

Computing Height Persistence and Homology Generators in \mathbb{R}^3 Efficiently

Tamal K. Dey*

Abstract

Recently it has been shown that computing the dimension of the first homology group $H_1(\mathcal{K})$ of a simplicial 2-complex \mathcal{K} embedded linearly in \mathbb{R}^4 is as hard as computing the rank of a sparse 0 – 1 matrix. This puts a major roadblock to computing persistence and a homology basis (generators) for complexes embedded in \mathbb{R}^4 and beyond in less than quadratic or even near-quadratic time. But, what about dimension three? It is known that when \mathcal{K} is a graph or a surface with n simplices linearly embedded in \mathbb{R}^3 , the persistence for piecewise linear functions on \mathcal{K} can be computed in $O(n \log n)$ time and a set of generators of total size k can be computed in $O(n + k)$ time. However, the question for general simplicial complexes \mathcal{K} linearly embedded in \mathbb{R}^3 is not completely settled. No algorithm with a complexity better than that of the matrix multiplication is known for this important case. We show that the persistence for *height functions* on such complexes, hence called *height persistence*, can be computed in $O(n \log n)$ time. This allows us to compute a basis (generators) of $H_i(\mathcal{K})$, $i = 1, 2$, in $O(n \log n + k)$ time where k is the size of the output. This improves significantly the current best bound of $O(n^\omega)$, ω being the exponent of matrix multiplication. We achieve these improved bounds by leveraging recent results on zigzag persistence in computational topology, new observations about Reeb graphs, and some efficient geometric data structures.

*Department of Computer Science and Engineering, The Ohio State University. tamaldey@cse.ohio-state.edu

1 Introduction

Topological persistence for a filtration or a piecewise linear function on a simplicial complex \mathcal{K} is known to be computable in $O(n^\omega)$ time [15] where n is the number of simplices in \mathcal{K} and $\omega < 2.373$ is the exponent of matrix multiplication. The question regarding the lower bound on its computation was largely open until Edelsbrunner and Parsa [12] showed that computing the rank of the first homology group $H_1(\mathcal{K})$ of a simplicial complex \mathcal{K} linearly embedded in \mathbb{R}^4 is as hard as the rank computation of a sparse $n \times n$ 0-1 matrix. The current upper bound for matrix rank computation is super-quadratic [7] and lowering it is a well-recognized hard problem. Consequently, computing the dimension of the homology groups and hence the topological persistence for functions on general complexes in better than super-quadratic time is difficult, if not impossible. But, what about the special cases that are still interesting? The complexes embedded in three dimensions which arise in plenty of applications present such cases.

It is easy to see that the Betti numbers β_i , the rank of the i th homology group $H_i(\mathcal{K})$ defined over a finite field for a simplicial complex \mathcal{K} linearly embedded in \mathbb{R}^3 can be computed in $O(n)$ time. For this, compute β_2 with a walk over the boundaries of the voids, compute β_0 as the number of components of \mathcal{K} , and then compute β_1 from the Euler characteristics of \mathcal{K} obtained as the alternating sum of the numbers of simplices of each dimension. Unfortunately, computation of other topological properties such as persistence and homology generators (basis) for such a complex \mathcal{K} is not known to be any easier than that of matrix multiplication ($O(n^\omega)$ time). In the special case when \mathcal{K} is a graph or a surface, the persistence for a PL function or a filtration on \mathcal{K} can be computed in $O(n \log n)$ time [1, 10]. In this paper, we show that when \mathcal{K} is more general, that is, a simplicial complex linearly embedded in \mathbb{R}^3 , the persistence of a height function on it can be computed in $O(n \log n)$ time. This special type of persistence which we term as the *height persistence* is not as general as the standard persistence. Nonetheless, it provides an avenue to compute a set of *basis cycles* in $O(n \log n + k)$ time where k is the total size of the output. Also, the height persistence provides a window to the topological features of the domain \mathcal{K} , the need for which arises in various applications.

To arrive at our result, we first observe a connection between the standard sublevel-set persistence [11, 17] and the level-set zigzag persistence [6] from the recent work in [3, 4, 6]. Then, with a sweep-plane algorithm that treats the level sets as planar graphs embedded in planes, we compute a *barcode graph* in $O(n \log n)$ time. A barcode is extracted from this graph using a slight but important modification of an algorithm in [1]. The barcode extracted from this graph provides a part of the height persistence. We show that the missing piece can be recovered from the Reeb graph which can be computed again in $O(n \log n)$ time [16]. We make other observations that allow us to extract the actual basis cycles from both pieces in $O(n \log n + k)$ time as claimed.

2 Background

A zigzag diagram of topological spaces is a sequence

$$\mathcal{X} : X_0 \leftrightarrow X_1 \leftrightarrow \cdots \leftrightarrow X_m \tag{2.1}$$

where each X_i is a topological space and each bidirectional arrow ' \leftrightarrow ' is either a forward or a backward continuous map. Applying the homology functor with coefficient in a field κ , we obtain a sequence of vector spaces connected by forward or backward linear maps, also called a zigzag module:

$$H_p(\mathcal{X}) : H_p(X_0) \leftrightarrow H_p(X_1) \leftrightarrow \cdots \leftrightarrow H_p(X_m)$$

When all vector spaces in $H_p(\mathcal{X})$ are finite dimensional, the Gabriel's theorem in quiver theory [13] says that $H_p(\mathcal{X})$ is a direct sum of a finite number of interval modules which are of the form

$$\mathcal{I}_{[b,d]} : I_1 \leftrightarrow I_2 \cdots \leftrightarrow I_m$$

where $I_j = \kappa$ for $b \leq j \leq d$ and $\mathbf{0}$ otherwise with the maps $\kappa \leftarrow \kappa$ and $\kappa \rightarrow \kappa$ being identities. The decomposition $H_p(\mathcal{X}) = \bigoplus_i \mathcal{I}_{[b_i,d_i]}$ provides a *barcode* (set of interval modules) for topological persistence when the topological spaces X_i originate as sublevel or level sets of a real-valued function $f : X \rightarrow \mathbb{R}$ defined on a space X . As shown in [6], classical persistence [11, 17], its extended version [8], and the more general zigzag persistence [6] arise as a consequence of choosing variants of the module \mathcal{X} in 2.1 that are derived from f .

2.1 Standard persistence

Standard persistence [11, 17] is defined by considering the sublevel sets of f , that is, X_i is $f^{-1}(-\infty, a_i]$ for some $a_i \in \mathbb{R}$. These values a_i are taken as the critical values of f so that the barcode captures the evolution of the homology classes of the sub-level sets across the critical values of f , which are defined below precisely.

For an interval $I \subseteq \mathbb{R}$, let $X_I := f^{-1}(I)$ denote the interval set. Following [3, 6], we assume that f is tame. It means that it has finitely many homological critical values $a_1 < a_2 < \cdots < a_m$ so that for each open interval $I \in \{(-\infty, a_1), (a_1, a_2), \dots, (a_{m-1}, a_m), (a_m, \infty)\}$, X_I is homeomorphic to a product space $Y \times I$, with $f(Y) \in I$. This homeomorphism should extend to a continuous function $X_{\bar{I}} \rightarrow Y \times \bar{I}$, with \bar{I} being the closure of I and each interval set X_I should have finitely generated homology groups.

It turns out that the description of the interval modules assumes one more subtle aspect when it comes to describing the standard persistence and zigzag persistence in general. Specifically, the interval modules can be *open* or *closed* at their end points. To elucidate this, consider a set of values $\{s_i\}$ of f interleaving with its critical values:

$$s_0 < a_1 < s_1 < \dots < a_m < s_m$$

Assuming $a_0 = -\infty$ and $a_{m+1} = \infty$, one can write the sub-level sets as $X_{[0,r]} := f^{-1}(-\infty, r]$. For standard persistence, we consider the sublevel set diagram and its corresponding homology module $H_p(\mathcal{SL}(f, X))$ for dimension $p \geq 0$:

$$\begin{aligned} \mathcal{SL}(f, X) : X_{[0,a_1]} &\rightarrow X_{[0,s_1]} \rightarrow X_{[0,a_2]} \cdots \rightarrow X_{[0,s_m]} \rightarrow X_{[0,a_{m+1}]} \\ H_p(\mathcal{SL}(f, X)) : H_p(X_{[0,a_1]}) &\rightarrow H_p(X_{[0,s_1]}) \rightarrow H_p(X_{[0,a_2]}) \cdots \rightarrow H_p(X_{[0,s_m]}) \rightarrow H_p(X_{[0,a_{m+1}]}) \end{aligned}$$

The summand interval modules, or the so called *bars*, for this case has the form $[a_i, s_j]$. This means that a p -dimensional homology class is born at the critical value a_i and it dies at the value s_j . The right end point of s_j is an artifact of our choice of the intermediate value $s_j \in (a_j, a_{j+1})$. Because of our assumption that f is tame, homology classes cannot die in any open interval between the critical values. In fact, they remain alive in the interval (a_j, a_{j+1}) and may die entering the critical value a_{j+1} . To accommodate this fact, we convert each bar $[a_i, s_j]$ of the standard persistence to a bar $[a_i, a_{j+1})$ that is open on the right end point.

