Interactive Visualization of 2-D Persistence Modules

Michael Lesnick[*]   Matthew Wright[†]

February 5, 2016

Abstract

The goal of this work is to extend the standard persistent homology pipeline for exploratory data analysis to the 2-D persistence setting, in a practical, computationally efficient way. To this end, we introduce RIVET, a software tool for the visualization of 2-D persistence modules, and present mathematical foundations for this tool. RIVET provides an interactive visualization of the barcodes of 1-D affine slices of a 2-D persistence module M. It also computes and visualizes the dimension of each vector space in M and the bigraded Betti numbers of M. At the heart of our computational approach is a novel data structure based on planar line arrangements, on which we can perform fast queries to find the barcode of any slice of M. We present an efficient algorithm for constructing this data structure and establish bounds on its complexity.

Contents

[*]Columbia University, New York, NY, USA; `mlesnick@alumni.stanford.edu`
[†]St. Olaf College, Northfield, MN, USA; `wright5@stolaf.edu`

1 Introduction

1.1 Overview

Topological data analysis (TDA) is a relatively new branch of statistics whose goal is to apply topology to develop tools for studying the coarse-scale, global, non-linear, geometric features of data. Persistent homology, one of the central tools of TDA, provides invariants of data, called barcodes, by associating to the data a filtered topological space $\mathcal{F}$ and then applying standard topological and algebraic techniques to $\mathcal{F}$. In the last 15 years, persistent homology has been widely applied in the study of scientific data [9, 31], and has been the subject of extensive theoretical work [5, 10, 17, 20, 24, 47].

For many data sets of interest, such as point cloud data with noise or non-uniformities in density, a single filtered space is not a rich enough invariant to encode the structure of interest in our data [11, 13, 19]. This motivates the consideration of multidimensional persistent homology, which in its most basic form associates to the data a topological space simultaneously equipped with two or more filtrations. Multi-D persistent homology yields algebraic invariants of data far more complex than in the 1-D setting; new methodology is thus required for working with these invariants in practice.

In the TDA community, it is widely appreciated that there is a need for practical data analysis tools for handling multidimensional persistent homology. However, whereas the community has been quick to develop fast algorithms and good publicly available software for 1-D persistent homology, it has been comparatively slow to extend these to the multi-D setting. Indeed, to date there is, to the best of our knowledge, no publicly available software which extends the usual persistent homology pipeline for exploratory data analysis to multidimensional persistence.

This work seeks to address this gap in the case of 2-D persistence: building on ideas presented in [13] and [14], we introduce a practical tool for working with 2-D persistent homology in exploratory data analysis applications, and develop mathematical and algorithmic foundations for this tool. Our tool can be used in much the same way that 1-D persistent homology is used in TDA, but offers the user significantly more information and flexibility than standard 1-D persistent homology does.

Our tool, which we call RIVET (the Rank Invariant Visualization and Exploration Tool), is *interactive*–it allows the user to dynamically navigate a two-parameter collection of persistence barcodes derived from a 2-D persistence module. This is in contrast to previous visualization tools for persistent homology, which have presented static displays. Because the invariants considered in this paper are larger and more complex than the standard 1-D persistence invariants, it is essential for the user have some nice way of "browsing" the invariant on the computer screen. We expect that as TDA moves towards the use of richer invariants in practical applications, interactive visualization paradigms will play an increasingly prominent role in the TDA workflow.

It should be possible to extend our approach for 2-D persistence to 3-D, at a compu-

tational cost. However, there are a number of practical challenges in this, not the least of which is designing and implementing a suitable graphical user interface. Since there is already plenty to keep us busy in 2-D, we restrict attention to the 2-D case in this paper.

In the remainder of this section, we review multidimensional persistent homology, introduce the RIVET visualization paradigm, and provide an overview of RIVET's mathematical and computational underpinnings. Given the length of this paper, we expect that some readers will be content to limit their first reading to this introduction. We invite you to begin this way.

Availability of the RIVET Software. We plan to make our RIVET software publicly available at http://rivet.online within the next few months. In the meantime, a demo of RIVET can be accessed through the website.

1.2 Multidimensional Filtrations and Persistence Modules

We start our introduction to RIVET by defining multidimensional filtrations and persistence modules, and reviewing the standard 1-D persistent homology pipeline for TDA. Here and throughout, we freely use basic language from category theory. An accessible introduction to such language in the context of persistence theory can be found in [8].

Notation. For categories $\mathcal{C}$ and $\mathcal{D}$, let $\mathcal{D}^{\mathcal{C}}$ denote the category whose objects are functors $\mathcal{C} \to \mathcal{D}$ and whose morphisms are natural transformations.

For $\mathcal{C}$ a poset category, $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ a functor, and $c \in \mathrm{obj}(\mathcal{C})$, let $\mathcal{F}_c = \mathcal{F}(c)$, and for $c \le d \in \mathrm{obj}(\mathcal{C})$, let $\mathcal{F}(c, d) : \mathcal{F}_c \to \mathcal{F}_d$ denote the image under $\mathcal{F}$ of the unique morphism in $\hom_{\mathcal{C}}(c, d)$. Let **Simp** denote the category of simplicial complexes and simplicial maps. For a fixed field K, let **Vect** denote the category of K-vector spaces and linear maps.

Define a partial order on $\mathbb{R}^n$ by taking

$$(a_1, \ldots, a_n) \le (b_1, \ldots b_n)$$

if and only if $a_i \le b_i$ for all i, and let $\mathbf{R}^n$ denote the corresponding poset category.

For S a set, we let $S \cup \infty$ denote the set $S \cup \{\infty\}$.

Finite Multidimensional Filtrations. Define an *n-D filtration* to be a functor $\mathcal{F} : \mathbf{R}^n \to \mathbf{Simp}$ such that for all $a \le b$, $\mathcal{F}(a, b) : \mathcal{F}_a \to \mathcal{F}_b$ is an inclusion. We will usually refer to a 2-D filtration as a *bifiltration*. *n-D* filtrations are the basic topological objects of study in multidimensional persistence.

In the computational setting, we of course work with filtrations that are specified by a finite amount of data. Let us now introduce language and notation for such filtrations.

We say an *n-D* filtration $\mathcal{F}$ *stabilizes* if there exists $a_0 \in \mathbb{R}^n$ such that $\mathcal{F}_a = \mathcal{F}_{a_0}$ whenever $a \ge a_0$. We write $\mathcal{F}_{\max} = \mathcal{F}_{a_0}$. We say a simplex s in $\mathcal{F}_{\max}$ *appears a finite number of times* if there is a finite set $A \in \mathbb{R}^n$ such that

1. for each $a \in A$, $s \in \mathcal{F}_a$, and

2. for each $b \in \mathbb{R}^n$ with $s \in \mathcal{F}_b$, there is some $a \in A$ with $a \leq b$.

If s appears a finite number of times, then a minimal such A is unique; we denote it $A(s)$, and call it the set of *grades of appearance* of s.

We say an n-D filtration $\mathcal{F}$ is *finite* if

1. $\mathcal{F}$ stabilizes,

2. $\mathcal{F}_{\max}$ is finite, and

3. each simplex $s \in \mathcal{F}_{\max}$ appears a finite number of times.

For $\mathcal{F}$ finite, we define $|\mathcal{F}|$, the *size of $\mathcal{F}$*, by

$$|\mathcal{F}| = \sum_{s \in \mathcal{F}_{\max}} |A(s)|.$$

In the computational setting, we can represent a finite n-D filtration $\mathcal{F}$ in memory by storing the simplicial complex $\mathcal{F}_{\max}$, along with the set $A(s)$ for each $s \in \mathcal{F}_{\max}$.

Multidimensional Persistence Modules. Define an *n-D persistence module* to be a functor $M : \mathbf{R}^n \to \mathbf{Vect}$. We say M is *pointwise finite dimensional (p.f.d.)* if $\dim(M_a) < \infty$ for all $a \in \mathbb{R}^n$. As we explain in Section 2.1, the category $\mathbf{Vect}^{\mathbf{R}^n}$ of persistence modules is isomorphic to a category of multi-graded modules over a suitable ring. In view of this, we may define presentations of n-D persistence modules; see Section 2.2 for details.

For $i \geq 0$, let $H_i : \mathbf{Simp} \to \mathbf{Vect}$ denote the ith simplicial homology functor with coefficients in K. For $\mathcal{F}$ a finite n-D filtration, there exists a finite presentation for $H_i\mathcal{F}$, which implies in particular that $H_i\mathcal{F}$ is p.f.d..

Barcodes of 1-D Persistence Modules. [27] shows that p.f.d. 1-D persistent homology modules decompose in an essentially unique way into indecomposable summands, and that the isomorphism classes of these summands are parameterized by (non-empty) intervals in $\mathbb{R}$.

Thus, we may associate to each p.f.d. 1-D persistence module M a multiset $\mathcal{B}(M)$ of intervals in $\mathbb{R}$, which records the isomorphism classes of indecomposable summands of M. We call $\mathcal{B}(M)$ the *barcode* of M; in general, we refer to any multiset of intervals as a barcode.

The barcode of a finitely presented 1-D persistence module consists of a finite set of intervals of the form $[a, b)$, with $a \in \mathbb{R}$, $b \in \mathbb{R} \cup \infty$. A barcode of this form can be represented as a *persistence diagram*, i.e., a multiset of points in $(a, b) \in \mathbb{R} \times (\mathbb{R} \cup \infty)$ with $a < b$. The right side of Fig. 1 depicts a barcode, together with its corresponding persistence diagram.
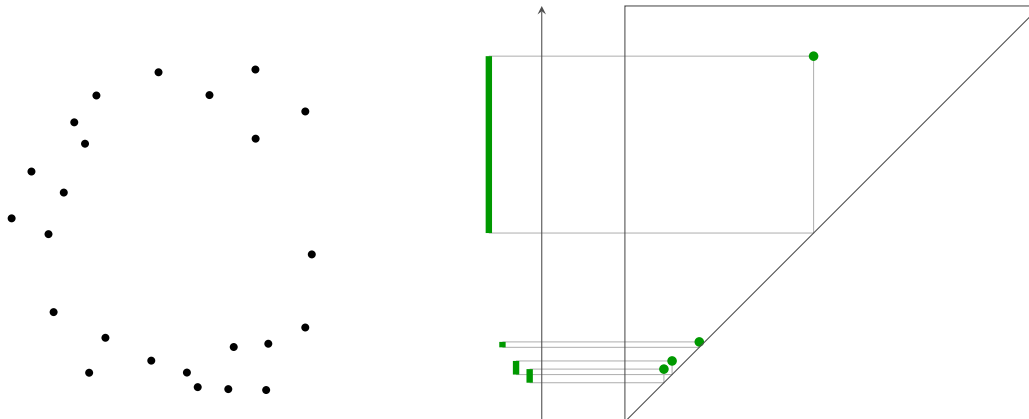
Figure 1: A point cloud circle (left) and its 1$^{\text{st}}$ persistence barcode (right), obtained via the Vietoris-Rips construction. A barcode can be visualized either by directly plotting each interval (green bars, oriented vertically) or a via persistence diagram (green dots, right). The single long interval in the barcode encodes the presence of a cycle in the point cloud.

1.3   Persistence Barcodes of Data

The standard 1-D persistent homology pipeline for data analysis associates a barcode $\mathcal{B}_i(P)$ to a data set P, for each $i \geq 0$. We regard each interval of each barcode $\mathcal{B}_i(P)$ as a topological feature of the data, and we interpret the length of the interval as a measure of robustness of the feature. The pipeline for constructing $\mathcal{B}_i(P)$ proceeds in three steps:

1. We associate to the data set P a finite 1-D filtration $\mathcal{F}(P)$.

2. We apply H_i to obtain a finitely presented 1-D persistence module $H_i\mathcal{F}(P)$.

3. We take $\mathcal{B}_i(P) = \mathcal{B}(H_i\mathcal{F}(P))$.

This pipeline for the construction of barcodes of data is quite flexible: in principle, we may work with a data set P of any kind, and may consider any number of choices of the filtration $\mathcal{F}(P)$. Different choices of $\mathcal{F}(P)$ topologically encode different aspects of the structure of our data.

The barcodes $\mathcal{B}_i(P)$ are readily computed in practice; see [47] or Section 6.2 for details.

Persistence Barcodes of Finite Metric Spaces.   Here is one standard choice of P and $\mathcal{F}(P)$ in the TDA pipeline: Let P be a finite metric space, and let $\mathcal{F}(P) = \mathrm{Rips}(P)$, the *Vietoris-Rips filtration* of P, be defined as follows. For $t \geq 0$, $\mathrm{Rips}(P)_t$ is the unique largest simplicial complex with 0-skeleton P, and 1-simplices the pairs $[p, q]$ with $d(p, q) \leq 2t$; for $t < 0$, $\mathrm{Rips}(P)_t$ is the empty simplicial complex.

5

Informally, the long intervals in the barcodes $\mathcal{B}_i(P)$ correspond to cycles in the data; see Fig. 1 for an illustration.

Stability. The following well-known result shows that these barcode invariants of finite metric spaces are robust to certain perturbations of the data:

Theorem 1.1 (Stability of Barcodes of Finite Metric Spaces [18]). *For all finite metric spaces P, Q and $i \geq 0$,*

$$d_{GH}(P, Q) \geq d_B(\mathcal{B}_i(P), \mathcal{B}_i(Q)),$$

where d_{GH} denotes Gromov-Hausdorff distance and d_B denotes the bottleneck distance on barcodes; see [18] for definitions.

See also [21] for a generalization of this theorem to compact metric spaces.

1.4 Multidimensional Persistent Homology

In many cases of interest, a 1-D filtration is not sufficient to capture the structure of interest in our data. In such cases, we are naturally led to associate to our data an n-D filtration, for some $n \geq 2$. As in the 1-D case, applying homology with field coefficients to an n-D filtration yields an n-D persistence module. The question then arises of whether we can associate to this n-D persistence module an n-dimensional generalization of a barcode in a way that is useful for data analysis.

In what follows, we motivate the study of multidimensional persistence by describing one natural way that bifiltrations arise in the study of finite metric spaces; other ways that bifiltrations arise in TDA applications are discussed, for example, in [13], [40], and [19]. We then discuss the algebraic difficulties involved in defining a multidimensional generalization of a barcode.

Bifiltrations of Finite Metric Spaces. In spite of Theorem 1.1, which tells us that the barcodes $\mathcal{B}_i(P) = \mathcal{B}_i(H_i \operatorname{Rips}(P))$ of a finite metric space P are well behaved in a certain sense, these invariants have a couple of important limitations. First, they are highly unstable to the addition and removal of outliers; see Fig. 2 for an illustration. Second, and relatedly, when P exhibits non-uniformities in density, the barcodes $\mathcal{B}_i(P)$ can be insensitive to interesting structure in the high density regions of P; see Fig. 2.

To address these issues, [13] proposed that we associate a bifiltration to P, and study the persistent homology of this bifiltration. We describe here both the proposal of [13], which depends on a choice of bandwidth parameter, and a simple parameter-free variant which, to the best of our knowledge, has not appeared elsewhere.

The construction of the bifiltration proposed in [13] depends on a choice of *codensity function* $\gamma : P \to \mathbb{R}$, a function on P whose value is high at dense points and low at outliers [13, 45]. For example, [13] proposes to take γ to be a K-nearest neighbors density function;

6
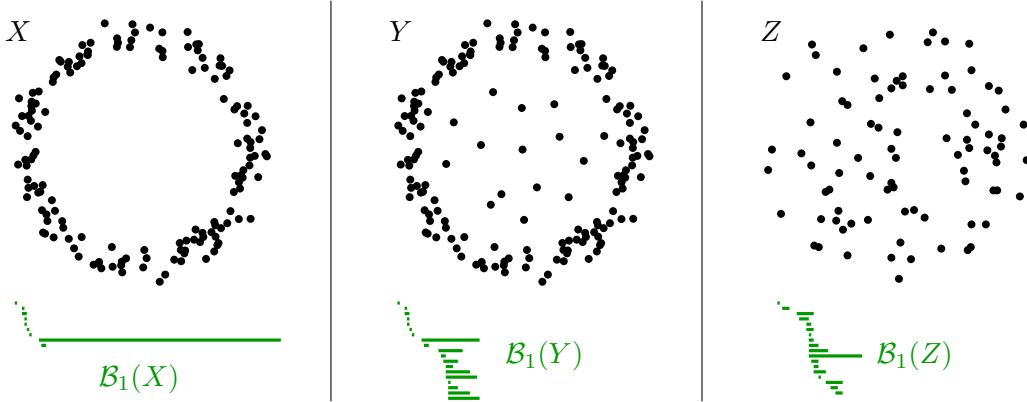
Figure 2: Barcodes are unstable with respect to addition of outliers, and can be insensitive to interesting structure in high density regions of the data. Thus, though the point clouds X and Y share a densely sampled circle and differ only by the addition of a few low-density outliers, $\mathcal{B}_1(X)$ and $\mathcal{B}_1(Y)$ are quite different from one another: $\mathcal{B}_1(X)$ has a long interval not appearing in $\mathcal{B}_1(Y)$. In contrast, the point cloud Z contains no densely sampled circle, but the longest intervals of $\mathcal{B}_1(Y)$ and $\mathcal{B}_1(Z)$ are of similar length.

in general, the choice of a density function depends on a choice of a bandwidth parameter K.

Given γ, we may define a 2-dimensional filtration $\mathrm{Rips}(\gamma)$ by taking

$$\mathrm{Rips}(\gamma)_{(a,b)} = \mathrm{Rips}(\gamma^{-1}(-\infty, a])_b.$$

If $a \geq a'$ and $b \leq b'$, then $\mathrm{Rips}(\gamma)_{(a,b)} \subseteq \mathrm{Rips}(\gamma)_{(a',b')}$. Thus, the collection of all such simplicial complexes, together with these inclusions, yields a functor

$$\mathrm{Rips}(\gamma) : \mathbf{R}^{\mathrm{op}} \times \mathbf{R} \to \mathbf{Simp}.$$

$\mathbf{R}^{\mathrm{op}} \times \mathbf{R}$ is naturally isomorphic to $\mathbb{R}^2$—for example there is an isomorphism sending each object (a,b) to $(-a,b)$—so upon the identification of these two categories, we may regard $\mathrm{Rips}(\gamma)$ as a bifiltration. Note that the definition of $\mathrm{Rips}(\gamma)$ in fact makes sense for *any* function $\gamma : P \to \mathbb{R}$. As discussed in [13], there are numerous possibilities for interesting choices of γ in our study of data, aside from density estimates.

To obtain a parameter-free variant of $\mathrm{Rips}(\gamma)$, for $(a,b) \in \mathbb{R}^2$ we first define the graph $G_{a,b}$ to be the subgraph of the 1-skeleton of $\mathrm{Rips}(P)_b$ consisting of vertices whose degree is at least a. We then define B-$\mathrm{Rips}(P)_{(a,b)}$ to be the maximal simplicial complex with 1-skeleton $G_{a,b}$. As above, upon the identification of $\mathbf{R}^{\mathrm{op}} \times \mathbf{R}$ with $\mathbf{R}^2$, the collection of simplicial complexes $\{\text{B-}\mathrm{Rips}(P)_{(a,b)}\}_{(a,b) \in \mathbb{R}^2}$ defines a bifiltration B-$\mathrm{Rips}(P)$.

H_i B-Rips(P) is a richer algebraic invariant than H_i Rips(P), and in particular is sensitive to interesting structure in high density regions of P in a way that H_i Rips(P) is not.

Barcodes of Multi-D Persistence Modules? We now explain the algebraic difficulties with defining the barcode of a n-D persistence module for $n > 1$, closely following a discussion in [40].

As for the case $n = 1$, for $n > 1$, finitely presented n-D persistence modules also decompose in an essentially unique way into indecomposable summands; this follows easily from a standard formulation of the Krull-Schmidt theorem [2]. However, it is a consequence of standard quiver theory results, as described for example in [30], that the set of isomorphism classes of indecomposable multi-D persistence modules is, in contrast to the 1-D case, extremely complicated.[1] In particular, for $n \geq 2$ the dimension of a vector space in a finitely presented indecomposable n-D persistence module can be arbitrarily large. Thus, while in principle we could define the barcode of a multi-D persistence module to be its multiset of isomorphism classes of indecomposables, as in the 1-D case, for $n > 1$ this invariant will typically not be useful for data visualization and exploration in the way that the 1-D barcode is.

In general, it seems that for the purposes of TDA, there is no entirely satisfactory way of defining the barcode of an n-D persistence module for $n > 1$, even if we consider invariants which are incomplete (i.e., invariants which can take the same value on two non-isomorphic modules).

Three Simple Invariants of a Multidimensional Persistence Module. Nevertheless, it is possible to define simple, useful, computable invariants of a multidimensional persistence module. Our tool RIVET computes and visualizes three such invariants of a 2-D persistence module M:

1. The *dimension function of M*, i.e., the function which maps $a \in \mathbb{R}^2$ to dim M_a.

2. The *fibered barcode* of M, i.e., the collection of barcodes of 1-D affine slices of M.

3. The (multigraded) Betti numbers of M.

The dimension function of M is a simple, intuitive, and easily visualized invariant, but is unstable and provides no information about persistent features in M. The next two subsections introduce the fibered barcode and the multigraded Betti numbers.

---

[1]For example, there is a fully faithful functor from the category of representations of the wild quiver E_9 to $\mathbf{Vect}^{\mathbf{R}^2}$, which maps indecomposables to indecomposables; see also [13] for a study of the possible isomorphism types of a multidimensional persistence module.

1.5 The Rank Invariant and Fibered Barcodes

The Rank Invariant. For $n \geq 1$, let $\mathcal{H}^n \subseteq \mathbb{R}^n \times \mathbb{R}^n$ denote the set of pairs (s, t) with $s \leq t$, let $\mathbb{N}$ denote the non-negative integers, and let M be an n-D persistence module. Following [13], we define

$$\mathrm{rank}(M) : \mathcal{H}^n \to \mathbb{N},$$

the *rank invariant of M*, by $\mathrm{rank}(M)(a, b) = \mathrm{rank}\, M(a, b)$. Using the structure theorem for 1-D persistence modules [27], it's easy to check that for M a p.f.d. 1-D persistence module M, $\mathrm{rank}(M)$ and $\mathcal{B}(M)$ determine each other [13].

For M an n-D persistence module, $n \geq 2$, $\mathrm{rank}(M)$ does not encode the isomorphism class of M; see Example 2.2. Nevertheless, the rank invariant does capture interesting "first order information" about the structure of a persistence module.

[14] observed that if M is a p.f.d. n-D persistence module, $\mathrm{rank}(M)$ carries the same data as a $(2n-2)$-parameter family of barcodes, each obtained by restricting M to an affine line in $\mathbb{R}^n$; we call this parameterized family of barcodes the *fibered barcode* of M. In particular, if M is a 2-D persistence module, the fibered barcode of M is a 2-parameter family of barcodes. In what follows, we give the definition of a fibered barcode of a 2-D persistence module.

The Space of Affine Lines in 2-D with Non-Negative Slope. Let $\bar{\mathcal{L}}$ denote the collection of all (unparameterized) lines in $\mathbb{R}^2$ with non-negative (possibly infinite) slope, and let $\mathcal{L} \subset \bar{\mathcal{L}}$ denote the collection of all (unparameterized) lines in $\mathbb{R}^2$ with non-negative, finite slope.

Note that $\bar{\mathcal{L}}$ is a submanifold (with boundary) of an affine Grassmannian of dimension 2. With the induced topology, $\bar{\mathcal{L}}$ is homeomorphic to $[0, 1] \times \mathbb{R}$. In this sense, it is appropriate to think of $\bar{\mathcal{L}}$ as a 2-parameter family of lines.

A standard point-line duality, to be described in Section 3.1, provides an identification between $\mathcal{L}$ and the half-plane $[0, \infty) \times \mathbb{R}$; we will make extensive use of this point-line duality later in the paper. The duality does not extend to $\bar{\mathcal{L}}$ (i.e., to vertical lines) in a natural way.

Definition of the Fibered Barcode. For $L \in \bar{\mathcal{L}}$, let $\mathbf{L}$ denote the associated full subcategory of $\mathbf{R}^2$. The inclusion $L \hookrightarrow \mathbb{R}^2$ induces a functor $i_L : \mathbf{L} \to \mathbf{R}^2$. For M a p.f.d. 2-D persistence module, we define $M^L = M \circ i_L$.

Define an *interval* I in L to be a non-empty subset of L such that whenever $a < b < c \in L$ and $a, c \in I$, we also have that $b \in I$. As $\mathbf{L}$ is isomorphic to $\mathbf{R}$, the structure theorem for p.f.d. 1-D persistence modules yields a definition of the barcode $\mathcal{B}(M^L)$ as a collection of intervals in L.

We define $\mathcal{B}(M)$, the *fibered barcode of M*, to be the map which sends each line $L \in \bar{\mathcal{L}}$ to the barcode $\mathcal{B}(M^L)$.

Proposition 1.2 ([14]). $\mathcal{B}(M)$ *and* $\mathrm{rank}(M)$ *determine each other.*

Proof. For each $a \leq b \in \mathbb{R}^2$, there exists a unique line $L \in \bar{\mathcal{L}}$ such that $a, b \in L$. Clearly then, $\mathrm{rank}(M)$ and the collection of 1-D rank invariants $\{\mathrm{rank}(M^L) \mid L \in \bar{\mathcal{L}}\}$ determine each other. As noted above, for N a p.f.d. 1-D persistence module, $\mathcal{B}(N)$ and $\mathrm{rank}(N)$ determine each other. The result follows. $\qquad\square$

Stability of the Fibered Barcode. Adapting arguments introduced in [14] and [6], a recent note by Claudia Landi [38] establishes that $\mathcal{B}(M)$ is *stable* in two senses. The results of [38] in fact hold for persistent homology modules of arbitrary dimension. However, in this paper we are primarily interested in the 2-D case.

In 2-D, Landi's first stability result says that for L a line which is neither horizontal nor vertical, the map $M \to \mathcal{B}(M^L)$ is Lipschitz continuous with respect to the interleaving distance on 2-D persistence modules and the *bottleneck distance* on barcodes. The Lipschitz constant c depends on the slope of L; $c = 1$ when the slope of L is 1, and c grows larger as the slope of L deviates from 1, tending towards infinity as the slope of L approaches 0 or ∞. We refer to this stability result as the *external stability* of $\mathcal{B}(M)$.

[38] also presents an *internal* stability result for $\mathcal{B}(M)$ which tells us that when M is finitely presented, the map $L \to \mathcal{B}(M^L)$ is continuous, in a suitable sense, at lines L which are neither horizontal nor vertical. In fact the result says something stronger, which put loosely, is that the closer the slope of the line L is to 1, the more stable $\mathcal{B}(M^L)$ is to perturbations of L.

In sum, the stability results of [38] tell us that for M a 2-D persistence module and L a line which is neither horizontal or vertical, the barcode $\mathcal{B}(M^L)$ is robust to perturbations both of M and L; the more diagonal L is, the more robust $\mathcal{B}(M^L)$ is.

1.6 Multigraded Betti Numbers

We next briefly introduce (multigraded) Betti numbers (called bigraded Betti numbers in the case of 2-D persistence); see Section 2.4 for a precise definition and examples.

For M a finitely presented n-D persistence module and $i \geq 0$, the ith (graded) Betti number of M is a function $\xi_i(M) : \mathbb{R}^n \to \mathbb{N}$. It follows from the *Hilbert Basis Theorem*, a classical theorem of commutative algebra, that $\xi_i(M)$ is identically 0 for $i > n$, so $\xi_i(M)$ is only of interest for $i \leq n$.

