Comput. Geom. 90 (2020) 101658

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Reconstructing embedded graphs from persistence diagrams

Robin Lynne Belton [a], Brittany Terese Fasy [a,b], Rostik Mertz [b], Samuel Micka [b], David L. Millman [b], Daniel Salinas [c], Anna Schenfisch [a], Jordan Schupbach [a,*], Lucia Williams [b]

[a] Depart. of Mathematical Sciences, Montana State University, United States of America
[b] School of Computing, Montana State University, United States of America
[c] Computer Science Depart., Westminster College, United States of America

ARTICLE INFO

Article history:
Available online 27 April 2020

Keywords:
Shape reconstruction
Persistent homology
Topological descriptors

ABSTRACT

The persistence diagram (PD) is an increasingly popular topological descriptor. By encoding the size and prominence of topological features at varying scales, the PD provides important geometric and topological information about a space. Recent work has shown that well-chosen (finite) sets of PDs can differentiate between geometric simplicial complexes, providing a method for representing complex shapes using a finite set of descriptors. A related inverse problem is the following: given a set of PDs (or an oracle we can query for persistence diagrams), what is underlying geometric simplicial complex? In this paper, we present an algorithm for reconstructing embedded graphs in $\mathbb{R}^d$ (plane graphs in $\mathbb{R}^2$) with n vertices from $n^2-n+d+1$ directional (augmented) PDs. Additionally, we empirically validate the correctness and time-complexity of our algorithm in $\mathbb{R}^2$ on randomly generated plane graphs using our implementation, and explain the numerical limitations of implementing our algorithm.

1. Introduction

Topological data analysis (TDA) provides a set of promising tools to help analyze data in fields as varied as materials science, transcriptomics, and neuroscience [19,24,29]. The wide applicability is due to the fact that many forms of data can be modeled as graphs or simplicial complexes, two widely-studied types of topological spaces. Topological spaces are described in terms of their invariants–such as the homotopy type or homology classes. Persistent homology considers the evolution of the homology groups in a filtered topological space.

Motivation The problem of manifold and stratified space learning is an active research area in computational mathematics. For example, Chambers et al. use persistent homology in a stratified space setting [5], describing an algorithm to identify shapes that simplify a noisier shape, and then confirm that the given simplification still satisfies the desired topological properties. Zheng et al. also address a problem in space learning, and study the 3D reconstruction of plant roots from multiple 2D images [31], using persistent homology to ensure the resulting 3D root model is connected. Another reconstruction

* Corresponding author.
E-mail addresses: robin.belton@montana.edu (R.L. Belton), brittany.fasy@montana.edu (B.T. Fasy), antonmertz@gmail.com (R. Mertz), samuelmicka@montana.edu (S. Micka), david.millman@montana.edu (D.L. Millman), dsalinasduron@westminstercollege.edu (D. Salinas), annaschenfisch@montana.edu (A. Schenfisch), jordan.schupbach@montana.edu (J. Schupbach), luciawilliams@montana.edu (L. Williams).

https://doi.org/10.1016/j.comgeo.2020.101658
0925-7721/

problem involves reconstructing road networks. Three common approaches to solving these problems involve Point Clustering, Incremental Track Insertion, and Intersection Linking [1]. Ge, Safa, Belkin, and Wang develop a point clustering algorithm using Reeb graphs to extract the skeleton graph of a road from point-cloud data [17]. The original embedding can be reconstructed using a principal curve algorithm [23]. Karagiorgou and Pfoser give an algorithm to reconstruct a road network from vehicle trajectory GPS data by identifying intersections with clustering, then using vehicle trajectories to connect them [22]. Ahmed et al. provide an incremental track insertion algorithm to reconstruct road networks from point cloud data [2]. The reconstruction is done incrementally, using a variant of the Fréchet distance to partially match input trajectories to the reconstructed graph. Ahmed, Karagiorgou, Pfoser, and Wenk describe all these methods in [1]. Finally, Dey, Wang, and Wang use persistent homology to reconstruct embedded graphs. This research has also been applied to input trajectory data [12]. Dey et al. use persistence to guide the Morse cancellation of critical simplices. We see from these applications the necessity for reconstruction algorithms, and in particular the necessity for reconstruction algorithms of graphs since much of the research involving reconstruction of road networks involves reconstructing graphs.

We explore the reconstruction of graphs from a widely used topological descriptor of data, *persistence diagrams*. The problem of reconstruction for simplicial complexes has received significant recent attention [10,18,30]. Our work is motivated by [30], which proves that one can reconstruct simplicial complexes in $\mathbb{R}^2$ and $\mathbb{R}^3$ using a subset of a parameterized infinite set of persistence diagrams. In this work, we use a version of persistence diagrams that includes information that is not normally considered; thus, for clarity, we refer to these descriptors as *augmented persistence diagrams* (APDs). Our approach differs from those listed above because we provide a deterministic algorithm using APDs generated from specific directional height filtrations to reconstruct the original graph.

Our contribution In this work, we focus on graphs embedded in $\mathbb{R}^d$ (plane graphs in $\mathbb{R}^2$) and use directional APDs to reconstruct a graph. In particular, our main contributions are an upper bound on the number of APDs required for reconstructing embedded graphs in $\mathbb{R}^d$, a polynomial-time algorithm for reconstructing plane graphs, and the first deterministic reconstruction algorithm for embedded graphs in arbitrary dimension.

The current paper is an extension of conference proceedings from CCCG 2018 [3]. We extend the proceedings paper in the following ways: (1) we revise proofs for clarity; (2) we extend our algorithms for graph reconstruction to $\mathbb{R}^d$; (3) we expand our literature review to include a discussion of recent results; (4) we publicly release code for our algorithm[1]; and (5) we provide an experimental section to demonstrate the implementation.

2. Preliminaries

We begin by summarizing the necessary background information, but refer the reader to [13] for a more comprehensive overview of computational topology.

Axis-aligned directions When considering a standard unit basis vector of $\mathbb{R}^d$ in dimension $i \in \{1, \dots, d\}$, we will write e_i to denote the direction.

Plane graphs Among our main objects of study are *plane graphs* with straight-line embeddings (referred to simply as *plane graphs* throughout this paper). A plane graph in $\mathbb{R}^2$ is a set of vertices and a set of straight-line connections between pairs of vertices called edges (denoted by V and E respectively) such that no two edges in the embedding cross. We frequently denote $|V| = n$ as the number of vertices in G. Throughout this paper, we make assumptions about the positioning of vertices in graphs.

General Position Assumption. Let G be a graph embedded in $\mathbb{R}^d$ with vertices V. We assume that for all $x, y \in V$ where $x = (x_1, \dots, x_d)$ and $y = (y_1, \dots, y_d)$, $x_i \neq y_i$ for any $i \in \{1, \dots, d\}$. Furthermore, we assume that no three vertices are collinear when projected onto the subspace spanned by e_1 and e_2.

Height filtration Let G be a plane graph. Consider a direction s in the unit sphere $\mathbb{S}^{d-1}$ in $\mathbb{R}^d$; we define the *lower-star filtration* with respect to direction s in two steps. First, let $h_s : G \to \mathbb{R}$ be defined for a simplex $\sigma \subseteq G$ by $h_s(\sigma) = \max_{v \in \sigma} v \cdot s$, where $v \cdot s$ is the inner (dot) product and measures height of v in the direction of s, since s is a unit vector. Thus, the height $h_s(\sigma)$ is the maximum height of all vertices in σ. Then, for each $t \in \mathbb{R}$, the subcomplex $G_t := h_s^{-1}((-\infty, t])$ is composed of all simplices that lie entirely below or at height t, with respect to direction s. Notice $G_r \subseteq G_t$ for all $r \leq t$ and $G_r = G_t$ if no vertex has height in the interval $[r, t]$. The sequence of all such subcomplexes, indexed by $\mathbb{R}$, is the *height filtration* with respect to s, denoted $F_s := F_s(G)$. Notice that the complex changes a finite number of times in this filtration. We note that the lower-star filtration is the discrete analog to a lower-levelset filtration, where the complex is intersected with a raising closed half-plane.

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

Augmented persistence diagrams The persistence diagram for a filtered simplicial complex K is a summary of the homology groups as the parameter ranges from $-\infty$ to ∞; in particular, the persistence diagram is a set of birth-death pairs of the form (b, d) in the extended plane, each with a corresponding dimension $k \in \mathbb{Z}_{\geq 0}$. In particular, each pair represents an independent generator of the k-th homology group $H_k(K_t)$ for $t \in [b, d)$. For technical reasons, all points in the diagonal $y = x$ are also included with infinite multiplicity. In [3], we introduced the notion of the *augmented persistence diagrams* (APDs), which is persistence diagram with an additional finite multi-set of points on the diagonal. These points can be considered as the points explicitly computed in an algorithm (such as the matrix reduction described in [14]), but a formal definition using the paired simplices of K is provided in [25, §2.5]. In the remainder of the paper, when we use "diagram," we mean APD; we write "persistence diagram" when we mean the traditionally considered diagram. We denote the space of all APDs by $\mathcal{D}$.

For height filtrations of a graph embedded in $\mathbb{R}^d$, a zero-dimensional birth occurs when the height filtration discovers a new vertex, representing a new connected component. A one-dimensional birth occurs when a one-cycle is completed. Zero-dimensional deaths correspond to connected components merging. One-dimensional deaths are all at ∞. All higher-dimensional homology groups are trivial.

For a direction $s \in \mathbb{S}^{d-1}$, let the *directional augmented persistence diagram* $D(F_s(K))$ be the set of birth-death pairs from the height filtration $F_s(K)$. Since each point in $D(F_s(K))$ has an associated dimension, we may restrict our attention to the points in a specified dimension. In particular, for $i \in \mathbb{Z}$, let $D_i(F_s(K))$ be the diagram restricted to the points with dimension i.[2] As with the height filtration, we simplify notation and define $D_i(s) := D_i(F_s(K))$ when the complex is clear from context. We denote β_i to be the i-th Betti number, i.e., the rank of the i-th homology group. In general, the complexity of computing a diagram is matrix multiplication time, with respect to the number n of simplicies in the filtration; that is, the complexity is $\Theta(n^\omega)$, where ω corresponds to the smallest known exponent for matrix multiplication time. In some cases (e.g., for computing $D_0(F)$ or $D_{d-1}(F)$ when F is a height filtration in $\mathbb{R}^d$), the computation time is $\Theta(n\alpha(n))$, where α is the inverse Ackermann function.

In what follows, for the unknown complex K, we assume that we have an *oracle* $\mathcal{O}$ that takes a direction $s \in \mathbb{S}^{d-1}$ and produces the diagram $D(s)$ in that direction. Additionally, we may restrict the diagram to specific dimension(s) by specifying a dimension $i \in \mathbb{Z}$ and denoting the (sub-)diagram as $D_i(s)$.

We define T_G such that $\Theta(T_G)$ is the time complexity for $\mathcal{O}$ to return this diagram. Notice that $T_G = \Omega(|D(s)|)$, where $|D(s)|$ denotes the number of off-diagonal points in $D(s)$.