One can see that there are two types of bars in the standard persistence, one of the type $[a_i, a_j)$, $j \neq m + 1$, which is bounded (finite) on the right, and the other of the type $[a_i, \infty)$ which is unbounded (infinite) on the right. The unbounded bars represent the essential homology classes since $H_p(X) \cong \bigoplus_i [a_i, \infty)$. The work of [3, 4, 6] implies that both types of bars of the standard persistence can be recovered from those of the level set zigzag persistence as described next. This observation leads to an efficient algorithm for computing the standard persistence in \mathbb{R}^3 .

2.2 Level set zigzag

In level set zigzag persistence, we track the changes in the homology classes in the level sets $X_r = f^{-1}(r)$ instead of the sub-level sets. We need maps connecting individual level sets, which is achieved by including the level sets into the adjacent interval sets. For this purpose we use the notation $X_i^j := X_{[s_i, s_j]}$ for the interval set between the two non-critical level sets. We have a zigzag sequence of interval and level sets connected by inclusions producing a level set zigzag diagram:

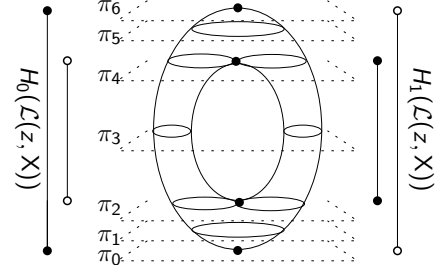
$$\mathcal{L}(f, X) : X_0^0 \rightarrow X_0^1 \leftarrow X_1^1 \rightarrow X_1^2 \cdots \rightarrow X_{m-1}^m \leftarrow X_m^m.$$

Applying the homology functor H_p with coefficients in a field κ , we obtain the zigzag persistence module for any dimension $p \geq 0$

$$H_p(\mathcal{L}(f, X)) : H_p(X_0^0) \rightarrow H_p(X_0^1) \leftarrow H_p(X_1^1) \rightarrow \cdots \rightarrow H_p(X_{m-1}^m) \leftarrow H_p(X_m^m). \quad (2.2)$$

The zigzag persistence of f is given by the summand interval modules of $H_p(\mathcal{L}(f, X))$. Each interval module is of the type $[r, r']$ where r and r' can be a_i or s_i for some $i \in [0, m+1]$. Just as in the sub-level set persistence, we identify the end points of the interval modules with the critical values

that were used to define the level set zigzag in the first place. In keeping with the understanding that even the level set homology classes do not change in the open interval sets, we convert an endpoint s_i to an adjacent critical value and make the interval module open at that critical value. Precisely we modify the interval modules as (i) $[a_i, a_j] \Leftrightarrow [a_i, a_j]$, (ii) $[a_i, s_j] \Leftrightarrow [a_i, a_{j+1}]$ (iii) $[s_i, a_j] \Leftrightarrow [a_i, a_j]$ (iv) $[s_i, s_j] \Leftrightarrow [a_i, a_{j+1}]$. The intervals in (i)-(iv) are referred as *closed-closed*, *closed-open*, *open-closed*, and *open-open* bars respectively. The figure above shows the two bar codes, one for H_0 and another for H_1 for a height function on a torus. The rightmost picture shows the barcode graph of $H_1(\mathcal{L}(z, X))$ which we explain later.



Using the results in [4, 6], we can connect the standard persistence with the level set zigzag persistence as follows:

Theorem 1.

1. $[a_i, a_j]$ is a bar for $H_p(\mathcal{SL}(f, X))$ iff it is so for $H_p(\mathcal{L}(f, X))$,
2. $[a_i, \infty)$ is a bar for $H_p(\mathcal{SL}(f, X))$ iff either $[a_i, a_j]$ is a closed-closed bar for $H_p(\mathcal{L}(f, X))$ for some $a_j > a_i$, or (a_j, a_i) is an open-open bar for $H_{p-1}(\mathcal{L}(f, X))$ for some $a_j < a_i$.

Proof. We know $H_p(\mathcal{SL}(f, X)) \cong (\oplus_{i,j} [a_i, a_j]) \oplus (\oplus_i [a_i, \infty))$. The first summand given by the finite intervals is isomorphic to a similar summand $\oplus_{i,j} [a_i, a_j]$ in the level set zigzag module $H_p(\mathcal{L}(f, X))$; see [6](Table 1, Type I). The second summand is isomorphic to $H_p(X)$, which by a result in [4] is isomorphic to $\mathring{B}_{p-1}(f, X) \oplus \bar{B}_p(f, X)$ where the open-open interval modules in $H_{p-1}(\mathcal{L}(f, X))$ generate $\mathring{B}_{p-1}(f, X)$ and the closed-closed interval modules in $H_p(\mathcal{L}(f, X))$ generate $\bar{B}_p(f, X)$. Then, the claimed result follows again from [6](Table 1, Type III and IV). \square

Overview and main results. Let \mathcal{K} be a simplicial complex consisting of n simplices that are linearly embedded in \mathbb{R}^3 . Let $|\mathcal{K}|$ denote the geometric realization arising out of this embedding. First, assume that \mathcal{K} is a pure 2-complex, that is, its highest dimensional simplices are triangles

and all vertices and edges are faces of at least one triangle. The algorithm for the case when it has tetrahedra and possibly edges and vertices that are not faces of triangles follows straightforwardly from the case when \mathcal{K} is pure, and is remarked upon at the end. Another assumption we make for our algorithm is that the coefficient field κ of the homology groups is \mathbb{Z}_2 .

A function $f : |\mathcal{K}| \rightarrow \mathbb{R}$ is called a *height* function if there is an affine transformation T of the coordinate frame so that $f(x) = z(T(x))$ for all points $x \in |\mathcal{K}|$ with z -coordinate being $z(x)$. Without loss of generality, assume that f is indeed the z -coordinate function and z is proper, that is, its values on the vertices are distinct. The standard topological persistence of z on $|\mathcal{K}|$ is called the *height persistence* which we aim to compute. Theorem 1 says that we can compute the barcode of the height persistence by computing the same for the level set zigzag persistence using the same height function. Precisely, we first compute the barcode for $H_1(\mathcal{L}(z, |\mathcal{K}|))$ from which we obtain a partial set of bars for $H_1(\mathcal{SL}(z, |\mathcal{K}|))$ and the complete set of bars for $H_2(\mathcal{SL}(z, |\mathcal{K}|))$. This is achieved by maintaining a level set data structure and tracking a set of *primary* cycles in them as we sweep through $|\mathcal{K}|$ along increasing z . At the same time, we build a barcode graph that registers the birth, death, split, and merge of the primary cycles. We show that this can be done in $O(n \log n)$ time. The bars of $H_1(\mathcal{L}(z, |\mathcal{K}|))$ are extracted from this graph again in $O(n \log n)$ time by adapting an algorithm of [1] to our case after a slight but important modification. According to Theorem 1, the closed-open and closed-closed bars of $H_1(\mathcal{L}(f, |\mathcal{K}|))$ constitute a partial set of bars for $H_1(\mathcal{SL}(z, |\mathcal{K}|))$. The open-open bars of $H_1(\mathcal{L}(f, |\mathcal{K}|))$, on the other hand, constitute a complete list of bars for the second homology module $H_2(\mathcal{SL}(z, |\mathcal{K}|))$ because the other summands for $H_2(\mathcal{SL}(f, |\mathcal{K}|))$ are trivial.

The rest of the bars of $H_1(\mathcal{SL}(z, |\mathcal{K}|))$ which are the open-open bars of $H_0(\mathcal{L}(z, |\mathcal{K}|))$ (Theorem 1) are shown to be captured by the Reeb graph of z on $|\mathcal{K}|$ which can be computed in $O(n \log n)$ time [16]. We show that the basis cycles for the first and second homology groups can be computed as part of the level set persistence and Reeb graph computations.

Theorem 2. *Let \mathcal{K} be a simplicial complex embedded in \mathbb{R}^3 with n simplices. Let $z : |\mathcal{K}| \rightarrow \mathbb{R}$ be a height function defined on it. One can compute the barcode for $H_i(\mathcal{L}(z, |\mathcal{K}|))$ for $i = 0, 1, 2$, in $O(n \log n)$ time where n is the number of simplices in \mathcal{K} . Furthermore, a set of basis cycles for $H_i(\mathcal{K})$, $i = 0, 1, 2$, can be computed in time $O(n \log n + k)$ where k is the total size of the output cycles.*

Similar statement holds for standard persistence.

Theorem 3. *Let \mathcal{K} be a simplicial complex embedded in \mathbb{R}^3 with n simplices. Let $z : |\mathcal{K}| \rightarrow \mathbb{R}$ be a height function defined on it. One can compute the barcode for $H_i(\mathcal{SL}(z, |\mathcal{K}|))$ for $i = 0, 1, 2$, in $O(n \log n)$ time where n is the number of simplices in \mathcal{K} .*

3 Level set data structure

Let v_1, v_2, \dots, v_m be the set of vertices of \mathcal{K} ordered by increasing z -values, that is, $z(v_j) > z(v_i)$ for $j > i$. Consider sweeping $|\mathcal{K}|$ in the increasing order of z -values. A level set $|\mathcal{K}|_r := z^{-1}(r)$, $r \in \mathbb{R}$, viewed as a graph embedded in the plane $\pi_r = \{x \in \mathbb{R}^3 \mid z(x) = r\}$, does not change its adjacency structure in any open interval $(z(v_i), z(v_{i+1}))$. This structure, however, may change as the level set sweeps through a vertex of \mathcal{K} . Consequently, for every vertex $v_i \in \mathcal{K}$, it suffices to track the changes when the level set jumps from the intermediate level $s_{i-1} < z(v_i)$ to the level $a_i := z(v_i)$ and then to the intermediate level $s_i > z(v_i)$ where $z(v_0) = -\infty$, and $z(v_{m+1}) = \infty$. All three level sets $|\mathcal{K}|_{s_{i-1}}$, $|\mathcal{K}|_{a_i}$, and $|\mathcal{K}|_{s_i}$ are plane graphs embedded linearly in the planes $z = s_{i-1}$, a_i , and s_i respectively. Let $G_r = (V_r, E_r)$ denote any such generic level set graph at a level r , where the vertex set V_r is the restrictions of the level set to the edges of \mathcal{K} and the edge set E_r is the restriction

of the level set to the triangles of \mathcal{K} . To avoid confusions, we will say *complex edges* and *complex triangles* to refer to the edges and triangles of \mathcal{K} respectively.

