interesting conversations, and lifelong friendships. I am honored to have been given the chance to work with him.

Firas Khasawneh is a brilliant man and a kind soul. He has been supportive, patient, and manages to put up with me despite my insanity. I have no idea how I would have made it through this without him. Ana ohebek, 7abibi. TaThkar, wahonaka da2man almal fi maoqaf almaoz.

Thank you to the many funding agencies who have given me so many fantastic opportunities, including the National Science Foundation, DDR&E, the family of Jo
Rae Wright, and the American Mathematical Society.

The Duke Math department has been an extremely enriching place to learn. I am indebted to Mike Reed, in particular, since I would have certainly dropped out without his help and support. Thank you also to my committee members Ezra Miller, Bill Pardon, and Kamesh Munagala for helpful discussions and putting up with my frantic e-mails. Thank you to Johathan Mattingly, Sayan Mukherjee, Paul Bendich, José Perea, Ingrid Daubechies, and Robert Calderbank for stimulating research discussions and for always pointing me in the right direction. Thank you to Bonnie, Monique, Tyffany, and Carolyn in the front office for all of their help. Writing this dissertation would have taken at least a few extra weeks had it not been for Mike Gratton’s LaTeX thesis template. Most especially, thank you to our fantastic computer staff, Andrew Schretter and Yunliang Yu, who make the bestlinux installation I have ever seen. Andrew, in particular, has dragged me out of quite a few self-induced system failures.

I have been very lucky in that I ended up at a school with so many fantastic teaching opportunities. Many thanks to Louis Blake, Jack Bookman, and Rann Barr-On for all of their help and mentoring in surviving the process.

There are some friends and fellow grad students who have been particularly supportive and deserve much more than a one line thank you in my dissertation.

- Jordan Wilson is the best friend the other half of my brain could ever have and has joined me for many long evenings of “No, wait, just one more turn…”
- Paul Bendich has provided many helpful discussions, attempted to restore my sanity during the dissertation writing process, and shot my bowels to hell with fabulous Szechuan food.
- José Perea is too wise for his years and an excellent dancer.
• Halcyon Derks dragged me away from a computer and into the gym periodically, and for that my arms and I are very grateful.

• Jeff Martin provides excellent beer and computer repair.

• To the rest of the sci-fi movie night crew, including but not limited to Ben Gaines, HAaron Jackson, Jenni Rinker, Chris O’Neill, Andrew Fontanella, Karen O'Connell, and Drew Crawford: thank you for many amusing Wednesday nights screaming at the TV. I am quite sure my life would not have been complete had I not been given the chance to see “Strippers Vs. Werewoles” in its entirety.
This thesis focuses on extending the now well established theory of persistent homology to time varying systems. There are three main contributions:

- The construction of a new mean for persistence diagrams which extends to a continuous pointwise mean on vineyards, the time-varying counterpart of the persistence diagram;

- An algorithm for predicting sensor network failure in a monitored system where the sensors have a probability of failure, and proof that exact computation of the probability of failure in the non-monitored case is \#P-hard;

- An application of persistent homology to behavioral predictions. We create behavior vectors for tracks of agents in satellite images and use 0-dimensional homology to cluster agents by behaviors. We also construct a flexible data structure to store and query the data in order to allow for the development for new and interesting behavior vectors to study.

In order to understand the context of this thesis, one must first gain an intuition for the uses of computational topology and persistent homology.
1.1 Persistent Homology

The idea is straightforward: watch how the homology groups of a topological space change as the space changes, and infer something about the original space from this information.

Consider a simple example where points are sampled from an annulus as in Figure 1.1a. Intuitively, we can see that this point cloud comes from a circular object, but how can we quantify this intuition so that we can still see the underlying structure if the cloud is embedded in, say, 100-dimensions? Think of centering equal-sized disks at each point in the cloud, then allow the radius of the disks to grow. As the radius of the disks changes, the blob formed by the union of these disks changes. At some radius, this blob will take on the approximate form of an annulus, until the growing radius causes the center hole to be filled in. We can then use the first homology
group, a tool from classical algebraic topology, to pick out the fact that this cloud had a hole which lasted for a while, and then went away. The information from this changing figure can be encoded into a persistence diagram as in Figure 1.1b. The hole of the annulus corresponds to the single point far from the diagonal.

Homology groups have been around for over a century, having been introduced in (Poincaré, 1895), but were not seen in applied settings until the introduction of persistent homology in (Edelsbrunner et al., 2000). For an overview of the main tools of the subject, see (Edelsbrunner and Harer, 2010).

With this powerful tool in hand, a great deal of work has been done to both improve the theory for topological data analysis (TDA) and to find new and exciting applications. Persistence has been used to study shape in proteins and protein complexes (Agarwal et al., 2006; Ban et al., 2004; Headd et al., 2007), plant root structure (Galkovskyi et al., 2012), and speech patterns (Brown and Knudson, 2009). Additionally, it has been used in image compression and segmentation (Carlsson et al., 2008; Edelsbrunner et al., 2003), neuroscience (Dabaghian et al., 2012), orthodontia (Gamble and Heo, 2010), and gene expression (Dequéant et al., 2008).

One particularly interesting application is use persistent homology to check for coverage in a sensor network. Sensors have a radius of coverage and we wish to cover a region with disks centered at the sensor locations. We could check the coverage of the sensors exactly if we had the exact location of each sensor, however we will assume we do not have this much information. Instead, we can allow the sensors to broadcast their identification to a fixed radius, so that nearby sensors know the identity of their neighbors. From this information, persistent homology can be used to check for coverage (Ghrist and Muhammad, 2005; de Silva and Ghrist, 2006, 2007; Tahbaz-Salehi and Jadbabaie, 2010). What happens if these sensors are likely to break? What can we understand about the probability of failure of the system over time when each node has a probability of failure? Chapter 4, which is taken
from work published in (Munch et al., 2012), discusses the methods for computation of this probability and proves that the problem is difficult in the sense that unless it is proven that $P = NP$, exact computation of the probability of failure is intractable. However, even though outputting an exact number is impossible, we present an algorithm which gives a warning of failure of a system which is monitored. The idea is that we can classify the sensors by whether their failure has the potential to cause system-wide failure, and give a warning whenever a new one of these sensor appears.

In Chapter 5, we will present a new application of persistent homology. Given a time series of aerial images of a busy city, we would like to automate the discovery of people acting strangely. However, since as mathematicians and not trained professionals we are not qualified to determine what constitutes strange or interesting behavior, we instead define a quantitative measurement for various behaviors which might be considered interesting. This then allows us to use 0-dimensional persistent homology to cluster people by similar behaviors. From here, we can look at subsets of the people, and create quantitative measurements for their behavior as a group to determine whether this particular subset might be of interest. We will also discuss the potential to use this information as feedback to improve the tracking algorithm which determines where each person is in each frame.

1.2 Time-varying systems, Vineyards, and Statistics

Since persistent homology is generally used to study point clouds drawn from manifolds, a large body of work has been dedicated to extending methods from probability and statistics to persistence diagrams. The approaches take many different forms. We can consider the homology of random simplicial complexes (Kahle, 2011; Kahle and Meckes, 2011), or understand and improve methods for sampling an underlying manifold (Chazal et al., 2009c,a). However, our interest lies with considering drawing several samples from the same underlying manifold, and then defining an
Figure 1.2: Examples of sensor networks. Sensors emit a disk of coverage, shown in Figure (a), and we want to check for coverage of the entire region. However, since their exact locations are unknown, we can instead consider the simplicial complex defined by the pairwise distances as shown in Figure (b). Holes in the coverage appear as one dimensional classes in the homology as seen in Figure (c).

“average” persistence diagram for these samples. While there have been several different definitions of means (Bubenik, 2012; Blumberg et al., 2012), our definition follows (Mileyko et al., 2011; Turner et al., 2011).

Here, the mean is defined geometrically rather than arithmetically. When taking an average of a set of numbers, we look for a number which minimizes the sum of the squared differences to each number. Similarly, the mean diagram should minimize the sum of the square distances to each diagram in a set. However, in this abstract space, finding such a minimizer is much more complicated. It has been shown (Mileyko et al., 2011) that such minimizers exist, however they are in general not unique. There is also an algorithm for finding such minimizers for finite sets of diagrams (Turner et al., 2011).

What has seen comparatively little work is the extension of persistent homology to time-varying systems. This is surprising since there is a great deal of data which has an associated time component. One such application will be discussed in Chapter 5. Due to the Stability Theorem (Cohen-Steiner et al., 2007), it can be shown that the motion of the persistence diagrams is continuous when the motion of the point cloud
it describes is continuous. Vineyards, which are time-varying persistence diagrams, were introduced in (Cohen-Steiner et al., 2006; Morozov, 2008) in order to give a new proof of the stability of persistence diagrams. In addition, these vineyards can be used to understand a dynamic point cloud. In Chapter 3, we give stability results for vineyards arising from a dynamic point cloud. Since the mean persistence diagram definition of (Mileyko et al., 2011; Turner et al., 2011) is not continuous with respect to a continuous vineyard, we give a modified definition of the mean. This construction is continuous for geodesic vineyards, and thus opens the door to doing statistics on vineyards.

1.3 Outline

In Chapter 2, we review the assumed background in persistent homology and the standard algorithms for its computation. Chapter 3 extends stability results to time varying persistence diagrams. It discusses previous work towards defining the mean of a set of diagrams, and gives a new definition for the mean which, unlike the previous work, is continuous when computed pointwise for sets of geodesic vineyards. We extend in Chapter 4 the application of persistent homology on static sensor networks to their time varying counterpart with a probability of failure for each sensor. In this chapter we also show that exact computation of the probability of failure of the system is \#P-hard, but give an algorithm which can give a warning for failure in a monitored system. Finally, in Chapter 5, we use persistent homology to automate the clustering of tracks of agents in satellite images by defining behavior vectors and give data structures which allow the data to be quickly queried.
In this chapter, we discuss the necessary background in persistent homology which is assumed throughout the rest of this dissertation. Section 2.1 discusses the general persistent homology theory. Section 2.2 applies these ideas to simplicial complexes and point clouds, and Section 2.3 gives the standard algorithms for computation.

This introduction follows Edelsbrunner and Harer (2010) and assumes a basic understanding of standard algebraic topology (Hatcher, 2002; Munkres, 1993). Note that here and for the remainder of the dissertation, we use homology with $\mathbb{Z}/2\mathbb{Z}$ coefficients.

2.1 Persistent Homology

2.1.1 General Case

When computing Persistent Homology, we start with a nested sequence of topological spaces,

$$\emptyset = X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots \subseteq X_n = X.$$  \hspace{1cm} (2.1)
Figure 2.1: The class $\gamma$ is born at $X_i$ because it is not in the image of the map from $H_p(X_{i-1})$. It dies entering $X_j$ because it was still not in the image of $H_p(X_{i-1})$ at $H_p(X_{j-1})$, but has merged with an older class upon entering $H_p(X_j)$.

Often this sequence arises from the level sets of a function, $f : \mathbb{X} \to \mathbb{R}$, where $X_i = f^{-1}((\infty, a_i])$ with $a_0 \leq a_1 \leq \cdots \leq a_n$. Although this is not required, the majority of this thesis will use persistent homology in this setting and thus we will assume this context for the rest of this section.

The sequence of inclusion maps from Equation (2.1) induces maps on homology for any dimension $p$,

$$0 \longrightarrow H_p(X_1) \longrightarrow H_p(X_2) \longrightarrow \cdots \longrightarrow H_p(X_n).$$

(2.2)

In order to understand the changing space, we look at where homology classes appear and disappear in this sequence.

Let $\varphi^i_j : H_p(X_i) \longrightarrow H_p(X_j)$ be the composition of necessary maps from Equation (2.2). Then the homology class $\gamma \in H_p(X_i)$ is said to be born at $X_i$ if it is not in the image of $\varphi^i_{i-1}$. This same class is said to die at $X_j$ if its image in $H_p(X_{j-1})$ is not in the image of $\varphi^j_{i-1}$, but its image in $H_p(X_j)$ is in the image of $\varphi^j_{i-1}$. See Figure 2.1.

In the case that the spaces arose from the level sets of a function $f$ as defined above, we define the persistence of a class $\gamma$ which is born at $X_i = f^{-1}((\infty, a_i])$ and dies at $X_j = f^{-1}((\infty, a_j]$ to be $\text{pers}(\gamma) = a_j - a_i$.

Note that there is no particular reason we chose absolute homology for Equation
We can define persistence equivalently with relative homology, which will be useful in Chapter 4. Instead of Equation 2.1, we start with

\[ \mathcal{Y} \subseteq \mathcal{X}_1 \subseteq \mathcal{X}_2 \subseteq \cdots \subseteq \mathcal{X}_n = \mathcal{X}. \]  

(2.3)

which induces maps on homology

\[ 0 \longrightarrow H_p(\mathcal{X}_1, \mathcal{Y}) \longrightarrow H_p(\mathcal{X}_2, \mathcal{Y}) \longrightarrow \cdots \longrightarrow H_p(\mathcal{X}_n, \mathcal{Y}). \]  

(2.4)

The definitions of birth and death of a class as well as the persistence of a class are equivalent to the absolute homology version. Notice that this equivalence can also be seen from working with persistence modules (Chazal et al., 2009b), an abstraction of the definition presented here. However this formal algebraic structure is unnecessary for our applications.

### 2.1.2 Persistence Diagrams

In order to visualize the changing homology, we draw a persistence diagram \( d_p \) for each dimension \( p \). A persistence diagram is a set of points with multiplicity in the upper half plane \( \{(b, d) \in \mathbb{R}^2 \mid d \geq b\} \) along with infinitely many copies of the points on the diagonal \( \Delta = \{(x, x) \in \mathbb{R}^2\} \). For each class \( \gamma \) which is born at \( \mathcal{X}_i \) and dies at \( \mathcal{X}_j \), we draw a point at \( (a_i, a_j) \). This gives a notion of “noise.” A point in the persistence diagram which is close to the diagonal represents a class which was born and died very quickly. On the other hand, a point which is far from the diagonal had a longer life. Depending on the context, this may have meant the class with a point far from the diagonal is more important, or more telling of the inherent topology of the space. See Figure 1.1 for an example.

Given two persistence diagrams, \( \mathcal{X} \) and \( \mathcal{Y} \), the distance is given by a modified version of the Wasserstein distance known in probability. Consider the set of bijections \( \varphi : \mathcal{X} \longrightarrow \mathcal{Y} \), where any number of off-diagonal points can be paired with the
diagonal. The $p^{th}$-Wasserstein distance chooses the best $\varphi$ to minimize the sum of the $p^{th}$ powers of the distances between the pairs.

**Definition 1.** The $p^{th}$-Wasserstein distance between two persistence diagrams $X, Y$ using $L_q$ distance (where $q$ could be $\infty$) between the points is defined to be

$$W_p[L_q](X, Y) = \inf_{\varphi: X \to Y} \left( \sum_{x \in X} (\|x - \varphi(x)\|_q)^p \right)^{1/p}.$$ 

Where the inside distance measure is clear from context, we will instead write $W_p(X, Y)$.

For the majority of our uses, we will use $L_{\infty}$ inside the sum, but for the applications of Chapter 3, this will change to $L_q, q \neq \infty$. This metric is a special case of the standard Wasserstein distance from probability as presented below.

**Definition 2.** The $p^{th}$-Wasserstein distance between two probability distributions, $\nu$ and $\eta$, on metrix space $(X, d_X)$ is

$$W_p[d_X](\nu, \eta) = \left[ \inf_{\gamma \in \Gamma(\nu, \eta)} \int_{X \times X} d_X(x, y)^p d\gamma(x, y) \right]^{1/p},$$

where $\Gamma(\nu, \eta)$ is the space of distributions on $X \times X$ with marginals $\nu$ and $\eta$ respectively. When $d_X$ is obvious from context, we will instead write $W_p(\nu, \eta)$.

This general metric will become useful in Chapter 3 when we work with the space of probability distributions on the space of persistence diagrams.

2.2 Point clouds and Simplicial Complexes

2.2.1 Simplicial Persistent Homology

This thesis will focus on computing persistent homology on a simplicial complex. In this case, we assume a finite simplicial complex $K$ with a real-valued function $f$ on the simplicies satisfying $f(\sigma) \leq f(\tau)$ if $\sigma$ is a face of $\tau$. We will often think of the
simplices as being added one at a time by finding a so-called compatible ordering of the simplices with the function values. This is an ordering of the simplices $\sigma_1, \ldots, \sigma_N$ such that

- $f(\sigma_i) \leq f(\sigma_{i+1})$ for all $i$, and
- if $f(\sigma_i) = f(\sigma_j)$ and $\sigma_i \leq \sigma_j$, then $i < j$.

We have a filtration of the simplicial complex, $\{K_i\}$, coming from the compatible ordering where the difference between $K_i$ and $K_{i-1}$ is only the $i^{th}$ simplex. We call simplex $\sigma_i$ positive if a class is born at $K_i$, and $\sigma_i$ negative if a class dies at $K_i$.

Once we have a compatible ordering, the simplicial complex is stored as a boundary matrix. This matrix has entry $[i,j] = 1$ if $\sigma_i$ is a codimension-1 face of $\sigma_j$, and is 0 otherwise. Since we have a compatible ordering, this matrix will be upper triangular. This representation of a simplicial complex will be used to compute persistence using the algorithms in Section 2.3.

2.2.2 Point clouds and simplicial complexes

To be even more specific, the applications of persistent homology in this thesis will build simplicial complexes from point clouds embedded in $\mathbb{R}^d$. Let $\chi$ be a set of points in $\mathbb{R}^d$. There are two main constructions which we will utilize: the Čech complex and the Rips complex.

The Čech complex $\mathcal{C}_\chi(r)$, or $\mathcal{C}(r)$ when $\chi$ is obvious from context, is an abstract simplicial complex with vertex set in 1-1 correspondence to $\chi$ and a simplex for every set of points whose balls of radius $r$ have a non-empty intersection. Specifically,

$$\mathcal{C}(r) = \left\{ \sigma = \langle p_0, \cdots, p_k \rangle \mid \bigcap_{i=0}^{k} B_r(p_i) \neq \emptyset \right\}$$

where $p_i \in \chi$. It is important to note that for any $q < r$, $\mathcal{C}(q) \subseteq \mathcal{C}(r)$. Thus for any
choice of increasing radii, we can construct a filtration of simplicial complexes where each space is the Čech complex of one of the radii.

An extremely useful property of the Čech complex is that it returns exactly the homology of the union of the balls centered at the points of $\chi$. This comes from the nerve lemma, presented below.

**Lemma 3. [Nerve Lemma]** Let $F = \{X_1, \cdots, X_n\}$ be a finite collection of closed sets such that every intersection between its members is either empty or contractible. Then $Nrv(F) = \{X \subseteq F \mid \cap X \neq \emptyset\}$ and the union of sets in $F$ have the same homotopy type.

Note that convex sets in $\mathbb{R}^d$, and balls in particular, satisfy the assumptions of the theorem, so we have the immediate corollary:

**Corollary 4.** For any set of points $\chi \subset \mathbb{R}^d$ and any $r \geq 0$,

$$H_k(C(r)) \cong H_k\left(\bigcup_{p \in \chi} B_r(p)\right).$$

While the Čech complex is mathematically desirable since it has the same homology as the union of balls, it is rather computationally difficult to construct given only the set of points and perhaps the pairwise distances between points. One can use the miniball algorithm, which takes $O(k)$ time for each subset of $k$ vertices, but since we must do this for every subset the computation can be prohibitive. Thus, we will approximate the Čech complex with an easier to compute complex which returns almost the correct homology.

The Vietoris-Rips complex, often shortened to Rips complex, is, like the Čech complex, a simplicial complex built on a set of points $\chi \subset \mathbb{R}^d$. However, unlike the Čech complex, the Rips complex $\mathcal{R}_\chi(r) = \mathcal{R}(r)$ has a simplex whenever all *pairs of*
Figure 2.2: A set of points equidistant from each other. The Čech and Rips complexes will both have all edges. However, the Čech complex will not have the corresponding 2-simplex since the three balls do not have a common intersection. On the other hand, the Rips complex will include the triangle since all points have pairwise distance less than twice the radius of the balls. Here, it can also be seen that the Rips complex does not give the homology of the union of balls since $H_1(\bigcup B_r(p)) \cong \mathbb{Z}_2$, while $H_1(\mathcal{R}(r)) \cong 0$.

The Rips complex is actually a specific case of the combinatorial simplicial complex known as either a flag complex or a clique complex. Given any 1-dimensional simplicial complex (more commonly known as a graph in computer science), the flag complex has a simplex $\sigma = \langle v_0, \cdots, v_d \rangle$ whenever all edges $\langle v_i, v_j \rangle$ are present in the original complex.

The Rips complex does not share the homotopy type of the union of balls of radius $r$ as can be seen in the counterexample in Figure 2.2. However, the Rips complex serves as an easy to compute approximation for the homology. In particular, we have the inclusions

$$\mathcal{R}(r) \subseteq C(\sqrt{2}r) \subseteq \mathcal{R}(\sqrt{2}r).$$

Thus, we can think of the Rips complex as a good approximation to the Čech complex.
and will use it as such in, for example, Chapter 4. Whether we work with the Rips or the Čech complex, we have an immediate notion of a function on the simplices defined by the time at which the simplex is added, so we can find a compatible ordering, and thus use the results of Section 2.2.1.

2.3 Algorithms for Computing Persistence

There are two main algorithms for computing the persistence of a simplicial complex. The most commonly used is the persistence algorithm, which we will discuss in 2.3.1. However, in the case that we are only interested in 0-dimensional homology, we can use UnionFind, Section 2.3.2, since it is a much faster method for computation although it only works for computing $H_0$.

2.3.1 The Persistence Algorithm

The idea behind the persistence algorithm is to do basic matrix column operations on the boundary matrix discussed in Section 2.2.1. Assume we have a simplicial complex with $m$ simplices with a compatible ordering, and thus can construct the boundary matrix $D$. The persistence algorithm takes the boundary matrix $D$ and turns it into a matrix $R$ via elementary column operations. This means $R = DV$ for some elementary matrix $V$.

The index of the lowest 1 in column $j$ of a matrix $R$ is denoted $\text{low}_R(j)$, or $\text{low}(j)$ when the matrix choice is obvious. If the column is completely 0, then $\text{low}(j)$ is undefined. A matrix is called reduced if $\text{low}(i) \neq \text{low}(j)$ for all $i, j$ where the low value is defined. Note that where it makes sense, we will use $\text{low}(\sigma)$ to denote the simplex associated to the row of the lowest value of the column associated to $\sigma$.