Next, we state a lemma relating birth-death pairs. We omit the proof, but refer the reader to [13, pp. 120–121 of §V.4] for more details.

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

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

As a result of Lemma 1, we can construct our filtration by adding one simplex at a time and form a bijection between simplices of K and birth-death events in the resulting APD. If G is any graph, then the maximum number of edges in G is $n(n-1)/2$, and so $|E| = O(n^2)$. In the case when G is a plane graph, $|E| = O(n)$ due to the planarity of G. Furthermore, an APD will have at least n points from the vertices in G corresponding to births in the zero-dimensional diagram. These observations give us the following corollary on the size of APDs for graphs.

Corollary 2 *(Size of augmented persistence diagrams). Let G be an embedded graph and n be the number of vertices in G. Then an augmented persistence diagram has $O(n^2)$ birth-death pairs. In the case when G is a plane graph, the diagram has $\Theta(n)$ birth-death pairs.*

3. Related work

In [30] Turner et al., introduced the *persistent homology transform (PHT)* that represents a shape in $\mathbb{R}^d$—such as a simplicial complex—as a family of persistence diagrams, parameterized by $\mathbb{S}^{d-1}$. In particular, the diagram for parameter $s \in \mathbb{S}^{d-1}$ is the persistence diagram defined by the height filtration in direction s. The *Euler characteristic transform (ECT)* is defined similarly and maps a shape to a parameterized family of Euler characteristic curves (ECCs), where the ECC is the graph of Euler characteristic by filtration parameter, for the same directional height filtrations. Turner et al. show that both of these maps are injective for simplicial complexes in $\mathbb{R}^2$ or $\mathbb{R}^3$. Recently, variations of the PHT and ECT have attracted interest in other research domains and researchers are realizing the potential of persistent homology as an effective data descriptor. For example, Crawford et al. [9] introduces the *smooth Euler characteristic transform (SECT)* as a method of predicting clinical

[2] The diagram $D_i(F_s(K))$ is actually a sub-diagram of $D(F_s(K))$. As such, the set of all diagrams for a given direction is counted as one diagram, not one for each dimension.

outcomes using MRIs from patients with glioblastoma multiforme. In [20], persistence diagrams are used as features in deep neural networks for classification of surfaces and graphs. Additional injectivity and representation results for ECT and other topological transforms are studied in [10,18]. Furthermore, a recent survey by Oudot and Solomon explores the current state of inverse problems in topological persistence as a potential tool for producing explainable data descriptors [28].

In order to leverage the injectivity of the PHT for shape comparison, two approaches can be taken: (1) show that the PHT has a finite representation; (2) provide an algorithm that reconstructs the shape from a finite set of directions. In fact, (2) is a harder problem than (1), since an algorithm for reconstruction will need to define a finite representation of the PHT. One method to tackle approach (1) is to observe that diagrams only provide new information when a transposition in the ordering of the filtration occurs; this observation is a direct result of [7] and guides the intuition behind the current paper, as well as [3,10,30]. For example, consider a finite geometric simplicial complex in $\mathbb{R}^d$ for some $d \geq 2$. Since transpositions can only happen when two vertices are transposed in the filter, the set of directions in $\mathbb{S}^{d-1}$ for which two vertices occur at the same height is finite. In other words, there are two hemispheres of $\mathbb{S}^{d-1}$ for two given vertices, say v and w: one where v is seen before w and one where w is seen before v. We take the hemispheres for all $\binom{n}{2}$ pairs of vertices and consider the regions for which the ordering is consistent. For each such region, the persistence diagram continuously varies and no transpositions (or 'knees') are witnessed [8]. In the current paper, we take approach (2) and show that we can use an oracle to select a finite set of directions $P \subset \mathbb{S}^{d-1}$ that allow us to reconstruct the original complex from the directional augmented persistence diagrams from directions in P. In particular, we prove that a quadratic number of directions (with respect to the number of vertices) is sufficient to reconstruct a graph.

4. Vertex reconstruction

Next, we present an algorithm for recovering the locations of vertices of an embedded graph. We begin with a plane graph G, where we are able to use three directional augmented persistence diagrams. We then extend this method for any embedded graph in $\mathbb{R}^d$, using $d+1$ directional augmented persistence diagrams.

4.1. Vertex reconstruction for plane graphs

The intuition behind vertex reconstruction is that for each direction, we identify the lines on which the vertices of G must lie. We show how to choose specific directions so that we can identify all vertex locations by searching for points in the plane where three lines intersect. We call these lines *filtration lines*:

Definition 3 *(Filtration hyperplanes and filtration lines).* Given a direction $s \in \mathbb{S}^{d-1}$ and a height $h \in \mathbb{R}$, the *filtration hyperplane at height h* is the $(d-1)$-dimensional hyperplane, denoted $\ell(s,h)$, through point $h * s$ and perpendicular to direction s, where $*$ denotes scalar multiplication. Given a finite set of vertices $V \subset \mathbb{R}^d$, the *filtration hyperplanes of V* are the set of hyperplanes

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

In the special case when $d=2$, we refer to filtration hyperplanes as *filtration lines*.

By construction, all hyperplanes in $\mathbb{L}(s,V)$ are parallel, and $\ell(s,h) = \ell(-s,-h)$. In particular, if $v \in V$, where V is the vertex set of a plane graph G, then the line $\ell(s,h_s(v))$ is perpendicular to s and occurs at the height where the filtration $F_s(G)$ includes v for the first time. In what follows, we adopt the following notation: given a direction $s_i \in \mathbb{S}^1$ and a point $p \in \mathbb{R}^2$, define $\ell_i(p) := \ell(s_i, h_{s_i}(p))$ as a way to simplify notation.

By Lemma 1, the births in the zero-dimensional augmented persistence diagram are in one-to-one correspondence with the vertices of the plane graph G. This means that, given a filtration line $\ell(s,h)$, exactly one vertex in V lies on $\ell(s,h)$. Using filtration lines from three directions, we show a one-to-one correspondence between three-way intersections of filtration lines and the vertices in G in the next lemma:

Lemma 4 *(Vertex existence). Let $G=(V,E)$ be a plane graph and let $n=|V|$. Let $s_1, s_2 \in \mathbb{S}^1$ be linearly independent and further suppose that $\mathbb{L}(s_1,V)$ and $\mathbb{L}(s_2,V)$ each contain n lines. Let A be the set of n^2 intersection points between lines in $\mathbb{L}(s_1,V)$ and in $\mathbb{L}(s_2,V)$. Let $s_3 \in \mathbb{S}^1$ such that each $a \in A$ has a unique height in direction s_3. Then, the following equality holds: $V = \mathbb{L}(s_3,V) \cap A$.*

Proof. We prove this equality in two steps. First, we prove $V \subseteq \mathbb{L}(s_3,V) \cap A$. Let $v \in V$. Then, for $i \in \{1,2,3\}$, there exists $\ell_i(v) \in \mathbb{L}(s_i,V)$ such that $v \in \ell_i(v)$. Thus, $v \in \ell_1(v) \cap \ell_2(v) \cap \ell_3(v)$ and

$$\ell_1(v) \cap \ell_2(v) \cap \ell_3(v) = \ell_1(v) \cap \Big(\ell_2(v) \cap \ell_3(v)\Big) \subset \mathbb{L}(s_i,V) \cap A,$$

by the definitions of filtration lines and A. Thus, $V \subseteq \mathbb{L}(s_3,V) \cap A$.

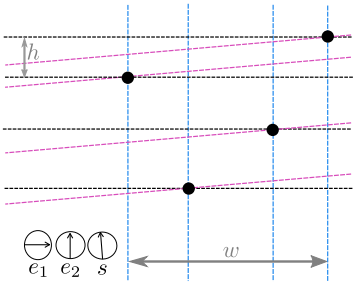


Fig. 1. A vertex set V of size four, with three sets of filtration lines. Here, notice that $e_1 \in \mathbb{S}^1$ and $e_2 \in \mathbb{S}^1$ are linearly independent, and the third direction s satisfies the assumptions of Lemma 4. The lines of $\mathbb{L}(e_1, V)$ are the blue vertical lines, $\mathbb{L}(e_2, V)$ are the black horizontal lines, and $\mathbb{L}(s, V)$ are the pink diagonal lines. The three-way intersection points (one from each set of filtration lines) is in one-to-one correspondence to the vertices in V. Lines 4 through 7 in Algorithm 1 provide details for finding s_3 using the width of the vertical lines (marked w) and the minimum height difference between horizontal lines (marked h).

Next, we prove $V \supseteq \mathbb{L}(s_3, V) \cap A$. Since each $v \in V \subseteq A$ has a unique height in direction s_3 and since every line of $\mathbb{L}(s_3, V)$ contains a vertex, we know that $\mathbb{L}(s_3, V)$ has n lines. Thus, we claim that each of these parallel lines intersects A. Assume, for contradiction, that there exists $\ell \in \mathbb{L}(s_3, V)$ such that $A \cap \ell = \emptyset$. Since ℓ is a filtration line in direction s_3, there exists $v \in V$ such that $\ell = \ell_3(v)$, meaning that this v lies on ℓ. However, $v \in A$ since $V \subseteq \mathbb{L}(s_3, V) \cap A \subseteq A$, contradicting the hypothesis that $A \cap \ell = \emptyset$. □

If we generate vertical lines, $L_V = \mathbb{L}(e_1, V)$, and horizontal lines, $L_H = \mathbb{L}(e_2, V)$, for our first two directions, then only a finite number of directions in $\mathbb{S}^1$ have been eliminated for the choice of s_3. In the next lemma, we choose a specific third direction by considering a bounding region defined by the largest distance between any two lines in L_V and smallest distance between any two consecutive lines in L_H. Then, we pick the third direction so that if one of the corresponding lines intersects the bottom left corner of this region then it will also intersect the along the right edge of the region. In Fig. 1, the third direction was computed using this procedure with the region having a width that is the length between the left most and right most vertical lines, and height that is the length between the top two horizontal lines. Next, we give a more precise description of the vertex localization procedure.

Lemma 5 *(Vertex localization). Let L_H and L_V be n horizontal and n vertical lines, respectively. Let w (and h) be the largest (and smallest) distance between two lines of L_V (and L_H, respectively). Let B be the smallest axis-aligned bounding region containing the intersections of lines in $L_H \cup L_V$. Let $s = (w, h/2)/||(w, h/2)||$, i.e., a unit vector oriented towards the point $(w, h/2)$. Any line parallel to s can intersect at most one line of L_H in B.*

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

We conclude the discussion of plane graph reconstruction with an algorithm to compute the coordinates of the vertices of the original graph in $\mathbb{R}^2$, using only three diagrams.

Theorem 6 *(Vertex reconstruction). Let G be a plane graph. We can compute the coordinates of all n vertices of G using three directional augmented persistence diagrams in $\Theta(n \log n + T_G)$ time, where $\Theta(T_G)$ is the time complexity of computing a single directional augmented persistence diagram for G.*

Proof. We proceed with a constructive proof that is presented as an algorithm in Algorithm 1. Let $\mathcal{O}$ be an oracle that takes a direction $s \in \mathbb{S}^1$ and returns the zero-dimensional directional APD for the unknown plane graph G in direction s in $\Theta(T_G)$ time.