Level set graph and homology basis. We need to track a set of cycles representing a homology basis of $H_1(G_{s_{i-1}})$ to that of $H_1(G_{a_i})$ and then to that of $H_1(G_{s_i})$ as we sweep through the vertex v_i . Consider any such generic level set graph $G_r = (V_r, E_r)$ representing $z^{-1}(r)$.

The embedding of G_r in the plane π_r produces a partition of π_r into 2-dimensional faces, 1-dimensional edges, and 0-dimensional vertices. The faces are the connected components of $\pi_r \setminus G_r$. Let \mathcal{F}_r denote the collection of all 2-faces in this partition. A face $F \in \mathcal{F}_r$ has boundary cycle ∂F consisting of possibly multiple components, each being a cycle. We orient F by orienting its boundary and denote it with \vec{F} . The orientation is such that $\partial \vec{F}$ has the face on its right. In Figure 1, the face F has two boundaries, one around the outer curve (shown solid) and another around the inner circle (shown dotted). The unique face in \mathcal{F}_r that is unbounded plays a special role and is denoted F^* .

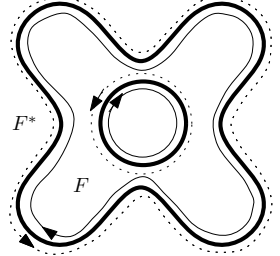


Figure 1: Primary and secondary cycles.

Observation 3.1. For a bounded face $F \in \mathcal{F}_r$, there is a unique oriented cycle $C_{\vec{F}} \in \partial \vec{F}$ that bounds a bounded face of $\pi_r \setminus C_F$ on its right. By definition, the unbounded face F^* has no such $C_{\vec{F}^*}$. In the figure above, $C_{\vec{F}}$ is the solid curve around outer boundary.

Because of the uniqueness of the cycles $C_{\vec{F}}$, we give them the special name of *primary* cycles. All other cycles are *secondary*. In Figure 1, the primary cycles are rendered solid and the secondary ones are rendered dotted. Recall that the elements of the first homology group H_1 are classes of cycles denoted $[C]$ for a cycle C . It turns out that the classes of unoriented primary cycles form a basis for $H_1(G_r)$ and thus tracking the primary cycles across the levels become the key to computing the level set zigzag persistence.

Proposition 4. The classes of unoriented cycles $\{[C_F] \mid C_{\vec{F}}$ is primary $\}$ form a basis of $H_1(G_r)$.

Proof. We observe the following facts:

- The classes of unoriented primary cycles form a sub-basis of $H_1(G_r)$.
- $H_1(G_r) \cong \tilde{H}_0(\pi_r \setminus G_r)$ where \tilde{H}_0 denotes the *reduced* zero-dimensional homology group.
- The faces in $\mathcal{F}_r \setminus F^*$ form a basis of $\tilde{H}_0(\pi_r \setminus G_r)$.

For the first fact, observe that the set of such cycles are independent meaning that there is no unoriented primary cycle C_F that can be written as the sum of other unoriented primary cycles. If it were true, let $C_F = C_{F_1} + C_{F_2} + \dots + C_{F_t}$. Then, the boundary of $R = F \cup_{i=1}^t F_i$ is empty. But, that is impossible unless $R = \pi_r$. Since $F^* \notin R$, we have $R \neq \pi_r$.

The second fact follows from Alexander duality because G_r is embedded in the plane π_r . The third fact follows from the definition of reduced homology groups.

Consider a map μ that sends each face $F \in \mathcal{F}_r \setminus F^*$ to its unoriented primary cycle C_F . This map is bijective due to Observation 3.1. Therefore, by the first and third facts, $\tilde{H}_0(\pi_r \setminus G_r)$ is isomorphic to the summand of $H_1(G_r)$ generated by the classes of unoriented primary cycles. Indeed, this summand is $H_1(G_r)$ itself since $\tilde{H}_0(\pi_r \setminus G_r)$ is isomorphic to $H_1(G_r)$ by the second fact. \square

The following Proposition complements Proposition 4. We do not use it, but remark about its connection to Reeb graphs at the end.

Proposition 5. *The H_0 -classes of unoriented secondary cycles form a basis of $H_0(G_r)$.*

Representing level set graphs. Proposition 4 implies that we can maintain a basis of $H_1(G_r)$ by maintaining the primary cycles alone. However, for realizing the zigzag maps that connect across the level sets (Eqn. 2.2), we need a different basis involving both primary and secondary cycles. For each bounded face $F \in \mathcal{F}_r \setminus F^*$, let $\partial F = C_F + \sum_i C_i$ be the *boundary cycle* for the face F which is the \mathbb{Z}_2 -addition of the primary cycle C_F with the secondary ones in F . The next assertion follows from Proposition 4 immediately.

Proposition 6. *The classes $\{[\partial F] \mid F \in \mathcal{F}_r \setminus F^*\}$ form a basis of $H_1(G_r)$.*

The importance of the boundary cycles in realizing the zigzag maps needed for the persistence module in Eqn. 2.2 is due to the following observation.

Observation 3.2. *For every $i \in \{1, \dots, m-1\}$ and for every boundary cycle ∂F in the intermediate level s_i , there are sum of boundary cycles $\sum \partial F_{i,j}$ and $\sum \partial F_{(i+1),j}$ at the critical levels $a_i = z(v_i)$ and $a_{i+1} = z(v_{i+1})$ respectively with $a_i < s_i < a_{i+1}$ so that the inclusions of $\partial F_{i,j}, \partial F$, and $\partial F_{(i+1),j}$ into the interval space $|\mathcal{K}|_{[a_i, a_{i+1}]}$ induce linear maps at the homology levels given by $[\partial F] \rightarrow [\sum \partial F_{i,j}]$, $[\partial F] \rightarrow [\sum \partial F_{(i+1),j}]$.*

By Proposition 6 and the above observation, the zigzag maps of the persistence module in Eqn. 2.2 can be tracked if we track the boundary cycles for each face. However, this requires additional bookkeeping for maintaining the primary and secondary cycles of a face together. Instead, we maintain each individual primary and secondary cycle independently being oblivious to their correspondence to a particular face though this information is maintained implicitly. Due to Proposition 4, it becomes sufficient to register the changes in the primary cycles for tracking the boundary cycles.

The primary and secondary cycles change as we sweep over vertices. Figure 2 illustrates some of these changes. A secondary cycle may split into two cycles one of which is primary and the other is not (C in Fig.), it may split into two secondary cycles (Z in Fig.), or two primary cycles may merge (D_1, D_2 in Fig.). Therefore, we need to maintain all oriented cycles in $\partial \vec{F}$, and keep track of the primary ones among them.

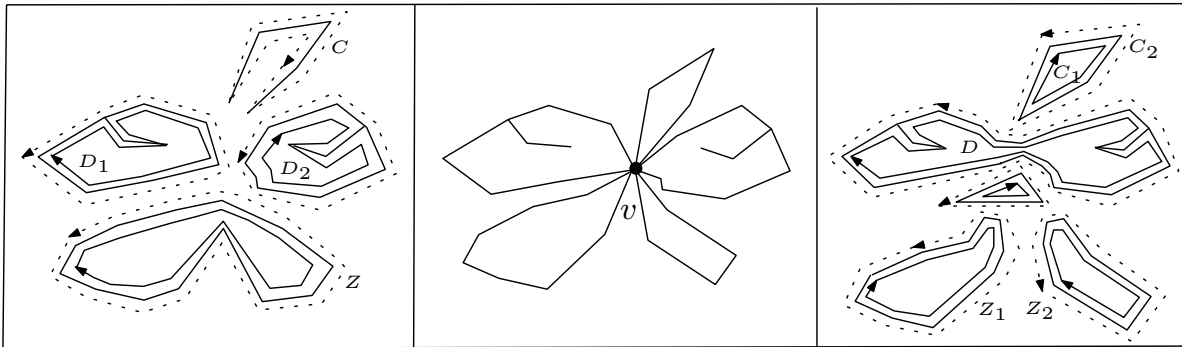


Figure 2: Level set graph going through changes after sweeping through a vertex: $G_{s_{i-1}}$ (left), G_{a_i} (middle), and G_{s_i} (right). The primary and secondary cycles are indicated with solid and dotted curves respectively. Notice how the secondary cycle C on the left got first pinched and then split into one primary cycle C_1 and another secondary cycle C_2 on the right.