The persistence algorithm turns $D$ into a reduced matrix $R$. From the reduced matrix $R$, we can read off the pairings of simplices, and thus the entire persistence diagram. Any simplex whose column in $R$ is completely zero is a positive simplex,
Algorithm 2.1 The standard Persistence Algorithm

Input: The boundary matrix $D$
Output: Returns reduced $R$ such that $R = DV$

for $j = 1, \ldots, m$ do
  while there exists $j_0 < j$ with $\text{low}(j_0) = \text{low}(j)$: do
    Add column $j_0$ to column $j$
  end while
end for

while a simplex with a non-zero column is a negative simplex. If a simplex $\tau$ is
negative, its addition kills the class born with the addition of the simplex $\text{low}(\tau)$.
Thus, the points of the persistence diagram are $(f(\text{low}(\tau)), f(\tau))$ for all simplices in
the complex.

This algorithm returns the persistence diagrams for all dimensions, however, it
takes worst-case $O(N^3)$ time, where $N$ is the number of simplicies. In the case of
the Rips complex built on a point cloud with $n$ points, each dimension $k$ has at
most $\binom{n}{k}$ simplices, which can make the computation time prohibitive. We can fix
this problem, however, by computing the persistence only in low dimensions, thus
limiting the dimension of the necessary simplices, or by limiting the maximum radius,
thereby limiting the total number of simplices. Alternatively, for the case where we
are only interested in the 0-dimensional homology, we have an even better fix as
shown in the next section.

2.3.2 UnionFind and 0-Dimensional Homology

The UnionFind algorithm is a standard in computer science, and it can be modified
slightly to return the persistence diagram of a 1-D simplicial complex, more com-
monly known as a graph. Assume we have a function $f$ on our graph $G = (V, E)$
with $f(v) < f(e)$ for either endpoint $v$ of edge $e$, and a compatible ordering on the
set of simplices $V \cup E$. The standard UnionFind is set up to determine components
of a graph by building a data structure which keeps track of these components. The
difference for our purposes is that we both care about the order in which edges and
vertices are added, and about the function value on edges which cause components to merge.

We will think about adding edges and vertices one at a time in the order of their function values. For each vertex $v$, the data structure stores a pointer to its parent, denoted $v$.parent. This is either another vertex, or None if it is the root of the tree. There are a few commands which are needed:

- **NewVertex**($v$): Adds a new vertex $v$ to the data structure and sets $v$.parent to None.

- **Find**($v$): Finds the name of the set containing $v$ by recursively following $v$.parent, until it finds and returns the vertex who has no parent.

- **Union**($v,w$): Merges the components containing $v$ and $w$ into one, or does nothing if $v$ and $w$ are already in one tree. To do this, the algorithm calls Find($v$) and Find($w$) to determine the roots of the respective trees. If the two vertices lie in the same component, the roots returned are the same node and the algorithm does nothing. Alternatively, if the roots returned are different nodes, the algorithm sets one to be the parent of the other. Without loss of generality, if $f(v) < f(w)$ then it sets $v$.parent = $w$.

If we think about adding the simplices one at a time, there are two possibilities:

- Add a vertex $v$ and thus add a new component to the complex. This calls NewVertex($v$). Since we are interested in the birth times of the components, this vertex will also store its function value as the birth time of its component, $v$.birth.

- Add an edge $e$. This uses Union($v,w$) to determine whether the endpoints of $e$ are in the same component. If they are, then we do nothing. If they are not
already in the same component, we set \( a.\text{parent} = b \) where \( a \) and \( b \) are the roots of the respective components and \( a.\text{birth} > b.\text{birth} \). We also set the death time of the younger component, in this case corresponding to \( a \), to be the function value of the just added edge.

By merely storing the function value at which we set a vertex \( v \) to have a parent, we can determine the birth times and death times of the component information. For each vertex \( v \), its component is born at time \( f(v) \), and dies at the time stored. Each edge addition takes time \( O(\log n) \) in this naïve implementation. Some minor modifications by updating the vertex parents on the way down the recursion can improve the amortized running time per edge to \( O(\alpha(n)) \) where \( \alpha \) is the extremely slow growing inverse of the Ackermann function.

When we are only interested in \( H_0 \), UnionFind is a very fast replacement for the standard persistence algorithm. Given a graph with \( n \) vertices, there are at most \( n^2 \) edges to add, so the 0-dimensional homology can be computed with this method in \( O(n^2\alpha(n)) \), as opposed to the \( O(N^3) \) time required for standard persistence.
This chapter concerns time-varying persistence diagrams as an abstract space. These time-varying persistence diagrams are more commonly known as vineyards, but can be thought of as paths in the space of persistence diagrams. Since persistent homology and topological data analysis in general have gained so much traction for understanding static point clouds, it is natural to extend these results to time-dependent systems. Examples of interesting time-varying systems are given in Chapters 4 and 5.

A large body of work in TDA has focused on applying persistent homology to samples from an underlying manifold. The goal is to then determine properties of the manifold and how it is embedded using the information given by the persistence diagram. This is lovely if we have exactly one point cloud sampled from the manifold, but what if we draw samples several times in order to improve the accuracy of our analysis? How do we give a summary of the information stored in multiple persistence diagrams?

The answer is to define the mean of a set of diagrams. Just as the arithmetic mean of a set of points gives information about a distribution of numbers, the Fréchet
mean gives information about a distribution of elements of a metric space. Although the Fréchet mean has been shown to be nonempty in the case of persistence diagrams (Mileyko et al., 2011), it is not always unique. This creates issues when we wish to extend the mean to a time-varying persistence diagram since non-unique Fréchet means translate to discontinuous paths.

The main contribution of this chapter is to modify the definition of the mean of a set of diagrams to be a distribution rather than a set, and prove that this new definition is continuous for vineyards which are piecewise-geodesic. We also conjecture that this construction is continuous in a more general setting.

This chapter represents joint work with Paul Bendich, John Harer, Jonathan Mattingly, Sayan Mukherjee, and Katharine Turner.

Outline  This chapter is organized as follows. We define the basic metrics which will be used in Section 3.1. In Section 3.2, we use these metrics to give a stability result for vineyards which is an extension of the stability result of Cohen-Steiner et al. (2010) for persistence diagrams. Section 3.3 describes the Fréchet mean of Mileyko et al. (2011) and the algorithm of Turner et al. (2011). Finally, Section 3.4 gives the new construction of a mean for diagrams and proves continuity for geodesic vineyards.

3.1 Standard Metric Definitions

The $L_p$ distance for two points in $\mathbb{R}^d$ is

$$
\|x - y\|_p := \left[ \sum_{i=1}^{d} (x_i - y_i)^p \right]^{1/p}
$$

while the $L_\infty$ distance is

$$
\|x - y\|_\infty := \max\{ |x_1 - y_1|, |x_2 - y_2|, \ldots, |x_d - y_d| \}.
$$
The \( L_\infty \) distance between two functions \( f, g : X \to \mathbb{R}^d \) for any topological space \( X \) is
\[
\| f - g \|_\infty = \sup_{x \in X} \| f(x) - g(x) \|_\infty.
\]

For two fixed, finite point clouds in \( \mathbb{R}^d \), there is an obvious notion of distance between them: The Hausdorff Distance.

**Definition 5.** Let \( X \) and \( Y \) be subsets of a metric space with metric \( d \). Then the Hausdorff distance is given by
\[
H[d](X, Y) := \max \left\{ \sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y) \right\}. \tag{3.1}
\]

See Figure 3.1a for an example. For the case in which we are interested, \( X \) and \( Y \) are finite point clouds, so the supremums and infimums of the Hausdorff distance can be replaced with maxima and minima.

Given two persistence diagrams \( X \) and \( Y \), the distance between them is given by the Wasserstein distance as in Definition 1, repeated here for completeness.
**Definition 6.** The Wasserstein distance between two persistence diagrams $X$ and $Y$ is given by

$$W_p[L_q](X, Y) := \inf_{\varphi: X \to Y} \left[ \sum_{x \in X} \|x - \varphi(x)\|_q \right]^{1/p}$$

where $1 \leq p, q \leq \infty$ and $\varphi$ ranges over bijections between $X$ and $Y$.

Notice that for $p = \infty$,

$$W_\infty[L_q](X, Y) := \inf_{\varphi: X \to Y} \sup_{x \in X} \|x - \varphi(x)\|_q.$$

$W_\infty[L_\infty]$ is often referred to as the bottleneck distance.

### 3.2 Stability

#### 3.2.1 Bottleneck Stability for Persistence Diagrams

A major breakthrough in the development of persistent homology as a tool for data analysis came with the stability theorem of Cohen-Steiner et al. (2007). Before we can state the theorem, we need a few definitions.

**Definition 7.** Let $\mathbb{X}$ be a topological space and $f$ a real function on $\mathbb{X}$. A homological critical value of $f$ is a real number $a$ for which there exists an integer $k$ such that for all sufficiently small $\varepsilon > 0$, the map $H_k(f^{-1}(-\infty, a - \varepsilon]) \to H_k(f^{-1}(-\infty, a + \varepsilon])$ induced by inclusion is not an isomorphism.

**Definition 8.** A function $f: \mathbb{X} \to \mathbb{R}$ is tame if it has a finite number of homological critical values and the homology groups $H_k(f^{-1}(-\infty, a])$ are finite-dimensional for all $k \in \mathbb{Z}$ and $a \in \mathbb{R}$.

Note that homological critical values are those where the homology of the sublevel sets changes. Thus, in the case of a finite simplicial complex, the homological critical values are a subset of the function values on the simplices. For the case of $\mathbb{X}$, a finite
point cloud, this implies via the nerve lemma that \( d_X(a) = \inf_{x \in X} \|x - a\| \) is a tame function. Given these definitions, we are ready to state the stability theorem from Cohen-Steiner et al. (2007).

**Theorem 9.** Let \( X \) be a triangulable space with continuous, tame functions \( f, g : X \to \mathbb{R} \), and let \( D(f) \) and \( D(g) \) be the persistence diagrams arising from the sublevel sets of the respective functions. Then the persistence diagrams satisfy

\[
W_\infty[L_\infty](D(f), D(g)) \leq \|f - g\|_\infty.
\]

Intuitively, this means that if a function on a topological space is wiggled a little bit, the persistence diagrams do not change much. In the case of finite point clouds \( X \) and \( Y \) in \( \mathbb{R}^d \), let \( d_X \) and \( d_Y \) be the induced distance functions on \( \mathbb{R}^d \). Since the point clouds are finite, \( d_X \) and \( d_Y \) are tame. \( d_X^{-1}(-\infty, a] \) is homotopic to \( C(a) \) by the nerve lemma, Lemma 3, so \( D(d_X) \) is the persistence diagram given by the filtration of Čech complexes. Then Theorem 9 can be used to give the following corollary for the Čech complex construction.

**Corollary 10.** Given two finite point clouds \( X \) and \( Y \) in \( \mathbb{R}^d \),

\[
W_\infty[L_\infty](D(d_X), D(d_Y)) \leq \|d_X - d_Y\|_\infty.
\]

Additionally, in the case of finite point clouds, Hausdorff distance and \( L_\infty \) distance for the induced distance functions are equivalent as shown in the following lemma.

**Lemma 11.** For finite point clouds \( X \) and \( Y \),

\[
\|d_X - d_Y\|_\infty = H[L_\infty](X, Y).
\]

**Proof.** For brevity, write \( H(X, Y) \) for \( H[L_\infty](X, Y) \) and \( \|x - y\| \) for \( \|x - y\|_\infty \).

Fix \( c \in \mathbb{R}^d \). Then \( d_X(c) = \min_{x \in X} \|c - x\| \), so there is a point \( x \in X \) which minimizes this. By definition of the Hausdorff distance, for any \( x \in X \), there is a
\[ y \in Y \text{ such that } \|x - y\| \leq H(X, Y). \text{ Note then that } \|c - y\| \geq d_Y(c). \]  

Thus

\[ d_Y(c) \leq \|y - c\| \leq \|y - x\| + \|x - c\| \leq H(X, Y) + d_X(c) \]

hence,

\[ d_Y(c) - d_X(c) \leq H(X, Y). \]

Repeating the same process for \( Y \) gives

\[ \|d_X(c) - d_Y(c)\| \leq H(X, Y) \]

for any point \( c \in \mathbb{R}^2 \), and thus \( \|d_X - d_Y\| \leq H(X, Y) \).

To show the other inequality, let \( x_0 \in X \) and \( y_0 \in Y \) be the points which achieve the Hausdorff distance, so that \( H(X, Y) = \|x_0 - y_0\| \). WLOG, let this be

\[ \|x_0 - y_0\| = \sup_{x \in X} \inf_{y \in Y} \|x - y\|. \]

Consider the distance functions at the point \( x_0 \). Since \( d_X(x_0) = 0 \), while \( d_Y(x_0) = H(X, Y) \), we have

\[ H(X, Y) = |d_X(x_0) - d_Y(x_0)| \leq \sup_{c \in \mathbb{R}^d} |d_X(c) - d_Y(c)| \]

and thus \( H(X, Y) \leq \|d_X - d_Y\| \), so the lemma follows.

\[ \Box \]

This gives another immediate corollary to Theorem 9 by replacing the \( L_\infty \) distance by the Hausdorff distance.

**Corollary 12.** Given two finite point clouds \( X \) and \( Y \) in \( \mathbb{R}^d \),

\[ W_\infty[L_\infty](D(d_X), D(d_Y)) \leq H[L_\infty](X, Y). \]

This corollary will be quite useful in the next section in order to prove similar stability results for time-varying persistence diagrams.
A dynamic point cloud \( X(t) = \{x_1(t), \cdots, x_N(t)\} \) is a point cloud which is moving continuously for a finite amount of time. For simplicity, assume time ranges from 0 to 1. Thus a dynamic point cloud is a map

\[
[0, 1] \rightarrow (\mathbb{R}^d)^N \\
\quad t \mapsto \{x_1(t), \cdots, x_N(t)\}.
\]

It is natural to consider the changing persistence diagrams arising from these dynamic point clouds. Vineyards, or time-varying persistence diagrams, were introduced in Cohen-Steiner et al. (2006) and Morozov (2008) as a tool to understand time-series data such as dynamic point clouds. Given a dynamic point cloud \( X \), there is a persistence diagram \( D(X(t)) \) for each time \( t \). This 1-parameter family of diagrams is called a vineyard,

\[
V(X) = \{D(X(t)) \mid t \in [0, 1]\}.
\]

where \( D_\infty \) represents the space of persistence diagrams. This section will use \( W_\infty[L_\infty] \) as the metric on the space of diagrams and \( H[L_\infty] \) on the point clouds, which will be written as \( W_\infty \) and \( H \) respectively.

**Corollary 13.** If a dynamic point cloud \( X(t) \) is continuous with respect to Hausdorff distance, the corresponding vineyard \( V(X) \) is continuous with respect to bottleneck distance.

**Proof.** Immediate from Corollary 12. \( \square \)

The first definitions of vineyards (Cohen-Steiner et al., 2006; Morozov, 2008) used them in the well-behaved case of a homotopy between two functions. In this case,
Figure 3.2: An example of a vineyard. For each time, given on the $z$-axis, there is a persistence diagram. Since vineyards arising from continuous point clouds are continuous, each point in the diagram traces out a path called a vine. These vines can have endpoints on the starting or ending times, or on the plane which projects to the diagonal.

Each off-diagonal point of $D(X(t))$ moves continuously in time and is called a vine. Vines can start and end at off diagonal points at times 0 or 1, or have starting or ending points on the diagonal for any $t$. See Figure 3.2. However, in the context of this chapter, and in particular in Section 3.4, we will allow vineyards to be arbitrary paths in persistence diagram space. It is not clear whether arbitrary vineyards can be decomposed into well-defined, individual vines.

In order to prove stability for vineyards, metrics are needed for both the space of dynamic point clouds, and the space of vineyards. Since there is a notion of distance between diagrams for each time $t$ as well as between static point clouds for each time $t$, these metrics can be integrated over time to obtain new metrics for their time-varying counterparts.
**Definition 14.** Given vineyards $V(\mathbb{X})$ and $V(\mathbb{Y})$ arising from dynamic point clouds $\mathbb{X}$ and $\mathbb{Y}$, the integrated bottleneck metric is given by

$$I[W_\infty](V(\mathbb{X}), V(\mathbb{Y})) := \int_0^1 W_\infty(D(\mathbb{X}(t)), D(\mathbb{Y}(t))) \, dt$$

and the integrated Hausdorff metric is given by

$$I[H](\mathbb{X}, \mathbb{Y}) := \int_0^1 H(\mathbb{X}(t), \mathbb{Y}(t)) \, dt.$$

To show that the functions as defined are, in fact, metrics, it must first be shown that the Hausdorff distance, and therefore the bottleneck distance, are continuous when the dynamic point clouds are continuous.

**Lemma 15.** For continuous, dynamic point clouds $\mathbb{X}$ and $\mathbb{Y}$, $H(\mathbb{X}(t), \mathbb{Y}(t))$ and $W_\infty(D(\mathbb{X}(t)), D(\mathbb{Y}(t)))$ are continuous functions of $t$.

**Proof.** Let $\varepsilon > 0$ be given and let $\| \cdot \|$ denote $L_\infty$ distance. Since $\mathbb{X}$ and $\mathbb{Y}$ are both continuous, for each point $x_i \in \mathbb{X}$ and each point $y_j \in \mathbb{Y}$, there are respective $\delta_i$ and $\delta_j$ such that $|t_1 - t_2| \leq \delta_i$ implies $\|x_i(t_1) - x_i(t_2)\| \leq \varepsilon/2$ and $|t_1 - t_2| \leq \delta_j$ implies $\|y_j(t_1) - y_j(t_2)\| \leq \varepsilon/2$.

Let $\delta$ be the minimum of all the $\delta_i$ and $\delta_j$ and assume $|t_1 - t_2| \leq \delta$. Assume also that $x_i$ and $y_j$ are the points which achieve the Hausdorff distance for the point clouds at time $t_1$; that is, $H(\mathbb{X}(t_1), \mathbb{Y}(t_1)) = \|x_i(t_1) - y_j(t_1)\|$. Note that $\|x_i(t_2) - y_j(t_2)\| \geq H(\mathbb{X}(t_2), \mathbb{Y}(t_2))$.

Then using continuity of the point clouds and the triangle inequality,

$$H(\mathbb{X}(t_2), \mathbb{Y}(t_2)) \leq \|x_i(t_2) - y_j(t_2)\|$$

$$\leq \|x_i(t_2) - x_i(t_1)\| + \|x_i(t_1) - y_j(t_1)\| + \|y_j(t_1) - y_j(t_2)\|$$

$$\leq \varepsilon/2 + H(\mathbb{X}(t_1), \mathbb{Y}(t_1)) + \varepsilon/2$$
and therefore
\[ H(\mathbb{X}(t_2), \mathbb{Y}(t_2)) - H(\mathbb{X}(t_1), \mathbb{Y}(t_1)) \leq \epsilon. \]

Reversing \( t_1 \) and \( t_2 \) gives the opposite direction, and thus
\[ \|H(\mathbb{X}(t_2), \mathbb{Y}(t_2)) - H(\mathbb{X}(t_1), \mathbb{Y}(t_1))\| \leq \epsilon \]

implying that \( H(\mathbb{X}(t), \mathbb{Y}(t)) \) is a continuous function of \( t \).

To see that the bottleneck distance is also continuous, choose the same \( \delta \) for the given \( \epsilon \). Then utilizing Corollary 12 for any \(|t_1 - t_2| \leq \delta\) and,
\[ \|W_\infty(D(\mathbb{X}(t_1)), D(\mathbb{Y}(t_1))) - W_\infty(D(\mathbb{X}(t_1)), D(\mathbb{Y}(t_1)))\| \leq \epsilon. \]

\[ \square \]

**Lemma 16.** The integrated bottleneck and integrated Hausdorff metrics of Definition 14 are, in fact, metrics.

**Proof.** The regular Hausdorff and bottleneck metrics are both non-negative by definition and are continuous by Lemma 15. So, the integral of these metrics over the interval \([0, 1]\) must be non-negative, and therefore \( I[H](\mathbb{X}, \mathbb{Y}) \geq 0 \) and \( I[W_\infty](V(\mathbb{X}), V(\mathbb{Y})) \geq 0 \). Moreover, since \( H \) is non-negative and continuous, if \( I[H](\mathbb{X}, \mathbb{Y}) = 0 \), then \( H(\mathbb{X}(t), \mathbb{Y}(t)) = 0 \) for all \( t \). By the properties of a distance function, this implies that \( \mathbb{X}(t) = \mathbb{Y}(t) \) for all \( t \). Likewise, if \( I[B](V(\mathbb{X}), V(\mathbb{Y})) = 0 \), then \( \mathbb{X}(t) = \mathbb{Y}(t) \) for all \( t \).

As \( H \) and \( W_\infty \) are both distances, \( H(A, B) = H(B, A) \) and \( W_\infty(A, B) = W_\infty(A, B) \), so
\[ I[W_\infty](V(\mathbb{X}), V(\mathbb{Y})) = I[W_\infty](V(\mathbb{Y}), V(\mathbb{X})) \]
and
\[ I[H](\mathbb{X}, \mathbb{Y}) = I[H](\mathbb{Y}, \mathbb{X}). \]
Since $H$ is a metric, thereby satisfying its own triangle inequality,

$$I[H](X, Y) = \int_0^1 H(X(t), Y(t)) \, dt$$

$$\leq \int_0^1 H(X(t), Z(t)) + H(Z(t), Y(t)) \, dt$$

$$= \int_0^1 H(X(t), Z(t)) \, dt + \int_0^1 d(Z(t), Y(t)) \, dt$$

$$= I[H](X, Z) + I[H](Z, Y)$$

for any point clouds $X$, $Y$, and $Z$. The triangle inequality for $I[W_\infty]$ is similar.

These definitions immediately give a stability theorem for vineyards arising from moving point clouds.

**Theorem 17.** Given finite dynamic point clouds $X$ and $Y$,

$$I[W_\infty](V(X), V(Y)) \leq I[H](X, Y).$$

**Proof.** From Corollary 12, for any $t$ we have

$$W_\infty(D(X(t)), D(Y(t))) \leq H(X(t), Y(t))$$

and integrating gives the theorem.