We start with requesting two directional augmented persistence diagrams from the oracle, $D_0(e_1)$ and $D_0(e_2)$. Note that, by the General Position Assumption, no two vertices of G share an x- or y-coordinate. By Corollary 2, the sets $\mathbb{L}(e_1, V)$ and $\mathbb{L}(e_2, V)$ (which we do not explicitly construct) each contain n distinct lines. Let $L_1 = \{h_1, h_2, \ldots, h_n\}$ be the resulting set of heights of lines in $\mathbb{L}(e_1, V)$, in increasing order. Likewise, let $L_2 = \{h'_1, h'_2, \ldots, h'_n\}$ be the ordered set of heights of lines in $\mathbb{L}(e_1, V)$, also in increasing order. As a consequence of Lemma 1, the birth times in the zero-dimensional diagrams are in one-to-one correspondence to heights of the filtration lines. Thus, we can compute L_1 and L_2 from the APD in direction

Algorithm 1 Reconstruct vertices.

Input: Oracle $\mathcal{O}$ for an unknown graph $G = (V, E) \subset \mathbb{R}^2$.
Output: Set of vertex locations.
1: Consult $\mathcal{O}$ to obtain diagrams $D_0(e_1)$ and $D_0(e_2)$
2: $L_1 \leftarrow$ birth times from points in $D_0(e_1)$
3: $L_2 \leftarrow$ birth times from points in $D_0(e_2)$
4: Sort L_1 and L_2 in increasing order
5: $w \leftarrow$ maximum minus minimum in L_1
6: $h \leftarrow$ minimum gap between two consecutive values in L_2
7: $s \leftarrow$ a unit vector perpendicular to the vector $[w, h/2]$
8: Consult $\mathcal{O}$ to obtain diagram $D_0(s)$
9: $L_s \leftarrow$ birth times from points in $D_0(s)$
10: Sort L_s in increasing order
11: **for** $i = 1, 2, \ldots, n$ **do**
12: $\quad \ell_i \leftarrow$ horizontal line with the i-th element of L_2 as the y-coordinate
13: $\quad \ell_i' \leftarrow$ line perpendicular to s at height equal to the i-th element of L_s
14: $\quad v_i = \ell_i \cap \ell_i'$
15: **end for**
16: **return** $\{v_i\}_{i=1}^n$

s in $\Theta(n)$ time by iterating through the points in the zero-dimensional augmented persistence diagram, then sorting in $\Theta(n \log n)$ time.

Let A be the set of n^2 intersection points between the lines in $\mathbb{L}(e_1, V)$ and in $\mathbb{L}(e_2, V)$. Exactly n of these points correspond to vertices of V. The next step is to identify a third direction s such that each line in $\mathbb{L}(s, V)$ intersects with only one point in A, which we will use in order to distinguish which n intersection points correspond to vertices in V.

Let $w = h_n - h_1$ and let h be the minimum of $\{h_i' - h_{i-1}'\}_{i=2}^n$. In words, w is the difference between the maximum and minimum heights of lines in $\mathbb{L}(e_1, V)$ and h is the minimum height difference between consecutive lines in $\mathbb{L}(e_2, V)$; see Fig. 1. Note that we can compute w in $\Theta(1)$ time from L_1 and h in $\Theta(n)$ time from L_2. Let B be the smallest axis-aligned bounding region containing the intersection points A, and let s be a unit vector perpendicular to the vector $[w, \frac{h}{2}]$. We request the set $D_0(s)$ from our oracle $\mathcal{O}$. As before, the heights of the lines in $\mathbb{L}(s, V)$ are the birth times of points in $D_0(s)$. We save this set of heights as L_s in $\Theta(n)$ time, and sort L_s in $\Theta(n \log n)$ time.

Finally, by Lemma 5, any line in $\mathbb{L}(s, V)$ intersects no more than one line of $\mathbb{L}(e_2, V)$ within B. Furthermore, by Lemma 4, the n vertices in G are in one-to-one correspondence with points in $\mathbb{L}(s, V) \cap A$. We compute these three-way intersections of $\mathbb{L}(s, V) \cap A$ by intersecting the i-th line of $\mathbb{L}(e_2, V)$ with the i-th line of $\mathbb{L}(s, V)$ in $\Theta(n)$ time.

In total, this algorithm, summarized in Algorithm 1, uses three directional diagrams, two requested from the oracle in Line 1 and one requested in Line 8. These two lines take $\Theta(T_G)$ time each, Lines 4 and 10 take $\Theta(n \log n)$ time each, and the for loop in Lines 11 through 15 takes $\Theta(n)$ time. All other lines are linear or constant, with respect to n. Thus, the total time complexity is $\Theta(n \log n + T_G)$. □

4.2. Vertex reconstruction in $\mathbb{R}^d$

The vertex reconstruction algorithm of the previous subsection generalizes to higher dimensions. In $\mathbb{R}^d$, a filtration line becomes a *filtration hyperplane*, a $(d-1)$-dimensional hyperplane that goes through one of the vertices in the vertex set (and is perpendicular to a given direction). Similar to filtration lines, filtration hyperplanes generated by a fixed direction are parallel and are in a one-to-one correspondence with the vertices (for almost all directions).

Lemma 7 *(Generalized vertex existence). Let $G = (V, E)$ be a straight-line embedded graph in $\mathbb{R}^d$. Let $s_1, s_2, \ldots, s_d$ be linearly independent directions in $\mathbb{S}^{d-1}$ and further suppose that $\mathbb{L}(s_i, V)$ contains n filtration hyperplanes for each $i \in \{1, 2, \ldots, d\}$. Choosing one hyperplane in each set $\mathbb{L}(s_i, V)$, the intersection of these hyperplanes is a point. Let A denote the n^d such intersection points. Let $s_{d+1} \in \mathbb{S}^{d-1}$ such that each $a \in A$ has a unique height in direction s_{d+1}. Then, the following equality holds: $V = \mathbb{L}(s_{d+1}, V) \cap A$.*

Proof. This proof follows the same structure of the proof of Lemma 4, generalizing $\mathbb{R}^2$ to $\mathbb{R}^d$.

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

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

Just as in the case of plane graphs, we can now describe a method for locating all vertices. The following lemma is a higher-dimensional analogue of Lemma 5.

Lemma 8 *(Generalized vertex localization). Let $G = (V, E)$ be a straight-line embedded graph in $\mathbb{R}^d$, with $n = |V|$. Choosing one hyperplane in each set $\mathbb{L}(e_i, V)$ for $1 \le i \le d$, the intersection of these hyperplanes is a point. Let A denote the n^d such intersection*

points. Then, we can find a direction $s \in \mathbb{S}^{d-1}$ such that each hyperplane in $\mathbb{L}(s, V)$ intersects at most one of the n^d points in A in $\Theta(dn \log n)$ time.

Proof. Let $L_i = \{h_1^{(i)}, h_2^{(i)}, \ldots, h_n^{(i)}\}$ be the ordered set of heights of hyperplanes in $\mathbb{L}(e_i, V)$. Let $w^{(i)} = h_n^{(i)} - h_1^{(i)}$; in other words, $w^{(i)}$ is the largest distance between any two hyperplanes in the set $\mathbb{L}(e_i, V)$, and let $w = \max_{1 \le i \le d} w^{(i)}$. Let $h^{(i)} = \min\{h_j^{(i)} - h_{j-1}^{(i)}\}_{j=2}^{n}$; in other words, $h^{(i)}$ is the smallest height difference between any two (adjacent) hyperplanes in the set $\mathbb{L}(e_i, V)$, and let $h = \frac{1}{2}\min_{1 \le i \le d} h^{(i)}$. Then, we consider the hyperplane that intersects the origin and each point $we_i + \langle 0, \ldots 0, \frac{h}{d-1} \rangle$ for $i \in \{1, 2, \ldots, d-1\}$. Next, we choose a vector orthogonal to the hyperplane. Without loss of generality, we choose

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

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

We now show that s satisfies the claim that each hyperplane in $\mathbb{L}(s, V)$ intersects at most one of the n^d points in A and that we can find s in $\Theta(dn \log n)$ time. Let $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ be points in A with $p \neq q$, and assume, for contradiction, that they lie on the same hyperplane in $\mathbb{L}(s, V)$. Then, $p \cdot s = q \cdot s$. By the definition of dot product, we have the following equation (after multiplying s by $||x||$):

$$\frac{d-1}{h}(p_d - q_d) - \sum_{i=1}^{d-1} \frac{1}{w}(p_i - q_i) = 0.$$

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

$$|p_d - q_d| = \frac{h}{w(d-1)} \left| \sum_{i=1}^{d-1} (p_i - q_i) \right|. \tag{1}$$

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

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

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

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

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

Theorem 9 *(Vertex reconstruction in higher dimensions). Let G be a straight-line embedded graph in $\mathbb{R}^d$ for $d > 1$. We can compute the coordinates of all n vertices of G using $d+1$ directional augmented persistence diagrams in $\Theta(dn^{d+1} + dT_G)$ time, where $\Theta(T_G)$ is the time complexity of computing a persistence diagram.*

Proof. We proceed with a constructive proof, generalizing the constructive proof from Theorem 6 and Algorithm 1. Let $\mathcal{O}$ be an oracle that takes a direction $s \in \mathbb{S}^{d-1}$ and returns the $D_0(s)$ in $\Theta(T_G)$ time.

For $i \in \{1, \ldots, d\}$, we use this oracle to obtain $D_0(e_i)$. Note that, by the General Position Assumption and Corollary 2, for each of these directions, we have exactly n distinct filtration hyperplanes, in one-to-one correspondence with the vertices. Note that, for a given direction e_i, we store the filtration hyperplanes as a list of the vertex heights. Choosing one hyperplane

in each direction yields d pairwise orthogonal hyperplanes; their intersection is a point in $\mathbb{R}^d$ and this point is a potential vertex location. In total, we have n^d potential vertex locations, of which only n are actual vertices. We denote this set of n^d potential vertex locations by A. Then, computing these lists of vertex heights takes $\Theta(T_G)$ time per dimension to account for computing and listing the points of the APD.

Let s be chosen as in Lemma 8 in $\Theta(dn \log n)$ time. By Lemma 8, each hyperplane $\ell_s(v)$ intersects at most one point in A for each $v \in V$. Thus, by Lemma 7, there are exactly n distinct intersections between $\mathbb{L}(s, V)$ and A, in one-to-one correspondence with the n vertices of G.

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

In total, this algorithm uses $d+1$ directional diagrams. The time complexity of constructing the $d+1$ sets of filtration hyperplanes is $\Theta(dn \log n + dT_G)$, and an additional $\Theta(dn^{d+1})$ time to compute the actual vertex locations. Thus, the total time complexity is $\Theta(dn^{d+1} + dT_G)$. □

5. Edge reconstruction

Given the vertices constructed in Section 4, we describe how to reconstruct the edges in an embedded graph using $n^2 - n$ augmented persistence diagrams. The key to determining whether an edge exists or not is counting the degree of a vertex for edges in the half plane "below" the vertex with respect to a given direction. We begin with a method for reconstructing plane graphs, and then extend our method to embedded graphs in $\mathbb{R}^d$.