We consider a directed version $\overrightarrow{G}_r = (V_r, \overrightarrow{E}_r)$ of G_r where each edge $e \in E_r$ is converted into two directed edges in \overrightarrow{E}_r that are oriented oppositely. The graph \overrightarrow{G}_r is represented with a set of oriented cycles $C(\overrightarrow{G}_r) = \cup_{F \in \mathcal{F}_r} \partial \overrightarrow{F}$ that bound the faces in \mathcal{F}_r on right. These cycles are represented with a sequence of directed edges.

A vertex in V_r either lies on a vertex $v \in \mathcal{K}$, or in the interior of a complex edge e in which case we denote it as the vertex $e \in V_r$. Any edge in E_r is an intersection of the level set with a complex triangle t , which we also denote as an edge $t \in E_r$. Let $t \in E_r$ be any edge adjoining a vertex $e \in V_r$. We have two directed copies \overrightarrow{t} and \overleftarrow{t} of t in \overrightarrow{G}_r . Assume that \overleftarrow{t} is directed away from e and \overrightarrow{t} is directed toward e .

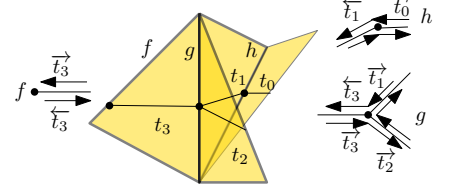


Figure 3: Connection rules.

We follow a **connection rule** for deciding the connections among the directed edges around $e \in V_r$ to construct the cycles in $C(\overrightarrow{G}_r)$ as follows. Let d and d' be a pair of directed edges, where the head of d is the tail of d' . The directed path dd' locally separates the plane around the meeting point of d and d' . The region to the right of dd' is called its *right wedge*, and the region to the left is called its *left wedge*. We have three cases for deciding the connections:

- e has only one edge $t = t_0$ (f in Figure 3): connect \overrightarrow{t} to \overleftarrow{t} .
- e has exactly two edges t_0 and t_1 (h in Figure 3): connect $\overrightarrow{t_0}$ to $\overleftarrow{t_1}$, and connect $\overleftarrow{t_0}$ to $\overrightarrow{t_1}$.
- e has three or more edges (g in Figure 3): consider a circular order of all edges $t \in E_r$ adjoining $e \in V_r$. Let $t_0, t_1, \dots, t_k, t_0$ be this circularly ordered edges around e . For any consecutive pairs of edges $t_i, t_{(i+1) \pmod k}$, determine if the right wedge of $\overrightarrow{t_i} \overleftarrow{t_{(i+1) \pmod k}}$ contains the edge $t_{(i-1) \pmod k}$. If so, connect $\overrightarrow{t_{(i+1) \pmod k}}$ to $\overleftarrow{t_i}$. If not, connect $\overrightarrow{t_i}$ to $\overleftarrow{t_{(i+1) \pmod k}}$.

The choice of our orientations and connections leads to the following observation:

Observation 3.3. *Let (d, d') be any pair of directed edges in \overrightarrow{G}_r . They are consecutive directed edges on the oriented boundary of a face $F \in \mathcal{F}_r$ if and only if d connects to d' by the connection rule around some vertex $e \in V_r$.*

The observation above relates the directed cycles in $C(\overrightarrow{G}_r)$ with a local connection rule. We exploit this fact to update the cycles locally in our algorithm.

Cycle trees. The directed cycles in $C(\overrightarrow{G}_r)$ are represented with balanced trees that help implementing certain operations on them efficiently. We explain this data structure now.

A directed edge d where $d = \overleftarrow{t}$ or $d = \overrightarrow{t}$ is represented with a node d that has three fields; $d \cdot \text{tri}$ points to the complex triangle t , $d \cdot \text{tail}$ and $d \cdot \text{head}$ point to the complex edges e_1 and e_2 respectively where d is directed from e_1 to e_2 . A cycle C of directed edges is represented with a balanced tree T_C , namely a 2-3 tree [2] where the directed edges of C constitute the leaf nodes of T_C with the constraint that the leaves of any subtree of T_C represent a path (directed) in C . The leaves of T_C are joined with a linked list in the order they appear on the directed cycle C . A pointer $d \cdot \text{next}$ in a leaf node d implements this link list. The node d also maintains another pointer $d \cdot \text{prev}$ to access the previous node on the linked list in $O(1)$ time. However, it is important to keep in mind that it is the next pointers that provide the orientation of the cycle C . Furthermore, the last node in both linked lists connected by next and prev pointers respectively is *assumed* to connect to the first one. This creates the necessary circularity without actually making the list circular. We denote the linked list of leaves of a tree T as $\text{list}(T)$. The 2-3 trees built on top of the paths support the following operations.

$\text{FIND}(d)$: returns the root of the tree d belongs to.
 $\text{SPLIT}(T, d)$: splits a tree T into two trees T_1 and T_2 where $\text{list}(T_1)$ is the sublist of $\text{list}(T)$ that contains all elements in $\text{list}(T)$ before d , and $\text{list}(T_2)$ is the sublist that contains all elements in $\text{list}(T)$ after and including d .
 $\text{JOIN}(T_1, T_2)$: takes two trees T_1 and T_2 and produces a single tree T with $\text{list}(T)$ as the concatenation of $\text{list}(T_1)$ and $\text{list}(T_2)$ in this order.
 $\text{PERMUTE}(T, d)$: makes d the first node in the cycle represented with T . It is implemented by calling $\text{SPLIT}(T, d)$ that produces T_1 and T_2 , and then returning $T := \text{JOIN}(T_2, T_1)$.
 $\text{INSERT}(d, d')$: inserts the element d after d' in $\text{list}(T)$ where $T := \text{FIND}(d')$.
 $\text{DELETE}(d)$: deletes d from $\text{list}(T)$ where $T := \text{FIND}(d)$.

All of the above operations maintain the trees well balanced allowing traversal of a path from a leaf to the root in $O(\log n)$ time where n is the total number of elements in the lists of the trees involved. This in turn allows each of these operations to be carried out in $O(\log n)$ time. Using these basic operations, we implement two key operations, splitting and merging of cycles.

$\text{SPLITCYCLE}(T, d, d')$: This splits a directed cycle into two. A cycle may get first pinched and then splits into more cycles as we sweep through a vertex. This operation is designed to implement this event. Given a tree T , it returns two trees T_1 and T_2 where $\text{list}(T_1)$ represents the path from d to d' in the directed cycle given by $\text{list}(T)$, and $\text{list}(T_2)$ represents the path from $d' \cdot \text{next}$ to $d \cdot \text{prev}$ in the same cycle. See Figure 4, bottom row.

It is implemented as follows: Let $T := \text{PERMUTE}(T, d)$. Call $\text{SPLIT}(T, d' \cdot \text{next})$ which returns two trees T_1 and T_2 as required.

$\text{MERGECYCLE}(d, d')$: This merges the two cycles that d and d' belong to. The new cycle has d' after d and $d \cdot \text{next}$ after $d' \cdot \text{prev}$. This is implemented as follows: Let $T_1 := \text{FIND}(d)$ and $T_2 := \text{FIND}(d')$. Let $T_1 := \text{PERMUTE}(T_1, d \cdot \text{next})$ and $T_2 := \text{PERMUTE}(T_2, d')$. Then, return $T := \text{JOIN}(T_1, T_2)$.

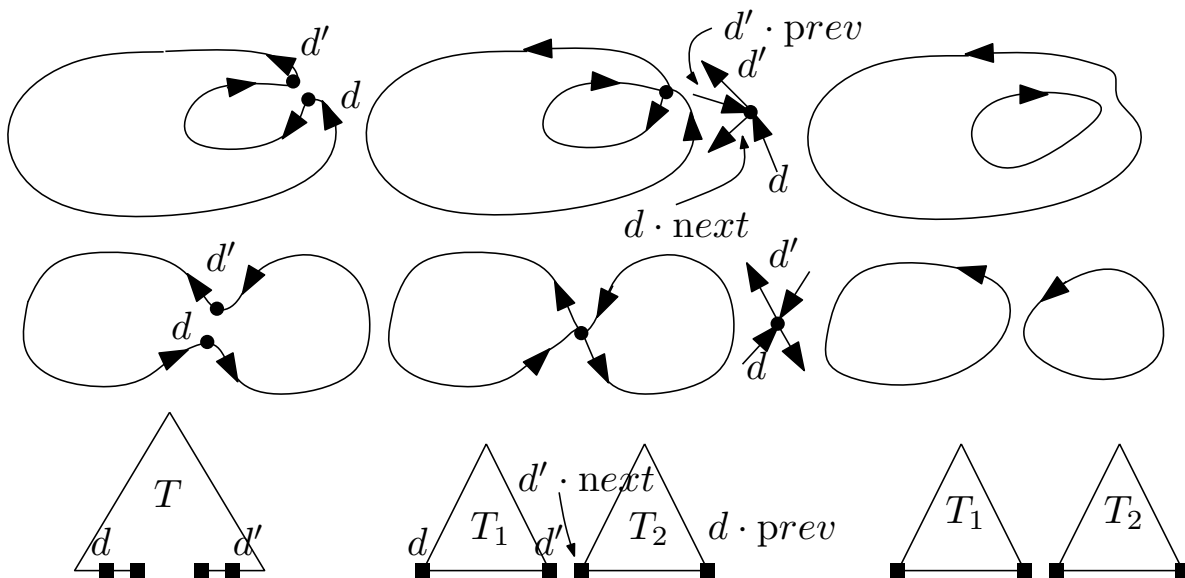


Figure 4: Secondary cycle splitting: split at the top generates one primary and another secondary cycle; split at the bottom generates two secondary cycles.