We will be especially interested in $\xi_0(M)$ and $\xi_1(M)$ in this paper. For $a \in \mathbb{R}^n$, $\xi_0(M)(a)$ and $\xi_1(M)(a)$ are the number of generators and relations, respectively, at index a in a minimal presentation for M; see Section 2.3. $\xi_2(M)$ has an analogous interpretation in terms of a minimal resolution of M.

Neither of the invariants $\mathcal{B}(M)$ nor $\{\xi_i(M) \mid i = 0, 1, \ldots, n\}$ determines the other, but the invariants are intimately connected. This connection in the case that $n = 2$ plays a central role in the present work.

One of the main mathematical contributions of this project is a fast algorithm for computing the multigraded Betti numbers of a 2-D persistence module M; our algorithm is described in the companion paper [41].

1.7   The RIVET Visualization Paradigm

Overview.   We propose to use the fibered barcode in exploratory data analysis in much the same way that 1-D barcodes are typically used. In particular, this requires us to have a good way of visualizing the fibered barcode. Though discretizations of fibered barcodes have been used in shape matching applications [15], to the best of our knowledge, there is no prior work on visualization of fibered barcodes.

This work introduces a paradigm, called RIVET, for the interactive visualization of the fibered barcode of a 2-D persistence module, and presents an efficient computational framework for implementing this paradigm. Our paradigm also provides for the visualization of the dimension function and bigraded Betti numbers of the module. The visualizations of the three invariants complement each other, and work in concert: Our visualizations of the dimension function and bigraded Betti numbers provide a coarse, global view of the structure of the persistence module, while our visualization of the fibered barcodes, which focuses on a single barcode at a time, provides a sharper but more local view.

We now give a brief description of our RIVET visualization paradigm. Additional details are in Appendix A.1.

Given a 2-D persistence module M, RIVET allows the user to interactively select a line $L \in \mathbb{R}^2$ via a graphical interface; the software then displays the barcode $\mathcal{B}(M^L)$. As the user moves the line L by clicking and dragging, the displayed barcode is updated in real time.

The RIVET interface consists of two main windows, the *Line Selection Window* (left) and the *Persistence Diagram Window* (right). Fig. 3 shows screenshots of RIVET for a single choice of M and four different lines L.

The Line Selection Window.   For a given finitely presented persistence module M, the Line Selection Window plots a rectangle in $\mathbb{R}^2$ containing the union of the supports of functions $\xi_i(M)$, $i \in \{0, 1, 2\}$.

The greyscale shading at a point a in this rectangle represents $\dim M_a$: a is unshaded when $\dim M_a = 0$, and larger $\dim M_a$ corresponds to darker shading. Scrolling the mouse over a brings up a popup box which gives the precise value of $\dim M_a$.

Points in the supports of $\xi_0(M)$, $\xi_1(M)$, and $\xi_2(M)$ are marked with green, red, and yellow dots, respectively. The area of each dot is proportional to the corresponding function value. The dots are translucent, so that, for example, overlaid red and green dots appear brown on their intersection. This allows the user to read the values of the Betti numbers at points which are in the support of more than one of the functions.

11
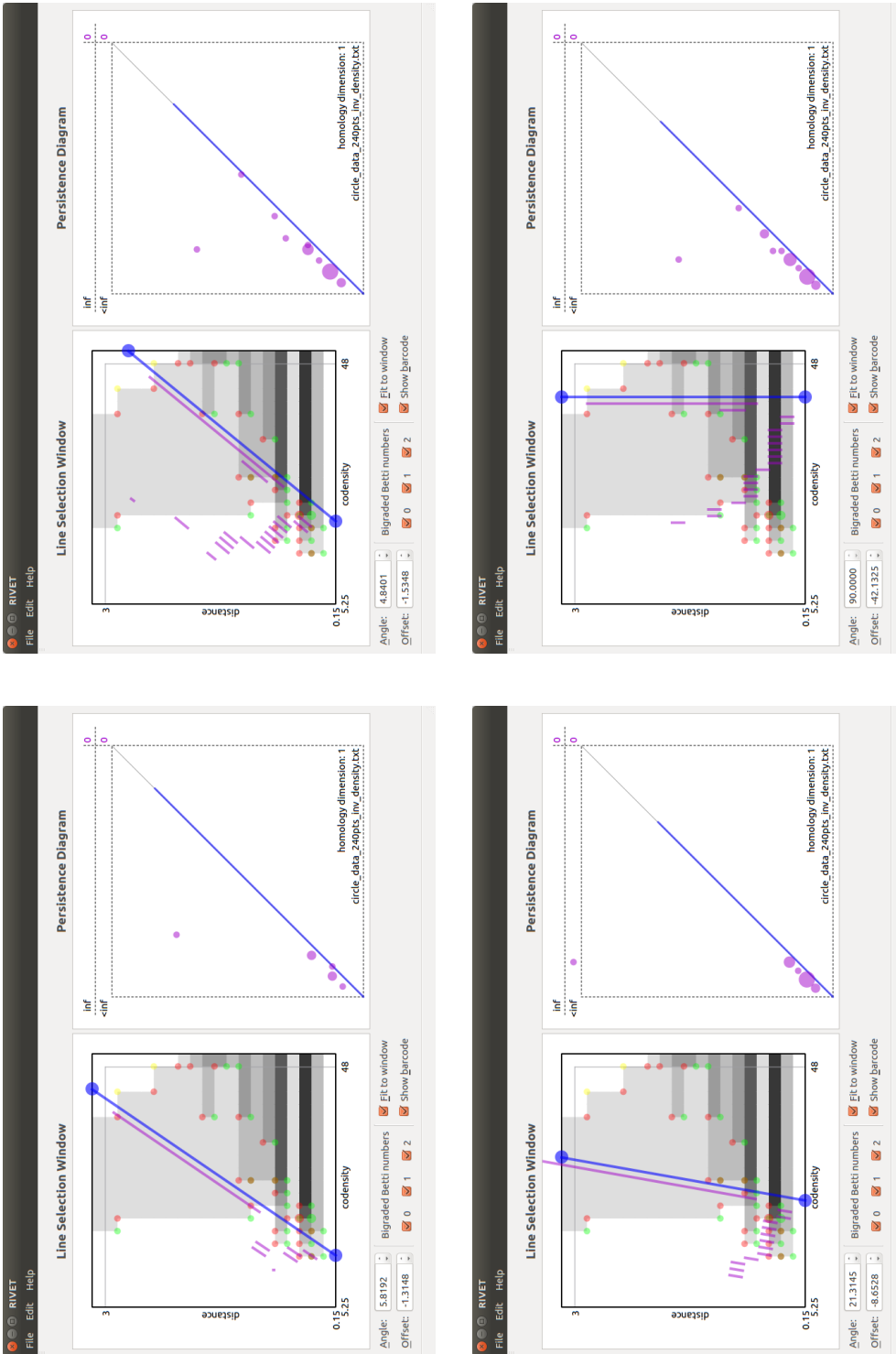
Figure 3: Screenshots of RIVET for a single choice of 2-D persistence module M and four different lines L. RIVET provides visualizations of the dimension of each vector space in M (greyscale shading); the 0th, 1st, and 2nd bi-graded Betti numbers of M (green, red, and yellow dots); and the barcodes of the 1-D slices M^L, for each L (in purple).

The line selection window contains a blue line of non-negative slope, with endpoints on the boundary of the displayed region of $\mathbb{R}^2$. This line represents a choice of $L \in \bar{\mathcal{L}}$. The intervals in the barcode $\mathcal{B}(M^L)$ are displayed in purple, offset from the line L in the perpendicular direction.

The Persistence Diagram Window. In the Persistence Diagram Window (right), a persistence diagram representation of $\mathcal{B}(M^L)$ is displayed. To represent $\mathcal{B}(M^L)$ via a persistence diagram, we need to choose an order-preserving parameterization $\gamma_L : \mathbb{R} \to L$ of L, which we regard as a functor; we then display $\mathcal{B}(M^L \circ \gamma_L)$. Our choice of the parameterizations γ_L is described in Appendix A.1.

As with the multi-graded Betti numbers, the multiplicity of a point in the persistence diagram is indicated by the area of the corresponding dot.

Interactivity. The user can click and drag the blue line in the left window with the mouse, thereby changing the choice of $L \in \bar{\mathcal{L}}$. Clicking the blue line away from its endpoints and dragging moves the line in the direction perpendicular to its slope, while keeping the slope constant. The clicking and dragging an endpoint of the line moves that endpoint while keeping the other fixed; this allows the user to change the slope of the line.

As the line moves, the both the interval representation of $\mathcal{B}(M^L)$ in the left window and its persistence diagram representation in the right window are updated in real time.

1.8 Querying the Fibered Barcode

Our algorithm for fast computation of Betti numbers of 2-D persistence modules, described in [41], performs an efficient computation of the dimension function of M as a subroutine. Thus, in explaining the computational underpinnings of RIVET, we will focus on the RIVET's interactive visualization of the fibered barcode.

Because our visualization paradigm needs to update the plot of $\mathcal{B}(M^L)$ in real time as we move the line L, it must be able to very quickly access $\mathcal{B}(M^L)$ for any choice of $L \in \bar{\mathcal{L}}$. In this paper, we introduce an efficient data structure $\mathcal{A}^\bullet(M)$, the *augmented arrangement* of M, on which we can perform fast queries to determine $\mathcal{B}(M^L)$, for $L \in \bar{\mathcal{L}}$. We present a theorem which guarantees that our query procedure correctly recovers $\mathcal{B}(M^L)$, and describe an efficient algorithm for computing $\mathcal{A}^\bullet(M)$.

Structure of the Augmented Arrangement. $\mathcal{A}^\bullet(M)$ consists of a line arrangement $\mathcal{A}(M)$ in $[0, \infty) \times \mathbb{R}$ (that is, a cell decomposition of $[0, \infty) \times \mathbb{R}$ induced by a set of intersecting lines), together with a collection $\mathcal{T}^e$ of pairs $(a, b) \in \mathbb{R}^2 \times (\mathbb{R}^2 \cup \infty)$ stored at each 2-cell e of $\mathcal{A}(M)$. We call $\mathcal{T}^e$ the *barcode template* at e. As we explain in Section 3.3, $\mathcal{T}^e$ is defined in terms of the barcode of a discrete 1-D persistence module derived from M.

Queries of $\mathcal{A}^\bullet(M)$. We now briefly describe how we query $\mathcal{A}^\bullet(M)$ for the barcodes $\mathcal{B}(M^L)$. Further details are given in Section 4.

As noted above, a standard point-line duality, described in Section 3.1, provides an identification of $\mathcal{L}$ with $[0, \infty) \times \mathbb{R}$. For simplicity, let us restrict attention for now to the generic case where L lies in a 2-cell e of $\mathcal{A}(M)$; the general case is similar and is treated in Section 4. To obtain $\mathcal{B}(M^L)$, for each pair $(a, b) \in \mathcal{T}^e$, we "push" the points of each pair $(a, b) \in \mathcal{T}^e$ onto the line L, by moving a and b upwards or rightwards in the plane along horizontal and vertical lines; this gives a pair of points $\mathrm{push}_L(a), \mathrm{push}_L(b) \in L$; if $b = \infty$, we take $\mathrm{push}_L(b) = \infty$.

Our Theorem 4.1, the central mathematical result underlying the RIVET paradigm, tells us that

$$\mathcal{B}(M^L) = \{\, [\mathrm{push}_L(a),\, \mathrm{push}_L(b)) \mid (a, b) \in \mathcal{T}^e \,\};$$

see Fig. 4 for an illustration. Thus, to obtain the barcode of $\mathcal{B}(M^L)$ it suffices to identify the cell e and then compute $\mathrm{push}_L(a)$ and $\mathrm{push}_L(b)$ for each $(a, b) \in \mathcal{T}^e$.
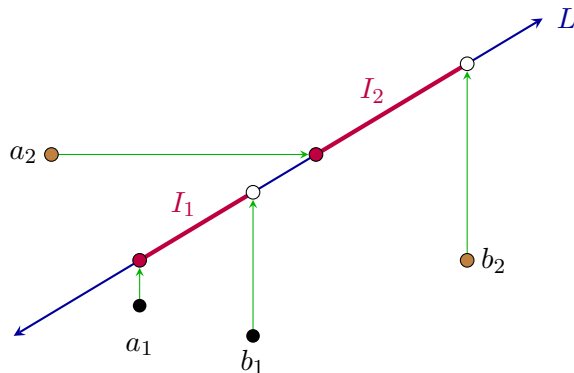
Figure 4: An illustration of how we recover the barcode $\mathcal{B}(M^L)$ from the barcode template $\mathcal{T}^e$ by "pushing" the points of each pair in the barcode template onto L. In this example, $\mathcal{T}^e = \{(a_1, b_1), (a_2, b_2)\}$, and $\mathcal{B}(M^L)$ consists of two disjoint intervals I_1, I_2.

1.9 Complexity Results for Computing, Storing, and Querying The Augmented Arrangement

Once $\mathcal{A}^\bullet(M)$ has been computed, computing $\mathcal{B}(M^L)$ via a query to $\mathcal{A}^\bullet(M)$ is far more efficient than computing $\mathcal{B}(M^L)$ from scratch; for typical persistence modules arising from data, the query of $\mathcal{B}(M^L)$ can be performed in real time, as desired. At the same time, the cost of computing and storing $\mathcal{A}^\bullet(M)$ is reasonable. The following two theorems provide a theoretical basis for these claims.

For M a 2-D persistence module and $i \in \{0, 1, 2\}$, let

$$\operatorname{supp} \xi_i(M) = \{a \in \mathbb{R}^n \mid \xi_i(M)(a) > 0\}.$$

Let $\kappa = \kappa_x \kappa_y$, for κ_x and κ_y the number of unique x and y coordinates, respectively, of points in $\operatorname{supp} \xi_0(M) \cup \operatorname{supp} \xi_1(M)$; we call κ the *coarseness of M*.

Theorem 1.3 (computational cost of querying the augmented arrangement)**.**

(i) *For $L \in \mathcal{L}$ lying in a 2-cell of $\mathcal{A}(M)$, we can query $\mathcal{A}^\bullet(M)$ for $\mathcal{B}(M^L)$ in time $O(\log \kappa + |\mathcal{B}(M^L)|)$, where $|\mathcal{B}(M^L)|$ denotes the number of intervals in $\mathcal{B}(M^L)$.*

(ii) *For all other lines $L \in \bar{\mathcal{L}}$, we can query $\mathcal{A}^\bullet(M)$ for $\mathcal{B}(M^L)$ in time $O(\log \kappa + |\mathcal{B}(M^{L'})|)$, for L' some arbitrarily small perturbation of L.*

Theorem 1.4. *For $\mathcal{F}$ a bifiltration of size m and $M = H_i(\mathcal{F})$,*

(i) *$\mathcal{A}^\bullet(M)$ is of size $O(m\kappa^2)$,*

(ii) *Our algorithm computes $\mathcal{A}^\bullet(M)$ from $\mathcal{F}$ using $O(m^3 \kappa + (m + \log \kappa)\kappa^2)$ elementary operations and $O(m^2 + m\kappa^2)$ storage.*

We prove Theorem 1.3 in Section 4.3 and Theorem 1.4 in Section 7.

To keep our exposition brief in this introduction, we have assumed in the statement of Theorem 1.4 that M is a persistent homology module of a bifiltration. However, our algorithm for computing augmented arrangements does handle purely algebraic input; see Section 5.1. We give more general complexity bounds in the algebraic setting in Section 7.

Coarsening and the Interpretation of Theorem 1.4. For a fixed choice of L, $\mathcal{B}(M^L)$ is of size $O(m)$, and the time required to compute $\mathcal{B}(M^L)$ via an application of the standard persistence algorithm is $O(m^3)$. Thus Theorem 1.4 indicates that, as one would expect, computation and storage of $\mathcal{A}^\bullet(M)$ is more expensive than computation and storage of $\mathcal{B}(M^L)$ for some fixed L.

For m as above, $|\kappa|$ is $O(m^2)$ in the worst case. Thus, in the worst case, our bounds on the size and time to compute $\mathcal{A}^\bullet(M)$ grow like m^5, which on the surface may appear problematic for practical applications. However, as we explain in Section 8.2, we can always employ a simple coarsening procedure to approximate M by a module M' for which κ is a small constant (say, $\kappa \le 400$). $\mathcal{A}^\bullet(M')$ encodes $\mathcal{B}(M')$ exactly, and so in view of Landi's external stability result [38], $\mathcal{A}^\bullet(M)$ encodes $\mathcal{B}(M)$ approximately. More details on coarsening are given in Section 8.2.

1.10 Computation of the Augmented Arrangement

Our algorithm for computing $\mathcal{A}^\bullet(M)$ decouples into three main parts:

1. Computing $\xi_0(M)$ and $\xi_1(M)$,

2. Constructing the line arrangement $\mathcal{A}(M)$,

3. Computing the barcode template $\mathcal{T}^e$ at each 2-cell e of $\mathcal{A}(M)$.

We next say a few words about each of these.

Computing Bigraded Betti Numbers. As noted above, one of the main mathematical contributions underlying RIVET is a fast algorithm for computing the bigraded Betti numbers of a 2-D persistence module M. Not only does RIVET provide a visualization of the Betti numbers, but it also makes essential use of the Betti numbers in constructing $\mathcal{A}^\bullet(M)$.

For M a persistent homology module of a bifiltration with n simplices, our algorithm for computes the bigraded Betti numbers of M in $O(m^3)$ time; see [41] and Section 5.3.

In Section 9 of this paper, we present preliminary experimental results on the performance of our algorithm for computing bigraded Betti numbers. These indicate that the cost of the algorithm is very reasonable in practice.

Computing the Line Arrangement. The second phase of computation constructs the line arrangement $\mathcal{A}(M)$ underlying $\mathcal{A}^\bullet(M)$. Line arrangements have been the object of intense study by computational geometers for decades, and there is well-developed machinery for constructing and working with line arrangements in practice [29]. Our algorithms for constructing and querying $\mathcal{A}^\bullet(M)$ leverage this machinery; see Section 4.3 and Section 5.

Computing the Barcode Templates. The third phase of our computation of $\mathcal{A}^\bullet(M)$ computes the barcode templates $\mathcal{T}^e$ stored at each 2-cell $e \in \mathcal{A}(M)$. As noted above, each $\mathcal{T}^e$ is defined in terms of the barcode $\mathcal{B}(M^e)$ of a certain 1-D persistence module M^e. To compute each $\mathcal{T}^e$ for each 2-cell e, we need to compute each $\mathcal{B}(M^e)$. This is the most expensive part of the computation of $\mathcal{A}^\bullet(M)$, both in theory and in practice. In Section 6, we introduce our core algorithm for this, based on the vineyard algorithm for updating persistent homology computations [25]. In Section 8.1, we present a modification of this algorithm which is much faster in practice.

In fact, as explained in Section 8.3, our algorithm for computing barcode templates is embarrassingly parallelizable.

Computational Experiments. Section 9 presents preliminary results on the performance of our algorithm for computing augmented arrangements. As explained there, our present implementation of RIVET is not yet fully optimized, and our timing results should be regarded as loose upper bounds on what can be achieved using the algorithms of this paper.

Still, the results demonstrate that even with our current code, computation of an augmented arrangement of a bifiltration containing millions of simplices is feasible on a standard personal computer, provided we employ some modest coarsening. Thus, the current implementation already performs well enough to be used in the analysis of modestly sized real world data sets. With more implementation work, including the introduction of parallelization, we expect RIVET to scale well enough to be useful in many of the same settings where 1-D persistence is currently used for exploratory data analysis.

1.11   Outline

We conclude this introduction with an outline of the remainder of the paper.

Section 2 reviews basic algebraic facts about persistence modules, their (minimal) presentations, and graded Betti numbers. We also discuss the connection between $\mathbb{R}^n$-indexed persistence modules and their $\mathbb{Z}^n$-indexed discretizations. Section 3 defines the augmented arrangement $\mathcal{A}^{\bullet}(M)$ of a 2-D persistence module M. Sections 4.1 and 4.2 give our main result on querying $\mathcal{A}^{\bullet}(M)$ for the barcodes $\mathcal{B}(M^L)$; this is Theorem 4.1. In Section 4.3, we describe how $\mathcal{A}^{\bullet}(M)$ is stored in memory, and apply Theorem 4.1 to give an algorithm for querying $\mathcal{A}^{\bullet}(M)$.

The remaining sections introduce our algorithm for computing $\mathcal{A}^{\bullet}(M)$: First, Section 5 specifies how persistence modules are represented as input to our algorithm, and explains our algorithm for computing $\mathcal{A}(M)$. Section 6 explains our core algorithm for computing the barcode templates $\mathcal{T}^e$; this completes the specification of our algorithm for computing $\mathcal{A}^{\bullet}(M)$, in its basic form. Section 7 analyzes the time and space complexity of our algorithm for computing $\mathcal{A}^{\bullet}(M)$. Section 8 describes several practical strategies to speed up the computation of $\mathcal{A}^{\bullet}(M)$. Section 9 presents our preliminary timing results for the computation of $\mathcal{A}^{\bullet}(M)$.

Appendix A.1 expands on the introduction to the RIVET interface given in Section 1.7, providing additional details.

2   Algebra Preliminaries

In this section, we present the basic algebraic definitions and facts we will need to define and study augmented arrangements of 2-D persistence modules.

2.1 A Module-Theoretic Description of Multi-D Persistence Modules

In Section 1.2 we defined a n-D persistence module to be an object of the functor category $\mathbf{Vect}^{\mathbf{R}^n}$. Here we give a module-theoretic description of $\mathbf{Vect}^{\mathbf{R}^n}$.

The Ring P_n.   Let the ring P_n be the analogue of the usual polynomial ring $K[x_1, \ldots, x_n]$ in n variables, where exponents of the indeterminates in P_n are allowed to take on arbitrary values in $[0, \infty)$ rather than only values in $\mathbb{N}$. For example, if $K = \mathbb{R}$, then $1 + x_2 + x_1^\pi + \frac{2}{5} x_1^3 x_2^{\sqrt{2}}$ is an element of P_2.

More formally, P_n can be defined as a *monoid ring* over the monoid $([0, \infty)^n, +)$ [39].

For $a = (a_1, \ldots, a_n) \in [0, \infty)^n$, we let x^a denote the monomial $x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n} \in P_n$. Let $I \subset P_n$ be the ideal generated by the set $\{x_i^\alpha \mid \alpha > 0, 1 \leq i \leq n\}$.

$\mathbb{R}^n$-graded P_n-Modules.   Since the field K is a subring of P_n, any P_n-module comes naturally equipped with the structure of a K-vector space.

We define an $\mathbb{R}^n$-*graded* P_n-module to be a P_n-module M with a direct sum decomposition as a K-vector space $M \simeq \bigoplus_{a \in \mathbb{R}^n} M_a$ such that the action of P_n on M satisfies $x^b(M_a) \subset M_{a+b}$ for all $a \in \mathbb{R}^n$, $b \in [0, \infty)^n$.

The $\mathbb{R}^n$-graded P_n-modules form a category whose morphisms are the module homomorphisms $f : M \to N$ such that $f(M_a) \subset N_a$ for all $a \in \mathbb{R}^n$. There is an obvious isomorphism between this $\mathbf{Vect}^{\mathbf{R}^n}$ and this category, so that we may identify the two categories. Henceforth, we will refer to $\mathbb{R}^n$-graded P_n-modules as (n-D) persistence modules, or n-modules for short.

Remark 2.1. As a rule, the familiar definitions and constructions for modules make sense in the category $\mathbf{Vect}^{\mathbf{R}^n}$. For example, as the reader may check, we can define submodules, quotients, direct sums, tensor products, projective/injective resolutions, and Tor functors in $\mathbf{Vect}^{\mathbf{R}^n}$. As we next explain, we also can define free n-modules and presentations of n-modules.

2.2 Free n-Modules and Presentations

n-graded Sets.   Define an n-*graded set* to be a pair $\mathcal{W} = (W, \mathrm{gr}_{\mathcal{W}})$ for some set W and function $\mathrm{gr}_{\mathcal{W}} : W \to \mathbb{R}^n$. Formally, we may regard $\mathcal{W}$ as the set of pairs

$$\{(w, \mathrm{gr}_{\mathcal{W}}(w)) \mid w \in W\},$$

and we'll sometimes make use of this representation. We'll often abuse notation and write $\mathcal{W}$ to mean the set W. Also, when $\mathcal{W}$ is clear from context we'll abbreviate $\mathrm{gr}_{\mathcal{W}}$ as gr.

We say a subset $\mathcal{Y}$ of an n-module M is *homogeneous* if

$$\mathcal{Y} \subseteq \bigcup_{a \in \mathbb{R}^n} M_a.$$

Clearly, we may regard $\mathcal{Y}$ as an n-graded set.

Shifts of Modules. For M an n-module and $v \in \mathbb{R}^n$, we define $M(v)$ to be the n-module such that for $a \in \mathbb{R}^n$, $M(v)_a = M_{a+v}$, and for $a \leq b \in \mathbb{R}^n$, $M(v)(a,b) = M(a+v, b+v)$.

For example, when $n = 2$, $M(1,1)$ is obtained from M by shifting all vector spaces of M down by one and to the left by one.

Free n-Modules. The usual notion of a free module extends to the setting of n-modules as follows: For $\mathcal{W}$ an n-graded set, let $\mathrm{Free}[\mathcal{W}] = \bigoplus_{w \in \mathcal{W}} P_n(-\mathrm{gr}(w))$. We identify $\mathcal{W}$ with a set of generators in $\mathrm{Free}[\mathcal{W}]$, in the obvious way. A *free n-module* F is an n-module such that $F \simeq \mathrm{Free}[\mathcal{W}]$ for some n-graded set $\mathcal{W}$. Equivalently, we can define a free n-module as an n-module which satisfies a certain universal property; see [13].

For $\mathcal{Y}$ a homogeneous subset of a free n-module F, let $\langle \mathcal{Y} \rangle$ denote the submodule of F generated by $\mathcal{Y}$.

Matrix Representations of Morphisms of Free Modules. Let $\mathcal{W}, \mathcal{W}'$ be finite n-graded sets with ordered underlying sets $W = \{w_1, \ldots, w_l\}$, $W' = \{w'_1, \ldots, w'_m\}$. For a morphism $f : \mathrm{Free}[\mathcal{W}] \to \mathrm{Free}[\mathcal{W}']$, we can represent f by a $|W| \times |W'|$ matrix $[f]$ with coefficients in K: If $\mathrm{gr}(w'_i) \leq \mathrm{gr}(w_j)$, we define $[f]_{ij}$ to be the unique solution to

$$[f]_{ij}\, w'_i = x^{\mathrm{gr}(w'_i) - \mathrm{gr}(w_j)} \rho_i \circ f(w_j),$$

where $\rho_i : \mathrm{Free}[\mathcal{W}] \to \mathrm{Free}[\{w_i\}]$ is the projection. If $\mathrm{gr}(w'_i) \not\leq \mathrm{gr}(w_j)$, we define $[f]_{ij} = 0$.

Presentations of n-Modules. A *presentation* of an n-module M is a pair $(\mathcal{W}, \mathcal{Y})$, where $\mathcal{W}$ is an n-graded set and $\mathcal{Y} \subseteq \mathrm{Free}[\mathcal{W}]$ is a homogeneous, with $M \simeq \mathrm{Free}[\mathcal{W}]/\langle \mathcal{Y} \rangle$. We denote the presentation $(\mathcal{W}, \mathcal{Y})$ as $\langle \mathcal{W} | \mathcal{Y} \rangle$. If there exists a presentation $\langle \mathcal{W} | \mathcal{Y} \rangle$ for M with $\mathcal{W}$ and $\mathcal{Y}$ finite, then we say M is *finitely presented*.