5.1. Edge reconstruction for plane graphs

We first define necessary terms, and then describe our algorithm for constructing edges.

Definition 10 *(Indegree of vertex).* Let G be a straight-line embedded graph in $\mathbb{R}^d$ with vertex set V. Then, for every vertex $v \in V$ and every direction $s \in \mathbb{S}^{d-1}$, we define:

$$\textsc{Indeg}(v, s) = |\{(v, v') \in E \mid s \cdot v' \leq s \cdot v\}|.$$

Thus, the indegree of v is the number of edges incident to v that lie below v, with respect to direction s; see Fig. 2.

Given a directional augmented persistence diagram, we prove that we can compute the indegree of a vertex with respect to that direction:

Lemma 11 *(Indegree from diagram). Let $G = (V, E)$ be a straight-line embedded graph in $\mathbb{R}^d$. Let $s \in \mathbb{S}^{d-1}$ such that no two vertices have the same height with respect to s (and thus $|\mathbb{L}(s, V)| = n$). Let $D_0(s)$ and $D_1(s)$ be the zero- and one-dimensional points of the augmented persistence diagram resulting from the height filtration $F_s(G)$. Then, for all $v \in V$,*

$$\textsc{Indeg}(v, s) = |\{(x, y) \in D_0(s) \mid y = v \cdot s\}| + |\{(x, y) \in D_1(s) \mid x = v \cdot s\}|.$$

Furthermore, if $n = |V|$ and $d = 2$ then $\textsc{Indeg}(v, s)$ can be computed in $\Theta(n)$ time. If $d > 2$, then $\textsc{Indeg}(v, s)$ can be computed in $O(n^2)$ time.

Proof. Let $v, v' \in V$ such that $s \cdot v' < s \cdot v$, i.e., the vertex v' is lower than v in direction s. Let $e = (v, v') \in E$. Then, by Lemma 1, we have two cases to consider when e is added to F_s:

Case 1: e joins two disconnected components. If e connects two previously disconnected components, then e is associated with a death in $D_0(s)$ at height $s \cdot v$. Moreover, since all deaths in $D_0(s)$ are associated with adding an edge, we know that the set of all edges that fall into this case with v as the top endpoint is $A = \{(x, y) \in D_0(s) \mid y = v \cdot s\}$.

Case 2: e creates a one-cycle. In this case, e is associated with a birth in $D_0(s)$ at height $s \cdot v$. Thus, we have that $B = \{(x, y) \in D_1(s) \mid x = v \cdot s\}$ is the set of edges that fall into this case with v as the top endpoint.

The union $A \cup B$ is the set of all edges ending at v with respect to s, hence $\textsc{Indeg}(v, s) = |A \cup B|$. Furthermore, by Corollary 2, if $d = 2$, then G is a plane graph and each of $D_0(s)$ and $D_1(s)$ have $\Theta(n)$ points, so we count and sum the points joining two disconnected components or creating one-cycles at height v in time $\Theta(n)$ time. If $d > 2$, then each of $D_0(s)$ and $D_1(s)$ have $O(n^2)$ points by Corollary 2, so we count and sum the points joining two disconnected components or creating one-cycles at height v in time $O(n^2)$ time. □

In order to decide whether an edge (v, v') exists between two vertices, we look at the degree of v as seen by two close directions such that v' is the only vertex in what we call a *wedge at v*:

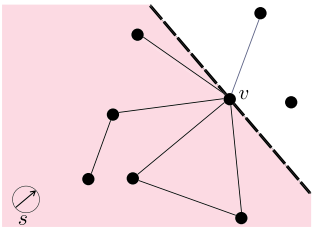


Fig. 2. A plane graph with a dashed line drawn intersecting v in the direction perpendicular to s. Since four edges incident to v lie below v indicated by the shaded region, with respect to direction s, $\textsc{Indeg}(v, s) = 4$.

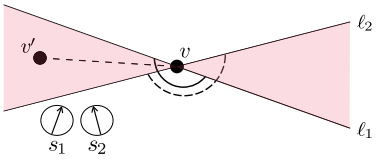


Fig. 3. Bow tie $\mathbb{B}$ at v, denoted by the shaded area. $\mathbb{B}$ contains exactly one vertex, v', so the only potential edge in $\mathbb{B}$ is (v, v'). In order to determine if there exists an edge between v and v', we compute $\textsc{Indeg}(v, s_1)$ and $\textsc{Indeg}(v, s_2)$, i.e., the number of edges incident to v in the solid and dashed arcs, respectively. An edge exists between v and v' if and only if $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1$.

Definition 12 *(Wedge).* Let $v \in V$, and choose $s_1, s_2 \in \mathbb{S}^{d-1}$. Then, a *wedge* at v is the closure of the symmetric difference between the half planes below v in directions s_1 and s_2. In the special case when $d = 2$, we refer to the wedge as a *bow tie*.

In what follows, we use a wedge that is defined by a direction perpendicular to an edge and slight tilts of that direction. In the proof of [30, Theorem 3.1], Turner et al. also develop a construction that is conceptually similar to the wedge for identifying links where changes in Betti numbers are observed when considering the effect of the continuous rotation of directions above and below the plane orthogonal to an edge.

Because we assume that no three vertices in our plane graph are collinear, for each pair of vertices $v, v' \in V$, we always find a bow tie centered at v that contains the vertex v' and no other vertex in V; see Fig. 3. We use bow tie regions to determine if there exists an edge between v and v'. In the next lemma, we show how to decide if the edge (v, v') exists in our plane graph.

Lemma 13 *(Edge existence). Let $G = (V, E)$ be a straight-line embedded graph in $\mathbb{R}^d$. Let $v, v' \in V$. Let $s_1, s_2 \in \mathbb{S}^{d-1}$ such that the wedge $\mathbb{B}$ at v defined by s_1 and s_2 satisfies: $\mathbb{B} \cap V \setminus \{v\} = v'$. Then,*

$$\left|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)\right| = 1 \iff (v, v') \in E.$$

Proof. Since edges in G are straight lines, any edge incident to v either falls entirely in the wedge region $\mathbb{B}$ or the interior of the edge is on the same side (above or below) of both hyperplanes that define $\mathbb{B}$. Let A be the set of edges that are incident to v and below both hyperplanes; that is, $A = \{(v, w) \in E \mid s_1 \cdot w < s_1 \cdot v \text{ and } s_2 \cdot w < s_2 \cdot v\}$. Next, we split the wedge into the two infinite cones defined by the two connected components of the interior of $\mathbb{B}$. Let $\mathbb{B}_1$ be the set of edges in one cone and $\mathbb{B}_2$ be the set of edges in the other cone. Then, by definition of indegree,

$$\begin{aligned}\left|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)\right| &= \left||A| + |\mathbb{B}_1| - |A| - |\mathbb{B}_2|\right| \\ &= \left||\mathbb{B}_1| - |\mathbb{B}_2|\right|.\end{aligned}$$

As $\mathbb{B}$ contains one vertex v', $\left||\mathbb{B}_1| - |\mathbb{B}_2|\right|$ is one when $(v, v') \in E$, and zero otherwise. Therefore, we conclude that $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1 \iff (v, v') \in E$, as required. □

Next, we prove that we can find the embedding of the edges in plane graphs using $\Theta(n^2)$ directional augmented persistence diagrams. See Appendix A for an example of walking through the reconstruction.

Theorem 14 *(Edge reconstruction). Let $G = (V, E)$ be a plane graph. If V is known, then we can compute E using $n^2 - n$ directional augmented persistence diagrams in $\Theta(n^2 T_G)$ time, where $\Theta(T_G)$ is the time complexity of computing a single diagram.*

Proof. We prove this theorem constructively, and summarize the construction in Algorithm 2. In the algorithm, we first preprocess in order to find a global bow tie half-angle. Then we iterate through each pair of vertices and test to see if the edge exists. This edge test is done in two steps: first create a bow tie that isolates the potential edge, then apply Lemma 13

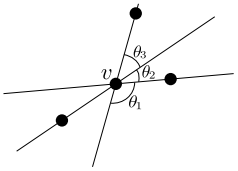


Fig. 4. Ordering of all vertices about v. Lines are drawn through all vertices and then angles are computed between all adjacent pairs of lines. The smallest angle is denoted as $\theta(v)$. Here, $\theta(v) = \theta_2$.

to determine if the edge exists or not by comparing the indegrees of v with respect to the two directions defining the bow tie.

Preprocessing (Lines 1–7 of Algorithm 2). We initialize a set E of edges to be the empty set in Line 1. Next, we compute an angle that is sufficiently small to be used to construct bow ties for every edge. For each vertex $v \in V$, we consider the cyclic ordering of the points in $V \setminus \{v\}$ around v; let $c[v]$ denote this ordered list of vertices. By Lemmas 1 and 2 of [26], we compute $c[v]$ for all $v \in V$ in $\Theta(n^2)$ total time.[3] Once we have these cyclic orderings, we compute all n lines through v and $v_i \in V \setminus \{v\}$ and can compute a cyclic ordering of all such lines through v in $\Theta(n)$ per vertex. (The step of obtaining the cyclic ordering of lines given the cyclic ordering of vertices is similar to the merge step of merge sort). Given two adjacent lines through v, consider the angle between these lines; see the angles labeled θ_i in Fig. 4. For each vertex v, the minimum of such angles, denoted $\theta(v)$, is computed in Line 5 in $\Theta(n)$ time. Finally, we define $\theta = \frac{1}{2}\min_{v \in V} \theta(v)$ in Line 7. The value θ will be used to compute bow ties in the edge test. The runtime for this preprocessing is $\Theta(n^2)$ and requires no augmented persistence diagrams.

Edge Test (Lines 9–17). Let $\mathcal{O}$ be an oracle that takes a direction s in $\mathbb{S}^1$ and returns the APD for the unknown plane graph G in direction s in time $\Theta(T_G)$. Let $v, v' \in V$ such that $v \neq v'$. We now provide the two steps necessary to test if $(v, v') \in E$ using only two diagrams.

The first step is to construct bow ties (Lines 9–12 of Algorithm 2). Let s be a unit vector perpendicular to vector $v' - v$, and let s_1, s_2 be the two unit vectors that form angles $\pm\theta$ with s. Note that v and v' are at the same height in direction s, but different heights in direction s_1 and s_2 (and, in fact, their order changes between directions s_1 and s_2). We consult the oracle $\mathcal{O}$ to obtain the APDs $D(s_1)$ and $D(s_2)$. Note that we need the zeroth- and first-dimensional diagrams only, and these are the only non-trivial diagrams for a graph. Let $\mathbb{B}$ be the bow tie between $\ell(s_1, h_{s_1}(v))$ and $\ell(s_2, h_{s_2}(v))$. Note that, by construction, $\mathbb{B}$ contains exactly one point from V, namely v'. This first step of the edge test takes $\Theta(T_G)$ time and will use two augmented persistence diagrams.