4 Updating level sets

Now we describe how we update the graph $\vec{G}_{s_{i-1}}$ to \vec{G}_{a_i} and then to \vec{G}_{s_i} . As we sweep through v_i , only the cycles in these graphs containing a vertex on a complex edge with v_i as an endpoint may change combinatorially. We only update the cycles for combinatorial changes to make sure that the combinatorics of the level set graphs are maintained correctly though their geometry is updated only when needed to infer the correct adjacencies. This allows us to inspect only $O(n_{v_i})$ simplices where n_{v_i} is the number of simplices adjoining v_i in \mathcal{K} . Summing over all vertices, this provides an $O(n)$ bound which gets multiplied with the $O(\log n)$ complexity for the tree operations that we perform for each such simplex. Also, local circular sorting of $O(n_{v_i})$ edges around each vertex v_i and complex edges connected to it accounts for $O(n \log n)$ time in total.

Primary cycle detection. The cycles in \vec{G}_r that change combinatorially may experience splitting, merging, edge contraction, edge expansion, or a combination of such events. Specifically, during splitting and merging, new cycles are generated which need to be characterized as primary or not. Figure 4 illustrates two cases of a secondary cycle splitting. Two similar cases arise for the primary cycle splitting. For merging also we have four cases mirroring the splitting case. It turns out that we can determine if the new cycles are primary or not by the orientations of the edges around the ‘pinching’ vertex if we know the type (primary or not) of the original cycles. We explain this for the case of splitting.

Let C be a cycle in $C(\vec{G}_{s_{i-1}})$ which splits at v_i . Let d and d' be any two non-consecutive directed edges in C that meet at v_i in \vec{G}_{a_i} . Assume that we know that C is secondary. The case when C is primary is similar. We need to distinguish the case when one of the two new cycles nests inside the other. This can be checked in $O(1)$ time by determining if the right wedge of $d(d \cdot \text{next})$ contains d' or not. If not, both new cycles remain secondary. Otherwise, we have a nesting, and exactly one of the two new cycles becomes primary. We can determine again which of the two becomes primary in $O(1)$ time. For this consider a ray with tail at v and entering the left wedge of $d(d \cdot \text{next})$. If this ray enters the left wedge of dd' , we declare the new cycle containing d and d' to be secondary and the other cycle containing $d' \cdot \text{prev}$ and $d \cdot \text{next}$ to be primary. If the ray enters the right wedge, we flip the assignment for the type of the two new cycles.

With these $O(1)$ local checks, we design the two routines below that decide the type of the new cycle(s) in both the splitting and merging cases assuming that we know if the input cycle(s) are primary or not.

SPLITPRIM(bool, d_1, d_2): This routine assumes that bool indicates if the cycle to be split which contains d_1 and d_2 is primary (true or false), and returns a pair (bool₁, bool₂) of booleans where bool_i is true if and only if the new cycle containing d_i is primary.

MRGPRIM(bool₁, bool₂, d_1, d_2): This routine assumes that the input boolean variables bool_i indicates if the cycle containing d_i is primary, and returns a boolean variable bool which is true if and only if the new merged cycle is primary.

Now we describe the actual updates of the graphs when the sweep goes through a vertex $v_i \in \mathcal{K}$. For convenience, we designate a complex triangle as *top*, *middle*, or *bottom* if it has v_i as the lowest, middle, or highest vertex respectively w.r.t. the height z . Similarly, a complex edge is called *top*, or *bottom* if it has v_i as the lowest or highest vertex respectively. As we continue with the sweep, we keep on recording the birth, death, splitting and merging of primary cycles by creating a barcode graph. *Current* primary cycles are represented by *current* edges in the barcode graph whose one endpoint is already determined, but the other one is yet to be determined. The nodes in the barcode

graph are created when a primary cycle is born, dies, splits, or merges with another cycle. It is important to note that the nodes of the barcode graph are created only at the intermediate levels s_i . Each tree T maintains a pointer $T \cdot \text{barcode}$ that points to a current edge in the barcode graph if its cycle is primary. Otherwise, this pointer is assumed to be a null pointer. Additionally, we assume that there is a boolean field $T \cdot \text{prim}$ which is set true if and only if T represents a primary cycle. The barcode graph at level r is denoted $B(G_r)$. As we move from level r to the next level r' , we keep updating this barcode graph by recording the birth, death, splitting and merging of primary cycles and still denote it as $B(G_r)$ till we finish processing level r' at which point we denote it as $B(G_{r'})$.

Updating $\vec{G}_{s_{i-1}}$ to \vec{G}_{a_i} . The combinatorics of $G_{s_{i-1}}$ change only by the edges $t \in E_{s_{i-1}}$ where t is a bottom or middle triangle. If the edge t has both vertices on bottom complex edges, then t is contracted to v_i in G_{a_i} . Otherwise, the edge t remains in G_{a_i} , but its adjacency at the vertex which becomes v_i in G_{a_i} changes. Also, in both cases classes in $H_1(\mathcal{L}(z, |\mathcal{K}|))$ may die or be born. We perform the combinatorial changes and detect the birth and deaths of homology classes as follows:

Contracting edges: When we contract edges, a cycle may simply contract and nothing else happens. But, we may also detect that a primary cycle of three edges is collapsed to two directed edges corresponding to a single undirected edge. This indicates a death of a class in $H_1(\mathcal{L}(z, |\mathcal{K}|))$ which occurs entering the level a_i but not exactly at a_i . So, we operate as follows.

Let t be any bottom complex triangle for v_i , and let d_1 and d_2 be two directed edges associated with t . Let $T_1 := \text{FIND}(d_1)$ and $T_2 := \text{FIND}(d_2)$. For $i = 1, 2$, we call $\text{DELETE}(d_i)$. If $T_i \cdot \text{prim} = \text{true}$ and T_i has two leaves, we terminate the current edge pointed by $T_i \cdot \text{barcode}$ with a closed node at level s_i in $B(G_{s_{i-1}})$ and remove T_i completely. The closed node indicates that the cycle dying entering the level a_i is still alive at the level s_{i-1} .

Cycle updates: The edges of a cycle in $C(\vec{G}_{s_{i-1}})$ can come together at v_i to create new cycles. After the edge contractions, the only edges that we need to update for possible combinatorial changes correspond to middle complex triangles. Let t be such a triangle and let g and h be its edges that are top and bottom edges for v_i respectively. For each directed edge d with $d \cdot \text{tri} = t$, we update $d \cdot \text{tail} = g$ or $d \cdot \text{head} = g$ if originally we had $d \cdot \text{head} = h$ or $d \cdot \text{tail} = h$ respectively.

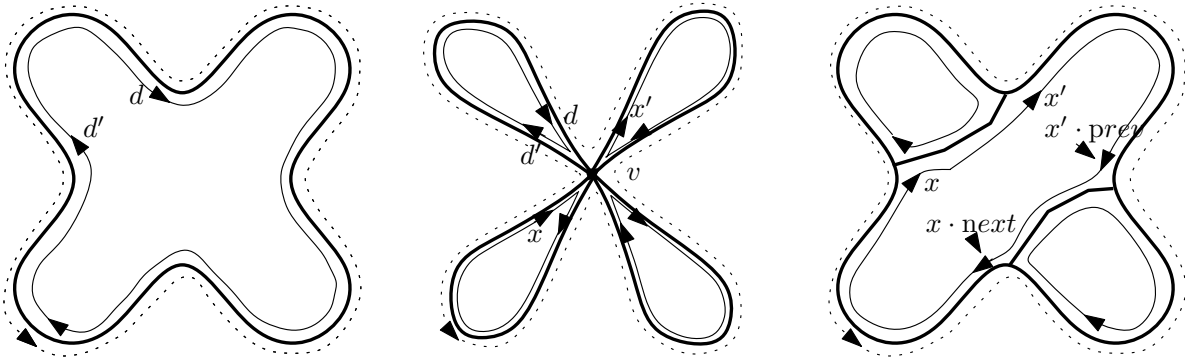


Figure 5: A primary cycle (left) splits into four primary cycles (middle), then two cycles merge into one where the other two only expand (right) when we sweep through vertex v_i .

Next, we update the cycles that may combinatorially change due to splitting or merging at v_i , and also record new births as a result. We consider every directed edge d so that the triangle $t = d \cdot \text{tri}$ is a middle triangle and determine a circular order of their undirected versions around v_i .

For every such directed edge d , we determine its pair directed edge d' using the *connection rule* that we described before. Observe that plane embedding of the level set graph is used here. Actually, the lack of such canonical ordering of edges around a vertex for level set graphs becomes the roadblock for extending this algorithm to persistence of functions that are not heights. Let $T := \text{FIND}(d)$ and $T' := \text{FIND}(d')$. We have two cases: the splitting case when $T = T'$ (see d and d' in Figure 5) and the merging case when $T \neq T'$.