Note that the inclusion $\mathcal{Y} \hookrightarrow \mathrm{Free}[\mathcal{W}]$ induces a morphism $\mathrm{Free}[\mathcal{Y}] \to \mathrm{Free}[\mathcal{W}]$; we denote this morphism as $\Phi_{\langle \mathcal{W} | \mathcal{Y} \rangle}$.

Example 2.2. Consider the 2-D persistence modules

$$M \simeq \langle (a, (1,0)), (b, (0,1)), (c, (1,1)) \mid x_2 a - x_1 b \rangle$$
$$N \simeq \langle (a, (1,0)), (b, (0,1)) \mid \emptyset \rangle.$$

The induced linear maps $M_{(1,0)} \oplus M_{(0,1)} \to M_{(1,1)}$ and $N_{(1,0)} \oplus N_{(0,1)} \to N_{(1,1)}$ do not have equal ranks. Hence M and N are not isomorphic. However, $\mathrm{rank}(M) = \mathrm{rank}(N)$. This shows that the rank invariant does not completely determine the isomorphism type of a persistence module.

2.3   Minimal Presentations of n-Modules

Let M be an n-module. We say a presentation $\langle \mathcal{W} | \mathcal{Y} \rangle$ of M is *minimal* if $\langle \mathcal{Y} \rangle \subseteq I \operatorname{Free}[\mathcal{W}]$ and $\ker \Phi_{\langle \mathcal{W} | \mathcal{Y} \rangle} \subseteq I \operatorname{Free}[\mathcal{Y}]$.

The following proposition is a variant of [33, Lemma 19.4], and is proved in the same way. It makes clear that minimal presentations are indeed minimal, in a reasonable sense.

Proposition 2.3. *A finite presentation $\langle \mathcal{W} | \mathcal{Y} \rangle$ is minimal if and only if $\mathcal{W}$ descends to a minimal set of generators for $\operatorname{coker} \Phi_{\langle \mathcal{W} | \mathcal{Y} \rangle}$ and $\mathcal{Y}$ is a minimal set of generators for $\langle \mathcal{Y} \rangle$.*

Remark 2.4. It follows immediately from Proposition 2.3 that every finitely presented n-D persistence module has a minimal presentation.

2.4   Graded Betti Numbers of Persistence Modules

For M an n-module, define $\operatorname{dimf}(M) : \mathbb{R}^n \to \mathbb{N}$, the *dimension function of M*, by $\operatorname{dimf}(M)(a) = \dim(M_a)$.

For $i \geq 0$, define
$$\xi_i(M) = \operatorname{dimf}(\operatorname{Tor}_i(M, P_n / IP_n)).$$

The functions $\xi_i(M)$ are called the *(graded) Betti numbers* of M. Betti numbers of multi-graded $K[x_1, \cdots, x_n]$-modules are defined analogously; these are discussed in many places, e.g., in [13]. In our study of augmented arrangements and fibered barcodes, we will only need to consider $\xi_0(M)$ and $\xi_1(M)$.

We omit the straightforward proof the following result:

Proposition 2.5. *If $\langle \mathcal{W} | \mathcal{Y} \rangle$ is a minimal presentation for M, then for all $a \in \mathbb{R}^n$,*

$$\xi_0(M)(a) = |\operatorname{gr}_{\mathcal{W}}^{-1}(a)|, \qquad \xi_1(M)(a) = |\operatorname{gr}_{\mathcal{Y}}^{-1}(a)|.$$

Example 2.6. The presentations of the modules M and N given in Example 2.2 are minimal. Using this, it's easy to see that

$$\xi_0(M)(a) = \begin{cases} 1 & \text{if } a \in \{(1,0), (0,1), (1,1)\} \\ 0 & \text{otherwise,} \end{cases} \qquad \xi_1(M)(a) = \begin{cases} 1 & \text{if } a \in \{(1,1)\} \\ 0 & \text{otherwise,} \end{cases}$$

$$\xi_0(N)(a) = \begin{cases} 1 & \text{if } a \in \{(1,0), (0,1)\} \\ 0 & \text{otherwise.} \end{cases} \qquad \xi_1(N) = \xi_2(N) = \xi_2(M) = 0.$$

Example 2.7. For $M \simeq \langle (a, (0,0)) \mid x_1 a, x_2 a \rangle$,

$$\xi_0(M)(a) = \begin{cases} 1 & \text{if } a \in \{(0,0)\} \\ 0 & \text{otherwise,} \end{cases} \qquad \xi_1(M)(a) = \begin{cases} 1 & \text{if } a \in \{(1,0), (0,1)\} \\ 0 & \text{otherwise,} \end{cases}$$

$$\xi_2(M)(a) = \begin{cases} 1 & \text{if } a \in \{(1,1)\} \\ 0 & \text{otherwise.} \end{cases}$$

Grades of Influence. For M a persistence module and $a \in \mathbb{R}^n$ let

$$\mathcal{I}_M(a) = \{y \leq a \mid \xi_0(y) > 0 \text{ or } \xi_1(y) > 0\}.$$

Lemma 2.8. *For M finitely presented and $a, b \in \mathbb{R}^n$ with $\mathcal{I}_M(a) = \mathcal{I}_M(b)$, $M(a, b)$ is an isomorphism.*

Proof. Let $\langle \mathcal{W} | \mathcal{Y} \rangle$ be a minimal presentation for M. Let M' be the n-module with $M'_a = \text{Free}[\mathcal{W}]_a / \langle \mathcal{Y} \rangle_a$, and $M'(a, a')$ the map induced by the inclusion $\text{Free}[\mathcal{W}](a, a')$. Clearly, M' isomorphic to M. Using Proposition 2.5, it's easy to see that for $a, b \in \mathbb{R}^n$ with $\mathcal{I}_M(a) = \mathcal{I}_M(b)$, the map $\text{Free}[\mathcal{W}](a, b)$ is an isomorphism sending $\langle \mathcal{Y} \rangle_a$ isomorphically to $\langle \mathcal{Y} \rangle_b$. Hence $M'(a, b)$ is an isomorphism. Since M and M' are isomorphic, $M(a, b)$ is an isomorphism as well. $\square$

2.5 Continuous Extensions of Discrete Persistence Modules

In the computational setting, the persistence modules we encounter are always finitely presented. It turns out that finitely presented persistence modules are, in a sense, *essentially discrete*; we now explain this.

A *discrete (or $\mathbb{Z}^n$-indexed) persistence module* is a functor $\mathbf{Z}^n \to \mathbf{Vect}$, where $\mathbf{Z}^n$ is the poset category of $\mathbb{Z}^n$. Let $A_n = K[x_1, \ldots, x_n]$ denote the ordinary polynomial ring in n variables. In analogy with the $\mathbb{R}^n$-indexed case, we can regard a discrete persistence module as a multi-graded A_n-module, in the obvious way.

All of the basic definitions and machinery we've described above for $\mathbb{R}^n$-indexed persistence modules can be defined for discrete persistence modules in essentially the same way. In particular, we may define the Betti numbers $\xi_i(Q) : \mathbb{Z}^n \to \mathbb{N}$ of a discrete persistence module Q.

Grid Functions. For $n \geq 1$, we define an *n-D grid* to be a function $\mathcal{G} : \mathbb{Z}^n \to \mathbb{R}^n$, given by

$$\mathcal{G}(z_1, \ldots, z_n) = (\mathcal{G}_1(z_1), \ldots, \mathcal{G}_n(z_n))$$

for some non-decreasing functions $\mathcal{G}_i : \mathbb{Z} \to \mathbb{R}$ with

$$\lim_{z \to -\infty} \mathcal{G}_i(z) = -\infty \qquad \text{and} \qquad \lim_{z \to \infty} \mathcal{G}_i(z) = \infty$$

for each i.

We define $\text{fl}_{\mathcal{G}}(a)$ to be the maximum element of $\text{im}(\mathcal{G})$ ordered before a in the partial order on $\mathbb{R}^n$. That is, for $\mathcal{G}$ a 1-D grid, we let $\text{fl}_{\mathcal{G}} : \mathbb{R} \to \text{im}(\mathcal{G})$ be given by

$$\text{fl}_{\mathcal{G}}(t) = \max\{s \in \text{im}(\mathcal{G}) \mid s \leq t\},$$

and for $\mathcal{G}$ an n-D grid function, we define $\text{fl}_{\mathcal{G}} : \mathbb{R}^n \to \text{im}(\mathcal{G})$ by

$$\text{fl}_{\mathcal{G}}(a_1, \ldots, a_n) = (\text{fl}_{\mathcal{G}_1}(a_1), \ldots, \text{fl}_{\mathcal{G}_n}(a_n)).$$

21

Continuous Extensions of Discrete Persistence Modules. For $\mathcal{G}$ an n-D grid, we define a functor $\mathrm{E}_{\mathcal{G}} : \mathbf{Vect}^{\mathbf{Z}^n} \to \mathbf{Vect}^{\mathbf{R}^n}$ as follows:

1. For Q a $\mathbb{Z}^n$-indexed persistence module and $a, b \in \mathbb{R}^n$,

$$\mathrm{E}_{\mathcal{G}}(Q)_a = Q_y, \qquad \mathrm{E}_{\mathcal{G}}(Q)(a, b) = Q(y, z),$$

where

$$y = \max \{ w \in \mathbb{Z}^n \mid \mathcal{G}(w) = \mathrm{fl}_{\mathcal{G}}(a) \},$$
$$z = \max \{ w \in \mathbb{Z}^n \mid \mathcal{G}(w) = \mathrm{fl}_{\mathcal{G}}(b) \}.$$

2. The action of $\mathrm{E}_{\mathcal{G}}$ on morphisms is the obvious one.

We say that an n-module M is a *continuous extension of Q along $\mathcal{G}$* if $M \cong \mathrm{E}_{\mathcal{G}}(Q)$.

Proposition 2.9. *Any finitely presented n-module M is a continuous extension of a finitely generated discrete persistence module along some n-D grid.*

Proof. Let $\mathcal{G} : \mathbb{Z}^n \to \mathbb{R}^n$ be any n-D grid such that $\operatorname{supp} \xi_0(M) \cap \operatorname{supp} \xi_1(M) \subseteq \operatorname{im} \mathcal{G}$. We regard $\mathcal{G}$ as a functor $\mathbf{Z}^n \to \mathbf{R}^n$ in the obvious way. Using Lemma 2.8, it's easy to check that M is a continuous extension of $M \circ \mathcal{G}$ along $\mathcal{G}$. Further, $M \circ \mathcal{G}$ is finitely generated; a finite presentation for M induces one for $M \circ \mathcal{G}$ of the same size. $\square$

Betti Numbers of Continuous Extensions.

Proposition 2.10. *Suppose an n-module M is a continuous extension of Q along an injective n-D grid $\mathcal{G}$. Then for all i,*

$$\xi_i(M)(a) = \begin{cases} \xi_i(Q)(z) & \text{if } a = \mathcal{G}(z) \text{ for some } z \in \mathbb{Z}^n, \\ 0 & \text{otherwise.} \end{cases}$$

RIVET exploits Proposition 2.10 to compute the Betti numbers of finitely presented $\mathbb{R}^2$-indexed persistence modules, by appealing to local formulae for the Betti numbers of $\mathbb{Z}^2$-indexed persistence modules; see Section 5.3.

Proof of Proposition 2.10. Let

$$\cdots \xrightarrow{\alpha^3} F^2 \xrightarrow{\alpha^2} F^1 \xrightarrow{\alpha^1} F^0$$

be a free resolution of Q. It's easy to see that $\mathrm{E}_{\mathcal{G}}$ preserves exactness, so

$$\cdots \xrightarrow{\mathrm{E}_{\mathcal{G}}(\alpha^3)} \mathrm{E}_{\mathcal{G}}(F^2) \xrightarrow{\mathrm{E}_{\mathcal{G}}(\alpha^2)} \mathrm{E}_{\mathcal{G}}(F^1) \xrightarrow{\mathrm{E}_{\mathcal{G}}(\alpha^1)} \mathrm{E}_G(F^0)$$

is a free resolution for M. Write $G^i = \mathrm{E}_{\mathcal{G}}(F^i)$ and $\beta^i = \mathrm{E}_{\mathcal{G}}(\alpha^i)$.

By definition, $\xi_i(M) = \mathrm{dimf}(\mathcal{H}^i)$, where $\mathcal{H}^i$ is the ith homology module of the following chain complex:

$$\cdots \xrightarrow{\beta_3 \otimes \mathrm{Id}} G^2 \otimes P_n/I \xrightarrow{\beta_2 \otimes \mathrm{Id}} G^1 \otimes P_n/I \xrightarrow{\beta_1 \otimes \mathrm{Id}} G^0 \otimes P_n/I.$$

We have two functors $\mathbf{Vect^{R^n}} \to \mathbf{Vect^{R^n}}$ acting respectively on objects by

$$N \to N \otimes P_n/I,$$
$$N \to N/IN,$$

with the action on morphisms defined in the obvious way; note that these are naturally isomorphic. Thus the above chain complex is isomorphic to the chain complex

$$\cdots \xrightarrow{\bar{\beta}^3} G^2/IG^2 \xrightarrow{\bar{\beta}^2} G^1/IG^1 \xrightarrow{\bar{\beta}^1} G^0/IG^0,$$

where $\bar{\beta}^i$ is the map induced by β^i. Since G^i is a continuous extension along $\mathcal{G}$, it is clear that if $a \notin \mathrm{im}\,\mathcal{G}$, then $(G^i/IG^i)_a = 0$. Hence, if $a \notin \mathrm{im}\,\mathcal{G}$ then $\xi_i(M)(a) = 0$, as claimed.

It remains to consider the case that $a \in \mathrm{im}\,\mathcal{G}$. Let J denote the maximal homogeneous ideal of A_n. $\xi_i(Q) = \mathrm{dimf}(\mathcal{K}^i)$, for $\mathcal{K}^i$ the ith homology module of the chain complex

$$\cdots \xrightarrow{\bar{\alpha}^3} F^2/JF^2 \xrightarrow{\bar{\alpha}^2} F^1/JF^1 \xrightarrow{\bar{\alpha}^1} F^0/JF^0,$$

where $\bar{\alpha}^i$ is the map induced by α^i. If $a = \mathcal{G}(z)$, then by the way we have defined the functor $\mathrm{E}_{\mathcal{G}}$, it is clear that we have isomorphisms $\{\psi_i : F_z^i \to G_a^i\}_{i \geq 0}$ sending JF_z^i isomorphically to IG_a^i, such that the following diagram commutes:

$$
\begin{array}{ccccc}
\cdots \longrightarrow & F_z^2 & \xrightarrow{\alpha_z^2} & F_z^1 & \xrightarrow{\alpha_z^1} & F_z^0 \\
& \downarrow{\psi_2} & & \downarrow{\psi_1} & & \downarrow{\psi_0} \\
\cdots \longrightarrow & G_a^2 & \xrightarrow{\beta_a^2} & G_a^1 & \xrightarrow{\beta_a^1} & G_a^0.
\end{array}
$$

Taking quotients, we obtain a commutative diagram:

$$
\begin{array}{ccccc}
\cdots \longrightarrow & (F^2/JF^2)_z & \xrightarrow{\bar{\alpha}_z^2} & (F^1/JF^1)_z & \xrightarrow{\bar{\alpha}_z^1} & (F^0/JF^0)_z \\
& \downarrow{\simeq} & & \downarrow{\simeq} & & \downarrow{\simeq} \\
\cdots \longrightarrow & (G^2/IG^2)_a & \xrightarrow{\bar{\beta}_a^2} & (G^1/IG^1)_a & \xrightarrow{\bar{\beta}_a^1} & (G^0/IG^0)_a.
\end{array}
$$

It follows that $\mathcal{K}_z^i \simeq \mathcal{H}_a^i$, so $\xi_i(M)_a = \xi_i(Q)_z$ as desired. $\qquad \square$

Barcodes of Discrete Persistence Modules. We discussed the barcodes of ($\mathbb{R}$-indexed) 1-D persistence modules in Section 1.2. The structure theorem of [46] tells us that the barcode $\mathcal{B}(Q)$ of a discrete 1-D persistence module Q is also well defined, provided $\dim(V_a) < \infty$ for all a less than some $a_0 \in \mathbb{Z}$. When Q is finitely generated, $\mathcal{B}(Q)$ is a finite multiset of intervals $[a, b)$ with $a < b \in \mathbb{Z} \cup \infty$.

Barcodes Under Continuous Extension. We omit the easy proof of the following:

Proposition 2.11. *For Q a finitely generated discrete 1-D persistence module and M a continuous extension of Q along $\mathcal{G}$,*

$$\mathcal{B}(M) = \{[\mathcal{G}(a), \mathcal{G}(b)) \mid [a, b) \in \mathcal{B}(Q), \mathcal{G}(a) < \mathcal{G}(b)\},$$

where we define $\mathcal{G}(\infty) = \infty$.

Remark 2.12. As already noted in Section 1.5, for $\mathbf{L}$ the poset category corresponding to a line $L \in \bar{L}$, $\mathbf{L}$ is isomorphic to $\mathbf{R}$. Hence, by adapting the definitions given above in the $\mathbb{R}$-indexed setting, we can define the grid function $\mathcal{G} : \mathbb{Z} \to L$, a function $\mathrm{fl}_{\mathcal{G}} : L \to \mathrm{im}\, G$, and the functor $\mathrm{E}_{\mathcal{G}} : \mathbf{Vect}^{\mathbf{Z}} \to \mathbf{Vect}^{\mathbf{L}}$. As in the $\mathbb{R}$-indexed case, we say $M : \mathbf{L} \to \mathbf{Vect}$ is a continuous extension of a $\mathbb{Z}$-indexed persistence module Q if $M \cong \mathrm{E}_{\mathcal{G}}(Q)$. Clearly then, Proposition 2.11 also holds for continuous extensions in the L-indexed setting. In Section 4.2, we will use Proposition 2.11 in the L-indexed setting to prove our main result on queries of augmented arrangements.

3 Augmented Arrangements of 2-D Persistence Modules

In this section we define the augmented arrangement $\mathcal{A}^{\bullet}(M)$ associated to a finitely presented 2-D persistence module M.

First, in Section 3.1 we define the line arrangement $\mathcal{A}(M)$ associated to M. Next, in Section 3.2 we present a characterization of the 1-skeleton of $\mathcal{A}(M)$. Finally, using this characterization, in Section 3.3 we define the barcode template $\mathcal{T}^e$ stored at a 2-cell e of $\mathcal{A}(M)$.

The augmented arrangement $\mathcal{A}^{\bullet}(M)$ is defined to be the arrangement $\mathcal{A}(M)$, together with the additional data $\mathcal{T}^e$ at each 2-cell e.

3.1 Definition of $\mathcal{A}(M)$

Let $S = \mathrm{supp}\, \xi_0(M) \cup \mathrm{supp}\, \xi_1(M)$. To keep our exposition simple, we will assume that each element of S has non-negative x-coordinate. Using the shift construction described in Section 2.2, we can always translate the indices of M so that this assumption holds, so there is no loss of generality in this assumption.

Point-Line Duality. Recall the definitions of $\mathcal{L}$ and $\bar{\mathcal{L}}$ from Section 1.5. As mentioned there, a standard point-line duality gives a parameterization of $\mathcal{L}$ by the half-plane $[0, \infty) \times \mathbb{R}$. We now explain this. Define dual transforms $\mathcal{D}_\ell$ and $\mathcal{D}_p$ as follows:

$$\mathcal{D}_\ell : \mathcal{L} \to [0, \infty) \times \mathbb{R} \qquad\qquad \mathcal{D}_p : [0, \infty) \times \mathbb{R} \to \mathcal{L}$$
$$y = ax + b \mapsto (a, -b) \qquad\qquad\qquad\qquad (c, d) \mapsto y = cx - d$$

This duality does not extend naturally to vertical lines, i.e. lines in $\bar{\mathcal{L}} - \mathcal{L}$.
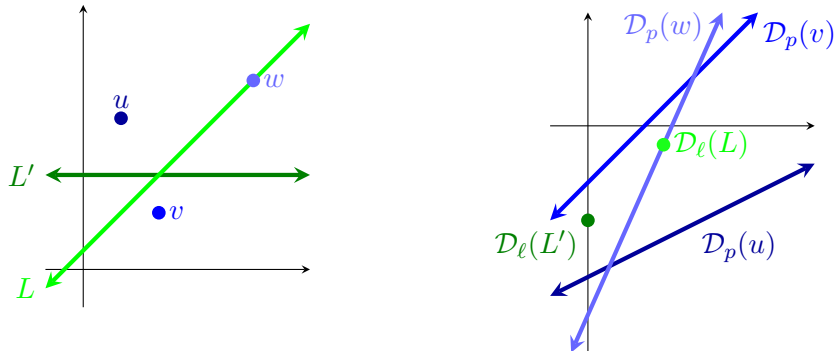
Figure 5: Illustration of point-line duality

The following lemma, whose proof we omit, is illustrated in Fig. 5 by the point w and line L.

Lemma 3.1. *The transforms $\mathcal{D}_\ell$ and $\mathcal{D}_p$ are inverses and preserve incidence, in the sense that for $w \in [0, \infty) \times \mathbb{R}$ and $L \in \mathcal{L}$, $w \in L$ if and only if $\mathcal{D}_\ell(L) \in \mathcal{D}_p(w)$.*

Line Arrangements in $[0, \infty) \times \mathbb{R}$. A *cell* is a topological space homeomorphic to $\mathbb{R}^n$ for some $n \geq 0$. We define a *cell complex* on $[0, \infty) \times \mathbb{R}$ to be a decomposition of $[0, \infty) \times \mathbb{R}$ into a finite number of cells, so that the topological boundary of each cell lies in the union of cells of lower dimension. By standard point-set topology, each cell in a cell complex on $[0, \infty) \times \mathbb{R}$ has dimension at most 2. According to the definition, a cell complex on $[0, \infty) \times \mathbb{R}$ is not a CW-complex, as some cells will necessarily be unbounded.

By a *line arrangement* in $[0, \infty) \times \mathbb{R}$, we mean the cell complex on $[0, \infty) \times \mathbb{R}$ induced by a set W of lines in $[0, \infty) \times \mathbb{R}$. In this cell complex, the 1-skeleton consists of the union of all lines in W, together with the line $x = 0$.

Definition of $\mathcal{A}(M)$. For $W = \{(a_1, b_1), (a_2, b_2), \ldots, (a_l, b_l)\}$ a finite subset of $\mathbb{R}^2$, let LUB(S), the *least upper bound of W* be given by

$$\text{LUB}(S) = (\max_i a_i, \ \max_i b_i).$$

For example, LUB$((3,5),(7,4)) = (7,5)$.

Say that a pair of distinct elements $u, v \in S$ is *weakly incomparable* if one of the following is true:

- u and v are incomparable with respect to the partial order on $\mathbb{R}^2$,
- u and v share either their first or second coordinate.

Call $\alpha \in [0, \infty) \times \mathbb{R}$ an *anchor* if $\alpha = $ LUB(u, v) for $u, v \in S$ weakly incomparable.

We define $\mathcal{A}(M)$ to be the line arrangement in $[0, \infty) \times \mathbb{R}$ induced by the set of lines

$$\{\mathcal{D}_p(\alpha) \mid \alpha \text{ is an anchor}\}.$$

In view of Lemma 3.1, then, the 1-skeleton $\mathcal{A}^1(M)$ of $\mathcal{A}(M)$ is given by

$$\mathcal{A}^1(M) = \{\mathcal{D}_\ell(L) \mid L \in \mathcal{L} \text{ contains an anchor}\} \cup (\{0\} \times \mathbb{R}). \tag{3.1}$$

It is clear that $\mathcal{A}(M)$ is completely determined by S.

0-cells of $\mathcal{A}(M)$. Note that for $\alpha \neq \beta$ two anchors, $\mathcal{D}_p(\alpha)$ and $\mathcal{D}_p(\beta)$ intersect in $[0, \infty)$ if and only if there exists some $L \in \mathcal{L}$ containing both α and β; such a line L exists if and only if α and β are comparable and have distinct x-coordinate.

Size of $\mathcal{A}(M)$. We now bound the number of cells in $\mathcal{A}(M)$ of each dimension. As in Section 1.9, let $\kappa = \kappa_x \kappa_y$, for κ_x and κ_y the number of unique x and y coordinates, respectively, of points in S. Clearly, the number of anchors for S is bounded above by κ. Hence the number of lines in $\mathcal{A}(M)$ is also bounded above by κ.

Precise bounds on the number of vertices, edges, and faces in an arbitrary line arrangement are well known, and can be computed by simple counting arguments. These bounds tell us that the number of vertices, edges, and faces in $\mathcal{A}(M)$ is each not greater than κ^2.

## 3.2 Characterization of the 1-Skeleton of $\mathcal{A}(M)$

We next give our alternate description of the 1-skeleton of $\mathcal{A}(M)$. To do so, we first define the set crit(M) of *critical lines* in $\mathcal{L}^\circ$. Here $\mathcal{L}^\circ$ denotes the topological interior of $\mathcal{L}$, i.e. the set of affine lines in $\mathbb{R}^2$ with positive, finite slope.

The Push Map. Note that for each $L \in \bar{\mathcal{L}}$, the partial order on $\mathbb{R}^2$ restricts to a total order on L. This extends to a total order on $L \cup \infty$ by taking $v < \infty$ for each $v \in L$. For $L \in \bar{\mathcal{L}}$, define the *push map* push$_L : \mathbb{R}^2 \to L \cup \infty$ by taking

$$\text{push}_L(a) = \min\{v \in L \mid a \leq v\}.$$

Note that:

- $\infty \in \operatorname{im} \operatorname{push}_L$ if and only if L is horizontal or vertical.

- For $a = (a_1, a_2) \in \mathbb{R}^2$ and $\operatorname{push}_L(u) = (v_1, v_2) \in L$, either $a_1 = v_1$ or $a_2 = v_2$; see Fig. 6.

- For $r < s \in \mathbb{R}^2$, $\operatorname{push}_L(r) \leq \operatorname{push}_L(s)$.
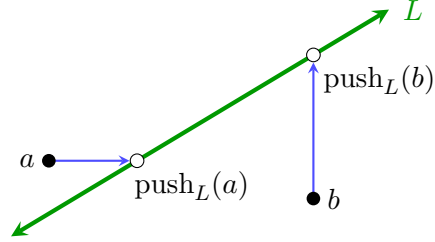
Figure 6: Illustration of the push map for lines of positive, finite slope

Continuity of Push Maps. For any $a \in \mathbb{R}^2$, the maps $\{\operatorname{push}_L\}_{L \in \bar{\mathcal{L}}}$ induce a map $\operatorname{push}_a : \bar{\mathcal{L}} \to \mathbb{R}^2 \cup \infty$, defined by $\operatorname{push}_a(L) = \operatorname{push}_L(a)$.

Recall that we consider $\bar{\mathcal{L}}$ as a topological space, with the topology the restriction of topology on the affine Grassmannian of 1-D lines in $\mathbb{R}^2$.