The second step of the edge test is to compute indegrees of v in order to determine if there exists an edge between v and v' (Lines 13–17 of Algorithm 2). By Lemma 11, we compute the indegrees $\textsc{Indeg}(v, s_1)$ and $\textsc{Indeg}(v, s_2)$ from $D(s_1)$ and $D(s_2)$, respectively, in $\Theta(n)$ time; see Lines 13 and 14. Then, using Lemma 13, we determine whether the edge (v, v') is in E by checking if $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1$. If this equality holds, the edge exists; if not, the edge does not; see Lines 15–17. The bottleneck of the edge test is the $\Theta(n)$ indegree computation, the second step of the edge test takes $\Theta(n)$ time. We do not compute additional diagrams in this step.

We apply the edge test for all $\binom{n}{2} = \frac{1}{2}(n^2 - n)$ distinct pairs in V. For all pairs, the complexity of the edge test uses $n^2 - n$ persistence diagrams and takes $\Theta(n^2 T_G + n^3)$ time. Observing that the time to compute a diagram D is $\Omega(|D|)$ and using Corollary 2, we observe that T_G is $\Omega(n)$. As a result, we can simplify $\Theta(n^2 T_G + n^3)$ to $\Theta(n^2 T_G)$. Thus, the runtime of Algorithm 2 is $\Theta(n^2 T_G)$ ($\Theta(n^2 T_G)$ for preprocessing and $\Theta(n^3)$ for the edge tests). □

Putting together Theorem 6 and Theorem 14 leads us to our primary result for plane graphs:

Theorem 15 *(Plane graph reconstruction). Let $G = (V, E)$ be a plane graph with n vertices embedded in $\mathbb{R}^2$. Algorithm 1 and Algorithm 2 calculate the vertex locations and edges using $n^2 - n + 3$ different directional augmented persistence diagrams in $\Theta(n^2 T_G)$ time, where $\Theta(T_G)$ is the time complexity of computing a single diagram.*

Proof. By Theorem 6, Algorithm 1 reconstructs the vertices V using three APDs in $\Theta(n \log n + T_G)$ time. By Theorem 14, Algorithm 2 reconstructs the edges E with $n^2 - n$ directional augmented persistence diagrams in $\Theta(n^2 T_G)$ time. Thus, we can reconstruct all vertex locations and edges of G using $n^2 - n + 3$ APDs in $\Theta(n^2 T_G)$ time. □

5.2. Edge reconstruction in $\mathbb{R}^d$

We can also reconstruct edges of graphs embedded in higher dimensions. We can form a higher-dimensional version of the bow tie, referred to as a *wedge*, which is the symmetric difference of two $(d-1)$-dimensional hyperplanes.

[3] Note that the naïve approach would be to sort about each vertex independently, which would take $\Theta(n^2 \log n)$ time, but the results of [26] improve this to $\Theta(n^2)$. The lemmas in [26] use big-O notation, but the presented algorithm is actually asymptotically tight.

Algorithm 2 Reconstruct edges.

Input: Oracle $\mathcal{O}$ for an unknown graph $G = (V, E) \subset \mathbb{R}^2$; the vertex set V
Output: the edge set E
1: $E \leftarrow \emptyset$
2: $c \leftarrow$ use [26] to compute cyclic orderings for all vertices in V stored as lists, indexed by $v \in V$
3: **for** $v \in V$ **do**
4: $\quad \ell[v] \leftarrow$ cyclic ordering of lines though v, computed from $c[v]$
5: $\quad \theta(v) \leftarrow$ minimum angle between any two lines in $\ell[v]$
6: **end for**
7: $\theta = \frac{1}{2} \min_{v \in V} \theta(v)$
8: **for** $(v, v') \in V \times V, v \neq v'$ **do**
9: $\quad s \leftarrow$ unit vector perpendicular to $(v' - v)$
10: $\quad s_1 \leftarrow s$ rotated by θ
11: $\quad s_2 \leftarrow s$ rotated by $-\theta$
12: $\quad$ Consult $\mathcal{O}$ to obtain $D(s_1)$ and $D(s_2)$ restricted to zero- and one-dimensional points
13: $\quad$ Compute $\textsc{Indeg}(v, s_1)$ from $D(s_1)$
14: $\quad$ Compute $\textsc{Indeg}(v, s_2)$ from $D(s_2)$
15: $\quad$ **if** $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| == 1$ **then**
16: $\quad\quad$ Add (v, v') to E
17: $\quad$ **end if**
18: **end for**
19: **return** E

Theorem 16 *(Edge reconstruction in higher dimensions). Let $G = (V, E)$ be a straight-line embedded graph in $\mathbb{R}^d$ for some $d > 1$. If V is known, then we can compute E using $n^2 - n$ directional augmented persistence diagrams in $O(n^2 T_G + n^4)$ time, where $\Theta(T_G)$ is the time complexity of computing a single diagram.*

Proof. Let $\mathbb{P}_{e_1,e_2}$ be the subspace of $\mathbb{R}^d$ spanned by e_1 and e_2. Let $\pi : \mathbb{R}^d \to \mathbb{R}^2$ be defined by $\pi(x_1, x_2, \ldots, x_d) = (x_1, x_2)$; in other words, π is the projection to $\mathbb{P}_{e_1,e_2}$. Let $V_* = \pi(V)$ and note that by the General Position Assumption, no three vertices are collinear in V_* and no two points share any coordinate values. Since the vertex set V_* lies in a two-dimensional plane, we can use Lines 1–7 of Algorithm 2 to compute an angle θ for V_*.

Let $v, v' \in V$. We use a method similar to the one defined in Lines 9–11 of Algorithm 2 to find an appropriate wedge to test whether an edge between v and v' exists. We define s_1 and s_2 such that the lines $\ell_1(\pi(v))$ and $\ell_2(\pi(v))$, which are perpendicular to s_1 and s_2 and go through $\pi(v)$, define a bow tie at $\pi(v)$ that isolates the potential edge $(\pi(v), \pi(v'))$. This bow tie extends to a wedge in $\mathbb{R}^d$ by replacing the lines with hyperplanes that intersect $\mathbb{P}_{e_1,e_2}$ orthogonally; specifically, the line $a_1x_1 + a_2x_2 = h$ in $\mathbb{P}_{e_1,e_2}$ corresponds to the $(d-1)$-dimensional hyperplane, $a_1x_1 + a_2x_2 + 0x_3 + \ldots + 0x_d = h$ in $\mathbb{R}^d$. Let $\hat{s}_1$ and $\hat{s}_2$ be directions in $\mathbb{S}^{d-1}$ that define this wedge. Because π is an orthogonal projection and points in V satisfy the General Position Assumption, v' must be the only vertex in this wedge, thus it isolates the edge (v, v'), should it exist.

We then compute the indegrees of v with respect to $\hat{s}_1$ and $\hat{s}_2$, just as we did in the two-dimensional case in Lines 13 and 14 of Algorithm 2. However, we note that since our dimension may be greater than two, Lemma 11 states that this step takes $O(n^2)$ time for each indegree computation. We perform indegree checks on $\binom{n}{2}$ pairs of vertices. Finally, by Lemma 13, we test for an edge by determining if the difference between the indegrees is one using the same technique as Lines 15-16 of Algorithm 2.

The runtime for computing the indegree for all $\binom{n}{2}$ pairs of vertices and reconstructing the edges is $O(n^2 T_G + n^4)$. Similar to Theorem 14, the reconstruction uses two diagrams for each pair of vertices, which is $n^2 - n$ diagrams. □

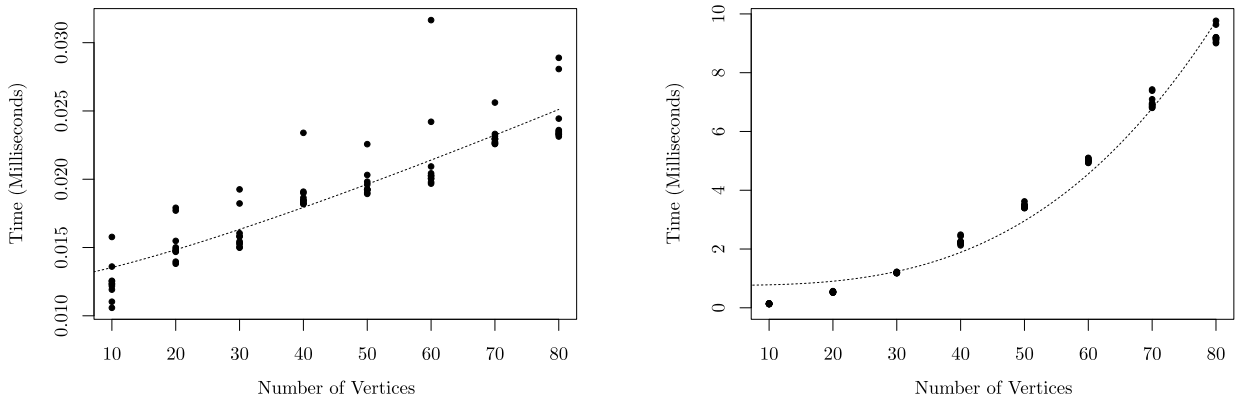
Putting together Theorem 9 and Theorem 16, we obtain a graph reconstruction algorithm for graphs embedded with straight-line edges in $\mathbb{R}^d$:

Theorem 17 *(Generalized graph reconstruction). If $G = (V, E)$ is a straight-line embedded graph in $\mathbb{R}^d$ for $d > 1$, then V, E, and the embedding of G can be computed using augmented persistence diagrams from $n^2 - n + d + 1$ different directions in time $O(dn^{d+1} + n^4 + (d + n^2)T_G)$, where $\Theta(T_G)$ is the time complexity of computing a single diagram.*

6. Experiments

In this section, we empirically validate both the correctness of our algorithm and the runtime of various steps of the algorithm. Furthermore, we explore the runtime of checking for the presence of edges for a fixed vertex set as the number of edges varies. We conclude with an analysis of the probability of having 'bad input;' that is, input with small angles. Timings were taken on a MacBook Pro with a 2.3 GHz Intel Core i5 processor, 8GB RAM, running macOS Mojave 10.14.6. Our implementations were written in C++, compiled with the LLVM toolchain version 10.0.1. Our code is publicly available and experiments were run using the reconstruction library at git commit hash *d9fd610*.[4]

[4] The library is available at https://github.com/compTAG/reconstruction.



(a) Vertex Reconstruction (b) Edge Reconstruction

Fig. 5. Experimental times for reconstructing plane graphs. For each value of n, we record the vertex and edge reconstruction times for 10 random graphs and compare to the theoretical runtimes for vertex reconstruction (Theorem 6) and edge reconstruction (Theorem 14). The experimental timings agree with the theoretical runtimes.

Implementation To complement the theoretical results of this paper, we have written the reconstruction code in C++. In the implementation, an oracle is given a direction and returns an augmented persistence diagram. We use the Dionysus 2.0 library [27] for computing the augmented persistence diagram. We store the computed birth-death pairs in two lists, one for zero-dimension points (representing connected components) and one for one-dimensional points (representing loops).