Splitting Case, $T = T'$: If $d' = d \cdot \text{next}$, the cycle C containing d and d' and represented by T does not change and we do nothing. Otherwise, the cycle C splits into two new cycles whose type needs to be determined. So, we call $\text{SPLITPRIM}(T \cdot \text{prim}, d \cdot \text{next}, d')$ which returns a pair of boolean values $(\text{bool}_1, \text{bool}_2)$ indicating if the two new cycles are primary or not. We split T to create the representations of the two new cycles. But, this operation destroys T whose type (primary or not) and barcode pointer are needed for assigning the same for the two new trees. So, we save $b := T \cdot \text{barcode}$ and $s := T \cdot \text{prim}$ first, and call $\text{SPLITCYCLE}(T, d, d')$ which returns two trees T_1 and T_2 representing the two cycles. Geometric constraints allow only the following two cases:

Case(i): $(s, \text{bool}_1, \text{bool}_2) = (\text{false}, \text{true}, \text{false})$ or $(\text{false}, \text{false}, \text{true})$: A new primary cycle is born at the level $z(v_i)$. This is an open-ended birth at the level s_{i-1} because the cycle exists at the level $z(v_i)$ but not at the level s_{i-1} . If $\text{bool}_i = \text{true}$, we set $T_i \cdot \text{prim} := \text{true}$ and $T_{i(\bmod 2)+1} \cdot \text{barcode} := \text{null}$, $T_{i(\bmod 2)+1} \cdot \text{prim} := \text{false}$. Then, we set $T_i \cdot \text{barcode} := b$ where b is a current edge created with an open end at level s_{i-1} in $B(G_{s_{i-1}})$.

Case(ii): $(s, \text{bool}_1, \text{bool}_2) = (\text{true}, \text{true}, \text{true})$: A new primary cycle is born at the level $z(v_i)$ due to a split of the cycle represented by the saved pointer $b := t \cdot \text{barcode}$. We set $T_i \cdot \text{prim} := \text{true}$, for $i = 1, 2$, and call $\text{SPLITBAR}(b, v_i)$ which splits b at level s_{i-1} and returns two current edge pointers b_1 and b_2 . We set $T_1 \cdot \text{barcode} := b_1$ and $T_2 \cdot \text{barcode} := b_2$.

Merging Case, $T \neq T'$: two cycles C_1 and C_2 represented by $T_1 := T$ and $T_2 := T'$ respectively merge to become one. As before, we first store aside the type of C_1 and C_2 and associated current edge pointers by setting $s_i := T_i \cdot \text{prim}$ and $b_i := T_i \cdot \text{barcode}$ for $i = 1, 2$. Next, we call $\text{MERGECYCLE}(d, d')$ which merges the two cycles containing d and d' and returns a tree T_3 representing this new cycle, say C_3 . A call to $\text{MRGPRIM}(s_1, s_2, d, d')$ returns a boolean variable bool which is true if and only if C_3 is primary. Again, we have only the following two cases.

Case(i): $(s_1, s_2, \text{bool}) = (\text{true}, \text{false}, \text{true})$ or $(\text{false}, \text{true}, \text{true})$. In this case no primary cycle dies, but the new cycle remains primary. So, no current edge is terminated and the current edge associated to the primary cycle among C_1 and C_2 is continued by C_3 . If $s_i = \text{true}$, we set $T_3 \cdot \text{barcode} := b_i$, $T_3 \cdot \text{prim} := \text{true}$.

Case(ii): $(s_1, s_2, \text{bool}) = (\text{false}, \text{false}, \text{false})$: No primary cycle dies and the new cycle is also not primary. We set $T_3 \cdot \text{barcode} := \text{null}$ and $T_3 \cdot \text{prim} := \text{false}$.

Updating \vec{G}_{a_i} to \vec{G}_{s_i} . To update \vec{G}_{a_i} to \vec{G}_{s_i} , we need to create directed edges corresponding to top triangles, that is, the complex triangles with v_i as the bottom vertex. These new edges change the combinatorics of \vec{G}_{a_i} in four ways: they may (i) expand the existing cycles without creating or destroying any primary class, (ii) create a new cycle giving birth to a new class, (iii) split a cycle pinched at v_i (it turns out that no new class is born in this case), (iv) merge two cycles meeting at v_i ; in this case, a primary cycle dies. Two cycles containing directed edges x and x' in \vec{G}_{a_i} in Figure 5 get merged into one cycle in \vec{G}_{s_i} . The details of the merge is shown in Figure 6. It is preceded by an insertion of a sequence of edges w, u, z that connect x and x' . Similarly, another sequence connects $x' \cdot \text{prev}$ and $x \cdot \text{next}$.

Expanding cycles: We iterate over all directed edges corresponding to the middle triangles. Let

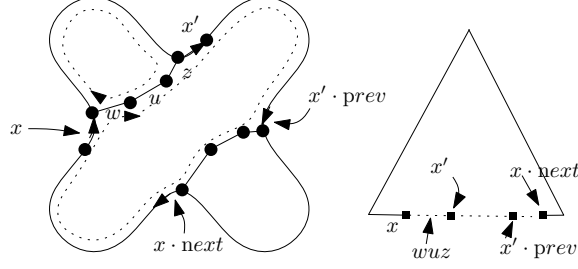


Figure 6: Merging lists.

d be any such directed edge in \vec{G}_{a_i} . The directed edge d belongs to a unique cycle C in the directed graph \vec{G}_{s_i} . Starting from d , we aim to create the missing edges in C . For this, we create a routine $\text{NEXTLINK}(d)$ that takes a directed edge d and creates all missing directed edges in C that lie between d and the next directed edge d' with $d' \cdot \text{tri}$ being a middle triangle.

$\text{NEXTLINK}(d)$: Consider the complex edge $e = d \cdot \text{head}$ and the circular order of directed edges of \vec{G}_{s_i} around e . In $O(1)$ time we determine the adjacent directed edge d' using the connection rule described before. If d' does not exist already, we create a directed edge for d' and insert it into the tree containing d by calling $\text{INSERT}(d', d)$. Replacing the role of d with d' , we continue. If $d' \cdot \text{tri}$ is a middle triangle, we stop and return d' .

To complete updating C containing the directed edge d , we call $\text{NEXTLINK}(d)$ which returns, say d' . Let $T := \text{FIND}(d)$ and $T' := \text{FIND}(d')$. We have two cases:

Splitting Case, $T = T'$: If $d' = d \cdot \text{next}$ in the graph \vec{G}_{a_i} , then this is a mere expansion of a cycle and we do not make any updates in the barcode graph. Otherwise, we do the same as in the case for updating $\vec{G}_{s_{i-1}}$ to \vec{G}_{a_i} except that the subcases become different.

Case(i): $(s, \text{bool}_1, \text{bool}_2) = (\text{true}, \text{true}, \text{false})$ or $(\text{true}, \text{false}, \text{true})$. In this case no new primary cycle is born. So, no new current edge is created. If $\text{bool}_1 = \text{true}$, we set $T_i \cdot \text{barcode} := T \cdot \text{barcode}$, $T_i \cdot \text{prim} := \text{true}$ and $T_{i(\bmod 2)+1} \cdot \text{barcode} := \text{null}$, $T_{i(\bmod 2)+1} \cdot \text{prim} := \text{false}$.

Case(ii): $(s, \text{bool}_1, \text{bool}_2) = (\text{false}, \text{false}, \text{false})$: No primary cycle is born. We set $T_i \cdot \text{barcode} := \text{false}$ and $T_i \cdot \text{prim} := \text{false}$.

Merging Case, $T \neq T'$: Let C_i be represented by T_i where $T_1 := T$ and $T_2 := T'$. Again, we do the same as in the case of merging while going from $\vec{G}_{s_{i-1}}$ to \vec{G}_{a_i} . The subcases become:

Case(i): $(s_1, s_2, \text{bool}) = (\text{true}, \text{true}, \text{true})$: Two primary cycles merge to become one. Here one primary class dies, but we do not know which one. So, we record the merging only. We join the current edges pointed by b_1 and b_2 at a node at level s_i and start a new current edge pointed by b from that node. We set $T_3 \cdot \text{barcode} := b$, $T_3 \cdot \text{prim} := \text{true}$.

Case(ii): $(s_1, s_2, \text{bool}) = (\text{false}, \text{true}, \text{false})$ or $(\text{true}, \text{false}, \text{false})$: A primary cycle dies. So, if $s_i = \text{true}$ for $i = 1$ or 2 , we terminate the current edge pointed by b_i at level s_i with an open end. Then, we set $T_3 \cdot \text{barcode} := \text{null}$ and $T_3 \cdot \text{prim} := \text{false}$.

New cycles: Some cycles in \vec{G}_{s_i} may not arise from the updates of the old cycles. All of their edges come from the top triangles that have the vertex v_i as the bottom vertex. These cycles may introduce new current edges with closed birth at level s_i . To create these cycles, we iterate over all top triangles for which at least one of the two directed edges has not been created yet. Let t be such a triangle where the directed edge from the complex edge e to e' has not yet been created.

We create the directed edge d with $d \cdot \text{tri} = t$, $d \cdot \text{tail} = e$, and $d \cdot \text{head} = e'$ and initialize a tree T with it. To complete the cycle C that d belongs to, we call `NEXTLINK(d)` which returns after completing the tree T . We check if the new cycle C containing d is primary or not by checking if it contains the point at infinity. This can be done in $O(n_v)$ time in total for all such new cycles. If C is primary, a current edge b begins with a closed edge end at level s_i in the barcode graph. So, we set $T \cdot \text{barcode} := b$ and $T \cdot \text{prim} := \text{true}$. Otherwise, set $T \cdot \text{barcode} = \text{null}$ and $T \cdot \text{prim} = \text{false}$.