This metric may not be the best metric for all cases. For instance, we can make $H(X(t), Y(t))$ as large as we like without changing $W_\infty(D(X(t)), D(Y(t)))$ by translating the point cloud $Y(t)$ far away from $X(t)$. Since translations and rotations of the point cloud do not change the persistence diagrams, we obtain the same stability theorem if we replace the dynamic point cloud metric by its minimum over all translations and rotations of the point clouds.
Consider the following example of two point clouds which are close although we may not want them to be close. The dynamic point cloud $X$ consists of two points at constant distance $h$ apart and moving straight forward. The dynamic point cloud $Y$ consists of $n$ points: one point following the exact path of one of the points of cloud $X$, while the rest of the $n - 1$ points cluster around the other point at distance at most $\epsilon$. For this case, it is easy to see that the integrated Hausdorff distance is at most $\epsilon$.

3.3 Probability on Persistence Diagrams

This section will present the previous work done to define the mean of a distribution of persistence diagrams. We will give some properties of the abstract space of persistence diagrams, $D_p$, define the mean of a diagram as the Fréchet mean, and give an algorithm for computation of the mean when the Wasserstein distance is given a mild modification.

3.3.1 The Space $(D_p, W_p(L_\infty))$

In order to define a framework for statistics, we will forget the connection to point clouds for a moment and instead focus on the space of persistence diagrams abstractly.

**Definition 18.** An abstract persistence diagram is a countable multiset of points along with the diagonal, $\Delta = \{(x, x) \in \mathbb{R}^2 \mid x \in \mathbb{R}\}$, with points in $\Delta$ having infinite multiplicity.

The distance between these abstract diagrams is the $p^{th}$ Wasserstein distance. Although persistence diagrams do not have a norm, consider the distance from any diagram to the empty diagram, $d_\emptyset$, which is the diagram consisting of only the diagonal. Let the persistence of a point be $\text{pers}(x) = x_2 - x_1$ and notice that
pers(x) = 2 \inf_{z \in \Delta} \|x - z\|_\infty. Then
\[ W_p[L_\infty](d, d_\emptyset)^p = \sum_{x \in d} \|x - \Delta\|_\infty^p \]
\[ = \sum_{x \in d} (\frac{1}{2} \text{pers}(x))^p \]

This means that the space of diagrams whose distance to \(d_\emptyset\) is finite is equivalent to the space of diagrams whose \(p^{th}\) total persistence,
\[ \text{Pers}_p(d) = \sum_{x \in d} (\text{pers}(x))^p, \]
is finite. Thus, we will limit ourselves to those diagrams who have finite \(p^{th}\) total persistence.

**Definition 19.** The space of persistence diagrams is
\[ D_p = \{ d \mid W_p[L_\infty](d, d_\emptyset) < \infty \} = \{ d \mid \text{Pers}_p(d) < \infty \} \]
along with the \(p^{th}\)-Wasserstein metric, \(W_p[L_\infty]\), from Definition 1.

Mileyko et al. (2011) shows that \(D_p\) is complete and separable, commonly known as Polish, and gives necessary and sufficient properties for compact sets in \(D_p\). In order to state these properties, several definitions are required.

**Definition 20.** A set \(S \subset D_p\) is birth-death bounded if there is a constant \(C > 0\) such that for all \(d \in S\) and for all \(x = (x_1, x_2) \in d\), \(\max\{|x_1|,|x_2|\} < C\).

A set \(S \subset D_p\) is off-diagonally birth-death bounded if for all \(\varepsilon > 0\), the set \(u_\varepsilon(S) = \{ u_\varepsilon(d) \mid d \in S \}\) is birth-death bounded, where
\[ u_\varepsilon(d) = \{ x \in d \mid \|x - \Delta\| > \varepsilon \} \]

**Definition 21.** A set \(S \subset D_p\) is uniform if for all \(\varepsilon > 0\) there exists \(\alpha > 0\) such that \(W_p(l_\alpha(d), d_\emptyset) < \varepsilon\) for all \(d \in S\) where
\[ l_\alpha(d) = \{ x \in d \mid \|x - \Delta\| < \alpha \}. \]
With these definitions, the necessary and sufficient properties of compact sets in $D_p$ proven in Mileyko et al. (2011) can be stated.

**Theorem 22.** A set $S \subset D_p$ is compact if and only if it is bounded, off-diagonally birth-death bounded, and uniform.

### 3.3.2 Fréchet Means

With the structure of $D_p$ in hand, we turn to the concept of Fréchet means, a mean which can be defined on any metric space. A probability space $(\Omega, \mathcal{F}, \mathcal{P})$ consists of three things:

- The sample space $\Omega$ which is the set of potential outcomes for an experiment. In this case, $\Sigma$ will be the set of diagrams $D_p$.
- A $\sigma$-algebra $\mathcal{F}$ which gives sets of possible events. In this case, we will use the Borel algebra $\mathcal{B}(D_p)$, which is the smallest $\sigma$-algebra which contains all the open sets of $D_p$.
- A probability measure $\mathcal{P}$ on $D_p$.

The probability measure $\mathcal{P}$ on $D_p$ is said to have a finite $q^{th}$ moment if

$$\mathcal{F}_p(X) = \int_{D_p} [W_p(X,Y)]^q d\mathcal{P}(Y) < \infty.$$ 

for all $X \in D_p$. We will require that our probability measures have finite second moment.

Given this probability space, we can define the Fréchet mean as follows.

**Definition 23.** Given a probability space $(D_p, \mathcal{B}(D_p), \mathcal{P})$, the quantity

$$\text{Var}_p = \inf_{X \in D_p} \left[ F_p(X) = \int_{D_p} W_p(X,Y)^2 d\mathcal{P}(Y) < \infty \right]$$

31
is the Fréchet variance of $\mathcal{P}$ and the set at which the value is obtained

$$\mathbb{E}(\mathcal{P}) = \{ X | F_P(X) = \text{Var}\mathcal{P} \}$$

is the Fréchet expectation, also called Fréchet mean.

Here, the mean is a set, not necessarily a single diagram. In fact, there is no guarantee that $\mathbb{E}(\mathcal{P})$ is either unique or non-empty. However, it was proved in Mileyko et al. (2011) that the Fréchet mean for $(D_p, W_p[L_\infty])$ is non-empty for certain types of well-behaved probability measures on $D_p$.

**Theorem 24.** Let $\mathcal{P}$ be a probability measure on $(D_p, \mathcal{B}(D_p))$ with a finite second moment. If $\mathcal{P}$ has compact support, then $\mathbb{E}(\mathcal{P}) \neq \emptyset$.

Since compactness of the support is rather restrictive, in the same paper, Mileyko et al. proved that the Fréchet mean is non-empty with less stringent conditions on the probability distribution. First, given a metric space $X$, a probability measure $\mu$ on the space $(X, \Omega)$ is called tight if for all $\varepsilon > 0$ there is a compact set $S \subset X$ such that $\mu(X - S) < \varepsilon$. The probability measure $\mu$ is said to have a rate of decay at infinity $q$ if for some $x_0 \in X$, there is a $C > 0$ and $R > 0$ such that for all $r \geq R$, $\mu(B_r(x_0)) \leq Cr^{-q}$.

**Theorem 25.** Let $\mathcal{P}$ be a tight probability measure on $(D_p, \mathcal{B}(D_p))$ with the rate of decay at infinity $q > \max\{2, p\}$. Then $\mathbb{E}(\mathcal{P}) \neq \emptyset$.

These two theorems are extremely interesting since they give cases where we are sure to have a Fréchet mean, however they do not give a method for computing a mean. In order to give specific examples, the distance metric must be modified slightly.
3.3.3 Algorithm for Persistence Diagram Mean

With a minor modification of the metric on $D_p$, a simple to understand but difficult to prove algorithm was stated in Turner et al. (2011). The idea is to use Wasserstein distance, Definition 1, with $L_2$ distance inside instead of $L_\infty$.

With $\text{pers}(x) = x_2 - x_1$, we can see that requiring a finite distance to the empty diagram $d_\emptyset$ is still equivalent to finite $p^{\text{th}}$ total persistence since

$$W_p[L_2](d, d_\emptyset)^p = \sum_{x \in d} \| x - \Delta \|_2^p$$

$$= \sum_{x \in d} \left( \frac{\sqrt{2}}{2} \text{pers}(x) \right)^p .$$

Thus, we have an equivalent notion of the space $D_p$ with a new metric. In particular, we will be interested in the case where $p = 2$.

Definition 26. The space of $L_2$ persistence diagrams is

$$D_p = \{ d \mid W_p[L_2](d, d_\emptyset) < \infty \} = \{ d \mid \text{Pers}_p(d) < \infty \}$$

along with the $L_2$ $p^{\text{th}}$-Wasserstein metric, $W_p[L_2]$, from Definition 1.

Since for any $x, y \in \mathbb{R}^2$,

$$\| x - y \|_\infty \leq \| x - y \|_2 \leq \sqrt{2} \| x - y \|_\infty ,$$

we have

$$W_p[L_\infty](d_1, d_2) \leq W_p[L_2](d_1, d_2) \leq \sqrt{2} W_p[L_\infty](d_1, d_2)$$

and therefore $W_p[L_2]$ and $W_p[L_\infty]$ are topologically equivalent metrics. In particular, $(D_p, W_p[L_2])$ is also complete and separable since $(D_p, W_p[L_\infty])$ is complete and separable, equivalent definitions for compact sets hold, and Theorems 24 and 25 are still true.
The first important property given by this change in metric is the notion of a geodesic. In order to show that \(D_2\) is a geodesic space, it must be shown that every pair of diagrams has a minimal geodesic between them. This geodesic can be defined using a bijection between the diagrams which minimizes Wasserstein distance.

Let diagrams \(X\) and \(Y\) be given and let \(\{x_i\}\) and \(\{y_i\}\) be the respective sets of off-diagonal points. Let \(\varphi : X \to Y\) be a bijection which minimizes the Wasserstein distance \(W_2[L_2](X,Y)\). Consider the vectors \(v_i\) in \(\mathbb{R}^2\) which start at \(x_i\) and end at \(\varphi(x_i)\). If \(x_i\) is paired with the diagonal, then the vector ends at \(y \in \Delta\) which minimizes \(\|x - \Delta\|\). Likewise, for all \(y_i\) paired with the diagonal, there is a vector from the minimizer of \(\|\Delta - y_i\|\) on \(\Delta\) to \(y_i\).

Then let \(\gamma(t)\) be the diagram with points \(x_i + t\vec{v}_i\) for all \(x_i\) and corresponding \(\vec{v}_i\), where the \(x_i\) or \(x_i + \vec{v}_i\) could be in the diagonal. Notice \(\gamma(1) = Y\), and the obvious bijection for the Wasserstein distance \(W_p(X, \gamma(t))\) pairs \(x_i\) with \(x_i + t\vec{v}_i\). This bijection is also optimal for each \(t\), so \(\gamma(t)\) is a minimal geodesic.

In addition to being a geodesic space, Turner et al. (2011) proved is that \((D_2, W_2[L_2])\) is a non-negatively curved Alexandrov space with curvature bounded from below. A space is Alexandrov if the intersection of any family, not necessarily finite, of open sets is open. This structure along with the geodesic structure allows for the construction of the tangent cone \(T_Y\) for any \(Y \in D_2\). Let \(\hat{\Sigma}\) be the set of geodesics from \(Y\). For any two geodesics, \(\sigma, \eta \in \hat{\Sigma}\), the angle between them can be computed using an analogue of the law of cosines,

\[\angle_Y(\sigma, \eta) = \arccos \left( \lim_{s,t \to 0} \frac{s^2 + t^2 - W_2(\sigma(s), \eta(t))^2}{2st} \right)\].

Two geodesics are said to start in the same direction if \(\angle_Y(\sigma, \eta) = 0\); denote this by \(\sigma \sim \eta\). Then let \((\Sigma_Y, \angle_Y)\), the space of directions from \(Y\), be the completion of \(\hat{\Sigma}/ \sim\). The tangent cone, \(T_Y\) is defined to be \(\Sigma_Y \times [0, \infty)/\Sigma_Y \times \{0\}\). This space has
a metric defined by
\[ d((\sigma, s), (\eta, t))^2 = s^2 + t^2 - 2st \cos \angle_Y(\sigma, \eta). \]
and an inner product
\[ \langle (\sigma, s), (\eta, t) \rangle = st \cos(\angle_y(\sigma, \eta)). \]

Given any diagram \( Y \), a vector in this tangent cone corresponds to a vector for each off-diagonal point, and at worst countably many vectors coming out of the diagonal where the sum of the square lengths of these vectors is finite.

Since we have a tangent cone, we can define differentials and gradients of a function. For \( s > 0 \), denote the rescaled space \((D_2, s \cdot W_2)\) by \( sD_2 \). Let \( i_s \) be the map \( sD_2 \to D_2 \). Then for any open \( A \subset D_2 \) and function \( f : A \to \mathbb{R} \), the differential of \( f \) at a diagram \( Y \in A \) is a map \( T_Y \to \mathbb{R} \) given by
\[ d_Y f = \lim_{s \to \infty} s(f \circ i_s - f(p)). \]
The differential is well defined for semiconcave functions \( f \), so we can understand the gradient of the function. The gradient of \( f \) at \( Y \) is defined to be \( \nabla_Y f \), the vector in \( T_Y \) such that \( d_Y f(x) \leq \langle \nabla_Y f, x \rangle \) for all \( x \in T_Y \), and \( d_Y f(\nabla_Y f) = \langle \nabla_Y f, \nabla_Y f \rangle \).

It is shown in Turner et al. (2011) that the Fréchet function is semiconcave for distributions with bounded support, so we can make use of a gradient descent algorithm to find local minima of the Fréchet function, Definition 23. In order to present the algorithm for computing the Fréchet mean, we must first describe the algorithm for computation of Wasserstein distance. The representation of a diagram is a list of its off-diagonal points, \( X = [x_1, \ldots, x_k] \). In order to compute the Wasserstein distance between two diagrams, we will reduce the problem to computing a minimum cost matching of a complete, weighted bipartite graph.

Let \( X = [x_1, \ldots, x_k] \) and \( Y = [y_1, \ldots, y_m] \) be diagrams. In order to compute \( W_2[L_2](X, Y) \), we construct a bipartite graph with vertex set \( V = U \cup V \). \( U \) has
a vertex for each $x_i$, as well as $m$ vertices representing the abstract diagonal $\Delta$. Likewise, $V$ has a vertex for each $y_i$ as well as $k$ vertices representing $\Delta$. Take all edges from $U$ to $V$ so that this is a complete bipartite graph. Each edge $(x_i, \Delta)$ and $(\Delta, y_j)$ has weight $\|x_i - \Delta\|$ and $\|y_j - \Delta\|$ respectively where $\|a - \Delta\| = \min_{z \in \Delta} \|a - z\|$. Edges between two vertices representing $\Delta$ are given weight 0. This weighted bipartite graph can be given to any off the shelf min cost matching algorithm, although we use the Hungarian algorithm of Munkres (Munkres, 1957). The reason for so many additional $\Delta$ vertices in the construction is to allow for the matching which pairs every $x_i$ with $\Delta$ and every $y_i$ with $\Delta$.

A minimum cost matching in the bipartite graph immediately gives a bijection $\varphi : X \rightarrow Y$ and the Wasserstein distance is given by the square root of the sum of the squares of the weights of the edges. Notice that since there could be multiple matchings for a bipartite graph which minimizes the cost, this immediately implies that there could be multiple matchings which minimize the Wasserstein distance. For the purposes of the algorithm to compute the mean diagram, we will actually be more interested in the bijection returned in this algorithm than in the distance itself. See Figure 3.3 for an example of a pair of diagrams and their corresponding bipartite graph.

Definition 27. Given a set of diagrams $X_1, \cdots, X_N$, a selection is a choice of one point from each diagram, where that point could be $\Delta$. The trivial selection for a particular off-diagonal point $x \in X_i$ is the selection $s_x$ which chooses $x$ for $X_i$ and $\Delta$ for every other diagram.

A matching is a set of selections so that every off-diagonal point of every diagram is part of exactly one selection.

Given a set of diagrams, a matching is the analog of a bijection for a pair of diagrams. A matching for $N$ diagrams which has $k$ selections can be stored as a
Figure 3.3: Computation of the Wasserstein distance between $d_{\mathbf{■}}$ and $d_{\cdot}$ in Figure (a). The problem is turned into the problem of computing a minimum cost matching on the weighted graph in Figure (b). The matching chosen, shown in the bold edges in (b), is used to determine the bijection for the diagrams in (a). Dashed edges in (b) correspond to $\Delta - \Delta$ pairings, which contribute nothing to the total distance.

$k \times N$ matrix $G$ where entry $G[j, X_i] = x$ means that the $j^{th}$ selection has point $x \in X_i$. Note that we consider matchings to be equivalent up to reordering of the selections. See Figure 3.4 for an example. In this case, the matching shown is given by the matrix

$$
\begin{pmatrix}
d_{\mathbf{■}} & d_{\mathbf{■}} & d_{\cdot} \\
1 & b & x & f \\
2 & a & \Delta & \Delta \\
3 & \Delta & y & g \\
4 & \Delta & z & \Delta \\
5 & a & \Delta & \Delta \\
6 & c & \Delta & \Delta \\
\end{pmatrix}
$$

(3.2)

where $\Delta$ represents the diagonal.

The mean of a selection is the point which minimizes the sum of the square
distances to the elements of the selection. Consider the mean of the selection \( m \) consisting of \( N \) points: \( \{p_1, \ldots, p_k\} \) with \( p_i = (x_i, y_i) \) off-diagonal, and the rest copies of the diagonal \( \Delta \). A quick computation gives this point as

\[
\text{mean}_X(m) = \frac{1}{2Nk} \left( (N + k) \sum x_i + (N - k) \sum y_i, \right. \\
\left. (N - k) \sum x_i + (N + k) \sum y_i \right).
\]

Sometimes it may be simpler to consider the mean of two selections in rotated coordinates with axes \((1/\sqrt{2}, 1/\sqrt{2})\) and \((-1/\sqrt{2}, 1/\sqrt{2})\). Writing \( p_i = (a_i, b_i) \) in these coordinates, Equation 3.3 becomes

\[
\text{mean}_X(m) = \left( \frac{1}{k} \sum_{i=1}^k a_i, \frac{1}{N} \sum_{i=1}^k b_i \right).
\]

In these coordinates, it is easier to see what happens for the mean of a trivial selection.
If the single off-diagonal point is at $x = (a, b)$ and there are a total of $N$ diagrams,

$$\text{mean}_{X}(m_{x}) = \left( a, \frac{1}{N} b \right)$$

so the point is placed at distance $\frac{1}{N}\|x - \Delta\|$ from the diagonal.

The \textit{mean of a matching}, $\text{mean}(G)$, is a diagram in $D_{p}$ with a point at the mean of each selection. When it is unclear as to the set of diagrams from which this mean arose, we will denote it as $\text{mean}_{X}(G)$. Note that the mean of the selection yields a point while the mean of a matching yields a diagram.

Now we are ready to give the algorithm for the Fréchet mean of a set of diagrams. Given a finite set of diagrams $\{X_{1}, \cdots, X_{N}\}$, start with a candidate for the mean, $Y$, and compute the bijection for $W_{2}(L_{2})(Y, X_{i})$. We denote this as $\text{WassersteinPairing}(Y, X_{i})$. From this, we have a matching $G$ where $G[y_{j}, X_{i}]$ gives the point in $X_{i}$ which was paired to point $y_{j} \in Y$. Set $Y' = \text{mean}(G)$. This new diagram is now the candidate for the mean and the process is repeated. The algorithm terminates when the Wasserstein pairing doesn’t change. Turner et al. (2011) uses the structure of $(D_{2}, W_{2}[L_{2}])$ to prove that this algorithm terminates at a local minimum of the Fréchet function. See Algorithm 3.1 for the pseudocode.

### 3.4 Probability on Vineyards

With an algorithm for computing the mean of a set of persistence diagrams in hand, we want to compute the mean of a set of time varying persistence diagrams. As with persistence diagrams, let us now consider the space of abstract vineyards as paths in persistence diagram space.

**Definition 28.** The space of abstract vineyards is

$$V = \{v : [0, 1] \to D_{p} \mid v \text{ continuous with respect to the metric } W_{2}[L_{2}]\}$$

with the metric $I[W_{2}[L_{2}]]$ from Definition 14.
Algorithm 3.1 Algorithm for computing the Fréchet Mean of a finite set of diagrams

**Input:** Persistence diagrams $X_1, \cdots, X_N$

**Output:** $Y$, a persistence diagram giving a local min of the Fréchet function

Choose one of the $X_i$ randomly, set $Y = X_i$

Initialize matching $G$

\[ G[y_j, X_i] = \text{the } x_k \in X_i \text{ matched with the point } y_j \in Y \]

stop = False

while stop == False do

for each diagram $X_i$ do

\[ P = \text{WassersteinPairing}(Y, X_i) \]

for each pair $(y_j, x_k) \in P$ do

Set $G[y_j, X_i] = x_k$

end for

end for

Initialize empty diagram $Y'$

for each point $y_j \in Y$ do

\[ y'_j = \text{mean}\{G[y_j, X_1], \cdots, G[y_j, X_N]\} \]

Add $y'_j$ to $Y'$

end for

if WassersteinPairing($Y, X_i$) = WassersteinPairing($Y', X_i$) $\forall i$ then

stop = True

end if

end while

return $Y$

Since we have a metric space, we can immediately define the Fréchet mean. However, as we will see, non-uniqueness of the mean in $D_p$ means that the pointwise mean we would like to construct is not necessarily continuous and therefore not in $V$. In Section 3.4.2, we will present a new definition for the mean of a set of diagrams which we will prove is continuous for pointwise computation on continuous vineyards.

### 3.4.1 Issues with extensions to Vineyards

Consider the example of Figure 3.5. Here we have two persistence diagrams overlaid: $d \square$ has square points 1 and 2, $d \bullet$ has circle points $a$ and $b$. Since the four points lie exactly on a square, the pairing to give the Wasserstein distance could either be $\{(a, 1), (b, 2)\}$ or $\{(a, 2), (b, 1)\}$. Thus there are two diagrams which give a minimum of the Fréchet function: the diagram with $u$ and $v$, or the diagram with $x$ and $y$. 