Lemma 3.2. *For each $a \in \mathbb{R}^2$, push_a is continuous on $\mathcal{L}^\circ$.*

Proof. Note that for any $a = (a_1, a_2)$ and $L \in \mathcal{L}^\circ$, $\operatorname{push}_a(L)$ is the unique intersection of L with

$$\{(a_1, y) \mid y \geq a_2\} \cup \{(x, a_2) \mid x \geq a_1\}.$$

From this, the result follows readily. $\square$

Critical Lines. For $L \in \mathcal{L}^\circ$, push_L induces a totally ordered partition S^L of S: elements of the partition are restrictions of levelsets of push_L to S, and the total order on S^L is the pullback of the total order on L. This partition is illustrated in Fig. 7.
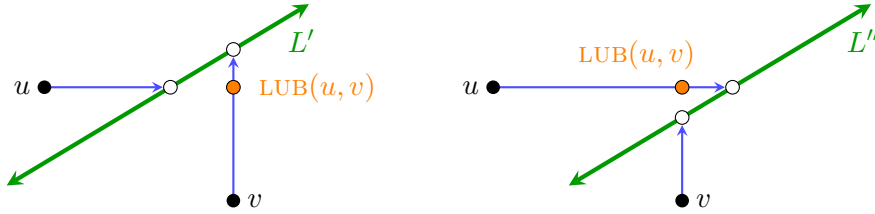
We call $L \in \mathcal{L}^\circ$ *regular* if there is an open ball $B \in \mathcal{L}^\circ$ containing L such that $S^L = S^{L'}$ for all $L' \in B$. We call $L \in \mathcal{L}^\circ$ *critical* if it is not regular. Let $\operatorname{crit}(M)$ denote the set of critical lines in $\mathcal{L}^\circ$.

Theorem 3.3 (Characterization of the 1-Skeleton of $\mathcal{A}(M)$)**.** *The 1-skeleton of $\mathcal{A}(M)$ is exactly*

$$\mathcal{D}_\ell(\operatorname{crit}(M)) \cup (\{0\} \times \mathbb{R}).$$
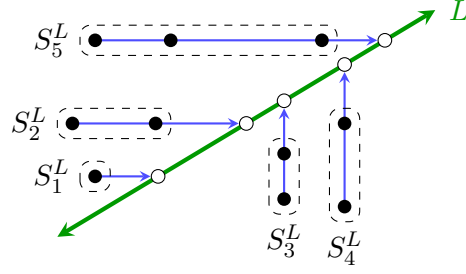
Figure 7: The totally ordered partition S^L of S. The ith element of the partition is labeled as S_i^L.

Figure 8: An illustration of the geometric idea behind Theorem 3.3: For $u, v, \in S$ and a line L' lying just above $\text{LUB}(u,v)$, $\text{push}_{L'}(u) < \text{push}_{L'}(v)$, whereas for a line L'' lying just below $\text{LUB}(u,v)$, $\text{push}_{L''}(v) < \text{push}_{L''}(u)$. Thus any line passing through $\text{LUB}(u,v)$ is critical.

Proof. In view of the description of the 1-skeleton of $\mathcal{A}(M)$ given by Eq. (3.1), proving the proposition amounts to showing that $L \in \mathcal{L}^\circ$ is critical if and only if L contains some anchor w.

Suppose $L \in \mathcal{L}^\circ$ contains an anchor w, and let $u, v \in S$ be weakly incomparable, with $w = \text{LUB}(u,v) \in L$. Then we must have that $\text{push}_L(u) = \text{push}_L(v) = w$. Further, it's easy to see that we can find an arbitrarily small perturbation of L so that either $\text{push}_L(u) < \text{push}_L(v)$ or $\text{push}_L(v) < \text{push}_L(u)$. Thus L is critical; see Fig. 8 for an illustration in the case that u and v are incomparable.

To prove the converse, assume that $L \in \mathcal{L}^\circ$ does not contain any anchor, and consider distinct $u, v \in S$ with $\text{push}_L(u) = \text{push}_L(v)$. Note that u, v must lie either on the same horizontal line or the same vertical line; otherwise u and v would be incomparable and we would have $\text{push}_L(u) = \text{push}_L(v) = \text{LUB}(u,v) \in L$. Assume without loss of generality that $u < v$ and u, v lie on the same horizontal line H. Then we must also have that $\text{push}_L(u) = \text{push}_L(v)$ lies on H. Since $v = \text{LUB}(u,v)$, v is an anchor. However, L does not contain any anchor, so we must have $v < \text{push}_L(v)$.

Any sufficiently small perturbation L' of L will also intersect H at a point p to the right of v, so that we have $\text{push}_{L'}(u) = \text{push}_{L'}(v) = p$. Thus for all L' in a neighborhood of L, $\text{push}(u) = \text{push}(v)$.

In fact, since S is finite, we can choose a single neighborhood $\mathcal{N}$ of L in $\mathcal{L}^\circ$ such that for any $u, v \in S$ with $\text{push}_L(u) = \text{push}_L(v)$, we have $\text{push}_{L'}(u) = \text{push}_{L'}(v)$ for all $L' \in \mathcal{N}$. By Lemma 3.2, choosing $\mathcal{N}$ to be smaller if necessary, may further assume that if $\text{push}_L(u) \neq \text{push}_L(v)$, then $\text{push}_{L'}(u) \neq \text{push}_{L'}(v)$ for all $L' \in \mathcal{N}$. Thus, the partition $S^{L'}$ is independent of the choice of $L' \in \mathcal{N}$. Moreover, by Lemma 3.2 again, the total order on $S^{L'}$ is also independent of the choice of $L' \in \mathcal{N}$. Therefore, L is regular. $\square$

Corollary 3.4. *If the duals of $L, L' \in \mathcal{L}$ are contained in the same 2-cell in $\mathcal{A}(M)$, then $S^L = S^{L'}$.*

Proof. Each 2-cell is connected and open, so this follows from Theorem 3.3. $\square$

Remark 3.5. In fact, Corollary 3.4 can be strengthened to show that for any $L, L' \in L^\circ$, the duals of L and L' lie in the same cell of $\mathcal{A}(M)$ if and only if $S^L = S^{L'}$. However, we will not need this stronger result.

3.3   The Barcode Templates $\mathcal{T}^e$

Using Corollary 3.4, we now define the barcode templates $\mathcal{T}^e$ stored at each 2-cell e of $\mathcal{A}(M)$. This will complete the definition of the augmented arrangement $\mathcal{A}^\bullet(M)$.

By Corollary 3.4, we can associate to each 2-cell e in $\mathcal{A}(M)$ a totally ordered partition S^e of S. Let S_i^e denote the ith element of the partition.

A 1-D Persistence Module at the 2-Cell e.   We next use S^e to define a discrete (i.e., $\mathbb{Z}$-indexed) persistence module M^e. If $M \cong 0$, we take $M^e = 0$. Assume then that $M \not\cong 0$. First, we define a map $\zeta^e : \mathbb{N} \setminus \{0\} \to \mathbb{R}^2$ by

$$\zeta^e(z) = \begin{cases} \text{LUB}(\cup_{i=1}^z S_i^e) & \text{if } 1 \leq z < |S^e|, \\ \text{LUB}(S) & \text{if } z \geq |S^e|. \end{cases}$$

We define $\mathcal{P}^e := \text{im}\, \zeta^e$, and call this the set of *template points* at cell e. Note that the restriction of ζ^e to $\{1, \ldots, |S^e|\}$ is an injection, so that $|\mathcal{P}^e| = |S^e|$.

Let $\mathbf{P}$ denote the poset category of positive integers. $\zeta^e(y) \leq \zeta^e(z)$ whenever $y < z$, so ζ^e induces a functor $\mathbf{P}$ to $\mathbf{R}^2$, which we also denote ζ^e. Finally, the functor $M \circ \zeta^e : \mathbf{P} \to \mathbf{Vect}$ extends to a functor $M^e : \mathbf{Z} \to \mathbf{Vect}$ (i.e., a $\mathbb{Z}$-indexed persistence module) by taking $M_z^e = 0$ whenever $z \leq 0$.

The Definition of Barcode Templates.   Clearly, M^e is p.f.d., so it has a well-defined barcode $\mathcal{B}(M^e)$, and it's easy to see that $\mathcal{B}(M^e)$ consists of intervals $[a, b)$, with $a < b \in \{1, \ldots, |\mathcal{P}^e|, \infty\}$. Let us write $\zeta(\infty) = \infty$.

We define $\mathcal{T}^e$ to be a collection of pairs of points in $\mathcal{P}^e \times (\mathcal{P}^e \cup \infty)$, as follows:

$$\mathcal{T}^e = \{(\zeta^e(a), \zeta^e(b)) \mid [a, b) \in \mathcal{B}(M^e)\}.$$

29

This completes the definition of $\mathcal{A}^\bullet(M)$.

Remark 3.6. $\mathcal{A}^\bullet(M)$ is completely determined by the fibered barcode $\mathcal{B}(M)$ and the set S. Indeed, $\mathcal{A}(M)$ is completely determined by S, and using Proposition 1.2, it's easy to see that $\mathcal{T}^e$ is completely determined by $\mathcal{B}(M)$ and S. The main result of the next section, Theorem 4.1, shows that conversely, $\mathcal{A}^\bullet(M)$ completely determines $\mathcal{B}(M)$, and does so in a simple way.

4   Querying the Augmented Arrangement

In the previous section, we defined the augmented arrangement $\mathcal{A}^\bullet(M)$ of a finitely presented 2-D persistence module M. We now explain how $\mathcal{A}^\bullet(M)$ encodes the fibered barcode $\mathcal{B}(M)$. That is, for a given $L \in \bar{\mathcal{L}}$, we explain how to recover $\mathcal{B}(M^L)$ from $\mathcal{A}^\bullet(M)$. The main result of this section, Theorem 4.1, is the basis of our algorithm for querying $\mathcal{A}^\bullet(M)$. We describe the computational details of our query algorithm in Section 4.3.

Recall that the procedure for querying $\mathcal{A}^\bullet(M)$ for a barcode $\mathcal{B}(M^L)$ was discussed for the case of generic lines L in Section 1.8. More generally, for any $L \in \bar{\mathcal{L}}$, querying $\mathcal{A}^\bullet(M)$ for the barcode $\mathcal{B}(M^L)$ involves two steps. First, we choose a 2-cell e in $\mathcal{A}(M)$; if $L \in \mathcal{L}$, then e is a 2-dimensional coface of the cell in containing $\mathcal{D}_\ell(L)$. Second, we obtain the intervals of $\mathcal{B}(M^L)$ from the pairs of $\mathcal{T}^e$ by pushing the points in each pair $(a, b) \in \mathcal{T}^e$ onto the line L, via the map push_L of Section 3.2.

We now describe in more detail the first step of selecting the 2-D coface e.

4.1   Selecting a 2-D Coface e of L

For $L \in \bar{\mathcal{L}}$, we choose the 2-D coface e of L as follows:

- If $L \in \mathcal{L}^\circ$, then there exists a 2-cell of $\mathcal{A}(M)$ whose closure contains $\mathcal{D}_\ell(L)$; we take e to be any such 2-cell.

- If L is horizontal, then the 2-D cofaces of the cell containing $\mathcal{D}_\ell(L)$ are ordered vertically in $[0, \infty) \times \mathbb{R}$; we take e to be the *bottom* coface. (Note that $\mathcal{D}_\ell(L)$ has only one 2-D coface unless L contains an anchor.)

- For L vertical, say L is the line $x = a$, let L_0 be the line in the arrangement $\mathcal{A}(M)$ of maximum slope, amongst those having slope less than or equal to a, if a unique such line exists; if there are several such lines, take L_0 to be the one with the largest y-intercept. If such L_0 exists, it contains a unique unbounded 1-cell in a. We take e to be the 2-cell lying directly above this 1-cell. If such a line L_0 does not exist, then we take e to be the bottom unbounded 2-cell of $\mathcal{A}(M)$; since we assume all x-coordinates of S to be non-negative, this cell is uniquely defined.

The selection of cofaces for several lines is illustrated in Fig. 9.
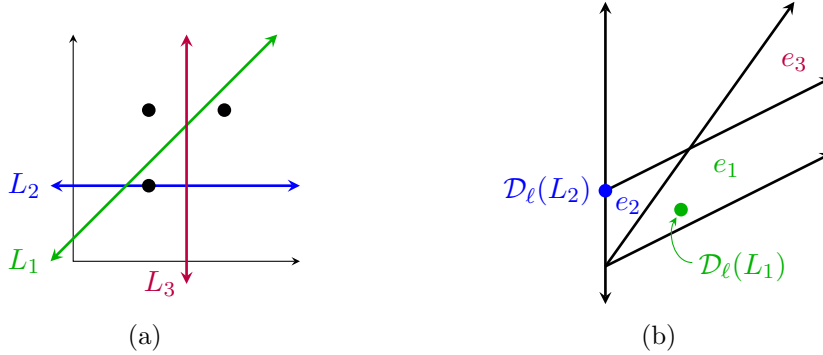
Figure 9: Three anchors are drawn as black dots in (a); the corresponding line arrangement $\mathcal{A}(M)$ is shown in (b). For each line L_i in (a), the dual point $\mathcal{D}_\ell(L_i)$ and the corresponding 2-cell e_i, chosen as in Section 4.1, are shown in (b) in the same color.

4.2 The Query Theorem

Here is the main mathematical result underlying RIVET:

Theorem 4.1 (Querying the Augmented Arrangement)**.** *For any line $L \in \bar{\mathcal{L}}$ and e a 2-cell chosen as in Section 4.1, the barcode obtained by restricting M to L is:*

$$\mathcal{B}(M^L) = \{\, [\text{push}_L(a), \text{push}_L(b)) \mid (a, b) \in \mathcal{T}^e, \ \text{push}_L(a) < \text{push}_L(b) \,\}.$$

Note that if L is such that $\mathcal{D}_\ell(L)$ lies in a 2-cell of $\mathcal{A}(M)$, then $\text{push}_L(a) < \text{push}_L(b)$ for all $(a, b) \in \mathcal{T}^e$, so the theorem statement simplifies for such L. In general, however, it is possible to have $\text{push}_L(a) = \text{push}_L(b)$.

Proof of Theorem 4.1. We prove the result for the case $L \in \mathcal{L}^\circ$; the proof for L horizontal or vertical is similar, but easier, and is left to the reader. The result holds trivially if $M \cong 0$. Assume then that $M \not\cong 0$.

Let e be a 2-D coface of the cell containing $\mathcal{D}_p(L)$. To keep notation simple, we will write $\text{push} = \text{push}^L$ and $\zeta = \zeta^e$.

Keeping Remark 2.12 in mind, we define a 1-D grid $\mathcal{G} : \mathbb{Z} \to L$. To do so, we first define the restriction of $\mathcal{G}$ to $\{1, \ldots, |\mathcal{P}^e|\}$ by taking

$$\mathcal{G}(z) = \text{push} \circ \zeta(z);$$

note that this is non-decreasing. We choose an arbitrary extension of this to a 1-D grid $\mathcal{G} : \mathbb{Z} \to L$.

By Proposition 2.11, to finish the proof it suffices to show that M^L is a continuous extension of M^e along $\mathcal{G}$; i.e., that there exists an isomorphism $\alpha : \mathrm{E}_{\mathcal{G}}(M^e) \to M^L$.

31

Given $t \in L$, let
$$z = \max\{w \in \mathbb{Z} \mid \mathcal{G}(w) = \mathrm{fl}_{\mathcal{G}}(t)\}.$$

Note that we have
$$\mathrm{E}_{\mathcal{G}}(M^e)_t = M_z^e = \begin{cases} M_{\zeta(z)} & \text{if } z \geq 1, \\ 0 & \text{if } z < 1. \end{cases}$$

Note also that
$$\mathcal{G}(z) \leq t < \mathcal{G}(z+1). \tag{4.1}$$

To define the maps $\alpha_t : \mathrm{E}_{\mathcal{G}}(M^e)_t \to M_t^L$, we will consider separately the three cases
$$z < 1, \qquad 1 \leq z < |\mathcal{P}^e|, \qquad \text{and} \qquad z \geq |\mathcal{P}^e|.$$

For $a \in \mathbb{R}^n$, recall the definition of $\mathcal{I}_M(a)$ from immediately above Lemma 2.8, and note that $\mathcal{G}(1) = \mathrm{push} \circ \zeta(1)$ is the minimal element a of L (with respect to the partial order on $\mathbb{R}^2$) such that $\mathcal{I}_M(a) \neq \emptyset$. Hence by Eq. (4.1), if $z < 1$ then $\mathcal{I}_M(t) = \emptyset$, so $M_t^L = 0$. Further, if $z < 1$, then $\mathrm{E}_{\mathcal{G}}(M^e)_t = 0$. Thus for $z < 1$, we (necessarily) take the isomorphism $\alpha_t : \mathrm{E}_{\mathcal{G}}(M^e)_t \to M_t^L$ to be the zero map.

For $1 \leq z < |\mathcal{P}^e|$, Eq. (4.1) and the definition of $\mathcal{G}$ give that
$$\mathrm{push} \circ \zeta(z) \leq t < \mathrm{push} \circ \zeta(z+1).$$

This implies
$$\zeta(z) \leq t \not\geq \zeta(z+1),$$

so $\mathcal{I}_M(\zeta(z)) = \mathcal{I}_M(t)$. Thus $M(\zeta(z), t)$ is an isomorphism by Lemma 2.8. Since $\mathrm{E}_{\mathcal{G}}(M^e)_t = M_{\zeta(z)}$, we may regard the map $M(\zeta(z), t)$ as an isomorphism $\alpha_t : \mathrm{E}_{\mathcal{G}}(M^e)_t \to M_t = M_t^L$.

For $z \geq |\mathcal{P}^e|$, we have that $\zeta(z) = \mathrm{LUB}(S)$, so $\mathcal{I}_M(\zeta(z)) = S$. Further, the chain of inequalities
$$\zeta(z) = \mathrm{LUB}(S) \leq \mathrm{push} \circ \mathrm{LUB}(S) = \mathcal{G}(|\mathcal{P}^e|) \leq \mathcal{G}(z) \leq t$$

gives that $\zeta(z) \leq t$, and further that $S = \mathcal{I}_M(\zeta(z)) \subseteq \mathcal{I}_M(t) \subseteq S$, so in fact
$$\mathcal{I}_M(\zeta(z)) = \mathcal{I}_M(t) = S.$$

Then by Lemma 2.8 again, $M(\zeta(z), t)$ is an isomorphism, which as above, can be interpreted as an isomorphism $\alpha_t : \mathrm{E}_{\mathcal{G}}(M^e)_t \to M_t^L$.

We have now defined isomorphisms $\alpha_t : \mathrm{E}_{\mathcal{G}}(M^e)_t \to M_t^L$ for all $t \in L$. Clearly, these isomorphisms commute with internal maps in $\mathrm{E}_{\mathcal{G}}(M^e)$ and M^L, so they define an isomorphism $\alpha : \mathrm{E}_{\mathcal{G}}(M^e) \to M^L$, as desired. $\qquad\square$

4.3 Computational Details of Queries

We next explain the computational details of storing and querying $\mathcal{A}^{\bullet}(M)$. We also give the complexity analysis of our query algorithm, proving Theorem 1.3.

DCEL Representation of $\mathcal{A}(M)$. As noted in the introduction, we represent the line arrangement $\mathcal{A}(M)$ that underlies $\mathcal{A}^\bullet(M)$ using the DCEL data structure [29], a standard data structure for representing line arrangements in computational geometry. The DCEL consists of a collection of vertices, edges, and 2-cells, together with a collection of pointers specifying how the cells fit together to form a decomposition of $[0, \infty) \times \mathbb{R}$.

Representing the Barcode Templates. To represent the augmented arrangement $\mathcal{A}^\bullet(M)$, we store the barcode template $\mathcal{T}^e$ at each 2-cell e in the DCEL representation of $\mathcal{A}(M)$. Recall that $\mathcal{T}^e$ is a multiset; this means a pair (a, b) may appear in $\mathcal{T}^e$ multiple times. We thus store $\mathcal{T}^e$ as a list of triples (a, b, k), where $k \in \mathbb{N}$ gives the multiplicity of (a, b) in $\mathcal{T}^e$.

Our Query Algorithm. Given a line $L \in \bar{\mathcal{L}}$, the query of $\mathcal{A}^\bullet(M)$ for $\mathcal{B}(M^L)$ proceeds in two steps. The first step performs a search for the 2-cell e of $\mathcal{A}(M)$ specified in Section 4.1. Once the 2-cell e is selected, we obtain $\mathcal{B}(M^L)$ from $\mathcal{T}^e$ by applying push_L to the endpoints of each pair $(a, b) \in \mathcal{T}^e$.

Let us describe our algorithm to find the 2-cell e in detail. In the case that $L \in \mathcal{L}$ (i.e., L is not vertical), it suffices to find the cell of $\mathcal{A}(M)$ containing $\mathcal{D}_\ell(L)$. In general, the problem of finding the cell in a line arrangement containing a given query point is known as the *point location problem*; this is a very well studied problem in computational geometry. When we need to perform many point location queries on an arrangement, or when we need to perform the queries in real time, it is standard practice to precompute a data structure on the which point locations can be performed very efficiently [44]; this is the approach we take. For v the number of vertices in the arrangement, there are a number of different strategies which, in time $O(v \log v)$, compute a data structure of size $O(v)$ on which we can perform a point location query in time $O(\log v)$. $\mathcal{A}(M)$ has $O(\kappa^2)$ vertices, so computing such a data structure for $\mathcal{A}(M)$ takes time $O(\kappa^2 \log \kappa)$. The data structure is of size $O(\kappa^2)$, and the point location query takes time $O(\log \kappa)$.

In the case that L is vertical, $\mathcal{D}_\ell(L)$ is not defined, and we need to take a different approach to find the 2-cell e. We precompute a separate (simpler) search data structure to handle this case: Let $\mathcal{Y}$ denote the set of lines J in $\mathcal{A}(M)$ such that there is no other line in $\mathcal{A}(M)$ with the same slope lying above J. We compute a 1-D array which contains, for each $J \in \mathcal{Y}$, a pointer to the rightmost (unbounded) 1-cell of $\mathcal{A}(M)$ contained in J, sorted according to slope. Given $\mathcal{A}(M)$, computing this array takes $O(n_l)$ time, where $n_l = O(\kappa)$ is the number of anchor lines. Once the array has been computed, for any vertical line L, we can find the appropriate 2-cell e via a binary search over the array. This takes $\log n_l$ time.

We are now ready to prove our result from the introduction on the cost of querying $\mathcal{A}^\bullet(M)$.

Proof of Theorem 1.3. From the discussion above, it is clear that once we have precom-

puted the appropriate data structures, finding the cell e takes $O(\log \kappa)$ time. Each evaluation of push_L takes constant time, so computing $\mathcal{B}(M^L)$ from $\mathcal{T}^e$ takes total time $O(|\mathcal{T}^e|)$. Thus, the total time to query $\mathcal{A}^\bullet(M)$ for $\mathcal{B}(M^L)$ is $O(|\mathcal{T}^e| + \log \kappa)$. If $\mathcal{D}_\ell(L) \in e$, then $|\mathcal{T}^e| = |\mathcal{B}(M^L)|$; this gives Theorem 1.3 (i). If, on the other hand, $\mathcal{D}_\ell(L) \notin e$, then we may not have $|\mathcal{T}^e| = |\mathcal{B}(M^L)|$, but we do have $|\mathcal{T}^e| = |\mathcal{B}(M^{L'})|$, for L' an arbitrarily small perturbation of L with $\mathcal{D}_\ell(L') \in e$. Theorem 1.3 (ii) follows. □

5 Computing the Arrangement $\mathcal{A}(M)$

We now turn to the specification of our algorithm for computing $\mathcal{A}(M)$. First, in Sections 5.1 and 5.2, we specify the algebraic objects which serve as the input to our algorithm, and explain how these objects arise from bifiltrations.

5.1 Free Implicit Representations of Persistence Modules: The Input to our Algorithm

For $k \geq 1$, define $[k]$ to be the set of integers $\{1, \ldots, k\}$, and let $[0]$ denote the empty set. Define a *free implicit representation (FI-rep)* of an n-D persistence module M to be a quadruple $\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$, such that:

- For $i = 1, 2$, $\mathrm{gr}_i : [m_i] \to \mathbb{R}^n$ is a function, for some $m_i \geq 0$.

- D_1 and D_2 are matrices with coefficients in K, of respective dimensions $m_0 \times m_1$ and $m_1 \times m_2$, for some $m_0 \geq 0$. Note that either D_1 or D_2 may be an empty matrix if $m_i = 0$.

- If $m_1 = 0$, then $M \cong 0$.

- If $m_1 > 0$, let $\mathrm{gr}_0 : [m_0] \to \mathbb{R}^n$ denote the constant function mapping to the greatest lower bound of $\mathrm{im}\, \mathrm{gr}_1$. Then, defining ordered n-graded sets $\mathcal{W}_i := ([m_i], \mathrm{gr}_i)$ for $i = 0, 1, 2$, we require that (in the notation of Section 2.2), $D_1 = [\partial_1]$ and $D_2 = [\partial_2]$ are the matrix representations, respectively, of maps

$$\partial_1 : \mathrm{Free}(\mathcal{W}_1) \to \mathrm{Free}(\mathcal{W}_0), \quad \partial_2 : \mathrm{Free}(\mathcal{W}_2) \to \mathrm{Free}(\mathcal{W}_1)$$

such that $\partial_1 \circ \partial_2 = 0$ and $M \cong \ker \partial_1 / \mathrm{im}\, \partial_2$.

We refer to (m_0, m_1, m_2), defined above, as the *dimensions* of Φ, and write $M \cong H\Phi$. Note that a presentation of M is an FI-rep of M in the degenerate case that $m_0 = 0$, so that D_1 is an empty matrix.

Our algorithm for computing the augmented arrangement of a finitely presented 2-D persistence module M takes as input an FI-rep of M.

34

Storing a Free Implicit Representation in Memory. We store the matrices D_1 and D_2 in a column-sparse data structure, as used in the standard persistent homology algorithm [47]. Columns of D_j are stored in an array of size m_j; the ith column is stored as a list in position i of the array. We also store $\text{gr}_j(i)$ at position i of the same array.

5.2 Motivation: Free Implicit Representations From Finite Filtrations

We are interested in studying the ith persistent homology module $H_i\mathcal{F}$ of a finite bifiltration $\mathcal{F}$ arising from data. Our choice to represent a 2-D persistence module via an FI-rep is motivated by the fact that in practice, one typically has ready access to an FI-rep of $H_i\mathcal{F}$. In contrast, we generally do not have direct access to a presentation of $H_i\mathcal{F}$ at the outset, and while there are known algorithms for computing one [36], they are computationally expensive [12].

We'll now describe in more detail how FI-reps of persistent homology modules arise in practice.

The Chain Complexes of Multi-filtrations. Recall from Section 1.2 that an n-D filtration is a functor $\mathcal{F} : \mathbf{R}^n \to \mathbf{Simp}$ such that the map $\mathcal{F}_a \to \mathcal{F}_b$ is an inclusion whenever $a \le b \in \mathbb{R}^n$. $\mathcal{F}$ gives rise to a chain complex $C\mathcal{F}$ of n-D persistence modules, given by

$$\cdots \xrightarrow{\partial_{j+2}} C_{j+1}\mathcal{F} \xrightarrow{\partial_{j+1}} C_j\mathcal{F} \xrightarrow{\partial_j} C_{j-1}\mathcal{F} \xrightarrow{\partial_{j-1}} \cdots ,$$

where

- we define $C_j\mathcal{F}$ by taking the vector space $(C_j\mathcal{F})_a$ to be generated by the j-simplices of $\mathcal{F}_a$, and taking the map $C_j\mathcal{F}(a,b)$ to be induced by the inclusion $\mathcal{F}_a \hookrightarrow \mathcal{F}_b$,

- the morphism ∂_j is induced by the jth boundary maps of the simplicial complexes $\mathcal{F}_a$.

Note that $H_j\mathcal{F} \cong \ker \partial_j / \operatorname{im} \partial_{j+1}$.