We note that we have made some changes from the graph reconstruction algorithm presented in the previous sections. In particular, instead of using [26] to compute a cyclic ordering of all vertices in $\Theta(n^2)$, in Algorithm 2 Line 2, we implemented a naïve approach of ordering around each vertex independently in $O(n^2 \log n)$.

Data Our experimental data is a set of random plane graphs embedded in $\mathbb{R}^2$. For $n \in \mathbb{Z}^+$, we randomly generate n points from the uniform distribution over the unit square. Next, we compute the Delaunay triangulation using the SciPy library [21] in Python. This random triangulation is similar to those of [4,6,11], but uses a binomial point process with a uniform density rather than a Poisson point process. Finally, we arrive at our random graph by setting a percentage α of edges E from the triangulation to keep and delete edges until $\alpha \cdot |E|$ remain. The result is a random subgraph of a Delaunay triangulation.

Timing For each experiment about timing, we subtract out the time spent computing diagrams as to capture only the time spent in the algorithm. In addition, we run each experiment five times and report the average of the runs.

6.1. Experimental runtimes

Our first experiment validates the theoretical runtime of the plane graph reconstruction algorithm. We fix the percent of edges at $\alpha = 10\%$ and vary the number of vertices $n \in \{10, 20, \ldots, 80\}$. For each value of n, we reconstruct ten random graphs as described in Paragraph 6.Data. We track the time spent in vertex and edge reconstruction and record the timings as described in Paragraph 6.Timing.

In Fig. 5a and Fig. 5b, we plot the time for reconstructing vertices and edges versus n, respectively. Times are measured in milliseconds and each mark is the mean for five runs on each random graph. In Fig. 5a we fit $t = \beta_0 + \beta_1 n \log n$ to the experimental data, which had a RMSE of 0.0019 ms. The figure agrees with Theorem 6, which showed that the vertex reconstruction runtime is $O(n \log n)$ (ignoring the time for computing diagrams). Fig. 5b must be interpreted with a little more care. Recall that Theorem 14 showed that the edge reconstruction runtime is $\Theta(n^2 T_G)$. However, the runtime of the loop in Lines 8–18 of Algorithm 2 is $O(n^3)$, and in the experiments, we subtract the time for computing diagrams. Thus, we expect the time to be upper-bounded by n^3. We fit the curve $t = \beta_0 + \beta_1 n^3$ to the experimental data, which had a RMSE of 0.45 ms. The figure agrees with Theorem 14.

6.2. Effect of edges on runtime

Our second experiment investigates the effect of edge density on our algorithm. Note that while our runtimes are expressed in the number of vertices, investigating the edges adds an additional insight. As in the previous section, each graph is generated as described in Paragraph 6.Data and we track the time spent in the vertex and edge reconstruction as described in Paragraph 6.Timing. Specifically, for a fixed Delaunay triangulation on a vertex set of size n, we vary the percent of edges α from the Delaunay triangulation. We use $\alpha \in \{10, 20, \ldots, 100\}$ and $n \in \{10, 25, 50, 100\}$. In Fig. 6a and Fig. 6b, each value of n is represented by a different line in the plot.

In the first part of this experiment, we investigate the effect of edge density on vertex reconstruction times. In Fig. 6b, we see a constant relationship in the average runtime per vertex as a function of the percent of edges. To confirm, we fit

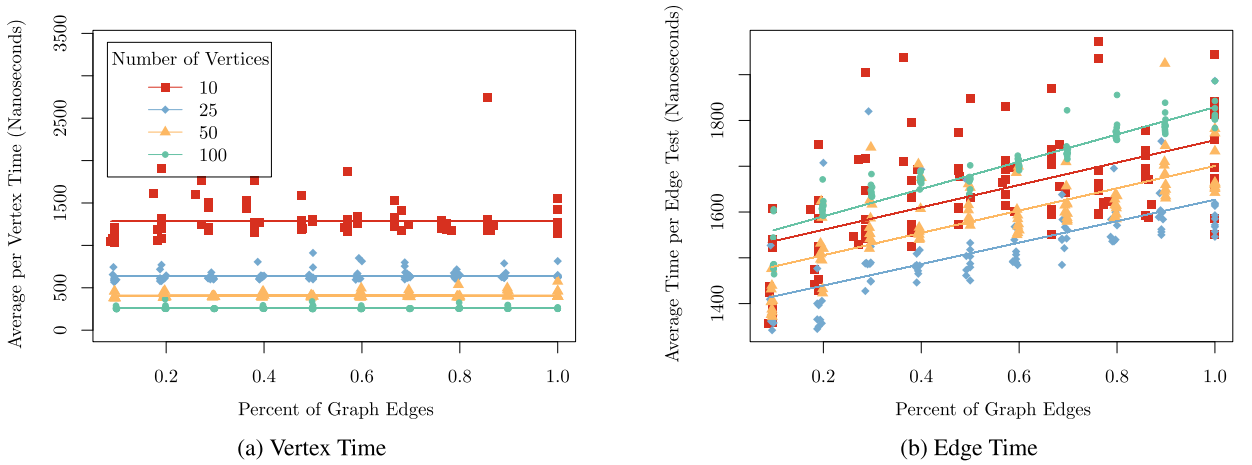


Fig. 6. Experimental reconstruction times for plane graphs as the number of edges increases. For each n, we record the vertex and edge reconstruction times for 10 random graphs as we increase the percent of edges from a Delaunay triangulation. *Left:* From (Theorem 6), we expect that the vertex reconstruction time is independent of the number of edges. *Right:* From (Theorem 14), for a graph with n vertices and m edges, we conduct $n^2 - n$ edge tests on a given graph. Each edge test compares two augmented persistence diagrams of size $O(n+m)$. Thus, we expect that the time grows linearly with respect to the number (and hence the percent) of edges. In both plots, the experimental timings agree with the theoretical run times.

the curve $t = \beta_0$, with RMSE 218.5 ns for $n = 10$, 62.4 ns for $n = 25$, 33.4 ns for $n = 50$, and 18.2 ns for $n = 100$. Thus, as predicted by the analysis of Theorem 6, the runtime of the vertex reconstruction algorithm is independent of the number of edges.

In the second part of this experiment, we focus on the reconstruction time of edges in the graph. In Fig. 6b, we see a linear relationship in the average runtime per edge as a function of the percent of edges. To confirm, we fit the curve $t = \beta_0 + \beta_1 n$, with RMSE 108.3 ns for $n = 10$, 72.9 ns for $n = 25$, 61.9 ns for $n = 50$, and 33.4 ns for $n = 100$. The linear growth is somewhat unexpected as the number of edges does not appear in the analysis of Theorem 14. But, recall that the edge reconstruction algorithm computes indegree. By Lemma 11, the computation takes time proportional to the size of the augmented persistence diagrams (in dimensions zero and one combined). Moreover, the size of the one dimensional diagram grows linearly with respect to the number of edges. Thus, when α is small, as would be the case in sparse graphs, we see fewer cycles, which makes $D_1(\cdot)$ smaller. This results in a small improvement in runtime.

6.3. Minimum angle

Very small angles cause numerical issues, which is a problem that we encountered in the above experiments. In order to mitigate the issues, our code has an assertion that bow tie half-angles are at least 10^{-6} radians. We observed that many of the experiments with over 50 vertices were failing this assertion. In addition, a bounded angle assumption was used in [10]. Thus, we investigate the probability of encountering small angles, both with a back-of-the-envelope calculation and empirical observations.

Back-of-the-envelope calculation Let a, b, and c be three points sampled i.i.d. from the uniform distribution on the unit square. We consider $\angle abc$. Without loss of generality, we can translate and rotate the points such that b is at the origin, and c is on the positive x-axis. Then, consider the line ba. If a was randomly chosen, then any angle it makes with the positive x-axis (ba) is equally likely, so

$$\mathbb{P}(\angle abc < 10^{-6}) = \frac{2 \cdot 10^{-6}}{2\pi} = \frac{10^{-6}}{\pi} \approx 3.18e{-}7.$$

Furthermore, let S_n be a set of n (different) points sampled i.i.d. from the uniform distribution on the unit square. Notice that we can make $\binom{n}{3} = n(n-1)(n-2)$ angles defined by the points in S_n. Let A_n be the event that there exists an angle less than 10^{-6} in S_n. Assuming independence of angles in S, we have:

$$\mathbb{P}(A_n) = 1 - \left(1 - \frac{10^{-6}}{\pi}\right)^{n(n-1)(n-2)}.$$

We observe that $\mathbb{P}(A_n) > 5\%$ when $n \geq 56$. In the above runtime experiments, it is not surprising that we see the assertion failing. In this calculation, we assume that all angles are independent, so $\mathbb{P}(A_n)$ is an overestimation of the true probability. We conjecture:

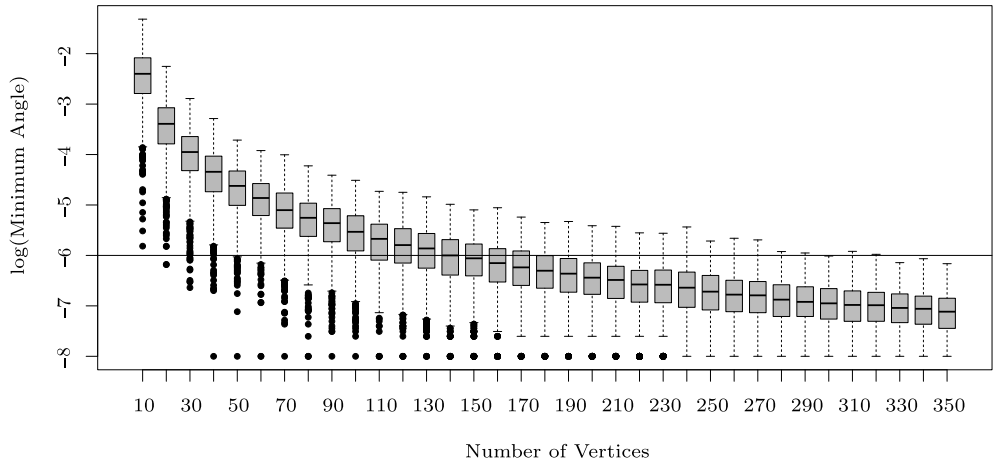


Fig. 7. Box plots for distributions of minimum angles for 100 random graphs with n vertices. Any graph that has a minimum angle less than 10^{-6}, denoted by the horizontal line, encounters numerical errors from the reconstruction algorithm. As the number of vertices increases, the number of graphs with a minimum angle of 10^{-6}, and thus encountering numerical errors, increases.

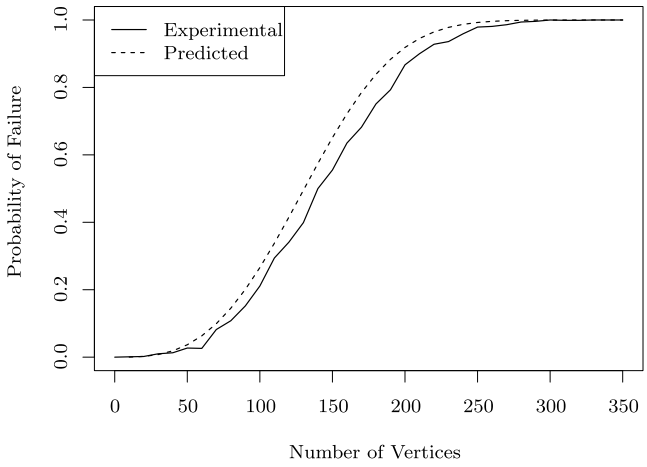


Fig. 8. Probability of failure (encountering an angle less than 10^{-6}) from experimental data (solid line) and the theoretical model (dashed line). The theoretical model provides an upper bound for the experimental probability of finding a small angle.