40
Figure 3.5: A counterexample to uniqueness of the Fréchet mean in $D_p$. Figure (a) shows two diagrams overlaid: $d_\square$ has points 1 and 2, $d_\bullet$ has points $a$ and $b$. Since the matching given by the Wasserstein distance is not unique, neither is the Fréchet mean. The two possible means are given in Figure (b): one has points $x$ and $y$, the other has points $u$ and $v$. See Section 3.4.1.

If two vineyards pass through this configuration, the mean of the vineyards constructed by finding the mean at each time will not be continuous. Consider for example two vineyards of two points each who start in the elongated configuration of Figure 3.6a and move along the dotted lines to the configuration in Figure 3.6b. At the bend of the dotted line, the points are at the corners of a square, so as in the example of Figure 3.5, there are two possible choices for the mean. One is close to the means from the previous times, and one is close to the means from the following times.

Since the obvious construction does not work, we must work harder to build our mean of vineyards.

3.4.2 The Mean as a Distribution

We would like to find some way around the issue created by the example of Figure 3.5. Intuitively, what we would like to do is create a mean which is in fact a distribution.
Figure 3.6: Two vineyards whose pointwise mean is not continuous. The two vineyards start at the configuration in Figure (a) and end in the configuration of Figure (b). The mean is continuous until the points get to the turn of the dotted line, where they form a square, and the mean jumps discontinuously.

In persistence diagram space, with the goal that this distribution is continuous when faced with a continuous vineyard.

In order to prove continuity later, the diagrams will be limited to \( S = S_{M,K} \), the set of diagrams in \( D_2 \) with at most \( K \) off-diagonal points, and all points \( x = (x_1, x_2) \) satisfy \( 0 \leq x_1, x_2 \leq M \). We will show that this set is compact using Theorem 22 and the comments following Definition 26.

**Lemma 29.** The set \( S_{M,K} \) is compact in \( D_2 \).

**Proof.** Since diagrams in \( S_{M,K} \) have finitely many points inside a finite triangle, the diagram farthest from \( d_\emptyset \) is the diagram which has all \( K \) points at the point \( (0, M) \) and is at distance \( \frac{KM}{\sqrt{2}} \). Using the triangle inequality, this implies that every diagram is within distance \( KM\sqrt{2} \) of any other diagram. Thus, \( S_{M,K} \) is bounded.

Because all diagrams have points contained in a finite triangle, the set is birth-death bounded and therefore trivially off-diagonally birth-death bounded.

Finally, assume that the set is not uniform. Then there is an \( \varepsilon > 0 \) such that for
all $\alpha > 0$, there is a $d_\alpha \in S$ such that $W_\rho(l_\alpha(d_\alpha), d_0) > \varepsilon$. If $|d_\alpha|$ is the number of off-diagonal points which are in $d_\alpha$ with $\|x - \Delta\| < \alpha$, then

$$\varepsilon^p < \sum_{x \in l_\alpha(d_\alpha)} \|x - \Delta\|^p < \sum_{x \in l_\alpha(d_\alpha)} \alpha^p = |d_\alpha| \cdot \alpha^p.$$  

Since $\varepsilon$ is fixed, as $\alpha \to 0$, $|d_\alpha| \to \infty$. This contradicts the fact that there are at most $K$ points in any $d \in S_{M,K}$, so the set must be uniform.

Consider the space $\mathcal{P}(S)$, the space of probability measures with finite second moment on $S \subset D_2$. This is of course a metric space with the standard probability Wasserstein distance $W_2[W_2[L_2]]$ where the outside $W_2$ is the Wasserstein distance of Definition 2 and the inside $W_2$ is the Wasserstein distance of Definition 1. We will create a probability distribution in $\mathcal{P}(S)$ by looking at distributions of possible matchings.

The Drawing Procedure

Let $X = \{X_1, \cdots, X_N\}$ be given diagrams in $S$. Sometimes, where appropriate, we will also use $X = \bigcup_i X_i$ to represent the set of off-diagonal points in all the $X_i$.

In order to determine a distribution on matchings, diagrams $X'_1, \cdots, X'_N$ are drawn which are close to the original given diagrams. Just how close is determined by a chosen parameter $\alpha$. One might wish to draw a random diagram from the ball of radius $\alpha$ around each $X_i$, however due to the diagonal, this is difficult to control in terms of points coming out of the diagonal. Instead, we randomly select a diagram $X'_i$ for $X_i$ with at most as many points, taking care that points near the diagonal might not have a point drawn for them at all.

Fix $\alpha > 0$ and a distribution $\eta_0$ on $\mathbb{R}^2$ whose support is contained in $B_\alpha(0) \subset \mathbb{R}^2$ and mean is 0. This distribution could be, for example, the uniform distribution.
Figure 3.7: The method for drawing points. For a point \( x \in X_i \) which is more than distance \( \alpha \) from the diagonal as at right, a point is drawn from the distribution \( \eta_x \), which has mean \( \alpha \) and is contained in the ball of radius \( \alpha \) centered at \( x \). This point is then added to the diagram \( X'_i \). For a point \( x \in X_i \) which is less than distance \( \alpha \) from the diagonal as at left, a point is still drawn from the distribution \( \eta_x \), however the point is only added to \( X'_i \) if it is inside the ball of radius \( \beta = \|x - \Delta\| \) centered at \( x \).

\[ \mathcal{U}(B_\alpha(0)) \] or the truncated normal distribution \( \mathcal{N}(0, \sigma^2, \alpha) \) which is just the portion of the standard normal distribution \( \mathcal{N}(0, \sigma^2) \) contained inside of \( B_\alpha(0) \) and normalized to have total mass 1. When this distribution is translated to have mean \( x \), denote it \( \eta_x \).

For each off diagonal point \( x \in X_i \) which is more than distance \( \alpha \) from the diagonal, draw a point from the distribution \( \eta_x \) and add it to the diagram \( X'_i \). For each off-diagonal point \( x \in X_i \) which is less than distance \( \alpha \) from the diagonal, draw a point \( x' \) from \( \eta_x \) and add it to the diagram \( X'_i \) only if it is contained in the ball of radius \( \beta = \|x - \Delta\| \) centered at \( x \). In this way, the probability that a point gets added to the diagram from one of these points close to the diagonal decreases as the distance to the diagonal decreases. See Figure 3.7.

Once these new diagrams \( X'_1, \ldots, X'_N \) have been drawn, the mean diagram can be computed via Algorithm 3.1. In particular, we need the matching used by the algorithm, and can discard the computed mean diagram. Since each point in \( X'_i \) is associated to a point in \( X_i \), we consider the matching used to compute the mean of \( \{X'_i\} \) to be the same as the matching using the corresponding points of \( X_i \). However,
since some points in the $X_i$ did not get corresponding points in $X'_i$, we can extend this to a full matching by adding the trivial selection for these points. That is, if a point $x \in X_i$ did not have a point in $X'_i$, we add the selection which chooses $x$ for $X_i$ and $\Delta$ for every other diagram.

In order to determine the distribution in $\mathcal{P}(S)$, the probability of a given draw returning a matching $G$ is associated to the diagram $\text{mean}_X(G) \in D_p$.

**Definition 30.** Let $X = \{X_1, \cdots, X_N\}$ and let $H$ be the random variable which gives the matching returned from a given draw. Then the mean distribution is defined to be

$$
\mu_X = \sum_G \mathbb{P}(H = G) \cdot \delta_{\text{mean}_X(G)}
$$

where the sum is over all possible matchings $G$ on the $X_i$s, $\mathbb{P}(H = G)$ is the probability of getting matching $G$, $\text{mean}_X(G) \in S$ is the mean diagram for the matching $G$, and $\delta_Y$ is the delta measure on the diagram $Y$.

This drawing procedure is clearer with an example. Consider the three overlaid diagrams in Figure 3.8a. Points are drawn in the ball of radius $\alpha$ centered at each point. Since $a, c, h, g, y, \text{ and } z$ are near the diagonal there is a chance that no point is drawn for them. In this particular draw, given in Figure 3.8b, no point is drawn for $a$ or $c$.

When computing the mean of the diagrams in Figure 3.8b, the matching used is

$$
\begin{pmatrix}
\begin{array}{ccc}
\ast & \square & \circ \\
1 & b' & x' \ f' \\
2 & \Delta & y' \ g' \\
3 & \Delta & \Delta \ h' \\
4 & \Delta & z' \ \Delta \\
\end{array}
\end{pmatrix}
$$

(3.4)
Figure 3.8: An example of corresponding matchings for a given draw. The original diagrams are $d_\star$, $d_\square$ and $d_\circ$ in Figure (a). A point is drawn near each point away from the diagonal, and points are drawn for some points near the diagonal to construct $d'_\star$, $d'_\square$ and $d'_\circ$ in Figure (b). The matching for the mean of these three diagrams is computed using Algorithm 3.1 and the associated matching is given in Equation 3.4. Then the matching is converted to a matching for $d_\star$, $d_\square$ and $d_\circ$ in Equation 3.5 and drawn in Figure (a).

So, to find the corresponding matching for the original diagrams, we replace each point with its corresponding point, and add in the trivial selection for the points that were not chosen:

$$
\begin{pmatrix}
1 & b & x & f \\
2 & \Delta & y & g \\
3 & \Delta & \Delta & h \\
4 & \Delta & z & \Delta \\
5 & a & \Delta & \Delta \\
6 & c & \Delta & \Delta \\
\end{pmatrix}
$$

(3.5)

Repetitions of the drawing procedure are tallied to determine a distribution on the matchings, and the mean distribution is returned.

This process is specifically tailored to draw at most one dot for each corresponding
point, so we can think of splitting up the single persistence diagram into many copies of $\mathbb{R}^2$, each with their own distribution, and then drawing one point from each of these spaces. Let $X = \bigcup_i X_i$ be the set of all off-diagonal points in all the $X_i$, and assume that $\omega \subset X$ is the set of points added to their respective diagrams in the draw. Then we give each of these points a copy of $\mathbb{R}^2$, with their associated distribution. In particular, if $x$ is more than $\alpha$ away from the diagonal, we just have the distribution $\eta_x$. If $x$ is less than $\alpha$ from the diagonal, we have the distribution $\eta'_x = \frac{1}{\eta_x(B_{\|x-\Delta\|}(x))}\eta_x\big|_{B_{\|x-\Delta\|}(x)}$ (3.6)

where $\eta_x(B_{\|x-\Delta\|}(x))$ gives the volume of the measure contained in the ball of radius $\|x-\Delta\|$ and $\eta_x\big|_{B_{\|x-\Delta\|}(x)}$ denotes $\eta_x$ restricted to the ball of radius $\|x-\Delta\|$. It is also useful to notice that when $x$ is more than $\alpha$ from the diagonal, $\eta_x(B_{\|x-\Delta\|}(x)) = 1$ and $\eta_x\big|_{B_{\|x-\Delta\|}(x)} = \eta_x$, so in this case $\eta'_x = \eta_x$.

Now consider a set $\omega \subset X$, the associated space $(\mathbb{R}^2)^{\omega}$, and a fixed matching $G$. We can treat $G$ as a completely combinatorial object, so we can look at the set of points in this space which return $G$. Call this set $U_{\omega,G} \subset (\mathbb{R}^2)^{\omega}$. Notice that this set is completely independent of the locations of the points from the original diagrams; it relies only on the combinatorial structure of $G$.

Let $\eta'_\omega = \prod_{x \in \omega} \eta'_x$ be the product measure on $(\mathbb{R}^2)^{\omega}$ and let $\mathcal{P}_\omega$ be a random variable which is 1 if exactly the points $\omega$ are chosen, and 0 otherwise. Then the probability that $G$ is the matching used assuming $\omega$ is chosen is

$$\mathbb{P}(\mathcal{H} = G \mid \mathcal{P}_\omega = 1) = \eta_\omega(U_{\omega,G}).$$

For example, say $X$ consists of two diagrams, $X_1$ and $X_2$. $X_1$ has a single off-diagonal point $x_1$, and $X_2$ has a single off diagonal point $x_2$. Then there are exactly
two possible matchings:

\[
G_1 = \frac{1}{2} \begin{pmatrix} X_1 & X_2 \\ x_1 & \Delta \\ \Delta & x_2 \end{pmatrix}, \quad G_2 = \begin{pmatrix} X_1 & X_2 \\ x_1 & x_2 \end{pmatrix}.
\]

Let \( \omega \) be all the points, so \( \omega = \{x_1, x_2\} \), then the space we are interested in is \((\mathbb{R}^2)^2\).

So \( U_{\omega,G_1} \) is the set of points which would rather be paired with the diagonal than with each other, thus

\[
U_{\omega,G_1} = \{(a, b) \in (\mathbb{R}^2)^2 \mid \|a - \Delta\|^2 + \|b - \Delta\|^2 \leq \|a - b\|^2\}.
\]

Likewise, \( U_{\omega,G_2} \) is the set of points where being paired to each other is better, so

\[
U_{\omega,G_2} = \{(a, b) \in (\mathbb{R}^2)^2 \mid \|a - b\|^2 \leq \|a - \Delta\|^2 + \|b - \Delta\|^2\}.
\]

For the purposes of determining the probability of \( G_1 \), we must take the locations of \( x_1 \) and \( x_2 \) into account. Now we are only interested in the \((a, b) \in U_{\omega,G_1}\) for which \( a \) is in the support of \( \eta'_{x_1} \) and \( b \) is in the support of \( \eta'_{x_2} \). So, the probability that \( G_1 \) is chosen assuming both points are picked is

\[
\eta'_{U_{\omega,G_1}} = (\eta'_{x_1} \times \eta'_{x_2})U_{\omega,G_1}.
\]

Likewise, the probability that \( G_2 \) is chosen assuming both points are picked is

\[
\eta'_{U_{\omega,G_2}} = (\eta'_{x_1} \times \eta'_{x_2})U_{\omega,G_2}.
\]

This construction is particularly useful since we can consider the same matching for two sets of diagrams which are close. Since \( U_{\omega,G} \) is independent of the actual diagrams, the only thing varying is the measure. This, as we show here, is continuous.

**Lemma 31.** The map
\[ \mathbb{R}^2 \rightarrow \mathcal{P}(\mathbb{R}^2) \]
\[ x \mapsto \eta'_x \]

is continuous.

**Proof.** Fix \( x \in \mathbb{R}^2 \) and let \( \varepsilon > 0 \) be given. Let \( \beta_x = \min\{\|x - \Delta\|, \alpha\} \), so it is the radius of the ball containing the support of \( \eta'_x \). First, consider all the measures in which we are interested centered at 0 in order to compare the amount of mass at associated points. For any \( \beta < \|x - \Delta\| \), we know that \( B_\beta(0) \subset B_{\beta_x}(0) \), so by inclusion of sets,

\[
\frac{\eta_0(B_\beta(0))}{\eta_0(B_{\beta_x}(0))}
\]

monotonically increases to 1 as \( \beta \) increases to \( \beta_x \). Thus, there is an \( r < \beta_x \) such that for all \( r < \beta < \beta_x \),

\[
\frac{\eta_0(B_\beta(0))}{\eta_0(B_{\beta_x}(0))} \geq 1 - \frac{\varepsilon}{6\alpha}.
\]

Likewise, for \( \beta > \beta_x \), \( B_{\beta_x}(0) \subset B_{\beta}(0) \). Then there is a \( \beta_x \leq R \leq \alpha \) such that if \( \beta_x < \beta < R \),

\[
\frac{\eta_0(B_{\beta_x}(0))}{\eta_0(B_\beta(0))} \geq 1 - \frac{\varepsilon}{6\alpha}.
\]

Let

\[ \delta < \min \left\{ \frac{\varepsilon}{2}, \alpha, \beta_x - r, R - \beta_x \right\} \]

and let \( y \) be given so that \( \|x - y\| < \delta \). Let \( \beta_y = \min\{\|y - \Delta\|, \alpha\} \), so that the ball of radius \( \beta_y \) contains the support of \( \eta'_y \). There are two cases to consider: either \( \beta_y \leq \beta_x \) or \( \beta_y > \beta_x \).

First, assume \( \beta_y \leq \beta_x \). Since \( \|x - y\| < \delta < \beta_x - r \leq \|x - \Delta\| - r \),

\[
\|y - \Delta\| > \|x - \Delta\| - \|x - y\| \\
> \|x - \Delta\| - (\|x - \Delta\| - r) \\
= r
\]

49
Figure 3.9: The picture for the cases of Lemma 31. The transportation plans in each case move as much mass as possible from one circle of smaller radius to the other, then bound the amount of mass contained in the region outside that smaller circle.

and thus $\beta_y > r$. By definition of $r$, this implies that

$$1 - \frac{\eta_0(B_{\beta_y}(0))}{\eta_0(B_{\beta_x}(0))} < \frac{\varepsilon}{6\alpha}.$$

Then consider the transportation plan which sends the mass from $\eta'_x$ which is contained in $B_{\beta_y}(x)$ to the associated points in $B_{\beta_y}(y)$. See Figure 3.9. The total amount of mass moved is

$$\eta'_x(B_{\beta_y}(y)) = \frac{\eta_x(B_{\beta_y}(x))}{\eta_x(B_{\beta_x}(x))} = \frac{\eta_0(B_{\beta_y}(0))}{\eta_0(B_{\beta_x}(0))}$$

and is moved a distance of $\|x - y\|$. The leftover mass is

$$1 - \frac{\eta_0(B_{\beta_y}(0))}{\eta_0(B_{\beta_x}(0))}$$
and moves at most distance $\beta_x + \beta_y + \|x - y\| \leq 3\alpha$. Thus,

$$W_2(\eta_x', \eta_y') \leq \left( \frac{\eta_0(B_{\beta_y}(0))}{\eta_0(B_{\beta_x}(0))} \right) \|x - y\| + \left( 1 - \frac{\eta_0(B_{\beta_y}(0))}{\eta_0(B_{\beta_x}(0))} \right) (3\alpha)$$

$$\leq 1 \cdot \frac{\varepsilon}{2} + \frac{\varepsilon}{6\alpha} \cdot 3\alpha$$

$$\leq \varepsilon$$

as desired.

The case where $\beta_y \leq \beta_x$ is extremely similar, except that our transportation plan moves the mass contained in $\eta_y(B_{\beta_y}(y))$ to the mass at $\eta_x(B_{\beta_x}(x))$. The total of mass moved in this plan is

$$\frac{\eta_0(B_{\beta_x}(0))}{\eta_0(B_{\beta_y}(0))}$$

and is again moved a distance of $\|x - y\|$. Since $\|x - y\| \leq R - \|x - \Delta\|$, 

$$\beta_y \leq \|y - \Delta\| \leq \|x - y\| + \|x - \Delta\| < R$$

so for the leftover mass,

$$1 - \frac{\eta_0(B_{\beta_x}(0))}{\eta_0(B_{\beta_y}(0))} \leq \frac{\varepsilon}{6\alpha}.$$ 

This leftover mass is moved at most distance $3\alpha$, so

$$W_2(\eta_x', \eta_y') \leq \left( \frac{\eta_0(B_{\beta_y}(0))}{\eta_0(B_{\beta_y}(0))} \right) \|x - y\| + \left( 1 - \frac{\eta_0(B_{\beta_x}(0))}{\eta_0(B_{\beta_y}(0))} \right) (3\alpha)$$

$$\leq 1 \cdot \frac{\varepsilon}{2} + \frac{\varepsilon}{6\alpha} \cdot 3\alpha$$

$$\leq \varepsilon.$$ 

\[\square\]

**Continuity for Sets of Vineyards**

The goal for the construction of this new mean was continuity for vineyards, so we prove that here. Let $\mu_V(t)$ be the distribution defined by the drawing procedure using
the set of diagrams \( \{V_1(t), \cdots, V_N(t)\} \) as in Definition 30. We will prove continuity for a geodesic vineyard, which implies continuity for piecewise geodesics.

**Theorem 32.** Let vineyards \( \{V_1, \cdots, V_N\} \) with \( V_i : [0, 1] \to \mathcal{P}(S_{M,K}) \) be continuous geodesics defined by the matchings \( \varphi_i : V_i(0) \to V_i(1) \). Then the map

\[
[0, 1] \longrightarrow \mathcal{P}(S_{M,K})
\]

\[
t \longmapsto \mu_{V(t)}
\]

is continuous.

The main idea of the proof is to create a bijection from most of the matchings of the \( \{V_i(0)\} \) to most of the matchings of the \( \{V_i(1)\} \) in such a way that the probability of getting the corresponding matchings is similar and the associated mean diagrams are close. This allows for a transportation for the Wasserstein distance which moves most of the mass a short way, and we can then argue that although the rest of the mass could be moved a long distance, its weight is negligible. Equation 3.9 encodes this information.

The proof will be done in several steps. Initially fixing a \( t \) gives a particular set of diagrams. First, we determine how far along each vineyard we can travel without allowing the probability of getting a given matching to change too much. This allows us to choose a \( \delta \) and fix a time \( t + \tau \) to obtain a second set of diagrams. With these two sets of diagrams fixed, we can construct the correspondence between matchings, and bound each of the pieces of Equation 3.9.

**Proof of Theorem 32.** Let \( \varepsilon > 0 \) be given and fix \( t \in [0, 1] \). This gives a set of diagrams \( V_i(t), \cdots, V_N(t) \) in \( S_{M,K} \). We need to find a \( \delta \) so that for any \( t' \) with \( |t - t'| < \delta \), \( W_2(\mu_{V(t)}, \mu_{V(t')}) < \varepsilon \). Recall that the distance between any diagrams in \( S_{M,K} \) is at most \( \overline{M} = \sqrt{2} \cdot MK \).

Let \( H_{V(t')} \) be the random variable giving the matching used for the mean of a draw of the diagrams \( \{V_i(t')\} \). First, we will show that \( \mathbb{P}(H_{V(t)} = G) \) is a continuous
function of $t$ for any fixed matching $G$, and thus there is a $T$ such that for all $t'$ with $|t - t'| < T$,  
\[
\sum_{G} \left| \mathbb{P}(\mathcal{H}_{V(t)} = G) - \mathbb{P}(\mathcal{H}_{V(t')} = G) \right| < \frac{\varepsilon}{3M}.
\]  

(3.7)

For this to make sense, we must be clear about what it means to have the same matching for a different time. Given a matching $G$ at time $t$, there is a matching at time $t'$ which replaces each $x \in V_i(t)$ with its pair under the $\varphi_i$ matching, $\varphi_i(x) \in V_i(t')$. Any additional points $y \in V_i(t')$ for which $\varphi^{-1}_i(y) = \Delta$ are given the trivial selection.