1-Critical and Multi-critical Filtrations. To explain how FI-reps arise from finite bifiltrations, it is helpful to first consider a special case: Following [12], we define a *1-critical filtration* $\mathcal{F}$ to be a finite n-D filtration where for each $s \in \mathcal{F}_{\max}$, $|A(s)| = 1$; here, as in Section 1.2, $A(s)$ denotes the set of grades of appearance of s. If $\mathcal{F}$ is finite and not 1-critical, we say $\mathcal{F}$ is *multi-critical*.

Bifiltrations arising in TDA applications are often (but not always) 1-critical. For example, for P a finite metric space and $\gamma : P \to \mathbb{R}$ a function, the bifiltration $\text{Rips}(\gamma)$ of Section 1.3 is 1-critical, but the filtration B-Rips(P) of that section generally is not 1-critical.

It's easy to see that if $\mathcal{F}$ is 1-critical, then each $C_j\mathcal{F}$ is free; we have an obvious isomorphism between $C_j\mathcal{F}$ and $\mathrm{Free}[\mathcal{F}_j]$, for $\mathcal{F}_j$ the n-graded set given by

$$\mathcal{F}_j \equiv \{(s, A(s)) \mid s \text{ a } j\text{-simplex in } \mathcal{F}_{\max}\}.$$

Thus, choosing an order for each $\mathcal{F}_j$, the boundary map $\partial_j : C_j\mathcal{F} \to C_{j-1}\mathcal{F}$ can be represented with respect to $\mathcal{F}_j$, $\mathcal{F}_{j-1}$ by a matrix $[\partial_j]$ with coefficients in the field K, as explained in Section 2.2; $[\partial_j]$ is exactly the usual matrix representation of the jth boundary map of $\mathcal{F}_{\max}$.

Free Implicit Representations of $H_i\mathcal{F}$ in the 1-Critical Case. The ordered n-graded sets $\mathcal{F}_j$ and matrices $[\partial_j]$ determine the chain complex $C\mathcal{F}$, and hence each of the homology modules $H_j\mathcal{F}$ up to isomorphism. In fact, for $\mathcal{O}_j : \mathcal{F}_j \to \{1, 2, \ldots, |\mathcal{F}_j|\}$ the total order on $\mathcal{F}_j$, we have that

$$(\mathrm{gr}_{\mathcal{F}_j} \circ \mathcal{O}_j^{-1}, \mathrm{gr}_{\mathcal{F}_{j+1}} \circ \mathcal{O}_{j+1}^{-1}, [\partial_j], [\partial_{j+1}],)$$

is an FI-rep of $H_j\mathcal{F}$, for any $j \geq 0$.

Free Implicit Representations of $H_i\mathcal{F}$ in the Multi-Critical case. For $\mathcal{F}$ a multi-critical filtration, the modules $C_j\mathcal{F}$ are not free. Nevertheless, as explained in [16], there is an easy construction of an FI-rep

$$\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$$

of $H_j\mathcal{F}$, generalizing the construction given above in the 1-critical setting. Letting

$$l_j = \sum_{\substack{s \text{ a } j\text{-simplex} \\ \text{in } \mathcal{F}_{\max}}} |A(s)|,$$

the dimensions (m_0, m_1, m_2) of Φ satisfy the following bounds when $\mathcal{F}$ is a bifiltration:

$$m_0 \leq l_{j-1}, \qquad m_1 \leq l_j, \qquad m_2 \leq l_{j+1} + l_j - 1; \tag{5.1}$$

see [16] for details.

Computation of the Free Implicit Representation of a Bifiltration. As explained in Section 1.2, we can store a finite simplicial bifiltration in memory as a simplicial complex, together with a list of grades of appearance for every simplex. Let $\mathcal{F}$ be a finite (one-critical or multi-critical) bifiltration of size l. It's not hard to show that given this input, for any $i \geq 0$ we can compute the FI-rep of $H_i\mathcal{F}$ described above in time $O(l \log l)$.

One Homology Index at a Time, or All Homology Indices at Once?   The standard algorithms for computing persistence barcodes of a 1-D filtration $\mathcal{F}$ compute $\mathcal{B}(H_i\mathcal{F})$ for all i up to a specified integer, in a single pass. When one is interested in the barcodes at each homology index, this is more efficient than doing the computations one index at a time, because the computations of $\mathcal{B}(H_i\mathcal{F})$ and $\mathcal{B}(H_{i+1}\mathcal{F})$ share some computational work.

In contrast, the algorithm described in this paper, and implemented in the present version of RIVET, computes $\mathcal{A}^\bullet(H_i\mathcal{F})$ of a finite bifiltration $\mathcal{F}$, for a single choice of $i \geq 0$. This approach allows us to save computational effort when we are only interested in a single homology module, and seems to be the more natural approach when working with multi-critical filtrations.

That said, within the RIVET framework, for a 1-critical bifiltration $\mathcal{F}$ one can also handle all persistence modules $H_i\mathcal{F}$, for i up to a specified integer, in a single pass.[2]

5.3   Computation of Betti Numbers

Our first step in the computation of $\mathcal{A}(M)$ is to compute $S = \operatorname{supp}\xi_0(M) \cup \operatorname{supp}\xi_1(M)$. Since RIVET also visualizes the Betti numbers of M directly, we choose to compute this by fully computing ξ_0 and ξ_1. (In any case, we do not know of any algorithm for computing S that is significantly more efficient than our algorithm for fully computing ξ_0 and ξ_1.)

Computing Betti Numbers of $\mathbb{Z}^2$-indexed Persistence Modules.   We can define a FI-rep $\Phi = (\operatorname{gr}_1, \operatorname{gr}_2, D_1, D_2)$ of a $\mathbb{Z}^2$-indexed persistence module Q just as we have for an $\mathbb{R}^2$-indexed persistence module above; in this case the functions gr_1 and gr_2 take values in $\mathbb{Z}^2$.

In a companion article [41], we show how to fully compute $\xi_0(Q)$, $\xi_1(Q)$, and $\xi_2(Q)$, given Φ; the algorithm runs in time $O(m^3)$, for (m_0, m_1, m_2) the dimensions of Φ and $m = m_0 + m_1 + m_2$.

One way to compute the bigraded Betti numbers of M, quite standard in computational commutative algebra, is to compute a free resolution for M [37]. However, this gives us more than we need for our particular application. Instead of following this route, our algorithm computes the Betti numbers via carefully scheduled column reductions on matrices, taking advantage of a well-known characterization of Betti numbers in terms of the homology of Kozul complexes [34].

---

[2]The natural way to do this is not to compute $\mathcal{A}^\bullet(H_i\mathcal{F})$ for each $1 \leq i \leq l$, but rather to compute the single augmented line arrangement $\mathcal{A}^\bullet(\bigoplus_{i=1}^l H_i\mathcal{F})$, labeling the intervals of the discrete barcode at each 2-cell of $\mathcal{A}(\bigoplus_{i=1}^l H_i\mathcal{F})$ by homology degree, so that a query of $\mathcal{A}^\bullet(\bigoplus_{i=1}^l H_i\mathcal{F})$ provides the homology degree of each interval of $\mathcal{B}(\bigoplus_{i=1}^l (H_i\mathcal{F})^L)$. This "labeled" variant of $\mathcal{A}^\bullet(\bigoplus_{i=1}^l H_i\mathcal{F})$ can be computed using essentially the same algorithm as presented in this paper for computation of a single augmented arrangement.

When $\mathcal{F}$ is not 1-critical, to compute $\mathcal{A}^\bullet(\bigoplus_{i=1}^l H_i\mathcal{F})$, we need to first replace $C(\mathcal{F})$ by a chain complex of free 2-D persistence modules. As noted in [12], this can be done via a mapping telescope construction, though this may significantly increase the size of $C(\mathcal{F})$.

Computing Betti Numbers of $\mathbb{R}^2$-indexed Persistence Modules. In fact, our algorithm for computing Betti numbers in the discrete setting can also be used to compute the bigraded Betti numbers of a finitely presented $\mathbb{R}^2$-indexed persistence module. Indeed, as we will now explain, a FI-rep $(\text{gr}_1, \text{gr}_2, D_1, D_2)$ of an $\mathbb{R}^2$-indexed persistence module M induces a discrete FI-rep $\Phi = (\text{gr}'_1, \text{gr}'_2, D_1, D_2)$ and an injective 2-D grid $\mathcal{G} : \mathbb{Z}^2 \to \mathbb{R}^2$ such that M is a continuous extension of $H\Phi$ along $\mathcal{G}$. (The matrices D_1, D_2 are the same in the two free implicit representations). Given this, we can compute the multigraded Betti numbers of $H\Phi$ using the algorithm of [41] and deduce the multigraded Betti numbers of M from those of $H\Phi$, by Proposition 2.10.

The construction of Φ and the 2-D grid $\mathcal{G}$ are simple: Let o_x (respectively, o_y) denote the ordered set of unique x-coordinates (y-coordinates) of elements of $\text{im}\,\text{gr}_1 \cup \text{im}\,\text{gr}_2$. Let $n_x = |o_x|$ and $n_y = |o_y|$. Let $\Omega_x : [n_x] \to o_x$ be the bijection sending i to the ith element of o_x; define $\Omega_y : [n_y] \to o_y$ analogously. We choose $\mathcal{G} : \mathbb{Z}^2 \to \mathbb{R}^2$ to be an arbitrary extension of

$$\Omega_x \times \Omega_y : [n_x] \times [n_y] \to o_x \times o_y.$$

For $j = 1, 2$, we define gr'_j to be the $\mathbb{Z}^2$-valued function $(\Omega_x \times \Omega_y)^{-1} \circ \text{gr}_j$.

We leave to the reader the easy check that M is a continuous extension of $H\Phi$ along $\mathcal{G}$.

5.4 Computation and Storage of Anchors and Template Points

Recall from Section 3.1 that an *anchor* is the least upper bound of a weakly incomparable pair of points in S, and that the set of anchors determines the line arrangement $\mathcal{A}(M)$. In this section, we will let A denote the set of anchors. To compute the line arrangement $\mathcal{A}(M)$, we need to first compute a list anchors of all elements of A.

Moreover, our algorithm for computing the barcode templates, described in Section 6, requires us to represent the set

$$\mathcal{P} := \bigcup_{e \text{ a 2-cell in } \mathcal{A}(M)} \mathcal{P}^e$$

of all template points using a certain sparse matrix data structure. It's easy to see that

$$\mathcal{P} = A \cup S.$$

We will see that because of this, it is convenient to compute the list anchors and the sparse matrix representation of $\mathcal{P}$ at the same time.

In this section, we specify our data structure for $\mathcal{P}$ and describe our algorithm for simultaneously computing both this data structure and the list anchors.

Sparse Matrix Data Structure for $\mathcal{P}$. Note that $S \subseteq \text{im}(\Omega_x \times \Omega_y)$. Given this, it's easy to see that also $\mathcal{P} \subseteq \text{im}(\Omega_x \times \Omega_y)$. Thus, to store $\mathcal{P}$, it suffices to store $(\Omega_x \times \Omega_y)^{-1}(\mathcal{P})$ and the maps Ω_x and Ω_y. We store Ω_x and Ω_y in two arrays of size n_x and n_y, and we

store $(\Omega_x \times \Omega_y)^{-1}(\mathcal{P})$ in a sparse matrix tPtsMat of size $n_x \times n_y$. The triple (tPtsMat, Ω_x, Ω_y) is our data structure for $\mathcal{P}$.

Henceforth, to keep notation simple, we will assume that Ω_x and Ω_y are the identity maps on $[n_x]$ and $[n_y]$ respectively, so that $(\Omega_x \times \Omega_y)^{-1}(\mathcal{P}) = \mathcal{P}$.

Let us now describe tPtsMat in detail. An example tPtsMat is shown in Fig. 10. Each element $u \in \mathcal{P}$ is represented in tPtsMat by a quintuple

$$(u, p_l, p_d, \mathsf{levSet}_1(u), \mathsf{levSet}_2(u)).$$

In this quintuple, p_l and p_d are pointers (possibly null); p_l points to the element of $\mathcal{P}$ immediately to the left of u, and p_d points to the element of $\mathcal{P}$ immediately below u. The objects $\mathsf{levSet}_1(u)$, $\mathsf{levSet}_2(u)$ are lists used in the computation of the barcode templates $\mathcal{T}^e$. Initially, these lists are empty. We discuss them further in Section 6.

The data structure tPtsMat also contains an array rows of pointers, of length n_y; the ith entry of rows points to the rightmost element of $\mathcal{P}$ with y-coordinate i.
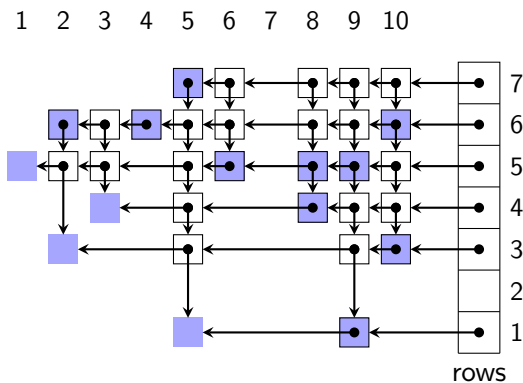
Figure 10: Example of tPtsMat for $n_x = 10$ and $n_y = 7$. Each element of $\mathcal{P}$ is represented by a square. Shaded squares represent elements of S, and squares with solid borders represent anchors. Each entry contains pointers to the next entries down and to the left; non-null pointers are illustrated by arrows. The lists $\mathsf{levSet}_1(u)$ and $\mathsf{levSet}_2(u)$ stored at each $u \in \mathcal{P}$ are not shown.

Computation of tPtsMat and anchors. Our algorithm for computing Betti numbers, given in [41], computes $\xi_0(M)(u)$ and $\xi_1(M)(u)$ at each bigrade $u \in [n_x] \times [n_y]$ by iterating through $[n_x] \times [n_y]$ in lexicographical order. As we iterate through $[n_x] \times [n_y]$, it is easy to also compute both tPtsMat and the list anchors. Let us explain this in detail.

Upon initialization, anchors is empty, tPtsMat contains no entries, and all pointers in rows are null. We create a temporary array columns of pointers, of length n_x, with each pointer initially null.

At each $u \in [n_x] \times [n_y]$, once the Betti numbers at u have been computed, if $u \in A$, then we add u to the list anchors. If $u \in S \cup A = \mathcal{P}$, then we add the quintuple $(u, \mathsf{rows}[u_2], \mathsf{columns}[u_1], [\,], [\,])$ to tPtsMat, and set $\mathsf{rows}[u_2]$ and $\mathsf{columns}[u_1]$ to both point to u. These updates to columns and rows ensure that:

- For each $i \in [n_y]$, $\mathsf{rows}[i]$ always points to the rightmost entry with y-coordinate i added to tPtsMat thus far;

- For each $j \in [n_x]$, $\mathsf{columns}[j]$ always points to the topmost entry with x-coordinate j added to tPtsMat thus far.

It remains to explain how we determine whether $u \in A$. Note that $u \in A$ if and only if at least one of the following two conditions holds when we visit u:

1. Both $\mathsf{rows}[u_2]$ and $\mathsf{columns}[u_1]$ are not null.

2. $u \in S$ and *either* $\mathsf{rows}[u_2]$ or $\mathsf{columns}[u_1]$ is not null.

Using this fact, we can check whether $u \in A$ in constant time.

Beyond the $O(m^3)$ time required to compute S, this algorithm for computing tPtsMat and anchors takes $O(n_x \times n_y) = O(m^2)$ time.

5.5 Building the Line Arrangement

Recall that the anchors of M correspond under point-line duality to the lines in the arrangement $\mathcal{A}(M)$. Thus, once the list of anchors has been determined, we are ready to build the DCEL representation of $\mathcal{A}(M)$. For this, our implementation of RIVET uses the well-known Bentley-Ottmann algorithm [29], which constructs the DCEL representation of a line arrangement with n lines and k vertices in time $O((n+k)\log n)$. Since our arrangement contains $O(\kappa)$ lines and $O(\kappa^2)$ vertices, the algorithm requires $O(\kappa^2 \log \kappa)$ elementary operations.

As explained in Section 3.1, the number of cells in $\mathcal{A}(M)$ is $O(\kappa^2)$. The size of the DCEL representation of any arrangement is of order the number of cells in the arrangement, so the size of the DCEL representation of $\mathcal{A}(M)$ is also $O(\kappa^2)$.

Remark 5.1. The $\kappa^2 \log \kappa$ term in the bound of Theorem 1.4 (ii) arises from our use of the Bentley-Ottmann algorithm. There are asymptotically faster algorithms for constructing line arrangements that would give a slightly smaller term in the bound of Theorem 1.4 (ii)— in fact, we can remove the $\log \kappa$ factor. However, the Bentley-Ottmann algorithm, which is relatively simple and performs well in practice, is a standard choice.

Each 1-cell e in $\mathcal{A}(M)$ lies on the line dual to some anchor α. In our DCEL representation of $\mathcal{A}(M)$, we store a pointer at e to the entry of tPtsMat corresponding to α.

Numerical Considerations.   Line arrangement computations, as with many computations in computational geometry, are notoriously sensitive to numerical errors that arise from floating-point arithmetic. Much effort has been invested in the development of smart arithmetic models for computational geometry which allow us to avoid the errors inexact arithmetic can produce, without giving up too much computational efficiency [7]. Because exact arithmetic is generally far more computationally expensive than floating-point arithmetic, these models typically take a hybrid approach, relying on floating-point arithmetic in cases where the resulting errors are certain to not cause problems, and switching over to exact arithmetic for calculations where the errors could be problematic. Our implementation of RIVET relies on a simple such hybrid model specially tailored to the problem at hand.

6   Computing the Barcode Templates

Once we have found all anchors and constructed the line arrangement $\mathcal{A}(M)$, we are ready to complete the computation of $\mathcal{A}^\bullet(M)$ by computing the barcode templates $\mathcal{T}^e$ for all 2-cells e of $\mathcal{A}(M)$. This section describes our core algorithm for this. In Section 8.1, we describe a refinement of the algorithm which performs significantly faster in practice.

The input to our algorithm consists of three parts:

1. A FI-rep Φ of M, represented in the way described in Section 5.1,

2. Our sparse matrix representation tPtsMat of the set $\mathcal{P}$ of all template points,

3. A DCEL representation of the line arrangement $\mathcal{A}(M)$.

Recall that Φ is given to us as input to our algorithm for computing $\mathcal{A}^\bullet(M)$; the computation of tPtsMat and $\mathcal{A}(M)$ from Φ has already been described above.

Recall from Section 3.3 that

$$\mathcal{T}^e = \{(\zeta^e(y), \zeta^e(z)) \mid [y, z) \in \mathcal{B}(M^e)\}.$$

Thus, to compute $\mathcal{T}^e$ at each 2-cell e, it suffices to compute the pair $(\mathcal{B}(M^e), \zeta^e)$ at each 2-cell e. This is essentially what our algorithm does, though it turns out to be unnecessary to explicitly store either $\mathcal{B}(M^e)$ or ζ^e at any point in the computation.

Note that $M \cong 0$ if and only if $S = \emptyset$, if and only if each $\mathcal{T}^e = \emptyset$. Thus, we may assume that $M \not\cong 0$.

6.1   Trimming the Free Implicit Representation

Let

$$\mathrm{box}(S) = \{y \in \mathbb{R}^2 \mid y \leq \mathrm{LUB}\, S\}.$$

We say a FI-rep $\Phi' = (\mathrm{gr}'_1, \mathrm{gr}'_2, D'_1, D'_2)$ for M is *trimmed* if

$$\mathrm{im}\,\mathrm{gr}'_1 \cup \mathrm{im}\,\mathrm{gr}'_2 \in \mathrm{box}(S).$$

As a preliminary step in preparation for the computation of the barcode templates, if Φ is not already trimmed we replace Φ with a smaller trimmed FI-rep. While it is possible to work directly with an untrimmed FI-rep to compute the barcode templates, it is more efficient to work with a trimmed one. In addition, assuming that our FI-rep is trimmed allows for some simplifications in the description of our algorithm.

For $j = 1, 2$, let $l_j = |\mathrm{gr}_j^{-1}(\mathrm{box}(S))|$. There is a unique order-preserving map

$$\eta_j : [l_j] \to \mathrm{gr}_j^{-1}(\mathrm{box}(S)).$$

We define $\mathrm{gr}'_j = \mathrm{gr}_j \circ \eta_j$. We define D'_1 to be the submatrix of D_1 whose columns correspond to elements of $\mathrm{im}\,\eta_1$, and we define D'_2 to be the submatrix of D_2 whose rows and columns correspond to elements of $\mathrm{im}\,\eta_1$ and $\mathrm{im}\,\eta_2$, respectively. Let

$$\Phi' = (\mathrm{gr}'_1, \mathrm{gr}'_2, D'_1, D'_2).$$

Proposition 6.1. $H(\Phi') \cong M$.

Proof. Associated to Φ and Φ' we have respective chain complexes of free modules

$$F_2 \xrightarrow{\partial_2} F_1 \xrightarrow{\partial_1} F_0$$

$$F'_2 \xrightarrow{\partial'_2} F'_1 \xrightarrow{\partial'_1} F'_0,$$

with $\ker \partial_1 / \mathrm{im}\,\partial_2 = H(\Phi) \cong M$ and $\ker \partial'_1 / \mathrm{im}\,\partial'_2 = H(\Phi')$, as in the definition of a FI-rep; further, since Φ' is a trimming of Φ, we have obvious maps $\psi^j : F'_j \hookrightarrow F_j$ for $j = 0, 1, 2$, making the following diagram commute:

$$
\begin{array}{ccccc}
F_2 & \xrightarrow{\partial_2} & F_1 & \xrightarrow{\partial_1} & F_0 \\
{\scriptstyle\psi^2}\big\uparrow & & {\scriptstyle\psi^1}\big\uparrow & & {\scriptstyle\psi^0}\big\uparrow \\
F'_2 & \xrightarrow[\partial'_2]{} & F'_1 & \xrightarrow[\partial'_1]{} & F'_0
\end{array}
$$

The maps ψ^i induce a map $\psi : H(\Phi') \to H(\Phi) \cong M$. It's easy to see that ψ_a is an isomorphism for $a \in \mathrm{box}(S)$. To finish the proof, it remains to check that ψ_b is also an isomorphism for $b \notin \mathrm{box}(S)$. For $b \notin \mathrm{box}(S)$, there is a unique element $a \in \mathrm{box}(S)$ minimizing the distance to b. Note that $a < b$. By commutativity, it suffices to see that $M(a, b)$ is an isomorphism and $H(\Phi')(a, b)$ is an isomorphism. Lemma 2.8 gives that $M(a, b)$ is an isomorphism, and since $\mathrm{im}\,\mathrm{gr}'_1 \cup \mathrm{im}\,\mathrm{gr}'_2 \in \mathrm{box}(S)$, it's easy to see directly that $H(\Phi')(a, b)$ is an isomorphism. $\qquad\square$

Clearly, Φ' is trimmed. Given tPtsMat and our column-sparse representation of Φ, we can compute Φ' from Φ in $O(m + d)$ time, where $d = O(m^2)$ is the number of non-zero entries of D_2.

Henceforth we will assume that Φ, the FI-rep of M given as input to our algorithm, is trimmed.

6.2 RU-Decompositions and Computation of Persistence Barcodes

To prepare for a description of our algorithm for computing the barcode templates, we begin with some preliminaries on the computation of persistence barcodes. There is a large and growing body of work on this topic; see [42] for a recent overview with an emphasis on publicly available software. We restrict attention here to what is needed to explain our algorithm.

The *standard algorithm* for computing persistence barcodes was introduced in [47], building on ideas in [32]; see also [31] for a succinct description of the algorithm, together with implementation details.

The algorithm takes as input a FI-rep $\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$ of a 1-D persistence module, with gr_1 and gr_2 non-decreasing, and returns $\mathcal{B}(H\Phi)$. Of course, in applications Φ typically comes from the chain complex of a 1-D filtration, with the simplices in each dimension ordered according to their grade of appearance.

The algorithm is a variant of Gaussian elimination; it performs column additions to construct certain factorizations of D_1 and D_2, from which the barcode $\mathcal{B}(M)$ can be read off directly. Let us explain this in more detail, drawing on ideas introduced in [25]: Let R be an $m \times n$ matrix, and for j the index of a non-zero column of R, let $\Lambda(R, j)$ denote the maximum row index of a non-zero entry in column j of R. We say R is *reduced* if $\Lambda(R, j) \neq \Lambda(R, j')$ whenever $j \neq j'$ are the indices of non-zero columns in R.

The standard persistence algorithm yields a decomposition $D = RU$ of any matrix D with coefficients in the field K, where R is reduced, and U is an upper-triangular matrix. For D an $r \times s$ matrix, the algorithm runs in time $O(rs^2)$.

We define an RU-decomposition of Φ simply to be a pair of RU-decompositions $D_1 = R_1U_1$ and $D_2 = R_2U_2$. We can read $\mathcal{B}(M)$ off of R_1 and R_2. To explain this, suppose Φ is of dimensions (m_0, m_1, m_2), and let $R_i(*, j)$ denote the jth column of R_i. Define

$$\mathrm{Pairs}(\Phi) := \{(\Lambda(R_2, j), j) \in [m_1] \times [m_2] \mid R_2(*, j) \neq 0\}$$
$$\mathrm{Ess}(\Phi) := \{j \in [m_1] \mid R_1(*, j) = 0 \text{ and } j \neq \Lambda(R_2, k) \text{ for any column } k \text{ of } R_2.\}$$

While the RU-decomposition of a matrix is not unique, it is shown in [25] that $\mathrm{Pairs}(\Phi)$ and $\mathrm{Ess}(\Phi)$ are independent of the choice of RU-decomposition of Φ.

Theorem 6.2 ([31, 47])**.**

$$\mathcal{B}(M) = \big\{[\mathrm{gr}_1(j), \mathrm{gr}_2(k)) \mid (j, k) \in \mathrm{Pairs}(\Phi)\big\} \cup \big\{[\mathrm{gr}_1(j), \infty) \mid j \in \mathrm{Ess}(\Phi)\big\}.$$

Vineyard Updates to Barcode Computations. Suppose that D is a $d_1 \times d_2$ matrix, and that D' is obtained from D by transposing either two adjacent rows or two adjacent columns of D. [25] introduces an algorithm, known as the *vineyard algorithm*, for updating an RU-decomposition of D to obtain an RU-decomposition of D' in time $O(d_1 + d_2)$. This algorithm is an essential subroutine in our algorithm for computing the barcode templates.

6.3   Permutations of Free Implicit Representations

As mentioned above, the standard persistence algorithm takes as input a FI-rep of a 1-D persistence module $\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$, with gr_1 and gr_2 non-decreasing. The reason we need gr_1 and gr_2 to be non-decreasing is that the formula Theorem 6.2 for reading the barcode off of the RU-decomposition holds only under this assumption on gr_1 and gr_2.

Now suppose that we are given a 1-D FI-rep $\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$ with either gr_1 or gr_2 *not* non-decreasing. How can we modify Φ to obtain a FI-rep Φ' of $H(\Phi)$ with non-decreasing grade functions, so that we can read the barcode of $\mathcal{H}(\Phi)$ off of an RU-decomposition of Φ'? We now answer this question.

It is easy to check that the following lemma holds.

Lemma 6.3. *Suppose* $\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$ *is a FI-rep of an n-D persistence module* M *of dimensions* (m_0, m_1, m_2). *Then for* σ_1 *and* σ_2 *any permutations on* $[m_1]$ *and* $[m_2]$, *respectively, and* P_1 *and* P_2 *the corresponding permutation matrices, we have that*

$$\Phi' := (\mathrm{gr}_1 \circ \sigma_1, \mathrm{gr}_2 \circ \sigma_2, D_1 P_1, P_1^T D_2 P_2)$$

is also a FI-rep of M.