5 Barcode graph

After processing the last vertex v_m of \mathcal{K} in the sorted order v_1, v_2, \dots, v_m , we obtain the barcode graph $R = B(G_{s_m})$. It has nodes for the intermediate levels between critical levels of \mathcal{K} (levels of vertices of \mathcal{K}). Now we proceed to justify why the bars extracted from a modified R are indeed the bars for $H_1(\mathcal{L}(z, |\mathcal{K}|))$.

By considering R as a graph linearly embedded in \mathbb{R}^3 , we can consider its level set zigzag module with height function z . Its vertices have values s_i , $i = 1, \dots, m-1$ that are linearly interpolated over the edges. Notice that the critical values of z on R are $s_0 < s_1 < \dots < s_m$ whereas the same on \mathcal{K} are $a_1 < \dots < a_m$. Writing the interval set $R_{[a_i, a_j]}$ as R_i^j , we get the level set zigzag module:

$$H_0(\mathcal{L}(z, R)) : H_0(R_1^1) \rightarrow H_0(R_1^2) \leftarrow H_0(R_2^2) \rightarrow \dots \rightarrow H_0(R_{m-1}^m) \leftarrow H_0(R_m^m). \quad (5.3)$$

Consider the level set zigzag module $H_1(\mathcal{L}(z, |\mathcal{K}|))$ by putting $p = 1$ in (2.2). Here, the interval sets are $\mathcal{K}_i^j = |\mathcal{K}|_{[s_i, s_j]}$ (notice the shift in interval sets). We have the level set zigzag module:

$$H_1(\mathcal{L}(z, |\mathcal{K}|)) : H_1(\mathcal{K}_0^0) \rightarrow H_1(\mathcal{K}_0^1) \leftarrow H_1(\mathcal{K}_1^1) \rightarrow \dots \rightarrow H_1(\mathcal{K}_{m-1}^m) \leftarrow H_1(\mathcal{K}_m^m). \quad (5.4)$$

Augmenting R with threading: For Proposition 7 below to be true, we need the homology group of every interval set in the above two modules to be identified with the homology group of the level set at the intermediate value of the vertex. That is, we want $H_0(R_i^{i+1}) = H_0(z^{-1}(s_i))$ and $H_1(\mathcal{K}_i^{i+1}) = H_1(z^{-1}(a_{i+1}))$. This condition is satisfied for the module for \mathcal{K} because the homology groups for \mathcal{K} do not change except at the critical values a_i , $i = 1, \dots, m$. However, this is not true for R because of the open ends of some of the edges. For example, consider a single edge with an open end at the value s_i . The homology group $H_0(R_i^{i+1})$ in this case has rank 1 whereas $H_0(z^{-1}(s_i))$ has rank 0 because of the open end node. To remedy this, we consider the reduced homology group $\tilde{H}_0(\cdot)$ for R and augment R with an added ‘thread’. In the above example, if we add a thread with monotonic values that attaches to the open end, and then consider the reduced homology group, we get that $\tilde{H}_0(R_i^{i+1}) \cong \tilde{H}_0(z^{-1}(s_i)) = 0$. Extending this idea, we augment the graph R by adding a ‘dummy’ thread that runs with monotone values in the range $(-\infty, \infty)$ while attaching to every open node of R at every level. See Figure 7. The ‘dummy’ component represented by the thread splits and merge at the ‘open’ degree-1 vertices. Call a degree-1 vertex *upward* (u in Figure 7, also see Figure 8(b)) or *downward* (v in Figure 7, also see Figure 8(b)) if it is connected to a vertex with smaller or larger value respectively. A real component joins with the dummy one at an upward vertex and splits from a dummy component at a downward vertex. The ‘thread’ representing the dummy component turns the open ends into merge or split vertices.

The nodes in R at an intermediate level s_i are not supposed to have any edge among themselves. However, because of our construction of R , we may have artificial edges between nodes at the same level. See Figure 8(b), (c). We modify R simply by contracting any such edge (Figure 8(d)). This operation, carried out in $O(n)$ time, brings all nodes at a fixed level in a connected component formed by artificial edges to a single node at that level. Let R still denote the resulting barcode graph.

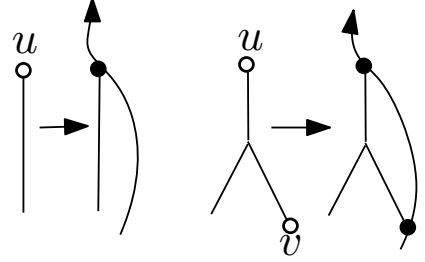


Figure 7: Threading.

Proposition 7. $H_1(\mathcal{L}(z, |\mathcal{K}|)) \cong \tilde{H}_0(\mathcal{L}(z, R))$.

Proof. Because of our construction and the threading, the homology group (vector space) of an interval set bracketing a single vertex either in R or in \mathcal{K} is isomorphic to that of the level set at the value of the vertex, that is, $\tilde{H}_0(R_i^{i+1}) = \tilde{H}_0(z^{-1}(s_i))$ and $H_1(\mathcal{K}_i^{i+1}) = H_1(z^{-1}(a_{i+1}))$. Writing the vector spaces for R as $\tilde{H}_0(z^{-1}(v)) = \mathbb{U}_v$ and the vector spaces for \mathcal{K} as $H_1(z^{-1}(v)) = \mathbb{V}_v$, and observing that $\mathbb{U}_v \cong \mathbb{V}_v$ for $v = a_i$ or s_i , we obtain the following diagram:

$$\begin{array}{ccccccc} H_1(\mathcal{L}(z, |\mathcal{K}|)) : & \mathbb{V}_{s_0} & \longrightarrow & \mathbb{V}_{a_1} & \longleftarrow & \mathbb{V}_{s_1} & \cdots \longrightarrow \mathbb{V}_{a_m} \longleftarrow \mathbb{V}_{s_m} \\ & \parallel & & \parallel & & \parallel & \parallel \\ \tilde{H}_0(\mathcal{L}(z, R)) : & \mathbb{U}_{s_0} & \longleftarrow & \mathbb{U}_{a_1} & \longrightarrow & \mathbb{U}_{s_1} & \cdots \longleftarrow \mathbb{U}_{a_m} \longrightarrow \mathbb{U}_{s_m} \end{array}$$

To make the above diagram commute, we reverse the arrows for one of the modules, say $\tilde{H}_0(\mathcal{L}(z, R))$ by considering the dual module on the dual vector spaces:

$$\begin{array}{ccccccc} H_1(\mathcal{L}(z, |\mathcal{K}|)) : & \mathbb{V}_{s_0} & \longrightarrow & \mathbb{V}_{a_1} & \longleftarrow & \mathbb{V}_{s_1} & \cdots \longrightarrow \mathbb{V}_{a_m} \longleftarrow \mathbb{V}_{s_m} \\ & \parallel & & \parallel & & \parallel & \parallel \\ \tilde{H}^0(\mathcal{L}(z, R)) : & \mathbb{U}_{s_0}^* & \longrightarrow & \mathbb{U}_{a_1}^* & \longleftarrow & \mathbb{U}_{s_1}^* & \cdots \longrightarrow \mathbb{U}_{a_m}^* \longleftarrow \mathbb{U}_{s_m}^* \end{array}$$

The above diagram commutes and thus $H_1(\mathcal{L}(z, |\mathcal{K}|)) \cong \tilde{H}^0(\mathcal{L}(z, R))$. By duality, $\tilde{H}^0(\mathcal{L}(z, R)) \cong \tilde{H}_0(\mathcal{L}(z, R))$ establishing the claim. \square

5.1 Extracting bars

We apply the procedure of Agarwal et al. [1] for extracting the bars out of a barcode graph that they compute for surfaces without boundary in \mathbb{R}^3 . Using the mergeable tree data structure of [14], this algorithm runs in $O(n \log n)$ time where R has a total of n edges and vertices. This algorithm in a sense mimics the definition of persistence pairs in different diagrams based on their types as elucidated in [3].

Figure 8 shows the sequence of operations applied to the barcode graph of a torus with a cylinder taken out to extract the bars. Applying the barcode extraction algorithm of [1] straightforwardly on the barcode graph provides a wrong answer if threading is not done. The barcode graph (Figure 8(b)) is threaded first which may create additional cycles (Figure 8(c)). Next, all vertices in a connected component of a level are contracted to a single vertex (Figure 8(d)). We apply the algorithm of [1] on this graph whose output are shown in Figure 8(e).