Given a particular matching $G$ for the set $\{V_1(t), \ldots, V_N(t)\}$, consider the probability that a draw returns the associated matching for the set $\{V_i(t+\tau)\}$. Let $P_x$ be the binary random variable which is 1 if the drawing process adds a point associated to $x$, and 0 otherwise. In order to determine $\mathbb{P}(\mathcal{H}_{V(t+\tau)} = G)$, consider all possible subsets of points which had points drawn. Given a subset of the points $\omega \subset V(t+\tau)$, let $P_\omega$ be the random variable which is 1 if the exactly the points in $\omega$ were drawn. Like matchings, these $\omega$ can be associated for different times by replacing the points in $\omega$ at time $t$ with their corresponding points at time $t + \tau$. The probability that exactly these points were drawn for at most the subset $\omega$ at time $t + \tau$ is

\[
\mathbb{P}(P_\omega = 1) = \prod_{x(t+\tau) \in \omega} \mathbb{P}(P_{x(t+\tau)} = 1) \cdot \prod_{x(t+\tau) \notin \omega} \left(1 - \mathbb{P}(P_x = 1)\right)
\]

\[
= \prod_{x(t) \in \omega} \eta_{x(t)}(B_{\|x-\Delta\|}(x(t + \tau))) \cdot \prod_{x(t+\tau) \notin \omega} \left(1 - \eta_{x(t+\tau)}(B_{\|x(t+\tau)-\Delta\|}(x(t + \tau)))\right)
\]

Since the $x(t + \tau)$ move continuously as a function of $t$ and the $\eta_{x(t+\tau)}$ translate with the moving $x(t + \tau)$, this probability is continuous as a function of $t$.

Let $|\omega|$ be the number of points in $\omega$. Associate each point in $\omega$ to a copy of $\mathbb{R}^2$. Then the particular set of draws with $P_\omega = 1$ where the matching used is $G$ can be
thought of as a subset of $(\mathbb{R}^2)^{|\omega|}$. Let
\[ U_{G,\omega} = \{(a_1, \ldots, a_K) \in (\mathbb{R}^2)^{|\omega|} \mid \mathcal{H}_A = G\} \]
where $A$ is the set of diagrams split up according to the location of the corresponding points in $\omega$, and $\mathcal{H}_A$ gives the matching used to compute the mean of these diagrams. For this set, the matching used is the same as $G$ if for every match in $G$ there is a match in $\mathcal{H}_A$ using points from the copies of $\mathbb{R}^2$ corresponding to the points in the match. In particular, this set is not dependent on the diagrams, only on the combinatorial structure of $G$.

Let $\eta'_{\omega,t+\tau}$ be the product measure $\eta'_{\omega,t+\tau} = \prod_{x(t+\tau) \in \omega} \eta'_x$ with $\eta'_x$ defined as in Equation 3.6 and which lines up $x(t+\tau) \in \omega$ with its associated copy of $\mathbb{R}^2$. Assuming that the points in $\omega$ are drawn for the diagrams at time $t+\tau$,
\[ \mathbb{P}(\mathcal{H}_V(t+\tau) = G \mid \mathcal{P}_\omega = 1) = \eta'_{\omega,t+\tau}(U_{G}(\omega)). \]
Since by Lemma 31 the $\eta'_x(t+\tau)$ move continuously with $\tau$, this probability is continuous.

Thus, the probability that matching $G$ is chosen for the set of diagrams given by a fixed $t$ is
\[ \mathbb{P}(\mathcal{H}_V(t+\tau) = G) = \sum_{\omega} \mathbb{P}(\mathcal{P}_\omega = 1) \cdot \eta_{\omega,t+\tau}(U_{G}(\omega)) \]
where the sum is over all subsets of the points $V(t+\tau) = \bigcup_i V_i(t+\tau)$. Since this probability is sums and products of continuous functions of $\tau$, it is continuous in $\tau$ for any $G$. Therefore, there is a $\delta_1$ such that that if $\tau < \delta_1$,
\[ \sum_G |\mathbb{P}(\mathcal{H}_X = G) - \mathbb{P}(\mathcal{H}_X(t) = G)| < \frac{\varepsilon}{3M}. \quad (3.8) \]

Since all of the vineyards are continuous, there is a $\delta_2$ such that
\[ \sum_i W_2(V_i(t), V_i(t'))^2 \leq \frac{\varepsilon}{3M} \]
for all $|t - t'| < \delta_2$. There is also a $\delta_3$ such that

$$\sum_i W_2(V_i(t), V_i(t'))^2 \leq \frac{\alpha^2 \varepsilon}{3M}.$$ 

Fix $\delta = \min\{\delta_1, \delta_2, \delta_3\}$ and let $t'$ be given so that $|t - t'| < \delta$. For the sake of notation, let $X_i = V_i(t)$ for each $i$, and let $Y_i = V_i(t')$ for each $i$. Let $\varphi_i$ be the matching for the Wasserstein distance $W_2(X_i, Y_i)$, so $\varphi_i : X_i \rightarrow Y_i$.

Let $\tilde{X}_i$ be the diagram with all points in $X_i$ such that $\varphi_i(x) \neq \Delta$. Likewise, let $\tilde{Y}_i$ be the diagram with all points in $Y_i$ such that $\varphi_i^{-1}(y) \neq \Delta$. Let $G_X$ be the set of matchings of the diagrams $X_1, \ldots, X_N$ and $G_{\tilde{X}}$ be the set of matchings for the diagrams $\tilde{X}_1, \ldots, \tilde{X}_N$. The definitions for $G_Y$ and $G_{\tilde{Y}}$ are similar.

There is a bijection $G_{\tilde{X}} \leftrightarrow G_{\tilde{Y}}$ which takes each entry $G(i, X_j)$ in $G \in G_{\tilde{X}}$ to $\varphi_j[G(i, X_j)]$. Since all ambiguity with the diagonal has been removed, this is well-defined. There is also an injection $G_{\tilde{X}} \hookrightarrow G_X$ which takes a matching $G \in G_{\tilde{X}}$ and maps it to the matching $G' \in G_X$ which has all the same selections along with the trivial selection for each unused point $x \in X_i \setminus \tilde{X}_i$. Call this map $i_{\tilde{X}}$. Likewise, there is an injection $i_{\tilde{Y}} : G_{\tilde{Y}} \hookrightarrow G_Y$.

Putting these maps together gives the diagram

$$
\begin{array}{ccc}
G_{\tilde{X}} & \rightarrow & G_{\tilde{Y}} \\
\downarrow i_{\tilde{X}} & & \downarrow i_{\tilde{Y}} \\
G_X & \longrightarrow & G_Y
\end{array}
$$

This diagram induces a bijection $\text{Im } (i_{\tilde{X}}) \rightarrow \text{Im } (i_{\tilde{Y}})$. Any matching in either $\text{Im } (i_{\tilde{X}})$ or $\text{Im } (i_{\tilde{Y}})$ is called paired, and its pair is the image under the given bijection. This means the set of matchings can be split into those that are paired and those that are unpaired.

In order to understand the bound we will put on $W_p(\mu_X, \mu_Y)$, it is best to use the earth mover analogy for Wasserstein distance. Each distribution is thought of as dirt
piles, and the cost of moving a dirt pile is the amount of dirt times the distance it must be moved. Under the aforementioned pairing of matchings, we can move most of the dirt from the diagram associated to one matching to the diagram matched to its pair. Then we will prove that the $\delta$ bound makes the amount of leftover dirt small enough so that the cost to move it anywhere is negligible.

In particular,

$$W_p(\mu_X, \mu_Y) \leq \sum_{(G,H) \in G_X \times G_Y \text{ Paired}} \min\{\mathbb{P}(\mathcal{H}_X = G), \mathbb{P}(\mathcal{H}_Y = H)\} \cdot W_p(\text{mean}_X(G), \text{mean}_Y(H))$$

$$+ \sum_{(G,H) \in G_X \times G_Y \text{ Paired}} |\mathbb{P}(\mathcal{H}_X = G) - \mathbb{P}(\mathcal{H}_Y = H)| \cdot \bar{M}$$

(3.9)

$$+ \sum_{G \in G_X \text{ unpaired}} |\mathbb{P}(\mathcal{H}_X = G)| \cdot \bar{M} + \sum_{H \in G_Y \text{ unpaired}} |\mathbb{P}(\mathcal{H}_Y = H)| \cdot \bar{M}$$

where $\bar{M}$ is the maximum distance between any two diagrams in $S_{K,M}$. In this equation, the first term corresponds to moving as much dirt as possible from mean_X(G) to mean_Y(H), the second term accounts for the leftover dirt from this motion, and the last two terms are the amount of dirt on matchings which do not have pairs.

In order to bound the first term in Equation 3.9, fix a $G \in G_X$ that is paired with $H \in G_Y$. Consider a single nontrivial selection $m = (x_1, x_2, \cdots, x_N)$ of $G$ where at most $N - 2$ of the $x_i$ could be $\Delta$. Since this is a nontrivial selection and it is in a paired grouping, for all off-diagonal $x_i$, $\varphi_i(x_i)$ is off-diagonal. Because $G$ is paired, there must be an associated selection in $H$ which is $m' = (y_1, \cdots, y_N)$ with $y_i = \varphi_i^{-1}(x_i)$ and $y_i$ is $\Delta$ if and only if $x_i$ is $\Delta$.

Consider the mean of the selections $m$ and $m'$ in rotated coordinates with axes $(1/\sqrt{2}, 1/\sqrt{2})$ and $(-1/\sqrt{2}, 1/\sqrt{2})$. Let $x_{\sigma(1)}, \cdots, x_{\sigma(k)}$ and $y_{\tau(1)}, \cdots, y_{\tau(k)}$ be the off-diagonal points, which leaves $N - k$ copies of the diagonal. Writing $x_{\sigma(i)} = (a_i, b_i)$
and \(y_{\tau(i)} = (u_i, v_i)\) in these coordinates, we have

\[
\text{mean}_X(m) = \left( \frac{1}{k} \sum_{i=1}^{k} a_i, \frac{1}{N} \sum_{i=1}^{k} b_i \right),
\]

\[
\text{mean}_Y(m') = \left( \frac{1}{k} \sum_{i=1}^{k} u_i, \frac{1}{N} \sum_{i=1}^{k} v_i \right).
\]

Then using Jensen’s inequality,

\[
\|\text{mean}_X(m) - \text{mean}_Y(m')\|^2 = \left( \frac{1}{k} \sum_{i=1}^{k} a_i - \frac{1}{k} \sum_{i=1}^{k} u_i \right)^2 + \left( \frac{1}{k} \sum_{i=1}^{k} b_i - \frac{1}{k} \sum_{i=1}^{k} v_i \right)^2
\]

\[
= \frac{1}{k^2} \left( \sum_{i=1}^{k} (a_i - u_i) \right)^2 + \left( \sum_{i=1}^{k} (y_i - v_i) \right)^2
\]

\[
\leq \left( \sum_{i=1}^{k} (a_i - u_i) \right)^2 + \left( \sum_{i=1}^{k} (y_i - v_i) \right)^2
\]

\[
\leq \left( \sum_{i=1}^{k} (a_i - u_i)^2 + \sum_{i=1}^{k} (y_i - v_i)^2 \right)
\]

\[
= \sum_{x_i \neq \Delta} \|x_i - \varphi_i(x_i)\|^2.
\]

For each trivial selection \(m_x\) for the off-diagonal point with rotated coordinates \(x = (a, b) \in X_i\), the mean of the selection is

\[
\text{mean}(m_x) = (a, \frac{b}{N})
\]

so

\[
\|\text{mean}_X(m_x) - \Delta\| = \frac{b}{N}.
\]

Since \(b = \|x - \Delta\|\), this implies that

\[
\|\text{mean}_X(m_x) - \Delta\| = \frac{1}{N} \|x - \Delta\| \leq \|x - \Delta\|.
\]

57
The equation is the same for any \( y = (a, b) \in Y \). Therefore,

\[
W_2(\text{mean}_X(G) - \text{mean}_Y(H))^2 \leq \sum_{\text{Nontrivial paired} \ m \in G, m' \in H} \|\text{mean}_X(m) - \text{mean}_Y(m')\|^2 \\
+ \sum_{\text{Trivial} \ m \in G} \|\text{mean}_X(m) - \Delta\|^2 \\
+ \sum_{\text{Trivial} \ m' \in H} \|\Delta - \text{mean}_Y(m')\|^2 \\
\leq \sum_{\text{Nontrivial} \ x \in m} \sum_{m \in G, x \neq \Delta} \|x_i - \varphi_i(x_i)\|^2 \\
+ \sum_{m \in G} \|x_i - \Delta\|^2 + \sum_{m \in G} \|\Delta - y_i\|^2.
\]

Since every \( x_i \) and \( y_i \) appears exactly once in this sum,

\[
W_2(\text{mean}_X(G) - \text{mean}_Y(H))^2 \leq \sum_{i} \sum_{x \in X_i} \|x - \varphi_i(x)\|^2 \\
= \sum_{i} W_2(X_i, Y_i)^2
\]

so \( \delta \leq \delta_2 \) implies

\[
W_2(\text{mean}_X(G) - \text{mean}_Y(G')) \leq \sum_{i} W_2(X_i, Y_i)^2 \\
\leq \frac{\varepsilon}{3M}.
\]

(3.10)

To bound the last term in Equation 3.9, fix a \( z \in X_i \) with the property \( \varphi_i(z) = \Delta \). Let \( \text{sel}_G(z) \) denote the selection in \( G \) which contains \( z \) and let \( G_z \) be the set

\[
G_z = \{G \in G_X \mid \text{sel}_G(z) \neq \text{trivial selection of } z \}.
\]

If, for a particular draw, \( G \in G_z \) was chosen, that set of diagrams must have drawn
a point for \( z \). So,
\[
\sum_{G \in \mathcal{G}_X \text{ unpaired}} \mathbb{P}(\mathcal{H}_X = G) \leq \sum_{\tilde{z} \in \mathcal{Z}} \sum_{\mathcal{G}_Z} \mathbb{P}(\mathcal{H}_X = G)
\]
\[
\leq \sum_{\tilde{z}} \mathbb{P}(\mathcal{P}_z = 1)
\]
\[
\leq \sum_{\tilde{z}} \frac{\|z - \Delta\|^2}{\alpha^2}
\]
\[
\leq \frac{1}{\alpha^2} \sum_i \sum_{\tilde{z} \in \mathcal{Z}_i} \|z - \Delta\|^2.
\]

Likewise,
\[
\sum_{H \in \mathcal{G}_Y \text{ unpaired}} \mathbb{P}(\mathcal{H}_Y = H) \leq \frac{1}{\alpha^2} \sum_i \sum_{\tilde{z} \in \mathcal{Z}_i} \|z - \Delta\|^2
\]
so because \( \delta < \delta_3 \),
\[
\sum_{G \in \mathcal{G}_X \text{ unpaired}} \mathbb{P}(\mathcal{H}_X = G) + \sum_{H \in \mathcal{G}_Y \text{ unpaired}} \mathbb{P}(\mathcal{H}_Y = H) \leq \frac{1}{\alpha^2} \sum_i W_2(X_i, Y_i)^2
\]
\[
\leq \frac{\varepsilon}{3M}.
\]

Therefore putting together Equations 3.8, 3.10, 3.11, and 3.9, we have
\[
W_2(\mu_X, \mu_Y) \leq \varepsilon
\]
as required.

\[\square\]

Although this proof only shows continuity for geodesic vineyards, we have an obvious corollary for piecewise geodesics.

**Corollary 33.** Given piecewise geodesic vineyards \( V_1, \cdots, V_N \), the map
\[ [0,1] \rightarrow \mathcal{P}(S_{M,K}) \]
\[ t \mapsto \mu_{\mathcal{V}(t)} \]

is continuous.

3.5 Conclusions and Future Work

Computationally, Corollary 33 is exactly what is necessary. Vineyards computed from finite point clouds have finitely many off-diagonal points and therefore also lie inside a bounded region in \( \mathbb{R}^2 \). They will also always be given at finitely many time points, so we are implicitly assuming that the vineyard is piecewise linear.

Although we are computationally satisfied with Corollary 33, it seems reasonable to conjecture that the conclusion is much stronger than the proof presented here.

**Conjecture 34.** The map

\[
S_{M,K} \times \cdots \times S_{M,K} \rightarrow \mathcal{P}(S_{M,K})
\]
\[ X_1, \cdots, X_N \mapsto \mu_X \]

is continuous.

This conjecture would be quite powerful since it would allow for continuity not only in the case of single-parameter paths, but in multi-parameter families in \( S \).

This construction can also be forced to return a single element by choosing the mean diagram with the highest probability. With the exception of time points where the matchings switch, these single elements will be continuous and therefore will return a piecewise continuous vineyard. Although this does not give as complete a description as the probability distribution, it is much easier to visualize and may, therefore, be better for presenting results of applications to non-mathematicians. It would be interesting to see if this highest probability persistence diagram always corresponds to the Fréchet mean in the cases where the Fréchet mean is unique.
We would also like to work towards a better understanding of the space $V_p$. Since the best stability results arise from bottleneck distance, we were able to show that the space of vineyards with the integrated bottleneck distance is stable. However, it is not as obvious to extend stability results for general $W_p$ metrics as in Cohen-Steiner et al. (2010) since these results involve a much more complicated constant based on the underlying space.
Topology enters the study of sensor networks when we consider questions like coverage. When does a set of sensors effectively monitor a region and when are there gaps? Phrasing this geometrically, we start with a set of sensors $\chi$ in a domain $\Delta \subset \mathbb{R}^2$ where each can detect objects in a circular region of fixed radius $r_c$, and we ask if the union of these discs covers all of $\Delta$. This problem has been studied quite a bit, but previous to de Silva and Ghrist (2006), most work fell into one of two groups - approaches that utilized geometric analysis to obtain an exact answer and those that sought a non-deterministic approximation but assumed significant capabilities of the sensors. For a survey of the literature, see Yick et al. (2008).

The former approach requires a great deal of prior knowledge about the geometry of the domain and the exact location of the sensors, or at least exact distances for every pair of sensors. The latter does not require this exactness, but often requires a uniform distribution of nodes or a high level of intelligence in the sensors. The main contribution of de Silva and Ghrist (2006) was a criterion for coverage that requires none of these things.

In this chapter, we take a middle ground and address the question of computing
the probability of failure of the criterion of de Silva and Ghrist (2006) given the probability of failure of each sensor. We show that computing the probability of failure for a generalized set of complexes is NP-hard, but we give an algorithm which can be used to solve small instances of the problem, and an alternative, dynamic algorithm to give an early warning of potential failure.

This chapter represents joint work with Michael Shapiro and John Harer.

Outline. The chapter is organized as follows. In Section 4.1, we summarize the problem and results of de Silva and Ghrist (2006). Section 4.2 adds the assumption that sensors have a probability of failure, and Section 4.3 discusses the complexity issues of determining the probability that there is coverage of the domain. Section 4.4 presents a deterministic algorithm for those times when the set of sensors is small enough, and Section 4.5 gives a dynamic algorithm for use when the set of sensors is too large.

4.1 The Coverage Criterion

Working in a simply-connected domain in the plane, suppose that we have a set of sensors with a fixed radius of coverage. Our goal is check that these sensors cover the whole domain. What makes this a challenging problem is that we do not assume that we know the locations of the sensors. This means that standard geometric techniques are not applicable. Instead we turn to topology to answer the coverage question by building the Rips complex on the set of sensors, thought of as points in the plane. We can then use homology to check for holes in the coverage.

4.1.1 The Standard Formulation of the dS-G Criterion

Let \( \chi \) be the set of points corresponding to the location of the set of sensors in a compact connected domain \( \Delta \subset \mathbb{R}^2 \) which has a piecewise linear boundary. Suppose
that each sensor has a fixed coverage radius $r_c > 0$. The question is whether every point in $\Delta$ lies within distance $r_c$ of some sensor in $\chi$. We do not use the distance $r_c$ to build the Rips complex, instead we add an additional capability to each sensor. Let $r_b > 0$ be fixed, with $r_b \leq \sqrt{3}r_c$ for technical reasons. Each sensor is given a unique identification number to broadcast. If another node is within distance $r_b$, it can hear the signal and identify the ID number, but it has no information about the location of the broadcaster. In particular, it does not know its direction or its exact distance, only that that distance is less than $r_b$. Whenever two sensors can hear each others’ identification number, an edge is placed in the Rips complex. Higher dimensional simplices are then added when all of their faces are already there.

The boundary of the domain $\Delta$ is taken to be piecewise linear with a sensor at each of its vertices. The boundary is called the fence, and each node in the fence knows the identification number of its two fence neighbors, both of which are within distance $r_b$.

Summarizing, following de Silva and Ghrist (2006), the assumptions are:

1. Nodes $\chi$ broadcast their unique ID numbers. Each node can detect the identity of any node within broadcast radius $r_b$.

2. Nodes have radially symmetric covering domains of cover radius $r_c \geq r_b/\sqrt{3}$.

3. Nodes lie in a compact connected domain $\Delta \subset \mathbb{R}^2$ whose boundary $\partial\Delta$ is connected and piecewise-linear with vertices marked fence nodes $\chi_f$. Non-fence nodes are called interior nodes, and denoted $\chi_{int}$.

4. Fence nodes $\chi_f$ are ordered cyclically and each $v \in \chi_f$ knows the identities of its two neighbors on $\partial\Delta$. These neighbors both lie within distance $r_b$ of $v$.

Based on this information, build the Rips complex $\mathcal{R}$ with the fence $\mathcal{F}$ as a subcomplex. With this setup, de Silva and Ghrist in (de Silva and Ghrist, 2006) give their
controlled boundary criterion for coverage:

**Theorem 35** (de Silva/Ghrist Criterion (dS-G)). If there is a nontrivial element of the relative homology group \( H_2(\mathcal{R}, \mathcal{F}) \) which maps to a nonzero class under the connecting homomorphism \( H_2(\mathcal{R}, \mathcal{F}) \to H_1(\mathcal{F}) \), then the union of the disks of radius \( r_c \) about the nodes contains all of \( \Delta \).

The class \([\alpha] \in H_2(\mathcal{R}, \mathcal{F})\) is called **fundamental** if it satisfies the criterion of Theorem 35, but we stress that when there is such an element it is not necessarily unique. The term **absolute cycle** will be used for a class in \( H_2(\mathcal{R}, \mathcal{F}) \) that comes from \( H_2(\mathcal{R}) \), which is equivalent to saying that it maps to 0 under the connecting homomorphism.