Thus, in the case that M is 1-D, finding a FI-rep of M with non-decreasing grade functions amounts to finding permutations σ_1 and σ_2 as above with $\mathrm{gr}_1 \circ \sigma_1$ and $\mathrm{gr}_2 \circ \sigma_2$ non-decreasing, and applying the corresponding permutations to the rows and columns of P_1 and P_2.

Finding the Permutations σ_j by Sorting. For $j = 1, 2$, we may use a sorting algorithm to find a permutation σ_j^{-1} which puts the list

$$\mathrm{gr}_j(1), \mathrm{gr}_j(2), \ldots, \mathrm{gr}_j(m_j)$$

in non-decreasing order. The function $\mathrm{gr}_i \circ \sigma_j$ is then non-decreasing. To take advantage of the vineyard algorithm in our main algorithm, we will want to work with a sorting algorithm which generates the permutation σ_j^{-1} as a product of transpositions of adjacent elements in $[m_j]$. For this, we use the well known *insertion-sort* algorithm [26]. This yields σ_j as a minimum length product of adjacent transpositions.

6.4 Induced Free Implicit Representations at Each 2-Cell

Using Lemma 6.3 above, we next show that for any 2-cell e in $\mathcal{A}(M)$, Φ yields a FI-rep

$$\Phi^e = (\mathrm{gr}_1^e, \mathrm{gr}_2^e, D_1^e, D_2^e)$$

of the discrete persistence module M^e introduced in Section 3.3, with gr_1^e and gr_2^e non-decreasing. Thus, we can compute $\mathcal{B}(M^e)$ by computing an RU-decomposition of Φ^e.

Lift Maps. Recall the definition of the set of template points $\mathcal{P}^e$ from Section 3.3. Define a function

$$\mathrm{lift}^e : \mathrm{box}(S) \to \mathcal{P}^e$$

by taking $\mathrm{lift}^e(a) = u$, for u the minimum element of $\mathcal{P}^e$ such that $a \leq u$ (where as elsewhere, $<$ denotes the partial order order $\mathbb{R}^2$). In Fig. 11 we illustrate lift^e and $\mathrm{lift}^{e'}$ for a pair of adjacent 2-cells e and e'.
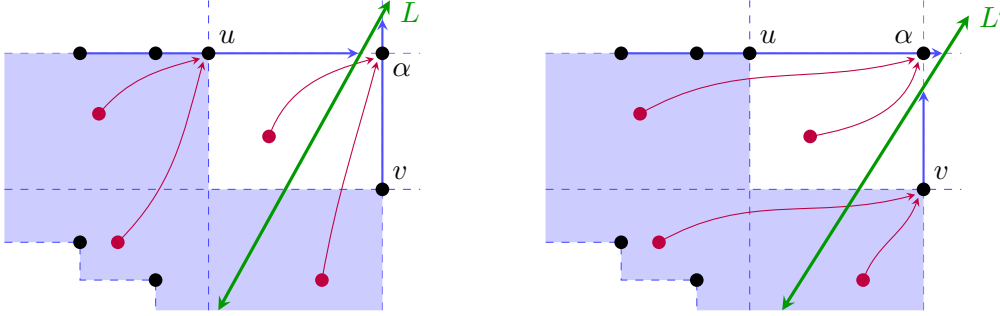
Figure 11: Illustration of lift^e (left) and $\mathrm{lift}^{e'}$ (right) at two adjacent cells e and e', containing the duals of lines L and L', respectively. The black dots represent points of $\mathcal{P} = \bigcup_e \mathcal{P}^e$. Note that $u, \alpha \in \mathcal{P}^e$ and $v, \alpha \in \mathcal{P}^{e'}$, but $u \notin \mathcal{P}^{e'}$ and $v \notin \mathcal{P}^e$. The maps lift^e, $\mathrm{lift}^{e'}$ are illustrated by red arrows, for a few sample points (purple dots). The shaded region in each figure is the subset of $\mathrm{box}(S)$ on which $\mathrm{lift}^e \neq \mathrm{lift}^{e'}$.

Let

$$\mathrm{ord}^e : \mathcal{P}^e \to \{1, 2, \ldots, |\mathcal{P}^e|\}$$

denote the unique order-preserving bijection, i.e., the inverse of the restriction of ζ^e to $\{1, 2, \ldots, |\mathcal{P}^e|\}$.

Free Implicit Representation of M^e. Suppose that $\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$, our FI-rep of M, is of dimensions (m_0, m_1, m_2). For $j = 1, 2$, let $\sigma_j^e : [m_j] \to [m_j]$ be any permutation such that

$$\mathrm{gr}_j^e := \mathrm{ord}^e \circ \mathrm{lift}^e \circ \mathrm{gr}_j \circ \sigma_j^e : [m_j] \to \mathbb{Z}$$

45

is non-decreasing, and write $\sigma^e = (\sigma_1^e, \sigma_2^e)$.

Let P_1 and P_2 denote the permutation matrices corresponding to σ_1^e and σ_2^e, respectively. Let $D_1^e = D_1 P_1$ and $D_2^e = P_1^T D_2 P_2$.

Proposition 6.4. $\Phi^e := (\mathrm{gr}_1^e, \mathrm{gr}_2^e, D_1^e, D_2^e)$ *is a FI-rep of* M^e.

Proof. It's not hard to check that

$$(\mathrm{ord}^e \circ \mathrm{lift}^e \circ \mathrm{gr}_1, \; \mathrm{ord}^e \circ \mathrm{lift}^e \circ \mathrm{gr}_2, D_1, D_2)$$

is a FI-rep of M^e. Given this, the result follows from Lemma 6.3. $\qquad\square$

Non-Uniqueness of Φ^e. In general, Φ^e is not uniquely defined, because it depends on the pair of permutations σ^e, which needn't be unique. We will sometimes write $\Phi^e = \Phi(\sigma^e)$ to emphasize the dependence on σ^e. We say any σ^e chosen as above is *valid*.

Reading off $\mathcal{T}^e$ from an RU-Decomposition of Φ^e. For $j = 1, 2$, let

$$f_j^e = \mathrm{lift}^e \circ \mathrm{gr}_j \circ \sigma_j^e = (\mathrm{ord}^e)^{-1} \circ \mathrm{gr}_j^e,$$

and write $f^e = (f_1^e, f_2^e)$. We call f^e the *template map* for Φ^e. Note that f^e is independent of the choice of a valid σ^e.

From Theorem 6.2, Proposition 6.4, and the definition of the barcode template $\mathcal{T}^e$ in Section 3.3, we have the following relationship between Φ^e, f^e, and $\mathcal{T}^e$:

$$\mathcal{T}^e = \big\{ (f_1^e(j), f_2^e(k)) \mid (j, k) \in \mathrm{Pairs}(\Phi^e) \big\} \cup \big\{ (f_1^e(j), \infty) \mid j \in \mathrm{Ess}(\Phi^e) \big\}. \qquad (6.1)$$

6.5   Our Algorithm

Eq. (6.1) tells us that for a 2-cell e in $\mathcal{A}(M)$, to compute the barcode template $\mathcal{T}^e$ it suffices to compute the template map f^e, along with the RU-decomposition of $\Phi^e = \Phi(\sigma^e)$, for some valid σ^e. This is exactly what our algorithm does; we give a description of the algorithm here, deferring some details to later sections.

We need to compute $\mathcal{T}^e$ at every 2-cell e. A naïve approach would be to do the computation from scratch at each 2-cell e. However, we can do better, by leveraging the work done at one cell to expedite the computation at a neighboring cell.

We proceed as follows: Let G denote the *dual graph* of $\mathcal{A}(M)$; this is the undirected graph with a vertex for each 2-cell e of $\mathcal{A}(M)$, and an edge $[e, e']$ for each pair of adjacent 2-cells $e, e' \in \mathcal{A}(M)$. The dual graph is illustrated in Fig. 12. We compute a path $\Gamma = e_0, e_1, \ldots e_w$ in G which visits each 2-cell at least once. Our algorithm for computing Γ is discussed below, in Section 6.6.

Let us adopt the convention of abbreviating an expression of the form $(-)^{e_i}$ by $(-)^i$. For example, we write f^{e_i} as f^i.

46
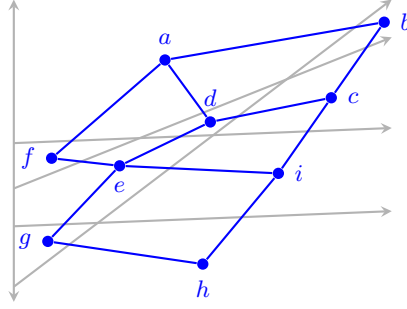
Figure 12: The line arrangement $\mathcal{A}(M)$ (in grey), together with its dual graph G (in blue). The path Γ through G might visit the vertices in the order $a, b, c, d, e, f, e, g, h, i$.

Once we have computed Γ, for each $i = 1, \ldots, w$, we compute the template map f^i and an RU-decomposition of $\Phi(\sigma^i) = (\mathrm{gr}_1^i, \mathrm{gr}_2^i, D_1^i, D_2^i)$, for some valid choice of σ^i. We proceed in order of increasing i.

For $j = 1, 2$, we store f_j^i in memory by separately storing its factors $\mathrm{lift}^{i-1} \circ \mathrm{gr}_j$ and σ_j^{i-1}; we discuss the data structures for this in Section 6.7. Thus, to compute f_j^i, we compute both $\mathrm{lift}^{i-1} \circ \mathrm{gr}_j$ and σ_j^{i-1}.

The initial cell e_0 is chosen in a way that allows for a simple combinatorial algorithm to compute $\mathrm{lift}^0 \circ \mathrm{gr}_j$ and σ_j^0; see Section 6.8. Letting P_j denote the matrix representation of σ_j^0, we have that

$$D_1^0 = D_1 P_1, \qquad D_2^0 = P_1^T D_2 P_2.$$

Thus, D_1^0 and D_2^0 can be obtained from D_1 and D_2 by performing row and column permutations. Given D_1^0 and D_2^0, we compute an RU-decomposition of Φ^0 via an application of the standard persistence algorithm.

For $1 \leq i \leq w$, we compute f^i as an update of f^{i-1}, and we compute the RU-decomposition of Φ^i as an update of the RU-decomposition of Φ^{i-1}. Let us explain this in more detail (with yet more detail to come in later sections). To update f_j^{i-1}, we update both $\mathrm{lift}^{i-1} \circ \mathrm{gr}_j$ and σ_j^{i-1}. First, we update $\mathrm{lift}^{i-1} \circ \mathrm{gr}_j$ to obtain $\mathrm{lift}^i \circ \mathrm{gr}_j$; details of this computation are given in Section 6.9. Second, we update σ_j^{i-1} to obtain σ_j^i as follows. Define

$$\tilde{\mathrm{gr}}_j^i := \mathrm{ord}_i \circ \mathrm{lift}^i \circ \mathrm{gr}_j \circ \sigma_j^{i-1}.$$

Note the distinction between $\tilde{\mathrm{gr}}_j^i$ and gr_j^i: the former is defined in terms of σ_j^{i-1}, while the latter is defined in terms of σ_j^i.

Applying the insertion-sort algorithm to $\tilde{\mathrm{gr}}_j^i$, we compute a sequence of transpositions of adjacent elements in $[m_j]$ such that, for $\tau_j^{-1} : [m_j] \to [m_j]$ the composition of these

47

transpositions, $\tilde{\text{gr}}_j^i \circ \tau_j$ is non-decreasing. We take $\sigma_j^i = \sigma_j^{i-1} \circ \tau_j$. Clearly, then,

$$\text{gr}_j^i = \text{ord}_i \circ \text{lift}^i \circ \text{gr}_j \circ \sigma_j^i$$

is non-decreasing, so σ^i is valid.

Note that for P_j the matrix representation of τ_j, we have

$$D_1^i = D_1^{i-1} P_1, \qquad D_2^i = P_1^T D_2^{i-1} P_2.$$

To compute an RU-decomposition of $\Phi^i(\sigma^e)$ from our RU-decomposition of $\Phi^{i-1}(\sigma^e)$, we exploit the decomposition of τ_j^{-1} as a sequence of transpositions provided by the insertion-sort algorithm, and apply the RU-update algorithm of [25] repeatedly, performing an update of the RU-decomposition for each transposition in the sequence. We note that neither τ, nor τ^{-1}, nor the decomposition of τ_j^{-1} into transpositions ever needs to be stored explicitly in memory. Rather, we use each transposition to perform part of the update as soon as it is computed; after this, there is no need to store the transposition.

It remains to explain how we compute Γ, $\text{lift}^0 \circ \text{gr}_j$, and σ^0, and how we update $\text{lift}^{i-1} \circ \text{gr}_j$ to obtain $\text{lift}^i \circ \text{gr}_j$. In what follows, we explain all of this, and also fill in some details about the data structures used by our algorithm.

6.6 Computing the Path Γ

We first explain how we choose the path $\Gamma = e_0, e_1, \ldots, e_w$. As above, let G be the dual graph of the line arrangement $\mathcal{A}(M)$. To compute Γ, we first compute a weight for each edge of G. The weight of $[e, e']$ is chosen to be an estimate of the amount of work our algorithm must do to pass between cells e and e', in either direction. We defer the details of how we define and compute these edge weights until Section 6.10. While the choice of edge weights impacts our choice of Γ, and hence the speed of our algorithm for computing the barcode templates, our asymptotic complexity bounds for our algorithm are independent of the choice of edge weights.

We take e_0 to be the topmost 2-cell in $\mathcal{A}(M)$; the points of e_0 correspond under point-line duality to lines that pass to the right of all points in S.

Call a path in G starting at e_0 and visiting every vertex of G a *valid* path. We'd like to choose Γ to be a valid path of minimum length, but we do not have an efficient algorithm for computing such a path. (Indeed, we expect the problem is NP-hard). Instead, we compute a path Γ whose length is *approximately* minimum.

Let Γ^* be a valid path of minimum length. It is straightforward to compute a valid path Γ such that $\text{length}(\Gamma) \leq 2\,\text{length}(\Gamma^*)$: First we compute a minimum spanning tree $\mathcal{M}$ for G via a standard algorithm such as Kruskal's Algorithm [26]. Via depth-first search of $\mathcal{M}$ starting at e_0, we can find a valid path Γ in $\mathcal{M}$ which traverses each edge of $\mathcal{M}$ at most twice. Since $\text{length}(\mathcal{M}) \leq \text{length}(\Gamma^*)$, we have that $\text{length}(\Gamma) \leq 2\,\text{length}(\Gamma^*)$.

In fact, an algorithm with a better approximation ratio is known: [35] shows that a variant of the Christofides $\frac{3}{2}$-approximation algorithm for the traveling salesman problem on a metric graph [23] yields a valid path Γ with length(Γ) $\leq \frac{3}{2}$length(Γ^*).

6.7 Data Structures

Before completing the specification of our algorithm for computing the barcode templates $\mathcal{T}^e$, we need to describe the data structures used internally by the algorithm.

Persistent Homology and Vineyard Update Data Structures. First, we mention that our algorithm uses each of the data structures specified in [25] for computing and updating RU-decompositions. These consist consists of 4 sparse matrices—one to store each of R_1, R_2, U_1 and U_2—as well as several additional 1-D arrays which aid in performing the persistence and vineyard algorithms, and in reading barcodes off of the matrices R_1, R_2. Since we use these data structures only in the way described in [25], we refer the reader to that paper for details.

Array Data Structures. For $j = 1, 2$, we also maintain arrays grades_j, lifts_j, sig_j, and sigInv_j, each of length $[m_j]$:

- grades_j is a static array with $\mathsf{grades}_j[k] = \mathrm{gr}_j(k)$.

- $\mathsf{lifts}_j[k]$ is an array of pointers to entries of $\mathsf{tPtsMat}$.

- After our computations at cell e_i are complete,

$$\mathsf{lifts}_j[k] = \mathrm{lift}^i \circ \mathrm{gr}_j(k), \quad \mathsf{sig}_j[k] = \sigma_j^i(k), \quad \mathsf{sigInv}_j[k] = (\sigma_j^i)^{-1}(k).$$

Remark 6.5. Note that after all computations at cell e_i are complete, we can use lifts_j and sig_j together to perform constant time evaluations of the template map

$$f_j^i = \mathrm{lift}^i \circ \mathrm{gr}_j \circ \sigma_j^i.$$

Together with the RU-decomposition of Φ^i, this allows us to efficiently read off $\mathcal{T}^e$ using Eq. (6.1).

The Lists $\mathsf{levSet}_j(u)$. We mentioned in Section 5.4 that for each $u \in \mathcal{P}$, we store lists $\mathsf{levSet}_1(u)$ and $\mathsf{levSet}_2(u)$ at the entry of $\mathsf{tPtsMat}$ corresponding to u. We now specify what these lists store: After our computations at cell e_i are complete, $\mathsf{levSet}_j(u)$ stores $(\mathrm{lift}^i \circ \mathrm{gr}_j \circ \sigma_j^i)^{-1}(u)$ for $u \in \mathcal{P}^i$, and $\mathsf{levSet}_j(u)$ is empty for $u \notin \mathcal{P}^i$. As we will see in Section 6.9, our algorithm uses the lists $\mathsf{levSet}_j(u)$ to efficiently perform the required updates as we pass from cell e_{i-1} to cell e_i.

6.8 Computations at the Initial Cell e_0

We next describe in more detail the computations performed at the initial cell e_0, building on the explanation of Section 6.5.

To begin our computations at cell e_0, for each $u \in \mathcal{P}$, we initialize

$$\mathsf{levSet}_j(u) = (\mathrm{lift}^0 \circ \mathrm{gr}_j)^{-1}(u).$$

Note that $(\mathrm{lift}^0 \circ \mathrm{gr}_j)^{-1}(u)$ is nonempty only if u is the rightmost element of $\mathcal{P}$ on the horizontal line passing through u. Thus, the elements $u \in \mathcal{P}$ such that $\mathsf{levSet}_j(u)$ is non-empty have unique y-coordinates.

To efficiently initialize the lists $\mathsf{levSet}_j(u)$, we use an $O(m \log m + m\kappa)$ time sweep algorithm, described in Appendix A.2. When we add $k \in [m_j]$ to the list $\mathsf{levSet}_j(u)$, we also set $\mathsf{lifts}_j[k] = u$.

Next, we concatenate the non-empty lists $\mathsf{levSet}_j(u)$ into single list of length m_j, in increasing order of the y-coordinate of u, and set sigInv_j equal to this list. For $j = 1, 2$, given sigInv_j, we construct sig_j in the obvious way in time $O(m_j)$. We define σ_j^0 to be the permutation whose array representation is sig_j.

Letting P_j denote the matrix representation of σ_j^0, we use the arrays sigInv_1 and sigInv_2 to compute

$$D_1^0 = D_1 P_1, \quad D_2^0 = P_1^T D_2 P_2.$$

Using the column sparse representation of D_1 and D_2 described in [25], which allows for implicit representations of row permutations, this takes $O(m)$ time.

As already explained in Section 6.5, we then apply the standard persistence algorithm to compute the RU-decompositions $D_1^0 = R_1^0 U_1^0$, $D_2^0 = R_2^0 U_2^0$.

This completes the work done by the algorithm at cell e_0.

6.9 Computations at Cell e_i, for $i \geq 1$

In Section 6.5, we outlined our algorithm for updating the template map f^{i-1} and RU-decomposition Φ^{i-1}, as we pass from cell e_{i-1} to e_1. We now give a more detailed account of this algorithm, filling in some details omitted earlier. As explained in Section 6.5, to update

$$f_j^{i-1} = \mathrm{lift}^{i-1} \circ \mathrm{gr}_j \circ \sigma^{i-1},$$

our algorithm separately updates the factors $\mathrm{lift}^{i-1} \circ \mathrm{gr}_j$ and σ_j^{i-1}.

While it is possible to first completely update $\mathrm{lift}^{i-1} \circ \mathrm{gr}_j$ and then update σ_j^{i-1} via an application of insertion sort, it is slightly more efficient to interleave the updates of the two factors, so that when we update the value of $\mathrm{lift}^{i-1} \circ \mathrm{gr}_j(k)$ for some $k \in [m_j]$, we immediately perform the insertion-sort transpositions necessitated by that update, along with the corresponding updates to the RU-decomposition. This is the approach we take.

We assume without loss of generality that e_{i-1} lies below e_i. The shared boundary of e_{i-1} and e_i lies on the line dual to some anchor $\alpha = (\alpha_1, \alpha_2)$. tPtsMat provides constant time access to the element $u = (u_1, u_2) \in \mathcal{P}$ immediately to the left of α and the element $v = (v_1, v_2) \in \mathcal{P}$ immediately below α, if such u and v exist. To keep our exposition simple, we will assume that u and v do both exist; the cases that either u or v do not exist are similar, but simpler. Note that

$$u, \alpha \in \mathcal{P}^{i-1}, \qquad v, \alpha \in \mathcal{P}^i, \qquad u \notin \mathcal{P}^i, \qquad v \notin \mathcal{P}^{i-1};$$

see Fig. 11.

Recall that for each i, $\mathrm{lift}^i \circ \mathrm{gr}_j$ is represented in memory using the data structures lifts_j and $\mathsf{levSet}_j(\cdot)$, whereas σ_j^i is represented using sig_j and sigInv_j.

To perform the required updates as we pass from cell e_{i-1} to cell e_i, we first iterate through the list $\mathsf{levSet}_j(\alpha)$ in decreasing order. For each $k \in \mathsf{levSet}_j(\alpha)$, if the y-coordinate of $\mathrm{gr}_j(k)$ is less than or equal to v_2, then $\mathrm{lift}^i \circ \mathrm{gr}_j(k) = v$; we remove k from $\mathsf{levSet}_j(\alpha)$, add k to the beginning of the list $\mathsf{levSet}_j(v)$, and set $\mathsf{lifts}_j[k] = v$.

If, on the other hand, the y-coordinate of $\mathrm{gr}_j(k)$ is greater than v_2, then $\mathrm{lift}^i \circ \mathrm{gr}_j(k) = \alpha$, and we do not perform any updates to lifts_j, $\mathsf{levSet}_j(\alpha)$, or $\mathsf{levSet}_j(v)$ for this value of k. If in addition, $\mathsf{lifts}_j[\mathsf{sigInv}_j[\mathsf{sig}_j[k] + 1]] = v$, then we apply insertion sort to update sig_j, sigInv_j, and the RU-decomposition; specifically, we compute $\mathsf{sortOneElement}(\mathsf{sig}_j[k])$, for $\mathsf{sortOneElement}$ the algorithm defined below.

Algorithm 1 $\mathsf{sortOneElement}(w)$

Input: $w \in \{1, \ldots, m_j - 1\}$ such that $\mathsf{lifts}_j[\mathsf{sigInv}_j[y]] \leq \mathsf{lifts}_j[\mathsf{sigInv}_j[z]]$ whenever $w < y < z$
Output: Updated sig_j and sigInv_j such that $\mathsf{lifts}_j[\mathsf{sigInv}_j[y]] \leq \mathsf{lifts}_j[\mathsf{sigInv}_j[z]]$ whenever $w \leq y < z$; correspondingly updated RU-decomposition
1: $y \leftarrow w$;
2: **while** $y < m_j$ and $\mathsf{lifts}_j[\mathsf{sigInv}_j[y]] > \mathsf{lifts}_j[\mathsf{sigInv}_j[y + 1]]$ **do**
3:      swap $\mathsf{sigInv}_j[y]$ and $\mathsf{sigInv}_j[y + 1]$
4:      $\mathsf{sig}_j[\mathsf{sigInv}_j[y]] \leftarrow y$
5:      $\mathsf{sig}_j[\mathsf{sigInv}_j[y + 1]] \leftarrow y + 1$
6:      Perform the corresponding updates to the RU-decomposition, as described in Section 6.5.
7:      $y \leftarrow y + 1$

Once we have finished iterating through the list $\mathsf{levSet}_j(\alpha)$, we next iterate through the list $\mathsf{levSet}_j(u)$ in decreasing order. For each $k \in \mathsf{levSet}_j(u)$, we perform updates exactly as we did above for elements of $\mathsf{levSet}_j(\alpha)$, with one difference: If the second coordinate of $\mathrm{gr}_j(k)$ is greater than v_2, then we must remove k from $\mathsf{levSet}_j(u)$, add k to the beginning of the list $\mathsf{levSet}_j(\alpha)$, and set $\mathsf{lifts}_j[k] = \alpha$.

When we have finished iterating through the list $\mathsf{levSet}_j(u)$, our updates at cell e_i are complete.

6.10 Choosing Edge Weights for G

We have seen in Section 6.6 that the path Γ depends on our choice of weights $w(E)$ on the edges E of G. We now explain how we choose and compute these weights. As we will explain in Section 8, computing these weights is also the first step in two practical improvements to our algorithm.

In practice, the cost of our algorithm for computing the barcode templates is dominated by the cost of updating the RU-decompositions. On average, we expect the cost of updating the RU-decomposition as we traverse edge E in G to be roughly proportional to the total number of insertion-sort transpositions performed. Thus, if it were the case that the average number $t(E)$ of transpositions performed as we traverse E were independent of the choice of path Γ, then it would be reasonable to take $w(E) = t(E)$. In fact, $t(E)$ does depend on Γ. Nevertheless, we can give a simple, computable estimate of $t(E)$ which is independent of Γ. We choose $w(E)$ to be this estimate. For $E = [e_1, e_2]$, our definition of $w(E)$ in fact depends only on the anchor line L containing the common boundary of e_1, e_2, so that we may write $w(E) = w(L)$.

To prepare for the definition of $w(L)$, we introduce some terminology, which we will also need in Section 7.3.

Switches and Separations. For e and e' adjacent 2-cells of $\mathcal{A}(M)$, we say $r, s \in \mathrm{box}(S)$ *switch at* (e, e') if either

$$\mathrm{lift}^e(r) < \mathrm{lift}^e(s) \quad \text{and} \quad \mathrm{lift}^{e'}(s) < \mathrm{lift}^{e'}(r),$$

or

$$\mathrm{lift}^{e'}(r) < \mathrm{lift}^{e'}(s) \quad \text{and} \quad \mathrm{lift}^e(s) < \mathrm{lift}^e(r).$$

Similarly, for r, s incomparable, we say r, s *separate at* (e, e') if either

$$\mathrm{lift}^e(r) \neq \mathrm{lift}^e(s) \quad \text{and} \quad \mathrm{lift}^{e'}(r) = \mathrm{lift}^{e'}(s),$$

or

$$\mathrm{lift}^e(r) = \mathrm{lift}^e(s) \quad \text{and} \quad \mathrm{lift}^{e'}(r) \neq \mathrm{lift}^{e'}(s).$$

We omit the straightforward proof of the following:

Lemma 6.6. *Suppose (e, e') and (f, f') are pairs of adjacent 2-cells of $\mathcal{A}(M)$, with the shared boundary of each pair lying on the same anchor line L. Then*

(i) $r, s \in \mathbb{R}^2$ switch at (e, e') if and only if r, s switch at (f, f').

(ii) r, s separate at (e, e') if and only if r, s separate at (f, f').

In view of Lemma 6.6, for L an anchor line, we say that $a, b \in \mathbb{R}^2$ *switch at L* if a, b switch at e, e' for any adjacent 2-cells e, e' whose boundary lies on L. Analogously, we speak of $a, b \in \mathbb{R}^2$ *separating at L*.