Conjecture 18 *(Probability of encountering small angles). Let $G = (V, E)$ be a randomly generated plane graph embedded in $\mathbb{R}^2$. If $|V| > 55$, then with probability at least 5%, the minimum angle as described in Theorem 14 is less than 10^{-6} radians.*

Our final experiment is an investigation of the minimum angle of random point sets. This is motivated by understanding the frequency of small angles, and the extent to which our assertion and the assumption of [10] may or may not be limiting in practice.

Empirical observations This experiment illustrates how the bounded angle assumption is actually quite limiting, and that small angles appear with high probability as the number of vertices increases.

We generated random sets of vertices of size $n \in \{10, 20, \ldots, 350\}$, and measured the minimum angle between all triples of vertices. In Fig. 7, we show the box plots of the minimum angles for 1000 random graphs for each n. In Fig. 8, we show the probability of a graph having a minimum angle less than 10^{-6} as predicted by the theoretical model and by the experimental data. We notice at $n = 20$, we begin to see minimum angles of 10^{-6} radians appear in our random graphs. At $n = 70$, the number of random graphs with a minimum angle of 10^{-6} radians or less becomes even more substantial, with 8% of the angles less than 10^{-6} experimentally. This experiment suggests that for plane graphs with $n \geq 50$, our algorithm may encounter numerical errors due to small angles. And, as n grows into the hundreds, the probability gets close to one. For the range of n values we tested experimentally, we see—as we expected—that the theoretical back-of-the-envelope calculation upper bounds the experimental probability of finding a small angle. As we mentioned above, the differences in Fig. 8 are due to the fact that the predicted probability is done assuming independence of angle.

7. Discussion

In this paper, we address the problem of reconstructing embedded graphs using persistence diagrams through the following three main contributions.

1. We provide the first deterministic algorithm for reconstructing a plane graph using (directional, augmented) persistence diagrams. For a graph with n vertices embedded in $\mathbb{R}^2$, this algorithm uses $n^2 - n + 3$ directional APDs in $\Theta(n^2 T_G)$ time.
2. We extend the algorithm for reconstructing plane graphs to reconstructing graphs embedded in $\mathbb{R}^d$ for $d \geq 2$. This algorithm uses $n^2 - n + d + 1$ directional APDs in $O(dn^{d+1} + n^4 + (d + n^2)T_G)$ time.
3. We experimentally validate the correctness and time complexity of our algorithm for reconstructing plane graphs by implementing the algorithm and testing the implementation on randomly generated plane graphs. We found that the empirical probability of encountering numerical issues are likely to occur even for graphs with only 20–50 vertices.

We identify a number of avenues for future work. We recently extended the methods given here to reconstruct geometric simplicial complexes (not just embedded graphs) using APDs; see the preprint [16]. It seems possible, however, that the algorithm for computing the generalization of indegree could be improved. Additionally, it would be interesting to explore reconstruction problems with other topological descriptors, such as Euler characteristic curves (ECCs). Indeed, in [15], we identified some of the challenges in reconstructing with ECCs. Thus, algorithmic reconstruction with ECCs is still largely unexplored. In other extensions of this work, one could consider variants of the inverse problem and classify simplicial complexes, up to some topological invariant (e.g., homotopy type) instead of up to geometric representation. One can hope that computing such a classification would require fewer persistence diagrams than is required for a complete geometric reconstruction.

Finally, our algorithms rely on *augmented* persistence diagrams, but we conjecture that the corresponding persistence diagrams (without the explicitly stored on-diagonal points) can be used to sufficiently differentiate one shape from another. If proven, this conjecture would close the gap between our theoretical results for APDs and the use of PDs in practice, such as in [20].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

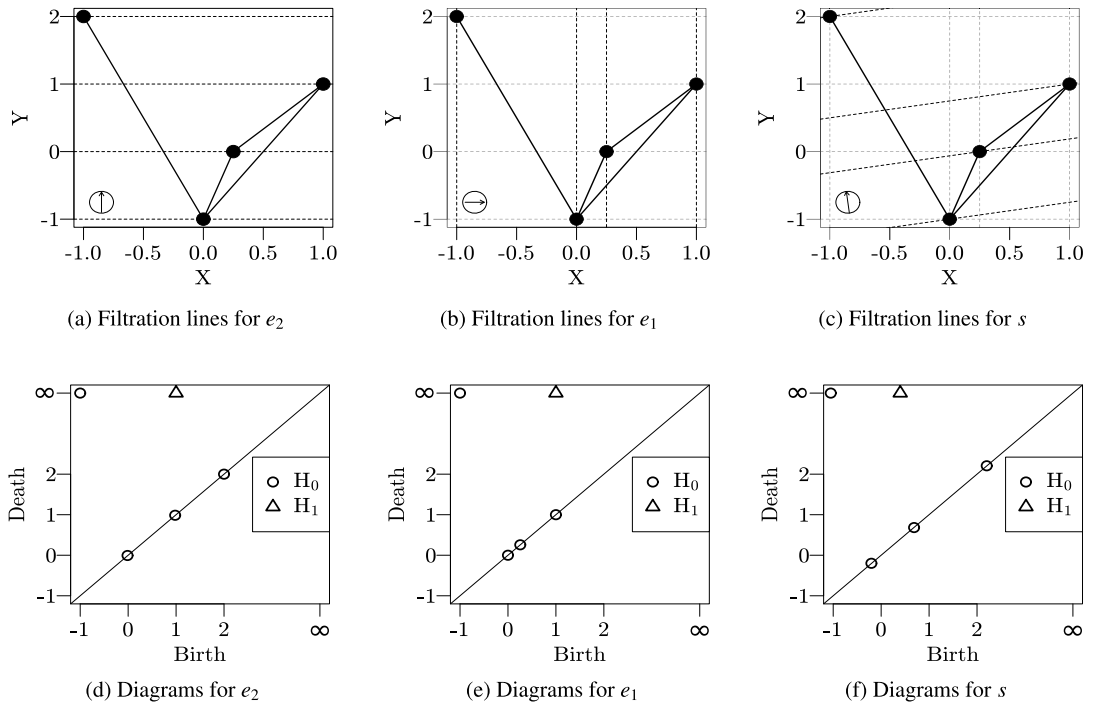
This material is based upon work supported by the National Science Foundation under the following grants: CCF 1618605 (BTF, SM, RM), DBI 1661530 (DLM, LW), DGE 1649608 (RLB, AS), DMS 1664858 (RLB, BTF, AS, JS), and DMS 1854336 (BTF). Additionally, RM thanks the Undergraduate Scholars Program for both travel funding and undergraduate research grants. All authors thank the CompTaG club at Montana State University. In particular, we thank Brad McCoy for checking the examples for correctness. Finally, we thank the reviewers for their careful reading and constructive comments that helped improve this paper.

Appendix A. Demonstration of plane graph reconstruction

We give an example of reconstructing a plane graph embedded in $\mathbb{R}^2$. Consider the complex, G, given in Fig. A.9. The vertices of G are $V = \{(-1, 2), (0, -1), (0.25, 0), (1, 1)\}$, and edges are given by the following pairs of vertices, $E = \{((-1, 2), (0, -1)), ((0, -1), (0.25, 0)), ((0, -1), (1, 1)), ((0.25, 0), (1, 1))\}$.

Vertex reconstruction We find vertex locations using the algorithm described in Section 4. Note, that in this example, $n = 4$. Using the persistence diagrams from height filtrations in directions $e_1 = (1, 0)$ and $e_2 = (0, 1)$, we construct the set of lines $\mathbb{L}(e_1, V) = \{(-1, y), (0, y), (0.25, y), (1, y) \mid y \in \mathbb{R}\}$ and $\mathbb{L}(e_2, V) = \{(x, 2), (x, -1), (x, 0), (x, 1) \mid x \in \mathbb{R}\}$ as shown in Fig. A.9c. The set $\mathbb{L}(e_1, V) \cup \mathbb{L}(e_2, V)$ of $2n = 8$ lines has $n^2 = 16$ possible locations for the vertices at the intersections in A. We show these filtration lines and intersections in Fig. A.9b.

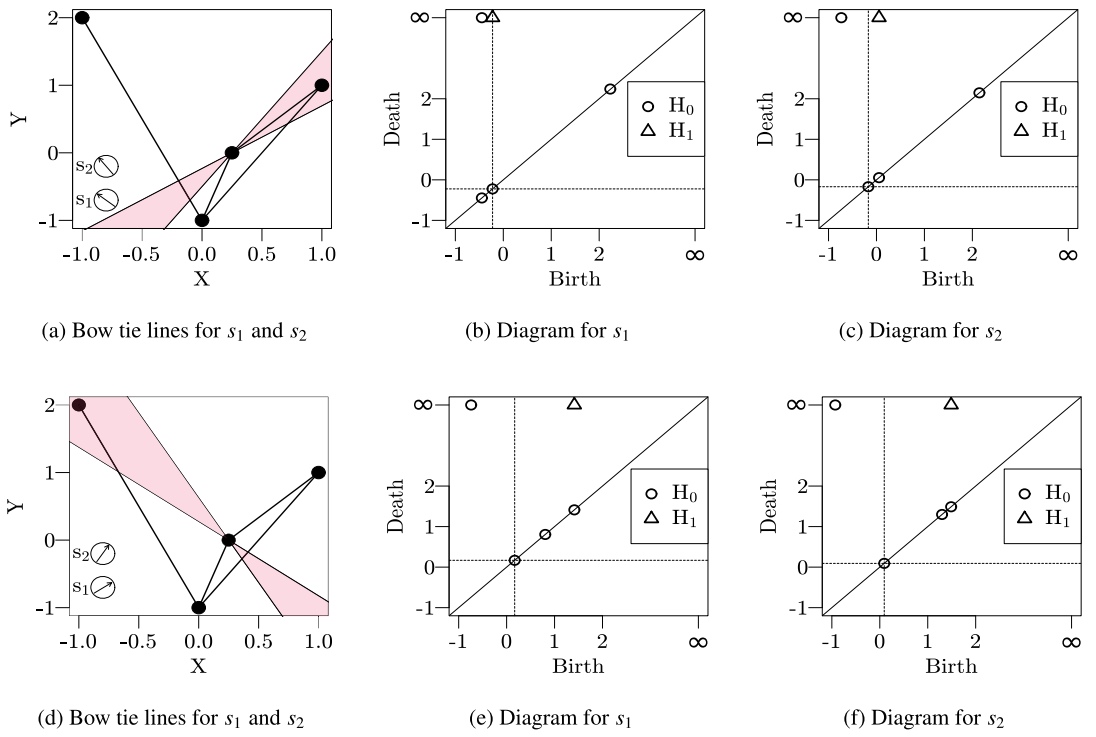
We compute the third direction, s, using the algorithm outlined in Theorem 6. Recall, that we need to find the maximum width between two lines in $\mathbb{L}(e_1, V)$, denoted by w, and smallest height between two adjacent lines in $\mathbb{L}(e_2, V)$, denoted by h. In our example, $w = 1 - (-1) = 2$ and $h = 2 - 1 = 1$. Then, we use the rectangle of width, w, and height, h, to choose a direction s. We pick s to be a unit vector perpendicular to $[w, h/2]$. In particular, we compute $s = (-0.243, 0.970) \in \mathbb{S}^1$. Then, the four three-way intersections in $\mathbb{L}(e_1, V) \cup \mathbb{L}(e_2, V) \cup \mathbb{L}(s, V)$ identify all Cartesian coordinates of the vertices in the graph.