The assumption that \( r_c \geq r_b / \sqrt{3} \) is required to compensate for the fact that the Rips complex does not accurately reflect the topology of the cover. While this bound promises that holes in the cover appear also as holes in the Rips complex, we can still create examples where phantom holes appear in the Rips complex even though no hole exists in the cover itself. In a perfect world, this theory would be built on Čech complexes, however the lack of location data for the nodes makes this method impossible.

**4.1.2 Alternative Formulations of the dS-G Criterion**

It should be noted that the dS-G criterion can be equivalently formulated in terms of absolute homology rather than relative.

**Lemma 36.** The following are equivalent:

1. \((\mathcal{R}, \mathcal{F})\) passes the dS-G criterion.

2. There exists a 2-chain in \( \mathcal{R} \) whose boundary is equal to the fence cycle.

3. The fence cycle is 0 in \( H_1(\mathcal{R}) \).
Proof. If \((\mathcal{R}, \mathcal{F})\) passes the dS-G criterion, there is a class \(\alpha \in H_2(\mathcal{R}, \mathcal{F})\) whose boundary is nontrivial in \(H_1(\mathcal{F})\). The boundary of the chain \(c\) representing the class \(\alpha\) must be contained in \(\mathcal{F}\) since it is in \(H_2(\mathcal{R}, \mathcal{F})\). Since the class associated to \(\partial c\) in \(H_1(\mathcal{F})\) is the generator, it must be equal to the fence cycle.

If we have a 2-chain in \(\mathcal{R}\) whose boundary is equal to the fence cycle, then the fence cycle must be 0 in \(H_1(\mathcal{R})\) by definition.

If the fence cycle is 0 in \(H_1(\mathcal{R})\), then we can use the portion of the exact sequence

\[ H_2(\mathcal{R}, \mathcal{F}) \rightarrow H_1(\mathcal{F}) \rightarrow H_1(\mathcal{R}). \]

The generator of \(H_1(\mathcal{F})\) must map to 0 in \(H_1(\mathcal{R})\) and therefore there is an \(\alpha \in H_2(\mathcal{R}, \mathcal{F})\) which maps to the generator of \(H_1(\mathcal{F})\). Hence, \((\mathcal{R}, \mathcal{F})\) passes the dS-G criterion.

While these alternative versions are useful for both intuition and proofs, we will continue to use the relative homology statement of the dS-G criterion whenever possible.

4.2 Sensor Failure

4.2.1 Relative Homology Version

Over time, sensors have a likelihood of failure which increases the longer the system is in place, caused perhaps by malicious actions, environmental conditions or mechanical failure. As nodes fail, there are two possible effects on the coverage: either the death of a subset of nodes creates a hole in the Rips complex, or the removal of the nodes does not affect the existence of a fundamental class. Once again, we emphasize that we are specifically not looking for the probability of failure of the cover over time, just the failure of the dS-G criterion. For this reason, we also assume that only interior nodes can fail. The loss of a fence node causes instant failure of the dS-G criterion, so there is nothing to check in this case.
Figure 4.1: A small scale example of the Rips complex $\mathcal{R}$ built from a set of points in the plane. The outer ring of nodes labeled with letters is the fence $\mathcal{F}$.

Let us start with a Rips complex pair $(\mathcal{R}, \mathcal{F})$ built from a set of nodes $\chi$. At time $t = 0$, we assume we have a fundamental class $[\alpha] \in H_2(\mathcal{R}, \mathcal{F})$. If a set of interior sensors $B \subset \chi_{int}$ fails, any simplex in $\mathcal{R}$ that has a vertex in the set $B$ is lost. Therefore this subcomplex, $\mathcal{R}_B$, can be thought of as the largest subcomplex of $\mathcal{R}$ that has $\chi - B$ as its vertices. We could then determine whether $\mathcal{R}_B$ fails the dS-G criterion by looking for a fundamental class in $H_2(\mathcal{R}_B, \mathcal{F})$, but this is a very narrow view of the problem. Much more information is available in a filtration that we will now construct. Note that it will contain $\mathcal{R}_B$ as one of its subcomplexes.

Let $|\chi_{int}| = n$. Order the nodes so that $B = \{v_1, \cdots, v_k\}$ and $\chi_{int} - B = \{v_{k+1}, \cdots, v_n\}$. Let $V_i = \{v_1, \cdots, v_i\}$ so that $V_k = B$, and let $\mathcal{R}_i = \mathcal{R}_{V_i}$ be the maximal subcomplex of $\mathcal{R}$ with vertices $\chi - V_i$. Then the filtration

$$\mathcal{F} = \mathcal{R}_n \subset \mathcal{R}_{n-1} \subset \cdots \subset \mathcal{R}_1 \subset \mathcal{R}_0 = \mathcal{R}$$

induces maps on relative homology

$$0 \to H(\mathcal{R}_{n-1}, \mathcal{F}) \to \cdots \to H(\mathcal{R}_1, \mathcal{F}) \to H(\mathcal{R}, \mathcal{F})$$

An example of the filtration in Equation 4.1 is illustrated in Figure 4.1.

4.2.2 Absolute Homology Version

As with the dS-G criterion itself, sensor failure can also be thought of in terms of absolute homology. The filtration of Equation 4.1 also induces maps on absolute homology:

$$0 \to H^0(\mathcal{R}_{n-1}, \mathcal{F}) \to \cdots \to H^0(\mathcal{R}_1, \mathcal{F}) \to H^0(\mathcal{R}, \mathcal{F})$$
homology
\[ H(F) \rightarrow H(\mathcal{R}_{n-1}) \rightarrow \cdots \rightarrow H(\mathcal{R}_1) \rightarrow H(\mathcal{R}). \] (4.3)

In this context, it is immediate from Lemma 36.2 that \((\mathcal{R}_i, \mathcal{F})\) passes the criterion when the generator of \(H(F)\) dies under the maps of Equation 4.3. For example, the 2-step filtration \(F \rightarrow \mathcal{R}\) passes the criterion if \(H(F) \rightarrow H(\mathcal{R})\) is surjective.

4.2.3 Cake or Death

Intuitively, we expect that discovering a fundamental element at any point in the sequence in Equation 4.2 implies that there is a fundamental element in any subsequent group.

**Lemma 37.** Let \(A \subset B\) be subsets of \(\chi_{\text{int}}\). Then if \([\beta] \in H_2(\mathcal{R}_B, \mathcal{F})\) is fundamental, its image under the map

\[ H_2(\mathcal{R}_B, \mathcal{F}) \xrightarrow{i_*} H_2(\mathcal{R}_A, \mathcal{F})\]

is also fundamental.

**Proof.** The lemma is immediate from Lemma 36.2 since the boundary of a 2-chain is the same whether viewed in \(\mathcal{R}_B\) or \(\mathcal{R}_A\). \(\square\)

This lemma shows that if \(\mathcal{R}_B\) passes the dS-G criterion, then \(\mathcal{R}_A\) passes the dS-G criterion for all \(A \subset B\). But it also shows that if \(\mathcal{R}_B\) fails the dS-G criterion, then \(\mathcal{R}_A\) fails the dS-G criterion for all \(A \supset B\). Thus if \(B\) is a death set, and \(B \subset A\), Lemma 37 implies that \(A\) is also a death set. This leads us to make the following definitions:

**Definition 38.** A set \(B \subset \chi_{\text{int}}\) whose removal causes failure of the dS-G criterion is called a **death set**. A death set \(B \subset \chi_{\text{int}}\) is a **minimal death set** if no subset of \(B\) is itself a death set.\(^1\)

\(^1\) See Izzard and Jordan (1999).
Definition 39. If the removal of $B$ does not cause failure, we call $B$ a cake set. A cake set $B \subset \chi_{\text{int}}$ is a maximal cake set if no superset of $B$ is also a cake set.

As we will show in section 4.4.2, minimal death sets are directly related to the failure of the dS-G criterion. However, we first look at the issues arising from the complexity of the problem.

4.3 Complexity Issues

The first issue to address is whether computing the probability of failure is computationally complex. In this section, we will show that a generalization of the problem is NP-hard, specifically #P-hard. The subproblem arising from points in the plane remains open.

4.3.1 Use of the 2-skeleton

A simplifying step is to work with the 2-skeleton $R^2$ of $R$ rather than the full complex. To justify this, notice that passing to the 2-skeleton does not affect our observance of the dS-G criterion:

Lemma 40. The dS-G criterion is satisfied for $R$ if and only if it is satisfied for $R^2$, where $R^2$ is the 2-skeleton.

Proof. This lemma is immediate in view of Lemma 36.2 since a 2-chain which spans the fence cycle is contained in the 2-skeleton of $R$.

This lemma implies that the sets of death sets, minimal death sets, cake sets, and maximal cake sets are equivalent to their counterparts when computed in the 2-skeleton. It also implies that the probability of failure of the dS-G criterion in the full Rips complex and the probability of failure of the dS-G criterion in the 2-skeleton are the same. And lastly, in $R^2$ there is exactly one cycle representing each homology
class \((Z_2 = H_2)\). For these reasons we will simplify notation and write \(R\) for the 2-skeleton of the Rips complex for the remainder of the paper.

### 4.3.2 \#P-Hard

The class of problems defined as \#P-complete was introduced by Valiant in (Valiant, 1979); they form a specific class of NP-hard problems. Typically, \#P problems are concerned with counting *how many* of something exists whereas general NP problems just ask *if* something exists.

A problem is \#P-complete if it is in \#P and if every problem in \#P reduces to it. It is \#P-hard if every problem in \#P reduces in it, even if it is not in \#P itself. Reducing problem \(A\) to problem \(B\) means that we take any instance of problem \(A\), use it to create an instance of problem \(B\), and conclude that the answer to solving problem \(B\) gives an answer to problem \(A\). To prove NP-hard or \#P-hard, both turning an instance of problem \(A\) into an instance of problem \(B\) and returning the answer to problem \(A\) given the solution to problem \(B\) must be done in polynomial time. If we can reduce \(A\) to \(B\) in polynomial time, we write \(A \leq_P B\).

A reduction from \(A\) to \(B\) is called *parsimonious* if the number of solutions for \(A\) is in one-to-one correspondence to solutions for \(B\). This is an important property for proving that problems are \#P-complete since we need to be able to count the number of solutions of \(A\) based on the number of solutions of \(B\). In order to show that our sensor network problem is \#P-hard, we need to find a polynomial time, parsimonious reduction from a \#P-complete problem to our problem.

It has been known for several decades that the computer science problem of **network reliability** is \#P-complete (Garey and Johnson, 1979; Colbourn, 1987). We will specifically work with the two terminal network reliability problem as defined in Garey and Johnson (1979). An instance of the problem is a graph \(G = (V, E)\) with marked vertices \(\{s, t\}\), a rational failure probability \(p_e, 0 \leq p_e \leq 1\) for each
edge $e \in E$, and a positive rational number $q \leq 1$. Then, assuming edge failures are
independent of one another, we ask whether the probability that $s$ and $t$ have a path
between them with no failed edge is greater than or equal to $q$. The fact that this
problem is hard in the class of counting problems comes from needing to count the
possible paths from $s$ to $t$ when determining the probability of failure.

This problem has striking similarities to ours, and the closeness is even more
pronounced when we look at it in the following way. Considering the graph $G$ as
a one-dimensional simplicial complex, a path in $G$ with endpoints at $s$ and $t$ is a
fundamental class in $H_1(G, \{s, t\})$, where fundamental means that the boundary of
the class is homologous to $[s] + [t]$ in $H_0(\{s, t\})$.

Our goal is to reduce network reliability to our problem, which we will therefore
call 2-dimensional network reliability. An instance of the problem is a simplicial
complex $X$ with a subcomplex $Y$ that is homeomorphic to $S^1$. We also have rational
probabilities of failure $p_v, 0 \leq p_v \leq 1$, on the vertices not in $Y$, and a value $0 < q \leq 1$.
We ask the following question: Given the fact that failures of vertices are independent
of each other, is the probability that we have a fundamental class $\alpha \in H_2(X, Y)$ at
least $q$?

Notice that our definition of the problem takes no account of the geometry in-
herent in the originally defined problem as we are ignoring the fact that we obtained
this simplicial complex from a set of points in $\mathbb{R}^2$, and thus we are proving a larger
class of problems to be #P-hard. To prove that 2-dimensional network reliability is
#P-hard, we must take an arbitrary instance of the 1-dimensional network reliability
problem, turn it into an instance of the 2-dimensional case in polynomial time,
take the solution given there and turn it into an answer to the 1-dimensional case in
polynomial time.

**Theorem 41.** 2-dimensional network reliability is #P-hard.
Proof. Consider a finite graph $G$ with vertex set $V$ and probability of failure $p_e$ given on each edge $e$. We will construct a 2-dimensional simplicial complex $X$ with a subcomplex $Y \cong S^1$ so that there is a one-to-one correspondence between paths from $s$ to $t$ in $G$ and fundamental classes of $H_2(X, Y)$. This correspondence will also preserve the probability of failure of the class, so this will imply that the probability of failure in the 1-dimensional case can be computed by determining the probability of failure in the 2-dimensional case.

Suppose that $G$ has $n$ vertices. Order these vertices by choosing a map $r : V \to \mathbb{R}$ that sends each vertex to a distinct integer in $\{1, \cdots, n\}$, with $r(s) = 1$ and $r(t) = n$. Extend $r$ to all of $G$ by linear interpolation over each edge, and subdivide $G$ by adding vertices at all points of $r^{-1}(\{1, \cdots, n\})$. Call the result $G'$, the map $r' : G' \to$ is now piecewise linear. If an edge $e$ of $G$ is subdivided into $k$ subedges in $G'$, we set the probability of failure for one of the subedges equal to $p_e$ and the rest equal to 0. See Figure [4.2] for an example of building this graph.

Form the complex

$$G' \times I / \sim$$

where $(x, z) \sim (x', z')$ iff $r(x) = r(x')$ and either $z = z' = 1$ or $z = z' = 0$. Note that this collapses the top and bottom graphs each onto a separate copy of the interval $[r(s), r(t)]$. To make this a true simplicial complex, divide each rectangle of the form $e \times I$ into triangles by placing a vertex at the barycenter and adding the obvious four new edges and four new triangles. The resulting complex will be called $X$. Then define $Y$ to be the subcomplex $G' \times \{0, 1\} / \sim$ together with the two edges $s \times I$ and $t \times I$; $Y$ is homeomorphic to $S^1$ by construction. (In Figure [4.2] we have not subdivided the rectangles to keep the picture uncluttered.)

Set the probability of failure of vertices that were added to the centers of the rectangles equal to the probability of failure of the edge of $G'$ from which they arose.
Figure 4.2: $G$ is an instance of the 1-dimensional Network Reliability problem with an ordering $r$ placed on the vertices. This map can be extended by linear interpolation over each edge. We then subdivide the edges at all points of $r^{-1}(\{1, \cdots, n\})$. Finally, we define the complex $X = G' \times I/\sim$ where $(x, z) \sim (x', z')$ iff $r(x) = r(x')$ and either $z = z' = 1$ or $z = z' = 0$.

Notice that failure of one of these vertices leads to removal of the interior of the corresponding rectangle.

It is obvious that each path from $s$ to $t$ in $G$ gives rise to a fundamental class in $H_2(X,Y)$. To prove the opposite, first recall that since $X$ has no 3-simplices, each fundamental class in $H_2(X,Y)$ has a unique representative cycle ($B_2(X,Y) = 0$). Furthermore, since we are using homology with $\mathbb{Z}_2$ coefficients, each class is simply a subset of the set of 2-simplices in $X$. A relative cycle $\alpha \in Z_2(X,Y)$ has the added property that an even number of 2 simplices in $\alpha$ contain any edge of $X - Y$ and
a fundamental class must have $\partial \alpha$ equal to the sum of all of the simplices of $Y$. Notice also that if $\alpha$ contains any 2-simplex from a rectangle, it must contain all four 2-simplices from that rectangle, so it is equivalent to think of $\alpha$ as a set of rectangles from before the subdivision.

Our conclusion now follows easily. Every rectangle in $\alpha$ determines a unique edge of $G'$. Since $F$ is covered exactly once, no two edges of $G'$ that are equivalent under $\sim$ can occur as edges of rectangles in $\alpha$. Since $\alpha$ is a relative cycle, each vertical edge must lie on either 0 or 2 rectangles, so the edges patch together to give a path from $s$ to $t$. Hence there is a one to one correspondence between the two sets.

Since we set up each rectangle to have an equal probability to that of its corresponding edge, the probability that a fundamental class is still in $H_2(X,Y)$ is equal to the probability that the corresponding path in $G$ is still functioning. Thus, if we could compute the probability of failure in $X$ in a reasonable time frame, the solution would give the probability of failure in $G$ in a reasonable time frame. Since the latter problem is $\#P$-complete, our 2-dimensional version is $\#P$-hard.

4.4 A Deterministic Algorithm

Now that we know that the general problem is $\#P$-hard, we strive to find ways to work around the computational complexity issues. We will first show that, given a set of sensors which is relatively small or at the very least relatively sparse in the domain, we can write a deterministic algorithm to compute the probability of failure of the system. In section 4.5, we will consider the modified problem of predicting failure as sensors in the system fail.
4.4.1 The Hasse Diagram

Consider a set of sensors $\chi$ in $\Delta$. Recall that edges are added when sensors are within $r_b$ of each other, and 2-simplices are added wherever all three edges have already been included.

Consider all possible subsets $A \subset \chi_{\text{int}}$ and as before construct the Rips complex $\mathcal{R}_A$, the largest subcomplex of $\mathcal{R}$ which does not utilize the nodes in $A$. The collection of these Rips complexes forms a poset under inclusion, where $A \subset B$ gives the reverse inclusion $\mathcal{R}_B \subset \mathcal{R}_A$. Arrange all of these Rips complexes into a Hasse diagram, as shown in Figure 4.3 for the example in Figure 4.1. Here we place $\mathcal{R}_A$ in the row indexed by the number of elements in $A$, and we have shaded all the complexes $\mathcal{R}_A$ which fail. A line is drawn between $\mathcal{R}_A$ and $\mathcal{R}_B$ if $A$ is obtained from $B$ by removing a vertex.

![Figure 4.3: Hasse Diagram. A Rips complex $\mathcal{R}_A$ is placed on a row according to the size of $A$ and lines are drawn to show inclusion between complex is neighboring rows.](image)

In Figure 4.3 notice that, if $\mathcal{R}_B$ fails the dS-G criterion, the Rips complexes for all its supersets of $B$ do as well, so all its successors are also shaded. This means that when searching for failures we do not have to check every possible subset. Using breadth first search from $\mathcal{R}$, we only need check complexes where all the predecessors
are cake sets. From this search pattern, if it is necessary to check the set in the first place then it is not just a death set but a minimal death set. This means that there is no post processing needed to determine the list of minimal death sets.

Given this setup, we now consider the probability of failure of the dS-G criterion.

4.4.2 Probability of Failure

Let $X_i$ be a random variable which gives the time of death of node $v_i$. In many cases, $X_i$ will be an exponential random variable, but this has no effect on our result so we make no such assumption. We do however assume that $X_i$ and $X_j$ are independent for $i \neq j$.

Let $S_A$ be the random variable which gives the first time at which all nodes in the set have failed, clearly $S_A = \max\{X_1, X_2, \cdots, X_k\}$. Because the failures of the nodes are independent events, we have

$$P(S_A \leq t) = P(\max\{X_1, X_2, \cdots, X_k\} \leq t)$$

$$= P(X_1 \leq t)P(X_2 \leq t) \cdots P(X_k \leq t).$$

Next, let $D = \{A_1, \cdots, A_r\}$ be the collection of all death sets, not necessarily minimal. Let $C$ be the random variable which gives the time of failure of the dS-G criterion for the system. The value of $C$ gives the first time that all of the nodes in one of the $A_i$ have failed, i.e. $C = \min\{S_{A_1}, \cdots, S_{A_r}\}$. Hence the probability that the system has failed the dS-G criterion by time $t$ is given by

$$P(C \leq t) = P(\min\{S_{A_1}, \cdots, S_{A_r}\} \leq t).$$

Unfortunately, these events are not independent since many of the death sets have non-trivial intersections. Compounded with the fact that there could be as many as $2^n$ such $A_i$ makes this basically uncomputable. However, it turns out the probability can be computed using only the minimal death sets, which is a far smaller collection.
Theorem 42. Let \( \{A_1, \ldots, A_d\} \) be the set of minimal death sets for the Rips complex \((R, F)\). Then the probability that the complex has failed by time \( t \) is equal to

\[
\mathbb{P}(\text{Failure by time } t) = \mathbb{P}\left( \min_i \{S_{A_i} \} \leq t \right).
\]

Proof. Since there are a finite number of nodes, every death set contains a minimal death set. Failure occurs if and only if one of the death sets fail, which happens if and only if a minimal death set fails.

This means that the probability of failure of the system can be computed given the minimal death sets. Notice that there is still work to be done since the minimal death sets may have intersections. However, since we assume that we have a small number of sensors that are well distributed and are not extremely dense in the domain, the number of intersections will be small and therefore this later computation is feasible. Thus, we seek an algorithm to compute the minimal death sets although from our knowledge that the problem is \#P-hard, we expect that this algorithm will be exponential in the worst case.

4.4.3 Death Sets Algorithm

When constructing an algorithm to determine the set of minimal death sets, the search space is the Hasse diagram described in Section 4.4.1. Since this has size \( 2^{|\chi_{\text{int}}|} \), we expect this is the source of our complexity issues.

Search the Hasse diagram using breadth first search. This will exploit the property that if \( B \) is a death set and \( B \subset A \) then \( A \) is also a death set. Hence, to find the minimal death sets, we must only check complexes where every predecessor is still a cake set. If we are forced to check all of the nodes of the Hasse diagram, then we will need to check \( 2^{|\chi_{\text{int}}|} \) complexes. However, with our assumption that we do not have a very dense set of sensors, it should take removal of a small set of sensors in
order to break the dS-G criterion. This means the size of minimal death sets will be relatively small, and thus they will be close to the top of the Hasse diagram. More importantly, it means there will be relatively few of them so our output size will not be too large.