Lemma 6.7. *If r, s switch at any anchor line L, then r, s are incomparable.*

Proof. Suppose $L = \mathcal{D}_p(\alpha)$ for some anchor α. If r, s switch at L, then there exist u, v as in Fig. 11, and exchanging and r and s if necessary, we have that

$$r_1 \leq u_1 < s_1, \quad s_2 \leq v_2 < r_2. \qquad \square$$

Remark 6.8. Every time we cross an anchor line L, our algorithm performs one insertion-sort transposition for each pair $k, l \in [m_j]$ such that $\mathrm{gr}_j(k), \mathrm{gr}_j(l)$ switch at L. For each pair $k, l \in [m_j]$ such that $\mathrm{gr}_j(k), \mathrm{gr}_j(l)$ separate at L, the algorithm may only perform a corresponding insertion-sort transposition when crossing L in one direction—e.g., from above to below—and may sometimes not perform such a transposition even when crossing L in this direction.

It is reasonable to estimate, then, that for each pair $k, l \in [m_j]$ such that $\mathrm{gr}_j(k), \mathrm{gr}_j(l)$ separate at L, the algorithm performs a corresponding transposition roughly $\frac{1}{4}$ of the time.

Definition of $w(L)$. For an anchor line L, a finite set Y, and a function $f : Y \to \mathbb{R}^2$, define $\mathrm{sw}_L(f)$ (respectively $\mathrm{sep}_L(f)$) to be the number of unordered pairs $a, b \in Y$ such that $f(a), f(b)$ switch (respectively, separate) along L. Motivated by Remark 6.8, we define

$$w(L) = \mathrm{sw}_L(\mathrm{gr}_1) + \mathrm{sw}_L(\mathrm{gr}_2) + \frac{1}{4}\mathrm{sep}_L(\mathrm{gr}_1) + \frac{1}{4}\mathrm{sep}_L(\mathrm{gr}_2).$$

Computing the weights $w(L)$. The weights $w(L)$ can be be computed using a simplified version of our main algorithm for computing all barcode templates: First, we choose a path Q through the 2-cells of $\mathcal{A}(M)$ starting at e_0 and crossing every anchor line L once; for example, we can choose Q to be a path through the rightmost cells of $\mathcal{A}(M)$. We then run a variant of the algorithm for computing the barcode templates described above, using the path Q in place of P, and omitting all of the steps involving matrices and updates of RU-decompositions. For e, e' adjacent 2-cells in Q with shared boundary on the anchor line $L = \mathcal{D}_p(\alpha)$, we compute $w(L)$ as we pass from cell e to cell e'.

To explain how this works, let u, v be as in Section 6.9; for simplicity, assume that u and v exist, as we did there. For any pair of elements r, s that switch or separate at L, $\mathrm{lift}^e(r), \mathrm{lift}^e(s) \in \{u, \alpha\}$, so to compute $w(L)$ we only need to consider pairs whose elements lie in the lists $\mathsf{levSet}_j(u)$ and $\mathsf{levSet}_j(\alpha)$. Further, the lines $x = u_1$ and $y = v_2$ determine a decomposition of the plane into four quadrants, and whether r, s switch or separate is completely determined by which of these quadrants contain r and s; see Fig. 11. Using these observations, we can easily extend the update procedure described in Section 6.9 to compute the weight $w(L)$ as we cross from e into e'.

7 Cost of Computing and Storing the Augmented Arrangement

In this section, we prove Theorem 1.4, which bounds the cost of computing and storing $\mathcal{A}^{\bullet}(M)$. Recall that Theorem 1.4 is stated for persistence modules arising as the ith persistent homology of a bifiltration. Using language of FI-reps, we may state the result in a more general algebraic form:

Proposition 7.1. *Let M be 2-D persistence module of coarseness κ, and let Φ be a FI-rep of M of dimensions (m_0, m_1, m_2). Letting $m = m_0 + m_1 + m_2$, we have that*

1. *$\mathcal{A}^{\bullet}(M)$ is of size $O(m\kappa^2)$,*

2. *Our algorithm computes $\mathcal{A}^{\bullet}(M)$ using $O(m^3\kappa + m\kappa^2 \log \kappa)$ elementary operations, and requires $O(m^2 + m\kappa^2)$ storage.*

To see that Theorem 1.4 follows from Proposition 7.1, let $\mathcal{F}$ be a (1-critical or multi-critical) bifiltration of size l, and recall from section Section 5.2 that, using the construction of [16], in $O(l \log l)$ time we can compute a FI-rep of $H_i(\mathcal{F})$ of dimensions (m_0, m_1, m_2), with $m_0 + m_1 + m_2 = O(l)$.

7.1 Size of the Augmented Arrangement

We prove Proposition 7.1 (i) first. As noted in Section 5.5, the DCEL representation of $\mathcal{A}(M)$ is of size $O(\kappa^2)$.

At each 2-cell of $\mathcal{A}(M)$, we store the barcode template $\mathcal{T}^e$. By considering the *RU*-decomposition, we see that if
$$\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$$
is a FI-rep of a 1-D persistence module N of dimensions (m_0, m_1, m_2), then $|\mathcal{B}(N)| \leq m_1$. Hence, Proposition 6.4 implies that $|\mathcal{T}^e| \leq m_1$ for all 2-cells e. Therefore, our representation of $\mathcal{A}^{\bullet}(M)$ in memory is of total size $O(m_1\kappa^2) = O(m\kappa^2)$, as claimed.

7.2 Cost of Computing the Augmented Arrangement

We now turn to the proof of Proposition 7.1 (ii). As we have seen, our algorithm for computing $\mathcal{A}^{\bullet}(M)$ involves several sub-computations. For each sub-computation, a row in Table 1 lists the data computed by this sub-computation, a bound on the number of elementary operation required, and the sections in this paper where the details were discussed.

The bounds in the first four rows of Table 1 were explained earlier, so it remains to analyze the cost of our algorithm for computing the barcode templates $\mathcal{T}^e$. The computation of $\mathcal{T}^e$ itself involves a number of steps, whose individual time complexities we again list in Table 2.

Table 1: Cost of augmented arrangement sub-computations

Data	# Elem. Operations	Details in
Set $S = \operatorname{supp} \xi_0(M) \cup \operatorname{supp} \xi_1(M)$	$O(m^3)$	Section 5.3
Template points $\mathcal{P}$ (stored in tPtsMat) and list anchors	$O(m^2)$	Section 5.4
Arrangement $\mathcal{A}(M)$ (constructed via Bentley-Ottmann algorithm)	$O(\kappa^2 \log \kappa)$	Section 5.5
Data structures for point location	$O(\kappa^2 \log \kappa)$	Section 4.3
Barcode templates $\mathcal{T}^e$	$O(m^3\kappa + (m + \log \kappa)\kappa^2)$	Section 6

The bounds in all but the last four rows in Table 2 (above the double horizontal line) were either explained earlier, or are clear from the discussion presented. In the remainder of this section, we verify the last four bounds.

Cost of Updates of the $\mathsf{levSet}_j(\cdot)$ **and** lifts_j **at** e_i**,** $i \geq 1$**.** In the notation of Section 6.9, to update the lists $\mathsf{levSet}_j(\cdot)$ to their proper values at e_i, our algorithm considers performing an update for each element k in the lists $\mathsf{levSet}_j(u)$ and $\mathsf{levSet}_j(\alpha)$. In the worst case, for each cell e_i there are $O(m)$ such elements k to consider in total. For each k, the updates of $\mathsf{levSet}_j(\cdot)$ and lifts_j take constant time. Thus the total amount of work we need to do at cell e_i is $O(m)$. The path P contains $O(\kappa^2)$ 2-cells, so the total work to perform the updates over all 2-cells is $O(m\kappa^2)$, as claimed.

A Bound on the Total Number of Transpositions. To establish the next two bounds, we take advantage of the following result, which we prove in Section 7.3:

Proposition 7.2. *Our algorithm for computing all barcode templates performs a total of* $O(m^2\kappa)$ *insertion-sort transpositions on* sigInv_1 *and* sigInv_2*.*

Cost of Updates of sig_j**,** sigInv_j **at cells** e_i**,** $i \geq 1$**.** The cost of updating sig is proportional to the cost of updating sigInv_j. Thus it is immediate from Proposition 7.2 that the total cost of updating these arrays is $O(m^2\kappa)$.

Cost of Updates of RU**-decomposition at cells** e_i**,** $i \geq 1$**.** There are $O(\kappa^2)$ cells to consider; this gives us the $O(\kappa^2)$ term. For each transposition performed on sig_j, we call the vineyard algorithm, described in Section 6.2, at most twice. Each call to the vineyard algorithm takes time $O(m)$. By Proposition 7.2, then, the total cost of all vineyard updates performed is $O(m^3\kappa)$. This gives the desired bound.

Table 2: Cost of barcode template sub-computations

Data	# Elem. Operations	Details in
Trimming the FI-rep	$O(m^2)$	Section 6.1
Path Γ (found via the 2-approximation algorithm for the optimal path, using Kruskal's MST algorithm)	$O(\kappa^2 \log \kappa)$	Section 6.6
$\mathsf{levSet}_j(u)$, lifts_j, sig_j, sigInv_j at cell e_0	$O(m \log m + m\kappa)$	Section 6.8, Appendix A.2
RU-decomposition of Φ^0	$O(m^3)$	Section 6.2
Reading the barcode template $\mathcal{T}^e$ off of the RU-decomposition of Φ^i, for all $i \geq 0$	$O(m\kappa^2)$	Section 6.4, Section 6.7
$\mathsf{levSet}_j(u)$ and lifts_j at all cells e_i, $i \geq 1$	$O(m\kappa^2)$	Section 6.9
sig_j, sigInv_j at all e_i, $i \geq 1$	$O(m^2\kappa)$	Section 6.9
RU-decompositions at all e_i, $i \geq 1$	$O(m^3\kappa + \kappa^2)$	Section 6.9
Weights $w(L)$ for all anchor lines L	$O(m^2 + m\kappa)$	Section 6.10

Cost of Computing Weights $w(L)$. As explained in Section 6.10, we compute the edge weights $w(L)$ using a stripped-down variant of our algorithm for computing the barcode templates $\mathcal{T}^e$. Using Lemma 7.5 below, it can be checked that computing all edge weights takes time $O(m^2 + m\kappa)$.

Storage Requirements. By Proposition 7.1 (i), $\mathcal{A}^\bullet(M)$ itself is of size $O(m\kappa^2)$, so our algorithm for computing $\mathcal{A}^\bullet(M)$ requires at least this much storage. Our algorithm for computing the Betti numbers requires $O(m^2)$ storage, as do the persistence algorithm and the vineyard updates to RU-decompositions. The Bentley-Ottmann algorithm requires $O(\kappa^2)$ storage, as does Kruskal's algorithm. Constructing the search data structures used for queries of $\mathcal{A}^\bullet(M)$ also requires $O(\kappa^2)$ storage. From our descriptions of the data structures used in the remaining parts of our algorithm, it is clear that other steps of our algorithm for computing $\mathcal{A}^\bullet(M)$ do not require more than $O(m^2 + m\kappa^2)$ storage. The bound of Proposition 7.1 (ii) on the storage requirements of our algorithm follows.

7.3 Bounding Total Number of Transpositions Required to Compute all Barcode Templates

To complete our proof of Proposition 7.1 (ii), it remains to prove Proposition 7.2. To this end, for $r \in \text{box}(S)$, let

$$\mathcal{P}^{\geq r} := \{u \in \mathcal{P} \mid u \geq r\},$$
$$\text{lift}(r) := \{u \in \mathcal{P} \mid u = \text{lift}^e(r) \text{ for some 2-cell } e\}.$$

We leave to the reader the proofs of the following two lemmas.

Lemma 7.3. *For e a 2-cell, L a line with $\mathcal{D}_\ell(L) \in e$, and $u \in \mathcal{P}^{\geq r}$, $\text{lift}^e(r) = u$ if and only if the following two conditions hold:*

1. *$\text{push}_L(u)$ is the minimum element of $\text{push}_L(\mathcal{P}^{\geq r})$,*

2. *for all $u' \in \mathcal{P}^{\geq r}$, $u' \neq u$, with $\text{push}_L(u') = \text{push}_L(u)$, we have $u' < u$.*

Lemma 7.4. *For $u \in \mathcal{P}^{\geq r}$, we have that $u \in \text{lift}(r)$ if and only if the following two conditions hold:*
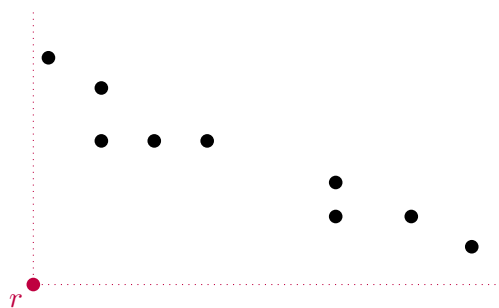
1. *there exists no $w \in \mathcal{P}^{\geq r}$ with*

$$w_1 < u_1 \quad and \quad w_2 < u_2.$$

2. *there exist no pair $v, w \in \mathcal{P}^{\geq r}$ with*

$$v_1 < u_1, \quad v_2 = u_2, \quad w_1 = u_1, \quad and \quad w_2 < w_2.$$

Fig. 13 illustrates the shape of the set $\text{lift}(r)$, as described by Lemma 7.4.

Figure 13: The shape of a set $\text{lift}(r)$, as described by Lemma 7.4.

The next lemma shows that the number of anchor lines at which a given pair of points in $\text{box}(S)$ can switch or separate is at most two; it is the key step in our proof of Proposition 7.2.

Lemma 7.5. *For $r, s \in \text{box}(S)$ incomparable,*

(i) *there is at most one anchor line L at which r and s switch, and if such L exists, then there is no anchor line at which r and s separate.*

(ii) *there are at most two anchor lines at which r and s separate.*

Proof. Assume without loss of generality that $r_1 < s_1$. Then, since r and s are incomparable, $r_2 > s_2$. Let

$$\mathcal{R} = \{(x, y) \in \text{lift}(r) \mid x < s_1\},$$
$$\mathcal{S} = \{(x, y) \in \text{lift}(s) \mid y < r_2\},$$
$$\mathcal{Q} = \{(x, y) = \text{lift}^e(r) = \text{lift}^e(s) \text{ for some } e \in \mathcal{A}(M)\}.$$

The following observations, illustrated in Fig. 14, follow from Lemma 7.4:

- If $\mathcal{R}$ is nonempty, there is an element $r' \in \mathcal{R}$ such that for all $u \in \mathcal{R}$, $u_1 \leq r'_1$ and $u_2 \geq r'_2$.

- Symmetrically, if $\mathcal{S}$ is nonempty, there is an element $s' \in \mathcal{R}$ such that for all $u \in \mathcal{S}$, $u_1 \geq s'_1$ and $u_2 \leq s'_2$.

- If $\mathcal{Q}$ is nonempty, there is an element $q^x \in \mathcal{Q}$ such that for all $u \in \mathcal{Q}$, $u_1 \geq q_1^x$ and $u_2 \leq q_2^x$.

- Symmetrically, if $\mathcal{Q}$ is nonempty, there is an element $q^y \in \mathcal{Q}$ such that for all $u \in \mathcal{Q}$, $u_1 \leq q_1^y$ and $u_2 \geq r_2$.

Clearly, r', s', q^x, and q^y are unique, when they exist. Using Lemma 7.3, it is straightforward to check that $u \in \mathcal{Q}$ if and only if $u \in \text{lift}(r) \cap \text{lift}(s)$ and one of the following is true:

1. u is incomparable to every element of $\mathcal{R} \cup \mathcal{S}$.

2. $\mathcal{R}$ is non-empty, $u_2 = r'_2$, and there is no $v \in \text{lift}(s)$ with $v_1 = u_1$ and $v_2 < u_2$,

3. $\mathcal{S}$ is non-empty, $u_1 = s'_1$, and there is no $v \in \text{lift}(r)$ with $v_1 < u_1$ and $v_2 = u_2$.

See Fig. 14 for an illustration of $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{Q}$.

To finish the proof of Lemma 7.5, we consider seven cases. For each, we explicitly describe the lines where r and s either switch or separate. The verification of the claimed behavior in each case, which uses Lemma 7.3 and the observations above, is left to the reader. Fig. 14 illustrates case 7.

1. $\mathcal{R}$ and $\mathcal{S}$ empty. $\text{lift}^e(r) = \text{lift}^e(s)$ for every 2-cell $e \in \mathcal{A}(M)$. Therefore, r and s never switch or separate.

2. $\mathcal{R}$ nonempty, $\mathcal{S}$ and $\mathcal{Q}$ empty. $\mathrm{lift}^e(r) < \mathrm{lift}^e(s)$ for every 2-cell $e \in \mathcal{A}(M)$. Again, no switches or separations.

3. $\mathcal{S}$ nonempty, $\mathcal{R}$ and $\mathcal{Q}$ empty. Symmetric to the above, no switches or separations.

4. $\mathcal{R}$ and $\mathcal{S}$ nonempty, $\mathcal{Q}$ empty. $\mathrm{lift}^e(r) < \mathrm{lift}^e(s)$ whenever e lies below $L = \mathcal{D}_p(\mathrm{LUB}(r', s'))$, and $\mathrm{lift}^e(s) < \mathrm{lift}^e(r)$ whenever e lies above L. Hence r and s switch at L.

5. $\mathcal{R}$ and $\mathcal{Q}$ nonempty, $\mathcal{S}$ empty. $\mathrm{lift}^e(r) < \mathrm{lift}^e(s)$ whenever e lies below $L = \mathcal{D}_p(\mathrm{LUB}(r', q_x))$, and $\mathrm{lift}^e(r) = \mathrm{lift}^e(s)$ whenever e lies above L. Hence r and s separate at L.

6. $\mathcal{S}$ and $\mathcal{Q}$ nonempty, $\mathcal{R}$ empty. Symmetric to the above, r and s separate at $\mathcal{D}_p(\mathrm{LUB}(q_y, s'))$.

7. $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{Q}$ all nonempty. $\mathrm{lift}^e(r) < \mathrm{lift}^e(s)$ whenever e lies below $L = \mathcal{D}_p(\mathrm{LUB}(r', q_x))$; $\mathrm{lift}^e(r) = \mathrm{lift}^e(s)$ whenever e lies above L and below $L' = \mathcal{D}_p(\mathrm{LUB}(q_y, s'))$; and $\mathrm{lift}^e(s) < \mathrm{lift}^e(r)$ whenever e lies above L'. Hence r and s separate at L and L'. $\qquad\square$
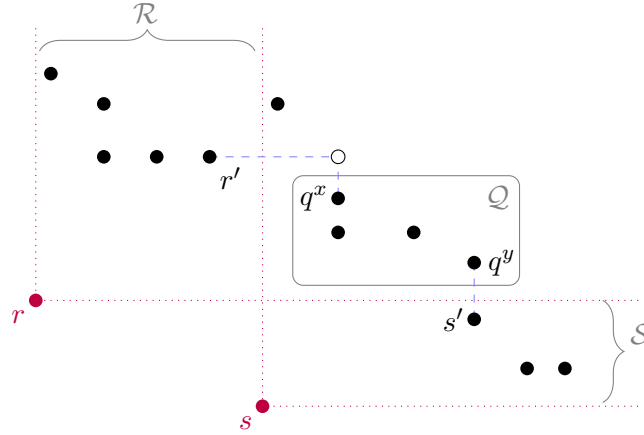
Figure 14: An illustration of the case where $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{Q}$ are each non-empty (case 7). Points in $\mathrm{lift}(r) \cup \mathrm{lift}(s)$ are drawn as black dots. Observe that r and s separate at $\mathcal{D}_p(\mathrm{LUB}(r', q_x))$ and $\mathcal{D}_p(\mathrm{LUB}(q_y, s'))$, and at no other anchor line do r and s either switch or separate.

Proof of Proposition 7.2. Fix $j \in \{1, 2\}$, and let $k \neq k' \in [m_j]$. First, we note that if $\mathrm{gr}_j(k)$ and $\mathrm{gr}_j(k')$ are comparable, then as we pass from cell e_{i-1} to cell e_i, our algorithm for computing barcode templates never performs an insertion-sort transposition of the values

k and k' in sigInv_j. This is because our initialization procedure at cell e_0, described in Section 6.8 and Appendix A.2, chooses sigInv_j such that if $\mathrm{gr}_j(k) < \mathrm{gr}_j(k')$, then $\mathsf{sig}_j[k] < \mathsf{sig}_j[k']$. Since $\mathrm{lift}^e(k) \le \mathrm{lift}^e(k')$ for all 2-cells e, there thus is never any need to swap k and k'.

Therefore, as we pass from cell e_{i-1} to cell e_i, our algorithm performs an insertion-sort transposition of the values k and k' in sigInv_j only if $\mathrm{gr}_j(k)$ and $\mathrm{gr}_j(k')$ either switch or separate at (e_{i-1}, e_i).

Clearly, the number of pairs $k, k' \in [m_j]$ such that $\mathrm{gr}_j(k)$ and $\mathrm{gr}_j(k')$ either switch or separate at *any* anchor line is less than $m_j^2/2$, the total number of pairs. The path Γ, constructed via the minimum spanning tree construction in Section 6.6, crosses each anchor line at most 2κ times. By Lemma 7.5 then, for each pair $k, k' \in [m_j]$, the insertion-sort component of our algorithm performs a total of at most 4κ transpositions of that pair in sigInv_j. Hence, the total number of transpositions performed by the algorithm altogether is at most $2(m_1^2 + m_2^2)\kappa < 2m^2\kappa$. $\qquad\square$

8 Speeding up the Computation of the Augmented Arrangement

In this section, we describe several simple, practical strategies to speed up the runtime of our computation of $\mathcal{A}^\bullet(M)$. Used together, these strategies allow us to the compute augmented arrangements of the persistent homology modules of much larger datasets than would otherwise be possible.

8.1 Persistence Computation from Scratch When RU-Updates are too Slow

Three Options for Computing a Barcode Template. While an update to an RU-decomposition involving few transpositions is very fast in practice, an update to an RU-decomposition requiring many transpositions can be quite slow; when many transpositions are required, it is sometimes much faster to simply recompute the RU-decomposition from scratch using the standard persistence algorithm. In our setting, the practical performance of our algorithm can be greatly improved if for consecutive 2-cells $e_{i-1}, e_i \in \Gamma$ with the edge weight $w(e_{i-1}, e_i)$ greater than some suitably chosen threshold t, we simply compute the RU-decomposition of Φ^i from scratch, directly from D_1^i and D_2^i.

Moreover, we can obtain significant additional speedups by avoiding the computation of the full RU-decomposition of Φ^i altogether at some cells e_i. To explain this, we first note that to obtain $\mathcal{T}^{e_i}$ via Eq. (6.1), we do not need the full RU-decomposition Φ^i, but only $\mathrm{PE}(\Phi^i) = (\mathrm{Pairs}(\Phi^i), \mathrm{Ess}(\Phi^i))$; in particular we do not need U_1^i and U_2^i. The algorithm for computing barcode templates described in Section 6 maintains the full RU-decomposition of each Φ^i because the vineyard algorithm requires this. But if we are willing to compute

$PE(\Phi^{i+1})$ from scratch, then it is not necessary to compute the full RU-decomposition of Φ^i; it suffices to compute $PE(\Phi^i)$. Further, if in this case we have that $e_i = e_j$ for some $j < i$, then we do not even need to compute $PE(\Phi^i)$ at all at cell e_i, since we have already done so at an earlier step.

In recent years, several algorithms for computing barcodes have been introduced which are much faster than the standard persistence algorithm [3, 4, 42]. For example, a few such algorithms are implemented in the software library PHAT [4]. Given a FI-rep Φ as input, these algorithms compute $PE(\Phi)$, but do not compute the full RU-decomposition of Φ.[3]

Let us restrict our attention a single such algorithm, say, the *clear and compress algorithm* implemented in PHAT [3].

To compute $PE(\Phi^i)$, then, we have three options available to us:

(A) Use the clear and compress algorithm if $e_i \neq e_j$ for all $j < i$; do nothing if $e_i = e_j$ for some $j < i$.

(B) Compute the full RU-decomposition of Φ^i from scratch using the standard persistence algorithm.

(C) Use vineyard updates. This option is only available if we chose option B or C at cell e_{i-1}, so that the full RU-decomposition of Φ^{i-1} was computed.

Clearly, there is a tradeoff between options A and B: Option A is much faster, but choosing option A at cell e_i precludes the use of option C at cell e_{i+1}. How, then, do we choose between these three options at each cell e_i? We formulate this problem as a discrete optimization problem, which can solved efficiently by reduction to a min-cut problem.

Estimating Runtimes of the Different Options. Our formulation of the problem requires us to first estimate the respective runtimes $c_i(A)$, $c_i(B)$, and $c_i(C)$ of options A, B, and C at each cell e_i in the path $\Gamma = e_0, e_1, \ldots, e_w$. We will describe a simple strategy for this here, and then explain below how to modify our approach to correct for a drawback of the strategy.

We take $c_i(A) = 0$ if $e_i = e_j$ for some $j < i$, and otherwise we take $c_i(A)$ to be some constant $c(A)$ independent of i. Similarly, we take $c_i(B) = c(B)$ to be independent of i. To compute $c(A)$, we compute $PE(\Phi^0)$ using option A, and set $c(A)$ to be the runtime of this computation. Similarly, to compute $c(B)$, we compute the the full RU-decomposition of Φ^0 from scratch and take $c(B)$ to be the runtime.

We set $c_0(C)$ arbitrarily, say $c_0(C) = 0$. To compute $c_i(A)$ for each $i \geq 1$, we perform several thousand random vineyard updates to the RU-decomposition of Φ^0. Using timing data from these computations, we compute, for $j = 1, 2$, the average runtime c_j^{vine} of an

---

[3]However, some fast algorithms for persistence computation can be readily adapted to compute R_1, R_2, U_1^{-1}, and U_2^{-1}. For example, as explained to us by Ulrich Bauer, this is true for the "twist" variant of the standard persistence algorithm [22].

update to the RU-decomposition corresponding to a transposition of adjacent elements in $[m_j]$. Letting L denote the anchor line containing the shared boundary of e_{i-1} and e_i, and recalling the notation of Section 6.10, we take

$$c_i := c_1^{\text{vine}}(\text{sw}_L(\text{gr}_1) + \frac{1}{4}\text{sep}_L(\text{gr}_1)) + c_2^{\text{vine}}(\text{sw}_L(\text{gr}_2) + \frac{1}{4}\text{sep}_L(\text{gr}_2)).$$

Some motivation for this choice of c_i is provided by Remark 6.8.

The Optimization Problem. To decide between options A, B and C at each cell $e_i \in \Gamma$, we solve the following optimization problem:

$$\text{minimize: } \sum_{i=0}^{w} c_i(X_i)$$
$$\text{subject to: } X_i \in \{\text{A}, \text{B}, \text{C}\},$$
$$X_i = \text{A} \implies X_{i+1} \neq \text{C},$$
$$X_0 \neq C.$$

Clearly, this problem is equivalent to the integer linear program (ILP):

$$\text{minimize: } \sum_{i=0}^{w} \big(c_i(\text{A})x_i + c_i(\text{B})y_i + c_i(\text{C})z_i\big)$$
$$\text{subject to: } x_i, y_i, z_i \in \{0,1\}$$
$$x_i + y_i + z_i = 1$$
$$x_i + z_{i+1} \leq 1,$$
$$z_0 = 0.$$

Using the constraints $x_i + y_i + z_i = 1$, we can eliminate the variables y_i from this ILP to obtain an equivalent ILP with a simpler set of constraints:

$$\text{minimize: } \sum_{i=0}^{w} \big((c_i(\text{A}) - c_i(\text{B}))x_i + (c_i(\text{C}) - c_i(\text{B}))z_i\big)$$
$$\text{subject to: } x_i, z_i \in \{0,1\}$$
$$x_i + z_{i+1} \leq 1,$$
$$z_0 = 0.$$

The constraint matrix associated to the latter ILP is of a standard form, well known to be totally modular [43]. While an ILP with totally unimodular constraint matrix can always be solved directly via linear programming relaxation, it is often the case that such an ILP

can be cast as a network flow problem, in which case we can take advantage of very efficient specialized algorithms.