(a) Filtration lines for e_2 (b) Filtration lines for e_1 (c) Filtration lines for s

(d) Diagrams for e_2 (e) Diagrams for e_1 (f) Diagrams for s

Fig. A.9. Example of vertex reconstruction from three directions, e_2, e_1 and s with corresponding persistence diagrams built for height filtrations from these directions. The filtration lines are the dotted lines superimposed over the complex.



(a) Bow tie lines for s_1 and s_2 (b) Diagram for s_1 (c) Diagram for s_2

(d) Bow tie lines for s_1 and s_2 (e) Diagram for s_1 (f) Diagram for s_2

Fig. A.10. Example of edge reconstruction for two edges. The first edge (top row) exists while the second edge (bottom row) does not. The bow tie is given on the left while the persistence diagrams $D_0(s_1)$ and $D_1(s_1)$ are given in the middle and the persistence diagrams $D_0(s_2)$ and $D_1(s_2)$ are given on the right. The dotted lines indicate $v \cdot s_1$ and $v \cdot s_2$ in diagrams for s_1 and s_2 respectively.

Edge reconstruction Next, we reconstruct all edges of G as described in Section 5. In order to do so, we first compute the bow tie half-angle denoted by θ. For each vertex $v \in V$, we consider the cyclic ordering of the points in $V \setminus \{v\}$ around v and compute $\theta(v)$, which is the minimum angle between all adjacent pairs of lines through v. Noting that the vertex set is $V = \{(-1, 2), (0, -1), (0.25, 0), (1, 1)\}$, we find $\theta(v)$ to be approximately 0.237, 0.219, 0.399, and 0.180 radians, respectively. Then, we fix θ so that $\theta = \min_v \theta(v)/2 = 0.09$.

Now, for each of the $n(n-1)/2$ pairs of vertices $(v, v') \in V^2$, we construct a bow tie B and then use this bow tie to determine whether an edge exists between the two vertices. We go through two examples: one for a pair of vertices that does have an edge between them, $((0.25, 0), (1, 1))$, and one for a pair that does not, $((0.25, 0), (-1, 2))$. We show the corresponding bow ties and diagrams for these two examples in Fig. A.10. First, consider the pair $v = (0.25, 0)$ and $v' = (1, 1)$. To construct the bow tie at v that contains v', we first find a unit vector perpendicular to the vector that points from v to v'. Here, our unit vector is $s = (-0.8, 0.6)$. Now, we find s_1, s_2 such that the bow tie at v has half-angle θ with s. We choose $s_1 = (-0.851, 0.526)$ and $s_2 = (-0.743, 0.669)$. By Lemma 11, we use the persistence diagrams from these two directions to compute $\textsc{Indeg}(v, s_1)$ and $\textsc{Indeg}(v, s_2)$. We observe that $D_0(s_1)$ contains exactly one birth-death pair (x, y) such that $y = v \cdot s_1$ and $D_1(s_1)$ has one birth-death pair such that $x = v \cdot s_1$. Thus, $\textsc{Indeg}(v, s_1) = 2$. On the other hand, $D_0(s_2)$ contains exactly one birth-death pair (x, y) such that $y = v \cdot s_2$, but $D_1(s_2)$ contains no birth-death pair such that $x = v \cdot s_2$. So $\textsc{Indeg}(v, s_2) = 1$. Now, since $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1$, we know that $(v, v') \in E$, by Lemma 13.

For the second edge example, consider the pair of vertices $v = (0.25, 0)$ and $v' = (-1, 2)$. Again, we construct the bow tie at v containing v', by finding a unit vector perpendicular to the vector, $(v' - v)$. We choose this to be $s = (0.848, 0.530)$. Then, the s_1 and s_2 that form angle θ with s are $s_1 = (0.892, 0.452)$ and $s_2 = (0.797, 0.604)$. Again by Lemma 11, we examine the zero- and one-dimensional persistence diagrams from these two directions to compute the indegree from each direction for vertex v. In $D_0(s_1)$, we have one pair (x, y) that dies at $y = v \cdot s_1$, but in $D_1(s_1)$, no pair is born at $x = v \cdot s_1$. So $\textsc{Indeg}(v, s_1) = 1$. We see the exact same for s_2, which means that $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 0$. Since Lemma 13 tells us that we have an edge between v and v' only if the absolute value of the difference of indegrees is one, we know that there is no edge between vertices $(0.25, 0)$ and $(-1, 2)$.

In order to reconstruct all edges, we perform the same computations for all pairs of vertices. After doing this, we obtain the desired plane graph as shown in Fig. A.9.

References

[1] M. Ahmed, S. Karagiorgou, D. Pfoser, C. Wenk, Map construction algorithms, in: Map Construction Algorithms, Springer, 2015, pp. 1–14.
[2] M. Ahmed, C. Wenk, Constructing street networks from GPS trajectories, in: European Symposium on Algorithms - ESA '12, Springer, 2012, pp. 60–71.
[3] R.L. Belton, B.T. Fasy, R. Mertz, S. Micka, D.L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, L. Williams, Learning simplicial complexes from persistence diagrams, in: Canadian Conference on Computational Geometry - CCCG '18, 2018, Also available at arXiv:1805.10716.
[4] B. Boots, A. Okabe, K. Sugihara, Spatial tessellations, in: Geographical Information Systems, vol. 1, 1999, pp. 503–526.
[5] E.W. Chambers, T. Ju, D. Letscher, M. Li, C.N. Topp, Y. Yan, Some heuristics for the homological simplification problem, in: Canadian Conference on Computational Geometry - CCCG '18, 2018.
[6] N. Chenavier, O. Devillers, Stretch factor in a planar Poisson–Delaunay triangulation with a large intensity, Adv. Appl. Probab. 50 (2018) 35–56.
[7] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, Stability of persistence diagrams, Discrete Comput. Geom. 37 (2007) 103–120.
[8] D. Cohen-Steiner, H. Edelsbrunner, D. Morozov, Vines and vineyards by updating persistence in linear time, in: Proceedings of the Twenty-Second Annual Symposium on Computational Geometry, ACM, 2006, pp. 119–126.
[9] L. Crawford, A. Monod, A.X. Chen, S. Mukherjee, R. Rabadán, Predicting clinical outcomes in glioblastoma: an application of topological and functional data analysis, J. Am. Stat. Assoc. (2019) 1–12.
[10] J. Curry, S. Mukherjee, K. Turner, How many directions determine a shape and other sufficiency results for two topological transforms, arXiv:1805.09782, 2018.
[11] O. Devillers, L. Noizet, Walking in a planar Poisson-Delaunay triangulation: shortcuts in the Voronoi path, Int. J. Comput. Geom. Appl. (2018) 1–12.
[12] T.K. Dey, J. Wang, Y. Wang, Graph reconstruction by discrete Morse theory, in: 34th International Symposium on Computational Geometry, vol. 99, SoCG 2018, 2018, 31.
[13] H. Edelsbrunner, J. Harer, Computational Topology: An Introduction, American Mathematical Society, 2010.
[14] H. Edelsbrunner, D. Letscher, A. Zomorodian, Topological persistence and simplification, Discrete Comput. Geom. (2002) 511–533.
[15] B.T. Fasy, S. Micka, D.L. Millman, A. Schenfisch, L. Williams, Challenges in reconstructing shapes from Euler characteristic curves, in: Fall Workshop on Computational Geometry - FWCG '18, 2018.
[16] B.T. Fasy, S. Micka, D.L. Millman, A. Schenfisch, L. Williams, Persistence diagrams for efficient simplicial complex reconstruction, arXiv:1912.12759, 2019.
[17] X. Ge, I.I. Safa, M. Belkin, Y. Wang, Data skeletonization via Reeb graphs, in: Advances in Neural Information Processing Systems - NIPS '11, 2011, pp. 837–845.
[18] R. Ghrist, R. Levanger, H. Mai, Persistent homology and Euler integral transforms, J. Appl. Comput. Topol. 2 (2018) 55–60.
[19] C. Giusti, E. Pastalkova, C. Curto, V. Itskov, Clique topology reveals intrinsic geometric structure in neural correlations, Proc. Natl. Acad. Sci. 112 (2015) 13455–13460.
[20] C. Hofer, R. Kwitt, M. Niethammer, A. Uhl, Deep learning with topological signatures, in: Advances in Neural Information Processing Systems, 2017, pp. 1634–1644.
[21] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: open source scientific tools for Python, http://www.scipy.org/, 2001.
[22] S. Karagiorgou, D. Pfoser, On vehicle tracking data-based road network generation, in: Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL '12, ACM, 2012, pp. 89–98.
[23] B. Kégl, A. Krzyzak, T. Linder, K. Zeger, Learning and design of principal curves, IEEE Trans. Pattern Anal. Mach. Intell. 22 (2000) 281–297.
[24] Y. Lee, S.D. Barthel, P. Dłotko, S.M. Moosavi, K. Hess, B. Smit, Quantifying similarity of pore-geometry in nanoporous materials, Nat. Commun. 8 (2017) 15396.
[25] S. Micka, Algorithms to Find Topological Descriptors for Shape Reconstruction and How to Search Them, Ph.D. thesis, Montana State University, 2020.
[26] D.L. Millman, V. Verma, A slow algorithm for computing the Gabriel graph with double precision, in: Canadian Conference on Computational Geometry - CCCG '11, 2011.

[27] D. Morozov, Dionysus, a c++ library for computing persistent homology, http://mrzv.org/software/dionysus2/, 2007.
[28] S. Oudot, E. Solomon, Inverse problems in topological persistence, arXiv:1810.10813, 2018.
[29] A.H. Rizvi, P.G. Camara, E.K. Kandror, T.J. Roberts, I. Schieren, T. Maniatis, R. Rabadan, Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development, Nat. Biotechnol. 35 (2017) 551.
[30] K. Turner, S. Mukherjee, D.M. Boyer, Persistent homology transform for modeling shapes and surfaces, Inf. Inference 3 (2014) 310–344.
[31] Y. Zheng, S. Gu, H. Edelsbrunner, C. Tomasi, P. Benfey, Detailed reconstruction of 3D plant root shape, in: Proceedings of the IEEE International Conference on Computer Vision, 2011, pp. 2026–2033.