Given this method to work through the sets, we need an efficient way to check the dS-G criterion. Consider the subcomplex $R_A$, thought of as the point in a filtration of $R$ where all simplices have been added except those which have vertices in $A$. Order the simplices so that those in the fence come first in the ordering. Initial intuition says that in order to talk about failure of the dS-G criterion when the nodes in $A$ are removed, we should filter $R$ so that all nodes, edges, and triangles which have any vertex in $A$ are last in the ordering. We could then construct the boundary matrix for this ordering, cut off the final columns corresponding to simplices which would be gone if the vertices in $A$ failed, and reduce the resulting matrix in order to read off the homology of $(R_A, F)$.

However, this turns out to be much more work than is needed. If we add all degree 1 and degree 0 simplices from the start, even if they have interior nodes which we assume to have failed, the failure of the dS-G criterion is not affected. The following expresses this and is elementary to prove.

**Lemma 43.** Let $Y$ be a 1-dimensional simplicial complex, $X$ a 2-dimensional simplicial complex whose vertex set may intersect nontrivially with $Y$, and $F \subset X$, then $H_2(X, F) \cong H_2(X \cup Y, F)$.

This implies that the time in the filtration when we add the 1-simplices is irrelevant to the homology group we are interested in, namely $H_2(R, F)$. Thus, we can order our filtration so that all the 2-simplices are at the end and consider the failure of a node as the failure only of the 2-simplices which contain it as a face.

Given this filtration, we reduce the matrix $D$ via the persistence algorithm (see,
e.g., Edelsbrunner and Harer (2010)), and consider the rightmost columns, which correspond to 2-simplices. If the row for a simplex has no lowest one in a row below those corresponding to the fence simplices, the addition of that 2-simplex creates a new class in $H_2(\mathcal{R}, \mathcal{F})$. If this column is not entirely 0 and has a lowest 1 in a row corresponding to a fence simplex, that class has a boundary which is nonzero in $H_1(\mathcal{F})$. Thus, our dS-G criterion reduces to looking for a column corresponding to a 2-simplex which has a lowest one in a row corresponding to a fence simplex.

This shows that we can quickly determine whether a complex satisfies the dS-G criterion once we have determined the correct filtration for $\mathcal{R}_A$ and have reduced the matrix $D$. We would like to not have to rewrite and re-reduce the matrix $R$ for each complex $\mathcal{R}_A$ to be checked. So, let us determine an efficient way to swap all the 2-simplices that have a vertex in our failure set $A$ to the end. For this, we turn to Cohen-Steiner et al. (2006), which gives an algorithm to quickly update and maintain the properties of $R$ and $U$, where $D = RU$, as we swap columns (Notice that $U = V^{-1}$ from the earlier discussion in Section 2.3.1). This means that we do not have to rerun the persistence algorithm each time to reduce the matrix $D$.

Let $D = RU$ be an $RU$-decomposition. That is, $R$ is the reduced matrix, and $U$ is upper triangular. Let $P$ be the matrix which swaps rows $i$ and $i + 1$, so that $PDP$ is the boundary matrix with simplices $\sigma_i$ and $\sigma_{i+1}$ switched. This can be written as $PDP = (PRP)(PUP)$, so we need to determine when $PRP$ is not reduced and $PUP$ is not upper triangular. Notice that $PRP$ is not reduced if and only if there are columns $k$ and $l$ with $\text{low}_R(k) = i$, $\text{low}_R(l) = i + 1$, and $R[i, l] = 1$. $PUP$ is not upper triangular if and only if $U[i, i + 1] = 1$.

In Cohen-Steiner et al. (2006), cases are split into whether $\sigma_i$ and $\sigma_j$ are positive or negative. The one case that will not occur for us is the possibility that $\sigma_i$ and $\sigma_{i+1}$ are positive, and there are rows $k$ and $l$ with $\text{low}_R(k) = i$ and $\text{low}_R(l) = i + 1$. This would imply that $\sigma_i$ and $\sigma_k$ are 3-simplices whose additions kill the classes born
by the addition of \( \sigma_i \) and \( \sigma_{i+1} \). As we are assuming \( \mathcal{R} \) is a 2-dimensional simplicial complex, this case is impossible, hence we can disregard it. With respect to the other cases, we can either swap rows and columns in \( R \) and \( U \), hence replace them with \( PRP \) and \( PUP \) with no issues, or we must replace them with \( PRWPW \) and \( WPWUP \), where \( W \) is the matrix which adds column \( i \) to column \( i + 1 \) in \( R \). We replace \( R \) and \( U \) with \( PRP \) and \( PUP \) if

- \( \sigma_i \) and \( \sigma_{i+1} \) are both positive simplices,
- \( \sigma_i \) and \( \sigma_{i+1} \) are both negative simplices and \( U[i, i + 1] = 0 \),
- \( \sigma_i \) is negative, \( \sigma_{i+1} \) is positive, and \( U[i, i + 1] = 0 \),
- \( \sigma_i \) is positive and \( \sigma_{i+1} \) is negative.

We instead replace \( R \) with \( PRWPW \) and \( U \) with \( WPWUP \) if

- \( \sigma_i \) and \( \sigma_{i+1} \) are both negative and \( U[i, i + 1] = 1 \)
- \( \sigma_i \) is negative, \( \sigma_{i+1} \) is positive, and \( U[i, i + 1] = 1 \).

Hence, the columns of 2-simplices which correspond to failure of specific vertices can be quickly swapped to the end of our matrix. Now that matrix \( R \) corresponding to the filtration placing the simplices in \( A \) at the end has been reduced, we can check whether our complex \( \mathcal{R}_A \) passes the dS-G criterion. In the language of persistence homology, the dS-G criterion is looking for a column which represents a positive simplex, and therefore a simplex which adds a new class to \( H_2(\mathcal{R}_A, \mathcal{F}) \). Additionally, the boundary of this new class is nonzero in \( H_1(\mathcal{F}) \). In terms of the matrices, we need to find a column \( i \) in the reduced matrix \( R \) which is has a lowest 1 corresponding to a simplex in \( \mathcal{F} \). A representation of this column is in Figure 4.4. Our algorithm is given in Algorithm 4.1.
Figure 4.4: The column representing the birth of a fundamental class. It corresponds to a 2-simplex and its lowest 1 corresponds to a simplex in $\mathcal{F}$.

Algorithm 4.1 CakeOrDeath($\mathcal{R}, \mathcal{F}$)

Input: The boundary matrix $D$ with filtration
{Fence, Remaining 1-simplices, 2-simplices}

Reduce $RU = D$

for $A \in \chi_{\text{int}}$, in the order of BFS in the Hasse diagram do
    Swap all columns corresponding to 2-simplices with a vertex in $A$
    to end of matrix $D$, and maintain $R'$ and $U'$.
    if there is not a column as in Figure 4.4 then
        mark $A$ as ‘Minimal Death.’
    else
        mark $A$ as ‘Cake.’
    end if
end for

4.4.4 Complexity of Algorithm

As a beginning aside, we point out why we chose breadth instead of depth first search. BFS has the property that we will only ever check sets which are cake or minimal death. On the other hand, DFS would require post-processing to determine which of the death sets found were minimal death sets. The perk of DFS, however, is that it requires less matrix swaps since multiple sets $A$ can be labeled as cake or death by reading off of one matrix. As we wish to have less post-processing, we choose to
use BFS for our algorithm.

Assume that the matrix $D$ is stored as a sparse matrix. This is done with a linear array of lists $D[1, \ldots, m]$ where $m$ is the total number of simplices in the 2-dimensional complex $R$. Also, assume the ordering of the simplices is \{fence, vertices and remaining 1-simplices, 2-simplices\}. Each entry $D[i]$ in this array stores a linked list denoting the locations of the codimension-1 faces of $\sigma_i$, or equivalently, the 1s in column $i$ of the full matrix $D$. This not only speeds up the operations required on the matrix, but reduces the storage size of $D$ to $O(m)$.

There are four major parts to the algorithm. The first is to reduce $D = RU$. Using the persistence algorithm, this takes time at most $O(m^3)$. See Edelsbrunner and Harer (2010) for details.

For each complex to be checked, all necessary columns must be swapped to the right side of matrix $R$. If there are $t$ 2-simplices, at worst there are $t^2/4$ swaps to be performed. While each swap has at worst an $O(m)$ running time, from Cohen-Steiner et al. (2006) there is an amortized time proportional to the number of 1s in the affected rows and columns, so this is $O(1)$. This step therefore has an amortized cost of $O(t^2/4)$.

Next, we check for a fundamental class. If this is done in $N$ complexes, then $N \leq 2^{\chi_{int} - 1}$. (Recall that because we assume that the dense set of sensors is not dense, $N$ will likely be much smaller than $2^{\chi_{int}}$.)

If a vector $\text{Low}$ giving the location of the lowest 1 in each column is maintained throughout the process of swapping, easily done via the cases in Cohen-Steiner et al. (2006), it only takes $O(t)$ time to check for a column which fits our requirements. Hence for each death set, the amortized running time is $O(t^2)$, and so the overall running time is $O(m^3 + Nt^2)$.

To make this running time feasible, one needs to keep $N$ under control. The easiest way to do this is to have a sparse set of sensors. For example, suppose the
area of the domain \( \Delta \) is \( R \) and we have \( n \) sensors. Each sensor covers an area of \( \pi r_c^2 \), so the sensors cover a total area (double counting overlap) of \( n \pi r_c^2 \). This means the expected number of sensors covering any point is \( x = n \pi r_c^2 / R \). Therefore the death sets should be of size approximately \( x \), so it is only necessary to check complexes through about the \( x \)th row. Thus \( N \approx \binom{n}{1} + \cdots + \binom{n}{x} \).

In conclusion, computing the probability exactly will be easier if the set of sensors is sparse, but what is gained in exactness of the computation is lost in the robustness of the network.

4.5 A Dynamic Algorithm for a Monitored System

Suppose we are in a situation where the deterministic algorithm is not feasible. Computing the probability of failure exactly is an NP-hard problem, and thus an exact computation is essentially impossible when the set of sensors is large. Instead, assume that a central monitoring station receives information as to whether or not each sensor has failed. In this case, a more practical question is to ask when the system is getting close to failure and so we seek a dynamic algorithm to predict which nodes would cause failure of the criterion should they fail soon. To do this, we will create a new criterion built from the old which will give an early warning for failure. It essentially gives a flag on each interior vertex warning that its failure would probably cause failure of the dS-G criterion.

For technical reasons, we will assume in this section that the domain is convex. A domain that is not convex can have a radius where the Rips complex has a nontrivial class in \( H_1(\mathcal{R}) \) even though \((\mathcal{R}, \mathcal{F})\) passes the dS-G criterion. This assumption is much stronger than is necessary since all we really need is that \( H_1(\mathcal{R}) = 0 \) whenever \((\mathcal{R}, \mathcal{F})\) passes the dS-G criterion.

The main idea for the new criterion comes from the following two theorems.
Theorem 44. Assume that the pair of simplicial complexes \((\mathcal{R}, \mathcal{F})\) with a convex domain passes the dS-G criterion and \(w\) is a vertex in \(\mathcal{R} - \mathcal{F}\).

If \(H_2(\mathcal{R}) = 0\) and \((\mathcal{R}_w, \mathcal{F})\) passes the dS-G criterion, then \(H_1(Lk(w)) = 0\).

Proof. Consider the canonical projection map \(p : \mathcal{R} \to \mathbb{R}^2\) given by sending each vertex to its location in \(\mathbb{R}^2\) and each simplex onto the convex hull of its vertices. The image of any subcomplex under this projection is called its shadow. By Chambers et al. (2009), \(\pi_1(p(\mathcal{R})) \cong \pi_1(\mathcal{R})\) and thus \(H_1(p(\mathcal{R})) \cong H_1(\mathcal{R})\). Since \((\mathcal{R}_w, \mathcal{F})\) passes the dS-G criterion and the domain is convex, the shadow of \(\mathcal{R}_w\) is the domain itself, which is contractible. Then by Chambers et al. (2009), \(H_1(\mathcal{R}_w)\) is trivial.

Mayer-Vietoris for \(\mathcal{R} = \mathcal{R}_w \cup \text{St}(w)\) gives the exact sequence

\[
H_2(\mathcal{R}) \to H_1(Lk(w)) \to H_1(\mathcal{R}_w) \oplus H_1(\text{St}(w, \mathcal{R}))
\]

Thus, since \(H_2(\mathcal{R})\), \(H_1(\mathcal{R}_w)\) and \(H_1(\text{St}(w, \mathcal{R}))\) are all trivial, \(H_1(Lk(w)) = 0\). \(\square\)

Theorem 45. Assume that the pair of simplicial complexes \((\mathcal{R}, \mathcal{F})\) passes the dS-G criterion and \(w\) is a vertex in \(\mathcal{R} - \mathcal{F}\).

If \(H_1(Lk(w)) = 0\) then \((\mathcal{R}_w, \mathcal{F})\) passes the dS-G criterion.

Proof. Consider the Mayer-Vietoris complex for the pairs \((\mathcal{R}_w, \mathcal{F})\) and \((\text{St}(w), \emptyset)\). This gives the exact sequence

\[
H_2(\mathcal{R}_w, \mathcal{F}) \oplus H_2(\text{St}(w), \emptyset) \to H_2(\mathcal{R}, \mathcal{F}) \to H_1(Lk(w)).
\]

The star of a vertex is contractible, and by assumption \(H_1(Lk(w)) = 0\), so this sequence becomes

\[
H_2(\mathcal{R}_w, \mathcal{F}) \to H_1(\mathcal{R}, \mathcal{F}) \to 0.
\]

Thus the map \(H_2(\mathcal{R}_w, \mathcal{F}) \to H_2(\mathcal{R}, \mathcal{F})\) must be surjective. Specifically, if \(\alpha\) is a fundamental class in \(H_2(\mathcal{R}, \mathcal{F})\), there must be a \(\beta \in H_2(\mathcal{R}_w, \mathcal{F})\) which maps to it.
Figure 4.5: The first circle contains a planar Rips complex with a nontrivial $H_2$. The six vertices are distributed around a circle of radius $1/\sqrt{3}$ and the Rips complex is built with edges of length 1. The second circle gives a counter example to show that the assumption $H_2(\mathcal{R}) = 0$ is necessary for Theorem 44. Vertex $w$ is added to the complex far enough away from the top vertex so as not to add the edge. Then $H_1(\text{Lkw})$ is nontrivial, as highlighted in the third circle, although the complex $(\mathcal{R}, \mathcal{F})$ would still pass the dS-G criterion.

Using the commutative diagram

$$
\begin{array}{ccc}
H_2(\mathcal{R}_w, \mathcal{F}) & \longrightarrow & H_2(\mathcal{R}, \mathcal{F}) \\
\downarrow & & \downarrow \\
H_1(\mathcal{F}) & & 
\end{array}
$$

we see that since $\alpha$ maps to something nonzero in $H_1(\mathcal{F})$ by definition of fundamental class, $\beta$ must also map to something nonzero in $H_1(\mathcal{F})$ and is therefore also a fundamental class.

These two theorems could comprise an if and only if statement except for the additional assumption that $H_2(\mathcal{R}) = 0$ for Theorem 44. Despite being counterintuitive, it is possible for a set of points in the plane to have a non-trivial second homology group Chambers et al. (2009). For example, distribute six vertices around a circle of radius $1/\sqrt{3}$ and form the rips complex for distance 1 as in Figure 4.5. This figure gives an example where the assumption $H_2(\mathcal{R}) = 0$ is necessary.
4.5.1 The new criterion and complexity

Given Theorems 44 and 45, we propose a new criterion to complement the de Silva - Ghrist criterion:

**Definition 46** (Link Condition). *If an interior vertex* $w$ *has* $H_1(Lk(w)) \neq 0$, we say it is flagged. Otherwise, we say it is not flagged.*

The idea is that if a vertex is flagged, there is a chance its removal will cause failure of the dS-G criterion. If it is not flagged, then its removal can do no harm. The pseudocode is given in Algorithm 4.2

**Algorithm 4.2** Monitored System Failure

**Input**: Simplicial complex pair $(\mathcal{R}, \mathcal{F})$

1. Check dS-G criterion $\triangledown$ We assume that this initial $\triangledown$ check will always pass
2. Compute link of each vertex $w$ and $H_1(Lk(w))$
   - if $H_1(Lk(w)) = 0$ then
     - Mark $w$ as flagged.
   end if
3. if vertex $v$ fails then
   - if $v$ is flagged then
     - Update matrix $\mathcal{R}$ to remove dead simplices.
     - Check dS-G criterion.
     - if $(\mathcal{R}_w, \mathcal{F})$ fails the dS-G criterion then
       - Break
     end if
   end if
4. Update links of vertices
5. Compute $H_1(Lk(w))$ for $w$ whose link has changed
6. Mark or unmark $w$ as flagged according to $H_1$ computation.

This algorithm turns out to be polynomial in the number of simplices in the worst case. We will split the complexity computation into two parts: the initialization step, done before any vertex has failed, and the time taken for the algorithm for each failed vertex.
**Initialization**

Let $m$ be the number of two simplices of dimension $\leq 2$. In section 4.3.1, we showed that this is the highest dimension simplex needed for the dS-G criterion, and since the link condition only looks at the first dimension of the complex, nothing above the second dimension is required.

The complexity of computing the link of $w$ is directly related to the number of simplices containing $w$ as a vertex. In fact, given a list of all simplices in $\mathcal{R}$ which include vertex $w$, print the simplex obtained by removing vertex $w$ from the simplex. This is the link, so given a listing of the simplices with vertex $w$, the link of $w$ can be computed in time $O(k)$ where $k$ is the number of adjacent simplices. Since $k$ is obviously less than $m$, the time to initially compute all the links is $O(mn)$.

The time to compute $H_1(\text{Lk}(w))$ is $O(k^3)$, again with $k$ equal to the number of simplices in the link of $w$. Since, $k < m$, the time taken for this initial step is $O(nm^3)$. Thus, the entirety of the initialization step takes time $O(m^3 + mn + nm^3) = O(nm^3)$.

**Failed Vertex**

In the worst case, every failed vertex is flagged and so the dS-G criterion must be recomputed each time. As seen in section 4.4.4, updating the matrix $R$ and checking the dS-G criterion takes time $O(t^2)$ where $t$ is the number of two simplices.

What is interesting about the link condition is that it is easy to maintain the links of all the interior vertices. Let $\text{Lk}(\sigma, X)$ be the link of $\sigma$ in the simplicial complex $X$ and let $X_w$ be the largest subcomplex of $X$ without the vertex $w$.

**Lemma 47.** For any vertices $v, w$ in a simplicial complex $X$,

$$\text{Lk}(v, X_w) = \text{Lk}(v, X) \cap X_w.$$  \hspace{1cm} (4.4)

**Proof.** If $\sigma \in \text{Lk}(v, X_w)$ then obviously $\sigma \in X_w$. Also, this implies that the simplex $\tau = \langle \sigma, v \rangle \in X_w$. Since $\sigma < \tau$ and $v \notin \sigma$, we must have that $\sigma \in \text{Lk}(v, X)$, so
Lk\((v, X_w) \subset \text{Lk}(v, X) \cap X_w\).

Let \(\sigma \in \text{Lk}(v, X) \cap X_w\). Since it is in \(\text{Lk}(v, X)\), the simplex, \(<\sigma, v> \in X\). As \(\sigma\) is also in \(X_w\), \(w \not\in \sigma\), so \(<\sigma, v> \in X_w\). Therefore, \(\sigma \in \text{Lk}(v, X_w)\), and equation 4.4 follows.

This lemma implies that the only update needed after the failure of a vertex is to delete any simplices in the link which were also deleted in the simplicial complex. In the worst case, the link of every vertex must be updated, the first homology recomputed, and the flag remarked as needed. Since the size of each link is at most \(m\), this step takes \(O(n(m + m^3)) = O(nm^3)\).

If this sequence of events happens for every \(n\), this second part of the algorithm takes time

\[
O\left(n(t^2 + t + mn + m^3n)\right) = O(m^3n^2).
\]

Combining this with the initializing step, the whole algorithm takes at worst time \(O(m^3n^2)\), so is polynomial in the number of simplices.

### 4.6 Conclusions and Possible Extensions

In this paper, we have extended the problem posed by de Silva and Ghrist in de Silva and Ghrist (2006) by assuming that sensors have a probability of failure, and asking for the probability of failure of the dS-G criterion for coverage. We determined that the generalized version of the problem is \#P-hard, and thus it is unlikely that there is an algorithm to answer this question in general which runs in a reasonable amount of time. Finally, we provided a deterministic algorithm which does work in the case of a small set of sensors, and a method to predict failure when the system is larger but is being monitored.

The obvious immediate extension of our work is to determine whether the the version of the problem posed by de Silva and Ghrist in de Silva and Ghrist (2007),
which allows for higher dimensions and looser boundary conditions, is also amenable
to an application of probability of failure. We conjecture that this extended problem
will also be NP-hard.

In the long term, we would like to see more applications of computational topol-
ogy to the design and analysis of sensor networks. Since we can make such strong
conclusions with such weak assumptions on the capabilities of the sensors, we expect
that such applications are abundant.
With an ever increasing quantity of sensors and surveillance in modern society comes an overpowering deluge of streaming data. In the past, this data has been monitored in real time by an army of trained professionals, but the enormous quantity of data has begun to create a burden on the amount of available human power.

What if we could automate the ability to say that two people are behaving in a similar fashion? Are these people behaving strangely? Are they working together to accomplish a goal? In this chapter, we give the beginnings of the framework to do just that. Say we are given the motion of a person as a track in $\mathbb{R}^2$. This is, in fact, just a time series of points in $\mathbb{R}^2$, so if we have tracks for a set of people, this can be thought of as a dynamic point cloud.

At the lowest level, we are given raw data in the form of aerial images. However, for the purposes of this chapter, we will assume that these have been processed into tracks, a list of time stamped locations for a set of agents. We will build a data structure which is easy to maintain as new information for the tracks is added. From this, we can build a behavior vector for any track which matches a quantity to a particular behavior.
Since we can then cluster tracks by their behavior vectors, we can pick out similarly behaving agents. Persistent 0-dimensional homology also gives a useful tool for representing this information, the dendrogram, which quickly conveys a great deal of information about the underlying point cloud. This allows a user to make a quick decision about which radius to use for clustering, and to find the agents who belong in each cluster. These behavior vectors not only allow us to pick out individuals who are interesting, but also interesting subsets of the agents for further study. We can then also create vectors which encode information about the behavior of a whole group of agents, without just treating them as individuals.