In fact, as explained to us by John Carlsson, the simplified ILP above can be cast as the problem of finding a minimum cut in a network: First, the ILP can be cast as a maximum-weight independent set problem in a bipartite graph with non-negative vertex weights. In any graph, the complement of an independent set is a vertex cover and vice versa, so the latter problem is in turn equivalent to a minimum-cost vertex cover problem in a bipartite graph. It is well known that such a problem can be solved by computing a minimum cut in a flow network [1, Section 8.4.3.1].

Dynamic Updates to our Estimates of Runtime Cost. As mentioned above, there is a drawback to the approach to barcode template computation we have proposed here: $c(A)$ and $c(B)$ may not be very good estimates of the respective average costs of option A and option B; after all, our estimates $c(A)$ and $c(B)$ are computed using very little data.

Here is one way to correct for this: We first solve the optimization problem above for just the first few cells (say 5%) in the path Γ. Using the solution, we then compute the barcode template for each of these cells. As we do this, we record the runtime of the computation at each cell. We next update the value of $c(A)$ to the average run time of all computations performed using option A thus far. We also update $c(B)$, c_1^{vine}, and c_2^{vine} in the analogous way. We then use these updated values as input to the optimization problem for the next 5% of cells in Γ. We continue in this way until the estimates of $c(A)$, $c(B)$, c_1^{vine}, and c_2^{vine} have stabilized. Finally, we solve the optimization problem for all of the remaining cells in Γ and use the solution to compute the remaining barcode templates.

8.2 Coarsening of Persistence Modules

We have seen that size of the augmented arrangement $\mathcal{A}^\bullet(M)$ depends quadratically on the coarseness κ of M, and that computing $\mathcal{A}^\bullet(M)$ requires $O(m^3\kappa + (m + \log \kappa)\kappa^2)$ elementary operations.

Thus, to keep our computations small, we typically want to limit the size of κ by coarsening our module. As we now explain, doing this quite simple. A similar coarsening scheme is mentioned in [12].

Let $\mathcal{G} = (\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n) : \mathbb{Z}^2 \to \mathbb{R}^2$ be any grid function, as defined in Section 2.5. $\mathcal{G}$ extends to a functor $\mathbf{Z}^n \to \mathbf{R}^n$, which we also denote by $\mathcal{G}$. For M a 2-D persistence module, let $M^{\mathcal{G}}$ be a continuous extension of $M \circ \mathcal{G}$. In view of Proposition 2.10, the coarseness of the grid $\mathcal{G}$ controls the coarseness κ of the module $M^{\mathcal{G}}$.

Let d_I denote the multidimensional interleaving distance, as defined in [40]. As explained there, d_I is a particularly well-behaved metric on persistence modules. The following proposition, whose easy proof we omit, makes precise the intuitive idea that a small amount of coarsening leads to a small change in our persistence module:

Proposition 8.1. *If* $|\mathcal{G}_j(z) - \mathcal{G}_j(z+1)| \leq \delta$ *for* $i = 1, 2$ *and all* $z \in \mathbb{Z}$*, then* $d_I(M, M^{\mathcal{G}}) \leq \delta$.

The "external stability theorem" of Landi [38], mentioned earlier in Section 1.5, shows that if 2-D persistence modules M and N are close in the interleaving distance, then the fibered barcodes $\mathcal{B}(M)$ and $\mathcal{B}(N)$ will be close, in a precise sense. This justifies the use of coarsening in conjunction with our visualization paradigm.

Coarsening Free Implicit Representations. As explained in Section 5.2, in practice, we typically have access to M via a FI-rep $\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2)$ for M. Since our algorithm for computing an augmented arrangement takes a FI-rep as input, to compute $\mathcal{A}^{\bullet}(M^{\mathcal{G}})$, we want to first construct a FI-rep $\Phi^{\mathcal{G}}$ of $M^{\mathcal{G}}$ from Φ.

Let $\mathrm{cl}^{\mathcal{G}} : \mathbb{R}^n \to \mathcal{G}$ be the function which takes each $a \in \mathbb{R}^n$ to the minimal $z \in \mathcal{G}$ with $a \leq z$.

Define $\Phi^{\mathcal{G}} = (\mathrm{cl}^{\mathcal{G}} \circ \mathrm{gr}_1, \mathrm{cl}^{\mathcal{G}} \circ \mathrm{gr}_2, D_1, D_2)$. We leave the proof of the following to the reader:

Proposition 8.2. $H(\Phi_G) \simeq M^{\mathcal{G}}$.

Thus, to obtain a FI-rep of a coarsening of M, it suffices to simply coarsen the grade functions in our FI-rep of M.

8.3   Parallelization

The problem of computing the barcode templates $\mathcal{T}^e$ is embarrassingly parallelizable. Here is one very simple parallelization scheme: Given l^2 processors, where l is less than or equal to the number of anchors, we can choose the l anchor lines L with the largest values of $w(L)$. These lines divide $\mathcal{A}(M)$ into at most l^2 polygonal cells $\{C_k\}$ with disjoint interiors, and the remaining anchor lines induce a line arrangement $\mathcal{A}(M)_k$ on each C_k. On processor k, we can run our main serial algorithm, described above, to compute the barcode templates at each 2-cell of $\mathcal{A}(M)_k$.

For this, we need to make just one modification to the algorithm: When choosing our path Γ through the 2-cells of $\mathcal{A}(M)_k$, we generally cannot choose our initial cell e to be the cell e_0 described in Section 6.6, since e_0 may not be be contained in $\mathcal{A}(M)_k$. Instead, we choose the initial cell e arbitrarily.

This means we cannot use the approach of Section 6.8 to initialize the data structures sig_j, sigInv_j, lifts_j, and $\mathsf{levSet}_j(\cdot)$ at cell e. One way to initialize these data structures is to chose an arbitrary affine line L with $\mathcal{D}_\ell(L) \in e$, and consider the behavior of the map push_L on $S \cup \mathrm{im}(\mathrm{gr}_1) \cup \mathrm{im}(\mathrm{gr}_2)$, using exact arithmetic where necessary.

9   Preliminary Runtime Results

We now present runtimes for the computation of augmented arrangements arising from synthetic data.

We emphasize that these computational results are preliminary, as our implementation of RIVET does not yet take advantage of some key optimizations. First, the code that produced these results employs a highly simplified variant of the scheme detailed in Section 8.1 for computing barcode templates; this variant chooses only between options (B) and (C) at each edge crossing, which is less efficient than what is proposed in Section 8.1. Secondly, this code stores the columns of our sparse matrices using linked lists; it is known that a smarter choice of data structure for storing columns can lead to major speedups in persistence computation [4]. Finally, as mentioned in Section 1.10, our current implementation runs only on a single processing core. We expect to see substantial speedups after parallelizing the computation of barcode templates as proposed in Section 8.3.

Our computations were run on a single (slow) 800 MHz core in a 32-core server with 96 GB of RAM. However, for the computations reported here, only a fraction of the memory was required. For example, RIVET used approximately 9.3 GB of RAM for our largest computation.

Noisy Circle Data. Data sets we consider are point clouds sampled with noise from an annulus, such as the center point cloud in Fig. 2. Specifically, 90% of the points in each data set X are sampled randomly from a thick annulus in a plane, and 10% are sampled randomly from a square containing the annulus. We define a codensity function $\gamma : X \to \mathbb{R}$ by taking $\gamma(p)$ to be equal to the number of points of X within some fixed distance to p. We then construct the Vietoris-Rips bifiltration $\mathcal{F} := \mathrm{Rips}(\gamma)$, described in Section 1.4, taking the metric on X to be the Euclidean distance, with the scale parameter for the Vietoris-Rips complexes capped at a value slightly larger than the inner diameter of the annulus.

Computing the Graded Betti Numbers. Table 3 displays the average runtimes for computing the graded Betti numbers ξ_0, ξ_1, and ξ_2 of $H_i(\mathcal{F})$, for $i \in \{1, 2\}$. Each row gives the averages from three point clouds of the specified size. For example, we generated three point clouds of 100 points, and the average number of 1-simplices in the resulting bifiltrations was 3,820, so computing H_0 homology required working with a bifiltration of average size 3,920 simplices.

Building the Augmented Arrangement. Table 4 displays the average runtimes (in seconds) to build $\mathcal{A}^{\bullet}(H_0(\mathcal{F}))$. As before, each row gives the averages from three point clouds of the specified size. The average runtimes for computing $\mathcal{A}^{\bullet}(H_0(\mathcal{F}))$ for each of four different coarsenings are displayed in the table. Similarly, Table 5 displays the average runtimes (in seconds) to build $\mathcal{A}^{\bullet}(H_1(\mathcal{F}))$.

Table 3: Average runtimes for computing the bigraded Betti numbers of the noisy circle data

	Points	Simplices	Runtime (sec.)
H_0	100	3,920	0.11
	200	15,719	0.70
	300	35,324	2.38
	400	62,415	3.99
H_1	100	91,876	4.01
	200	755,211	59.4
	300	2,560,718	264
	400	6,052,584	790

Table 4: Runtimes for computing the augmented arrangement for 0th homology of the noisy circle data

Points	Simplices	Runtimes (seconds)			
		Bins: 5×5	10×10	15×15	20×20
100	3,920	0.17	0.39	0.97	1.51
200	15,719	0.40	7.90	21.9	46.1
300	35,324	0.46	10.3	46.7	113
400	62,415	1.25	22.0	180	637

Table 5: Runtimes for computing the augmented arrangement for 1st homology of the noisy circle data

Points	Simplices	Runtimes (seconds)			
		Bins: 5×5	10×10	15×15	20×20
100	91,876	0.60	1.33	2.35	3.61
200	755,211	13.1	211	1,012	3,368
300	2,560,718	39.9	374	3,364	7,510
400	6,052,584	228	6,893	18,346	59,026

10   Conclusion

In this paper, we have introduced RIVET, a practical tool for visualization of 2-D persistence modules. RIVET provides an interactive visualization of the barcodes of 1-D affine slices of a 2-D persistence module, as well as visualizations of the dimension function and bigraded Betti numbers of the module. We have presented a mathematical theory for our visualization paradigm, centered around a novel data structure called an augmented ar-

rangement. We have also introduced and analyzed an algorithm for computing augmented arrangements, and described several strategies for improving the runtime of this algorithm in practice.

In addition, we have presented timing data from preliminary experiments on the computation of augmented arrangements. Though we have yet to incorporate several key optimizations into our code, the results demonstrate that our current implementation already scales well enough to be used to study bifiltrations with millions of simplices. With more implementation work, we expect RIVET to scale well enough to be used in many of the same settings where 1-D persistence is currently used for exploratory data analysis.

From here, there are several natural directions to pursue. Beyond continuing to improve our implementation of RIVET, we would like to:

- Apply RIVET to the exploratory analysis of scientific data.

- Develop statistical foundations for our data analysis methodology.

- Adapt the RIVET paradigm in the setting of 0-D homology to develop a tool for hierarchical clustering and interactive visualization of bidendrograms [11].

- Extend the RIVET methodology to other generalized persistence settings, such cosheaves of vector spaces over $\mathbb{R}^2$ [28], or cosheaves of 1-D persistence modules over $\mathbb{R}$.

We hope that RIVET will prove to be a useful addition to the existing arsenal of TDA tools. Regardless of how it ultimately fares in that regard, however, we feel that the broader program of developing practical computational tools for multidimensional persistence is a promising direction for TDA, and we hope that this work can draw attention to the possibilities for this. We believe that there is room for a diverse set of approaches.

Acknowledgements

A Appendix

A.1 Details of the RIVET Interface

Expanding on Section 1.7, we now provide some more detail about RIVET's graphical interface.

As discussed in Section 5, the module M is input to RIVET as a free implicit representation

$$\Phi = (\mathrm{gr}_1, \mathrm{gr}_2, D_1, D_2).$$

RIVET uses gr_1 and gr_2 to choose the bounds for the Line Selection Window and Persistence Diagram Window. To explain, let A and B denote the greatest lower bound and least upper bound, respectively, of $\mathrm{im}\,\mathrm{gr}_1 \cup \mathrm{im}\,\mathrm{gr}_2$. In order to avoid discussion of uninteresting edge cases, we will assume that $A_1 < B_1$ and $A_2 < B_2$.

Choice of Bounds for the Line Selection Window. We take the lower left corner and upper right corner of the Line Selection Window to be A and B, respectively.[4] By default, the Line Selection Window is drawn to scale. A toggle switch rescales (i.e., normalizes) the window so that it is drawn as a square on the screen.

Parameterization of Lines for Plotting Persistence Diagrams. We next explain how, given a line $L \in \bar{\mathcal{L}}$, RIVET represents $\mathcal{B}(M^L)$ as a persistence diagram. Let us first assume that the Line Selection Window is unnormalized; we treat the case where it is normalized at the end of this section. Further, by translating the indices of M if necessary, we may assume without loss of generality that $A = 0$.

As noted in Section 1.7, to plot $\mathcal{B}(M^L)$ as a persistence diagram, we need to first choose a parameterization $\gamma_L : \mathbb{R} \to L$ of L. We choose γ_L to be the unique order-preserving isometry such that:

1. If L has finite positive slope, then $\gamma_L(0)$ is the unique point in the intersection of L with the union of the non-negative portions of the coordinate axes.

2. If L is the line $x = a$, then $\gamma_L(0) = (a, 0)$.

3. If L is the line $y = a$, then $\gamma_L(0) = (0, a)$.

---

[4]For a more intrinsic choice of bounds for the Line Selection Window, one could instead take the lower left and upper right corners of the window to be the greatest lower bound and least upper bound, respectively, of the set $\{a \in \mathbb{R}^2 \mid \xi_i(M)(a) > 0 \text{ for some } i \in \{0, 1, 2\}\}$. However, we feel that for a typical TDA application, the extrinsic bounds for the Line Selection Window we have proposed provide a more intuitive choice of scale.

The Choice of Bounds for the Persistence Diagram Window. The bounds for the Persistence Diagram Window are are chosen statically, depending on M but not on the choice of L. RIVET chooses the viewable region of the persistence diagram to be $[0, |B|] \times [0, |B|]$.

Representation of Points Outside of the Viewable Region of the Persistence Diagram. It may be that for some choices of L, $\mathcal{B}(M^L)$ contains intervals $[\alpha, \beta)$, for $\alpha \geq |B|$ or $\beta \geq |B|$, so that $[\alpha, \beta)$ falls outside of the viewable region of the persistence diagram; indeed, the coordinates of some points in the persistence diagram can become huge (but finite), as the slope of the L approaches 0 or ∞. Thus, our persistence diagrams include some information on the top of the diagram, not found in typical persistence diagram visualizations, to record the points in the persistence diagram which fall outside of the viewable region:

- Above the main square region of the persistence diagram are two narrow horizontal strips, separated by a dashed horizontal line. The upper strip is labeled "inf," while the lower is labeled "< inf". In the higher strip, we plot a point with x-coordinate α for each interval $[\alpha, \infty)$ with $\alpha \leq |B|$. In the lower strip, we plot a point with x-coordinate α for each interval $[\alpha, \beta) \in \mathcal{B}(M^L)$ with $\alpha \leq |B|$ and $|B| < \beta < \infty$.

- Just to the right of each of the two horizontal strips is a number, separated from the strip by a vertical dashed line. The upper number is the count of intervals $[\alpha, \infty) \in \mathcal{B}(M^L)$ with $|B| < \alpha$; the lower number is the count of intervals $[\alpha, \beta) \in \mathcal{B}(M^L)$ with $|B| < \alpha, \beta < \infty$.

Persistence Diagrams Under Rescaling. If we choose to normalize the Line Selection Window, then RIVET also normalizes the persistence diagrams correspondingly. To do this, it computes respective affine normalizations gr_1', gr_2' of gr_1, gr_2, so that after normalization, $A = (0,0)$ and $B = (1,1)$. RIVET then chooses the parameterizations of lines $L \in \bar{\mathcal{L}}$ and computes bounds on the Persistence Diagram Window exactly as described above in the unnormalized case, but taking the input FI-rep to be $(\mathrm{gr}_1', \mathrm{gr}_2', D_1, D_2)$.

A.2 Computing the Lists $\mathsf{levSet}_j(u)$ at Cell e_0

As mentioned in Section 6.8, to efficiently compute the lists $\mathsf{levSet}_j(u)$ at cell e_0, we use a sweep algorithm. As we did in Section 5.4, to keep notation simple we assume that Ω_x and Ω_y are the identity maps on $[n_x]$ and $[n_y]$ respectively, so that $(\Omega_x \times \Omega_y)^{-1}(\mathcal{P}) = \mathcal{P}$. We assume that to start, gr_1 and gr_2 are both increasing with respect to colexicographical order; if this is not the case, then by Lemma 6.3, we can apply a sorting algorithm to modify Φ so that the assumption does hold. This sorting can be done in $O(m \log m)$ time.

Our sweep algorithm maintains a linked list Frontier of pointers to elements of $\mathcal{P}$. (These elements are stored in tPtsMat.) Both the x- and y-coordinates of the entries of Frontier are always strictly decreasing.

The algorithm iterates through the rows of the grid $[n_x] \times [n_y]$ from top to bottom. The list Frontier is initially empty, and is updated at each row s, with the help of tPtsMat. If row s is empty, then no update is necessary at row s. Otherwise, let u be the rightmost entry of $\mathcal{P}$ in row s, and let r be the column containing u. If the last element of Frontier is also in column r, then this element is removed from Frontier and replaced by u. Otherwise, we append u to the end of Frontier.

The algorithm then inserts each $k \in [m_j]$ with $\mathrm{gr}_j(k) = (r, s)$ for some r into the appropriate list $\mathsf{levSet}_j(-)$. (Since gr_j is assumed to be increasing with respect to colexicographical order, we have immediate access to all such k.) Specifically, k is added to the list $\mathsf{levSet}_j(u)$, for u the leftmost element of Frontier with $r \leq u_1$. The lists $\mathsf{levSet}_j(u)$ are maintained in lexicographical order. It is easy to check that $u = \mathrm{lift}^0 \circ \mathrm{gr}_j(k)$, as desired.

The frontier-sweep algorithm is stated in pseudocode in Algorithm 2.

Updating Frontier at each row of tPtsMat takes constant time, so the total cost of updating Frontier is $O(n_y)$. For each row s, we must iterate over Frontier once to identify the lists $\mathsf{levSet}_j(-)$ into which we insert elements of $[m_j]$. There are n_y rows, and the length of Frontier is $O(\kappa_x)$, so the total cost of these iterations over Frontier is $O(n_y \kappa_x)$. Inserting each $k \in [m_j]$ into the appropriate list $\mathsf{levSet}_j(-)$ takes constant time, so the total cost of such insertions is $O(m)$. Thus, the total number of operations for the frontier-sweep algorithm (including the cost of the initial sorting to put gr_1 and gr_2 in the right form), is $O(m \log m + n_y \kappa_x) = O(m \log m + m\kappa)$.

Algorithm 2 Frontier-sweep algorithm for building the lists $\mathsf{levSet}_j(u)$

Input: gr_j (represented as a list in colexicographical order), tPtsMat
Output: tPtsMat, updated so that for each $u \in \mathcal{P}^{e_0}$, $\mathsf{levSet}_j(u) = (\mathrm{lift}^0 \circ \mathrm{gr}_j)^{-1}(u)$, with each list sorted in lexicographical order
1: initialize Frontier as an empty linked list
2: **for** $s = n_y$ to 1 **do**
3:     **if** tPtsMat has an entry in row s **then**   ▷ update Frontier for row s
4:         let r be the column containing rightmost element of $\mathcal{P}$ in row s
5:         **if** last element of Frontier is in column r **then**
6:             remove the last entry from Frontier
7:         append the entry of $\mathcal{P}$ at row s, column r to the end of Frontier
8:     **for each** entry u in Frontier **do**   ▷ add elements with grades in row s to the lists $\mathsf{levSet}_j(-)$
9:         **if** u is the last element of Frontier **then**
10:             add all $k \in [m_j]$ such that $\mathrm{gr}_j(k) = (r, s)$ with $r \leq u_1$ to $\mathsf{levSet}_j(u)$
11:         **else**
12:             let v be the element after u in Frontier
13:             add all $k \in [m_j]$ such that $\mathrm{gr}_j(k) = (r, s)$ with $v_1 < r \leq u_1$ to $\mathsf{levSet}_j(u)$

References

[1] M. J. Atallah. *Algorithms and theory of computation handbook*. CRC press, 1998.

[2] M. F. Atiyah. On the krull-schmidt theorem with application to sheaves. *Bulletin de la Société Mathématique de France*, 84:307–317, 1956.

[3] U. Bauer, M. Kerber, and J. Reininghaus. Clear and compress: Computing persistent homology in chunks. In *Topological Methods in Data Analysis and Visualization III*, pages 103–117. Springer, 2014.

[4] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. Phat – persistent homology algorithms toolbox. In *Mathematical Software – ICMS 2014*, volume 8592 of *Lecture Notes in Computer Science*, pages 137–143. Springer Berlin Heidelberg, 2014.

[5] U. Bauer and M. Lesnick. Induced matchings of barcodes and the algebraic stability of persistence. In *Proceedings of the 2014 Annual Symposium on Computational Geometry*, page 355. ACM, 2014.

[6] S. Biasotti, A. Cerri, P. Frosini, and D. Giorgi. A new algorithm for computing the 2-dimensional matching distance between size functions. *Pattern Recognition Letters*, 32(14):1735–1746, 2011.

[7] H. Brönnimann, A. Fabri, G.-J. Giezeman, S. Hert, M. Hoffmann, L. Kettner, S. Pion, and S. Schirra. 2D and 3D geometry kernel. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.5.2 edition, 2015. http://doc.cgal.org/4.5.2/Manual/packages.html#PkgKernel23Summary.

[8] P. Bubenik, V. de Silva, and J. Scott. Metrics for generalized persistence modules. *Foundations of Computational Mathematics*, pages 1–31, 2014.

[9] G. Carlsson. Topological pattern recognition for point cloud data. *Acta Numerica*, 23:289–368, May 2014.

[10] G. Carlsson, V. de Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the 25th annual symposium on Computational geometry*, pages 247–256. ACM, 2009.

[11] G. Carlsson and F. Mémoli. Multiparameter hierarchical clustering methods. *Classification as a Tool for Research*, pages 63–70, 2010.

[12] G. Carlsson, G. Singh, and A. Zomorodian. Computing multidimensional persistence. *Algorithms and Computation*, pages 730–739, 2009.

[13] G. Carlsson and A. Zomorodian. The theory of multidimensional persistence. *Discrete and Computational Geometry*, 42(1):71–93, 2009.

[14] A. Cerri, B. Di Fabio, M. Ferri, P. Frosini, and C. Landi. Betti numbers in multidimensional persistent homology are stable functions. *Mathematical Methods in the Applied Sciences*, 36(12):1543–1557, 2013.

[15] A. Cerri, B. Di Fabio, G. Jablonski, and F. Medri. Comparing shapes through multi-scale approximations of the matching distance. *Computer Vision*, 2014.

[16] W. Chacholski, M. Scolamiero, and F. Vaccarino. Combinatorial presentation of multidimensional persistent homology. *arXiv preprint arXiv:1409.7936*, 2014.

[17] F. Chazal, D. Cohen-Steiner, M. Glisse, L. Guibas, and S. Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the 25th annual symposium on Computational geometry*, pages 237–246. ACM, 2009.

[18] F. Chazal, D. Cohen-Steiner, L. Guibas, F. Mémoli, and S. Oudot. Gromov-Hausdorff stable signatures for shapes using persistence. In *Proceedings of the Symposium on Geometry Processing*, pages 1393–1403. Eurographics Association, 2009.

[19] F. Chazal, D. Cohen-Steiner, and Q. Mérigot. Geometric inference for probability measures. *Foundations of Computational Mathematics*, pages 1–19, 2011.

[20] F. Chazal, V. de Silva, M. Glisse, and S. Oudot. The structure and stability of persistence modules. *arXiv preprint arXiv:1207.3674*, 2012.

[21] F. Chazal, V. De Silva, and S. Oudot. Persistence stability for geometric complexes. *Geometriae Dedicata*, 173(1):193–214, 2014.

[22] C. Chen and M. Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, volume 11, 2011.

[23] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Report 388, Graduate School of Industrial Administration, CMU*, 1976.

[24] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete and Computational Geometry*, 37(1):103–120, 2007.

[25] D. Cohen-Steiner, H. Edelsbrunner, and D. Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 119–126. ACM, 2006.

[26] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[27] W. Crawley-Boevey. Decomposition of pointwise finite-dimensional persistence modules. *Journal of Algebra and Its Applications*, 14(05):1550066, 2015.

[28] J. Curry. Sheaves, cosheaves and applications. *Ph.D. Dissertation, University of Pennsylvania*, 2014.

[29] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[30] H. Derksen and J. Weyman. Quiver representations. *Notices of the AMS*, 52(2):200–206, 2005.

[31] H. Edelsbrunner and J. Harer. *Computational topology: an introduction.* American Mathematical Society, 2010.

[32] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[33] D. Eisenbud. *Commutative Algebra: with a view toward algebraic geometry*, volume 150. Springer Science & Business Media, 1995.

[34] D. Eisenbud. *The geometry of syzygies: a second course in algebraic geometry and commutative algebra*, volume 229. Springer Science & Business Media, 2005.

[35] J. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991.

[36] M. Kreuzer and L. Robbiano. *Computational commutative algebra 2*, volume 1. Springer, 2005.

[37] R. La Scala and M. Stillman. Strategies for computing minimal free resolutions. *Journal of Symbolic Computation*, 26(4):409–431, 1998.

[38] C. Landi. The rank invariant stability via interleavings. *arXiv preprint arXiv:1412.3374*, 2014.

[39] S. Lang. Algebra, revised third edition. *Graduate Texts in Mathematics*, 2002.

[40] M. Lesnick. The theory of the interleaving distance on multidimensional persistence modules. *Foundations of Computational Mathematics*, 15(3):613–650, 2015.

[41] M. Lesnick and M. Wright. Computing multigraded betti numbers of 2-D persistent homology modules in cubic time. *In preparation*, 2015.

[42] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington. A roadmap for the computation of persistent homology. *arXiv preprint arXiv:1506.08903*, 2015.

[43] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[44] C. D. Toth, J. O'Rourke, and J. E. Goodman. *Handbook of discrete and computational geometry.* CRC press, 2004.

[45] L. Wasserman. *All of statistics: a concise course in statistical inference.* Springer Verlag, 2004.

[46] C. Webb. Decomposition of graded modules. *Proceedings of the American Mathematical Society*, 94(4):565–571, 1985.

[47] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete and Computational Geometry*, 33(2):249–274, 2005.

Notation Index

S union of the supports of the 0th and 1st bigraded Betti numbers of M, page 24

S^e totally ordered partition of S, page 29

σ_j permutation of $[m_j]$, page 44

$\mathcal{T}^e$ barcode template at cell e, page 13

$\xi_i(M)$ i^{th} (graded) Betti number of M, page 10

ζ^e map from positive integers to template points at cell e, page 29

$\mathbf{Z}^n$ Poset category of $\mathbb{Z}^n$, page 21