Additionally, we can study groups as a way to deal with noisy or incorrect tracking systems. A common issue for trackers is when a large number of agents go to the same area, get mixed up, and then leave. The tracker cannot decipher which entering agent is the same as which exiting agent, so it will either create a new track for the exiting individuals, or make a guess, often incorrectly. With the added information from the behavior vectors, we can improve the accuracy of the guess by comparing the behavior before and after the event. If we don’t want to make a guess, we can study the questioned tracks as a group instead of as individuals.

The goal of this chapter is not to determine the best behavior vectors for picking out interesting individuals, but rather to create a tool for trained professionals to automate the process of picking out agents with a behavior of interest.

This chapter represents joint work with Paul Bendich, Robert Calderbank, and John Harer.

Outline This chapter is organized as follows. Section 5.1 presents the Dynamic Data Structure used to store the data. Section 5.2 discusses the single track behavior vectors and methods for clustering. In Section 5.3, we present behavior vectors for finding patterns created using multiple agents, and in Section 5.4 we discuss the
potential extensions of this work to improve the selection of groups and feedback to
the tracker for its improved performance.

5.1 Dynamic Data Structure

Initially, the data is given as a time series of aerial images of a highly populated area
with agents moving within the frame. We will assume that these images have been
passed through a tracker, so the data given to us is a stream of locations with a time
stamp and ID number. We will initially assume that the information given to us by
the tracker does not suffer from common error issues such as incorrect locations, or
lost or mislabeled tracks, however in Section 5.4 we will discuss the potential of using
topology to both compensate for potential error as well as feed information back to
the tracker to improve its output.

Our first goal is to construct a data structure to store this information and which
can be updated and queried quickly for computing the behavior vectors of Section 5.2.
Prior to starting the stream of data, we chose a mesh $M$ to use on the underlying
region in $\mathbb{R}^2$ defined by the boundary of the images and store it as described in
Section 5.1.1. We also keep a data structure to store the information attached to a
particular ID number. Then, as new data is fed into the system, we store information
in two places: with each ID number and with each face of the mesh element. The
multiple data structures are interconnected, and this redundancy in storage allows
for fast computation. We call this set of data structures the Dynamic Data Structure
(DDS).

5.1.1 Mesh Storage

Before any information about agents or tracks is added, we will construct a mesh
to speed computation of the behavior vectors. Since we will be interested in agents
who are close together, we can define a neighbor to be anyone in a neighboring mesh
element. This way, we can avoid the $O(n^2)$ time needed to compute all pairwise distances between a set of points.

The mesh will need to do two things quickly: find all its neighbors, and return the mesh element containing a given query point. Our data lives in a box defined by the size of the image, so although we could use a very arbitrary polygonal mesh stored, for example, in some combination of a doubly-connected edge list and a trapezoidal map, we will use the extremely simple data structure known as a balanced quadtree. This exposition follows that of de Berg et al. (2008).

A quadtree is a rooted tree where each internal node has exactly four children. Each node corresponds to a square, and the children represent the four quadrants of this square. These children are labeled NE, NW, SE, and SW according to their position in the square. Each mesh element is represented by a leaf, and the depth of the quadtree is equal to the number of subdivisions to get to the smallest square in the mesh. We will call this mesh a quadtree subdivision. Note that although the faces of the quadtree subdivision are in the shape of the square, they may have more than four vertices. In order to be specific about whether we are working in the quadtree or its subdivision, the square associated to a node $\nu$ is denoted $\sigma(\nu)$. See Figure 5.1 for an example.

Determining which mesh element contains a query point $q$ is a simple check down the tree. Starting at the root, determine which quadrant contains $q$ by asking if it is above or below the horizontal split and if it is left or right of the vertical split. Thus, for a quadtree of depth $d$, the query can be done in $O(d)$ time.

Define two mesh elements to be neighbors if they share an edge or a vertex. The first issue to be tackled is that returning a list of neighbors could be bad if a mesh element has an unbounded number of neighbors, which can happen for quadtrees as in Figure 5.1. In order to combat this problem, we will use balanced quadtrees.

A quadtree subdivision is called balanced if every pair of squares which share an
Figure 5.1: A quadtree subdivision, at left, and its corresponding quadtree, at right. For the node $\nu$, $\sigma(\nu)$ is its corresponding square in the quadtree subdivision. The neighbors of $\sigma(\nu)$ are shaded and their corresponding nodes in the quadtree are filled in. Note that because this quadtree subdivision is not balanced, there can be a very large number of neighbors for a given face.

An edge differs in size by at most a factor of 2. A quadtree is balanced if its subdivision is balanced. Notice that this implies that any mesh element has at most 12 vertex or edge neighbors. See Figure 5.2 for the subdivision of a quadtree subdivision which is balanced.

There are two important algorithms still required: finding the neighbors of a given mesh element, and the ability to balance a given quadtree. First, given a square $\sigma(\nu)$ for a node $\nu$ in the quadtree, consider finding the node $\nu'$ representing a square $\sigma(\nu')$ which shares a side with $\sigma(\nu)$ and which is at least as big as $\sigma(\nu)$. Note that this means we may be looking for an internal node rather than a leaf, and the node $\nu'$ will have depth at most that of $\nu$. See Figure 5.3.

As an example, we will find the square which shares the north side of the square $\sigma(\nu)$. If $\nu$ happens to be the SE or SW child of its parent, then its northern neighbor is the NE or NW child of its parent, respectively. If, however, $\sigma(\nu)$ already touches the north side of its parent, we move up to the parent and look for its northern neighbor. If this neighbor corresponds to a leaf node, we return this neighbor, otherwise, we
Figure 5.2: A quadtree subdivision, shown at left, and its corresponding balanced quadtree subdivision. In order for a quadtree to be balanced, faces which share an edge can differ in size by at most a factor of 2. Although the largest SW square in this example does not initially fail this criterion, subdivisions of the NW square force its subdivision.

Figure 5.3: The node returned by the NorthNeighbor($\nu$, $T$) algorithm, Algorithm 5.1, is the lowest node in the tree which is at most as deep as $\nu$ and shares the north side of $\nu$. This node, $\nu'$ in the above example, might not be a leaf node. If we want the actual neighbor mesh elements of $\nu$, we follow all southern paths down until we find leaves.
follow the tree back down a step to find the appropriate child. Note that if the square is already on the northern border of the square for the root node, there is no northern neighbor. In this case, the algorithm returns \texttt{None}. See Algorithm 5.1 for the pseudocode, and the algorithms to find the eastern, western, and southern neighbors are similar. If we want to find the neighbor which shares a corner with the square \( \eta \), say the NW corner, we can find \texttt{NorthNeighbor(WestNeighbor(\nu, T), T)}
 and follow the quadtree down through SE children until we find a leaf.

\begin{algorithm}
\textbf{Algorithm 5.1} \texttt{NorthNeighbor}\((\nu, T)\)
\begin{algorithmic}
\begin{itemize}
\item[] \textbf{Input:} Node \( \nu \) in quadtree \( T \).
\item[] \textbf{Output:} Node \( \nu' \) which is at most the depth of \( \nu \) and where \( \sigma(\nu') \) shares the northern side of \( \sigma(\nu) \), or \texttt{None} if there is no such node.
\item[] if \( \nu = \text{root}(T) \) then
\item[] \hspace{1em} return \texttt{None}
\item[] else if \( \nu = \text{SW child of parent}(\nu) \) then
\item[] \hspace{1em} return NW child of parent(\nu)
\item[] else if \( \nu = \text{SE child of parent}(\nu) \) then
\item[] \hspace{1em} return NE child of parent(\nu)
\item[] end if
\end{itemize}
\item[] \( \mu = \text{NorthNeighbor(parent}(\nu), T) \)
\item[] if \( \mu = \text{None} \) or \( \mu \) is a leaf then
\item[] \hspace{1em} return \( \mu \)
\item[] else if \( \nu = \text{NW child of parent}(\nu) \) then
\item[] \hspace{1em} return SW child of \( \mu \)
\item[] else
\item[] \hspace{1em} return SE child of \( \mu \)
\item[] end if
\end{algorithmic}
\end{algorithm}

Since each recursive call takes \( O(1) \) time, and at most travels the depth of the tree, the northern neighbor \( \nu' \) can be found in \( O(d) \) time. Then, if we have a face \( \sigma(\nu) \) and wish to find all leaves \( \omega \) whose square \( \sigma(\omega) \) shares the northern side of \( \sigma(\nu) \), we can follow every path down from \( \nu' \) using only SE or SW children. If \( T \) is a balanced quadtree, there are at most 2 such nodes, so these can be found in \( O(1) \) time.

Thus, in order to bound the number of any given mesh element, we want to turn any quadtree into a balanced quadtree. We do this by placing the leaves into a list to
be checked. When we check a leaf $\nu$, we just need to see what each neighbor algorithm akin to Algorithm 5.1 returns. For example, if the NorthNeighbor algorithm returns a node $\nu'$ where either its SE or its SW child is not a leaf, the node $\nu$ must share its northern border with a square less than half its size. If this is the case, we split $\sigma(\nu)$ into four pieces by adding four children to $\nu$. We then add these to the list to be checked. We also check that the splitting of $\sigma(\nu)$ didn’t cause any neighbors to need to be split; if so, we add these as well to the list.

**Algorithm 5.2 BalanceQuadTree($T$)**

**Input:** A quadtree $T$

**Output:** A balanced version of $T$

$L = \{\text{Leaves of } T\}$

while $L \neq \emptyset$ do

Remove a leaf $\nu$ from $L$

if $\sigma(\nu)$ needs to be split then

Give $\nu$ four children.

Add these new leaves to $L$.

if any of $\sigma(\nu)$’s neighbors now need to be split then

Add these nodes to $L$.

end if

end if

end while

return $T$

At first sight, there doesn’t appear to be any control over the size of the new quadtree or how long this process would take since splitting nodes can propagate. However, as shown in de Berg et al. (2008), things are not as bad as they seem.

**Theorem 48.** Given a quadtree $T$ with $n$ nodes and depth $d$, BalanceQuadTree returns a balanced version with $O(n)$ nodes in $O((d + 1)n)$ time.

Putting this all together, we can take any quadtree mesh we like on our original images and turn it into a balanced quadtree mesh. Then we can both find the neighbors of a given mesh element and determine a query points mesh element in $O(d)$ time.
5.1.2 Track Storage

Once the structure for the mesh has been built, the structure for the tracks can be built. Unlike the mesh, the tracks will be given and changing in real time, so this data structure should be easily updatable and constantly changing. Also, unlike the mesh, this data structure is extraordinarily simple since all that is required of it is the ability to call up either the set of mesh element or specific locations visited by a particular ID number.

As the stream of data is given, we can add a new track every time a new ID number is seen. This track stores a dictionary with the time-stamped location information. The mesh determines which face contains this location in $O(d)$ time, so we also keep a pointer for each time stamp to the corresponding face. We can also update the list, stored with the mesh, of agents which have visited this particular face.

5.2 Single Track Behavior Vectors

In order to have a quantifiable representation of each track’s behavior, we construct behavior vectors. Behavior vectors are simple vectors in Euclidean space which give a snapshot of some aspect of a track’s behavior. These vectors can describe anything from the simplest of behaviors like acceleration or stopping times, to more complex patterns like loop formation. We quantify these vectors as “simple” since they can be quickly computed and updated from the DDS.

5.2.1 Stopping Time Vector

The easiest behavior vector to understand is the stopping time vector. The algorithm for computation takes in a track and a parameter $\epsilon$ which allows for noisy data. The idea is to look for times at which the track is standing still, and keep track of how long this occurs each time. We then create a sorted vector of the lengths of time for which the agent is not moving.
Depending on context and the size of the mesh, being stopped may mean that the agent actually stays at the same point, or at least within a small $\varepsilon$ of the same point, or that it stays within the same face for a given amount of time. Since this computation is extremely simple to update, it can either be maintained as the track information is entered into the data structure, or computed at the time it is wanted.

Fixing some time of interest, we look for the times the agent has been stopped in the interval. For each instance where the agent has stopped, we add the length of time to a vector. Once all the times are added, we reverse-sort the vector (so that the highest value is first) and call this the stopping time vector for the given interval.

### 5.2.2 Acceleration Vector

Like the stopping time vector, this vector is also extremely simple to compute. Again giving a fixed time interval of interest, each set of three consecutive time points gives an acceleration value for the time period. So, if we are interested in the triple of locations $p_i, p_{i+1}, p_{i+2}$ corresponding to times $t_i, t_{i+1},$ and $t_{i+2},$ we can approximate velocity and acceleration by

$$v_i = \frac{\|p_{i+1} - p_i\|}{t_{i+1} - t_i}, \quad a_i = \frac{v_{i+1} - v_i}{t_{i+1} - t_i}.$$ 

These accelerations, like the stopping times, can be put into a vector and sorted. In particular we will keep track of the positive values and the negative values for the purposes of comparison later.

### 5.2.3 Looping Vector

Like the stopping and acceleration vectors, the looping vector aims to condense one particular behavior characteristic into a small vector. However, unlike the previously discussed vectors, this computation has the ability to become a bit more complicated.

In its simplest form, the looping vector needs to find times when a given track
crosses itself. If the mesh is large enough that a given agent will never move more than one face away between each time stamp, loops are defined by any time an agent returns to a face previously visited. For this case, the time to complete the loop is the difference of the time stamps when the agent entered that face. Again, this information can be entered into a vector and sorted. If, however, the mesh is too small to be immediately useful for determining loop times, the loops can be computed exactly by checking each pair of edges in the path or an intersection, and returning the associated times. Whatever the method for computation, we can place the difference in the timestamps of the two crossed edges into a sorted vector, which we will call the looping vector.

The issue with this construction is that it is extremely non-robust to error in the given point locations for a track. Consider the example of Figure 5.4. In the first example, the agent does not cross its own path, and so the looping vector created would be empty. However, in the second example, the agent crosses its path twice and so has two large entries in its looping vector. While the tracks only differ by the locations of two points, the two loop vectors are quite different.

In order to compensate for this, we can use persistence to determine the locations
of almost-loops. Given a track $T = (p(t_1), \cdots, p(t_n))$, we will build a new simplicial complex which represents the distance function to the track, $d_T(x) = \inf_{p \in T} \|p - x\|$ where a point $p \in T$ can either be on a vertex or the edges between them.

We will construct a clique complex $K_T$ with a function on the edges, similar to the Rips complex, which can be used to represent the union of sublevel sets of $d_T$ in the plane. Like the normal construction with the union of balls around points, the sublevel set of the function around a line is convex. So, the homology of the set only changes when the blobs around two edges $(e_1, e_2)$ intersect at distance

$$d(e_1, e_2) = \inf_{(x_1, x_2) \in (e_1, e_2)} \|x_1 - x_2\|.$$

Let the simplicial complex $K_T$ have vertex set isomorphic to the edge set of $T$. Let the vertex associated to edge $e$ be denoted $v_e$. Then we can define the distance between $v_e$ and $v_{e'}$ to be $d(e, e')$. Computing the persistent homology of the filtration defined by the flag complex of this edge set gives the homology of the union of blobs around each edge.

A loop is created in this complex for some small distance $\delta$ if the original track formed an almost loop. We are, in fact, not interested in computing the persistence for the entire clique complex. This construction was built to deal with noise in the data returned, so we can fix a maximum $\delta$ based on the expected noise of the data set, and compute up to this maximum threshold. Then loops of interest are ones which are born before time $\delta$ and are not dead when we stop computing. See Figure 5.5.

A helpful side effect of this procedure is that it also ignores small loops which might be uninteresting in terms of the behaviors for which we are looking, or might themselves be created by noisy and inexact data.
5.2.4 Dendrograms

Once the vectors of interest have been computed for a set of tracks, the next step is to determine which of these vectors arise from similar behavior. Embed each vector into $\mathbb{R}^d$ for some dimension longer than the longest vector using some method which makes sense for the particular behavior being studied. For instance, the stopping time vectors or looping vectors can have zeros appended to the end until all vectors have the same length. For the acceleration vectors, since we would like to associate positive values with positive values and negative values with negative values, zeros can be added to the beginning and end of each vector so that the sign shift occurs in the same coordinate.

Whatever the method, given a set of points in $\mathbb{R}^d$, we can cluster the set of points. In particular, we can think of computing the 0-dimensional persistent homology for the Rips complex of the points. As we are interested only in the 0th-dimension, all the classes are born at time 0. This means that all the off-diagonal points of the persistence diagram lie on the $y$-axis. This picture is not very telling of the structure.
Figure 5.6: Figure (a) shows a set of short tracks exhibiting looping behaviors. The looping vectors are computed and the corresponding dendrogram is drawn. The dendrogram in (b) shows that the tracks with a single loop are similar, the tracks with disjoint double loops are similar, and the tracks with two intersecting loops are also similar. Since the looping vector computation did not utilize the persistence modification from Section 5.2.3, the tracks with two non-intersecting loops are more similar to the single loops than to the tracks with two intersecting loops.
of the clusters, so instead we use a dendrogram, a standard visualization tool used in clustering, to understand the structures.

In particular, if we think of computing 0-dimensional homology using the union-find algorithm, a dendrogram is the same as the searchable tree structure created by the algorithm which pays attention to the function value at which two clusters merged. So, each point in the cloud is represented by a leaf of the tree, and each other vertex is drawn at height $u$ where the addition of an edge $e$ with length $u$ caused the merging of the clusters represented by the vertex’s left and right subtrees.

Suddenly, it is very easy to see what function value makes sense for clustering since the number of trees in the dendrogram below a given threshold is the number of clusters for that radius, and the size of each cluster is the number of leaves in its corresponding subtree. For instance, in Figure 5.6b, cutting off the function at 200 means that there are three subtrees with two leaves each. In this case, the trees correspond to the paths with one, two, and two crossed loops respectively.

5.3 Group Behaviors

5.3.1 Group Creation

There are many ways to define groups of the individuals. The first and most obvious from the previous section is that we are interested in a group of agents with similar behavior vectors. If the dendrogram is cut off at a given function value, we consider a group to be all the agents in one of the defined clusters. This means that if agents are exhibiting similar behaviors, we can associate them even if they have never been to the same location.

However, it may be helpful to use common locations to create groups. For instance, we can determine a group by agents who have spent a great deal of time in the same or neighboring mesh elements. Since the DDS keeps a list of neighbors for each track, this is determined by data already stored. Likewise, we could deter-
mine a group by agents who frequently visit a location of interest. Again, since this information is stored with each mesh element, a determination of groups with this method is extremely quick.

5.3.2 Analyzing Group Behavior

Once we have some method for determining a subset of agents, we can look at the behaviors exhibited by them as a group which do not show up when the tracks are taken individually. Consider, for example, the group of agents in Figure 5.7. Although individually, none of the agents is doing anything interesting, as a group they started in the same place, split apart, and came back together.

There are two ways we can pick out this behavior. First consider the union of the tracks of interest in a time cube \((\mathbb{R}^2) \times \mathbb{R}\), where \(\mathbb{R}^2\) represents the location in the plane, and \(\mathbb{R}\) represents the timestamp. Then there will be a class in the persistent \(H_1\) which was born early and died late if this set of tracks had a group of agents in the same place at one time, separated for a while, and then came back together. The closer the agents were to being in the same place at the same time either at the beginning or end, the earlier the class will be born.

If, on the other hand, we are interested only in the shape created and not in the time at which it occurred, we can instead project the tracks onto the \(\mathbb{R}^2\) plane and forget the time stamps. Again, we can look for classes in \(H_1\), but here, an early birth of a class only corresponds to proximity to the same location without regarding time.

The idea of groups can also be used to associate tracks which are suspected to be the same track, but which the tracker mislabeled. For instance, Agent X stopped for a period of time, causing the tracker to lose her. Then, when she started moving again, the tracker gave her a new label. However, since she is still, assumingly, exhibiting the same behavior vector before and after the confusion, grouping agents by behavior vectors would put both of her half-tracks into the same pile. Looking
Figure 5.7: A group of tracks which create a loop. In this case, we are looking at the 2-dimensional projection of the track locations, so time is completely disregarded. This shape is encoded in the persistent $H_1$ of the union of tracks as a class which is born early, corresponding to the small distance between the tracks at left, and dies at a radius associated to the diameter of the center hole.

For, say, loops as above, having Agent X represented as one or two tracks makes no difference.

5.4 Future Work

In the future, we would like to improve our determination of groups. We will build a model based on spreading an infectious disease through a network (See, for example, Keeling (2005)). Say Agent X is a known person of interest. We consider her infected. Then, whenever there is an agent in a neighboring face to hers, we spread a little of the disease. Spending a large amount of time in a neighboring face causes the other person to be infected. Being in a neighboring face for a short period of time and then being away from the infected party causes the infection to dissipate.

We would also like to include the results of Chapter 3, in particular the mean of a
set of vineyards, to improve the determination of whether a given group is interesting. Given any of these groups, we can compute their corresponding vineyard for some time interval. If we have found several interesting groups, we can look at their average vineyard in order to understand something about their general patterns.

This project also seems like it would benefit greatly from the addition of machine learning. Since we do not have the expertise to determine which agents are actually exhibiting “interesting” behavior, we will use the techniques of machine learning to automate the process of deciding which behavior vectors do the best job of finding interesting agents.

In the long term, we would also like to use the behavior vector concept to improve current tracking algorithms. In particular, if a group of agents come together, the tracker cannot determine which agents leaving correspond to those that entered. Since each track can have many associated behavior vectors, these can be used to patch up broken tracks by pairing up pieces with similar characteristics.

Hopefully, this framework will allow for better and faster automated analysis of behavior of agents in aerial images as well as the ability to detect group actions faster than any human could.
Bibliography


Biography

Elizabeth Munch was born in Rochester, NY, on March 18, 1986. She attended the Eastman School of Music and the School of Arts and Sciences at the University of Rochester. In May, 2008, she earned a Bachelor of Music in Harp Performance with a Performer’s Certificate and a Bachelor of Science in Mathematics. She also holds a Master of the Arts in Mathematics from Duke University, December, 2011.

While at Duke, she has been funded by the OASIS grant through the DDR&E, the NSF Mathematical Biology and Data RTGs, and received the first Jo Rae Wright Fellowship for Outstanding Women in Science in 2012-2013. While at Duke, she was the first author of Munch et al. (2012). In addition, she is the president of the Noetherian Ring of the Duke Mathematics Department.

In addition to math, she performs harp regularly both with the Charlotte Symphony, and as part of the Leechford Harp and Sax Duo. She also knits, is severely obsessed with low budget sci-fi movies and indie video games, and is slowly learning Arabic.