Modifying the bars: The algorithm of [1] can be viewed as successively peeling off paths from the graph R with endpoints at values s_i , $i = 1, \dots, m - 1$ since R has vertices at these levels only. An

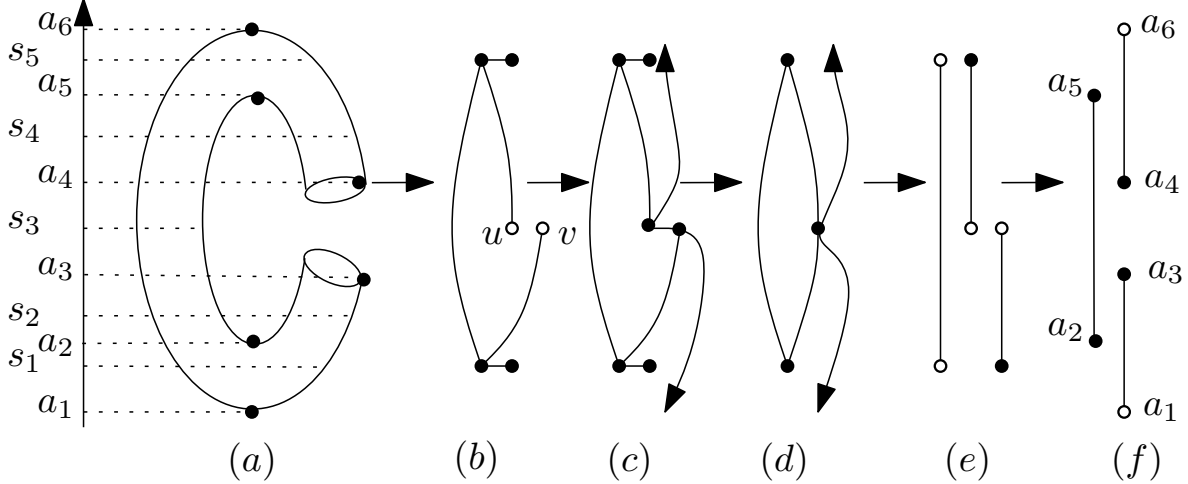


Figure 8: (a) Illustration of stages of barcode extraction on a torus with a cylinder taken out: six critical values a_1 - a_6 for H_1 ; (b) barcode graph; top two and bottom two vertices have the same values s_5 and s_1 respectively because of which they are drawn at the same levels; (c) threading connects the two open vertices at level s_3 ; (d) all vertices in the same connected component of a level are coalesced (here s_5, s_3, s_1); (e) extracted bar code; (f) modifying the bars to bring their ends at the critical levels while reversing their types.

endpoint of a bar which is peeled from a split or a merge vertex v is necessarily open (see bottom and top vertices of the bar between s_1 and s_5 in Figure 8(e)). The last copy of v that remains after all bars are peeled off of it remains to be closed (top and bottom vertices of the two shorter bars in Figure 8(e)). This accounts for the fact that a vertex contributes only a single component at its level. The other endpoints that arise from degree-1 vertices become open or closed according to the type of the vertex. After extracting all bars, we need to move their endpoints to the values a_i , $i = 1, \dots, m$ because the actual bars for \mathcal{K} have endpoints at the critical values.

Analogous to converting interval modules to bring their endpoints at critical values, we deploy the following conversions: (i) $(s_i, s_j) \Rightarrow [a_{i+1}, a_{j-1}]$, (ii) $[s_i, s_j] \Rightarrow (a_{i-1}, a_{j-1}]$, (iii) $(s_i, s_j] \Rightarrow [a_{i+1}, a_{j+1})$, (iv) $[s_i, s_j] \Rightarrow (a_{i-1}, a_{j+1})$ where open and closed brackets indicate the open and closed ends respectively. See Figure 8(f).

6 Reeb graph, barcode, and generators

Recall that $H_p(|\mathcal{K}|) \cong \mathring{B}_{p-1}(z, |\mathcal{K}|) \oplus \bar{B}_p(z, |\mathcal{K}|)$ where $\mathring{B}_{p-1}(z, |\mathcal{K}|)$ is generated by the open-open bars in $H_{p-1}(\mathcal{L}(z, |\mathcal{K}|))$ and $\bar{B}_p(z, |\mathcal{K}|)$ is generated by the closed-closed bars in $H_p(\mathcal{L}(z, |\mathcal{K}|))$ [4]. Our algorithm in the previous section produces the bars for the level set module $H_1(\mathcal{L}(z, |\mathcal{K}|))$ which allows us to obtain the closed-open and closed-closed bars for the sublevel set module (standard persistence) $H_1(\mathcal{SL}(z, |\mathcal{K}|))$ and the open-open bars for $H_2(\mathcal{SL}(z, |\mathcal{K}|))$. Although this completes the barcode for the second homology H_2 , we still need to compute the open-open bars for $H_0(\mathcal{L}(z, |\mathcal{K}|))$ to complete the barcode for $H_1(\mathcal{SL}(z, |\mathcal{K}|))$. We achieve this with the help of Reeb graphs.

Reeb graphs and barcodes. Given a continuous function $f : |\mathcal{K}| \rightarrow \mathbb{R}$, one defines the Reeb graph $Rb_f(|\mathcal{K}|)$ as the quotient space $|\mathcal{K}| \sim$ under the equivalence relation \sim where for any pair $x, y \in |\mathcal{K}| \times |\mathcal{K}|$, $x \sim y$ if and only if $f(x) = f(y)$ and the level set $f^{-1}(f(x))$ contains x and y in

the same connected component. We observe the following connection:

Proposition 8. $\mathring{B}_0(z, |\mathcal{K}|) \cong \mathring{B}_0(z, \text{Rb}_z(|\mathcal{K}|)) \cong H_1(\text{Rb}_z(|\mathcal{K}|))$.

Proof. By Theorem 1, $H_1(\text{Rb}_z(|\mathcal{K}|))$ is isomorphic to the direct sum $\mathring{B}_0(z, \text{Rb}_z(|\mathcal{K}|)) \oplus \bar{B}_1(z, \text{Rb}_z(|\mathcal{K}|))$. Consider an embedding that takes each vertex v of $\text{Rb}_z(|\mathcal{K}|)$ to points in \mathbb{R}^3 with the z -coordinate equaling the z -value of its pre-image in $|\mathcal{K}|$. Recall that an embedding necessarily maps distinct vertices to distinct points even if they have same z -values. An edge (v_i, v_j) is embedded as the line segment joining v_i and v_j . Observe that, because of the assumption that the height function z is proper for $|\mathcal{K}|$, the embedded Reeb graph, also denoted $\text{Rb}_z(|\mathcal{K}|)$ for simplicity, does not have any edge connecting two vertices with the same z -value. So, no edge lies entirely on any level set $z^{-1}(r)$ for any $r \in \mathbb{R}$. Assuming general position, no two edges cross.

A level set $z^{-1}(r)$ for $\text{Rb}_z(|\mathcal{K}|)$ has only isolated points where the edges intersect the plane $\pi_r : z = r$ transversely. Thus, there is no closed-closed bar in $H_1(\mathcal{L}(z, \text{Rb}_z(|\mathcal{K}|)))$ and hence the summand group $\bar{B}_1(z, \text{Rb}_z(|\mathcal{K}|))$ is trivial. It follows that

$$H_1(\text{Rb}_z(|\mathcal{K}|)) \cong \mathring{B}_0(z, \text{Rb}_z(|\mathcal{K}|)). \quad (6.5)$$

It follows from the definition of the Reeb graph that the 0-dimensional homology group of any level set $z^{-1}(r)$, $r \in \mathbb{R}$, for $|\mathcal{K}|$ and $\text{Rb}_z(|\mathcal{K}|)$ are isomorphic. Because of tameness of z , the same can be concluded for corresponding interval sets. Therefore, using our notation for interval sets between consecutive non-critical values, $\mathcal{K}_i^j = |\mathcal{K}|_{[s_i, s_j]}$ and $\mathcal{R}_i^j = \text{Rb}_z(|\mathcal{K}|)_{[s_i, s_j]}$, we have the following commutative diagram between the 0-dimensional level set zigzag persistence modules.

$$\begin{array}{ccccccccccc} H_0(\mathcal{L}(z, |\mathcal{K}|)) : & & H_0(\mathcal{K}_0^0) & \longrightarrow & H_0(\mathcal{K}_0^1) & \longleftarrow & H_0(\mathcal{K}_1^1) & \cdots & \longrightarrow & H_0(\mathcal{K}_{m-1}^m) & \longleftarrow & H_0(\mathcal{K}_m^m) \\ & & \parallel & & \parallel & & \parallel & & \parallel & & \parallel & \\ H_0(\mathcal{L}(z, \text{Rb}_z(|\mathcal{K}|))) : & & H_0(\mathcal{R}_0^0) & \longrightarrow & H_0(\mathcal{R}_0^1) & \longleftarrow & H_0(\mathcal{R}_1^1) & \cdots & \longrightarrow & H_0(\mathcal{R}_{m-1}^m) & \longleftarrow & H_0(\mathcal{R}_m^m) \end{array}$$

It follows that the two zigzag persistence modules $H_0(\mathcal{L}(z, |\mathcal{K}|))$ and $H_0(\mathcal{L}(z, \text{Rb}_z(|\mathcal{K}|)))$ are isomorphic. Therefore, $\mathring{B}_0(z, |\mathcal{K}|) \cong \mathring{B}_0(z, \text{Rb}_z(|\mathcal{K}|))$. Combining it with 6.5 we get the claim. \square

Computing H_1 -generators. Since $H_1(|\mathcal{K}|) \cong \mathring{B}_0(z, |\mathcal{K}|) \oplus \bar{B}_1(z, |\mathcal{K}|)$, we are required to generate a set of cycles whose classes form a basis for $\bar{B}_1(z, |\mathcal{K}|)$ and another set of cycles whose classes form a basis for $\mathring{B}_0(z, |\mathcal{K}|)$. A closed-closed bar $[a_i, a_j]$ in $H_1(\mathcal{L}(z, |\mathcal{K}|))$ is initiated by a cycle C at the level $z(v_i) = a_i$ for some vertex $v_i \in \mathcal{K}$. The homology class $[C]$ can be traced at each level set in the interval $[a_i, a_j]$ through the images and inverse images of the inclusion maps that produce the zigzag level set module $H_1(\mathcal{L}(z, |\mathcal{K}|))$. In other words, the class $[C] \in H_1(|\mathcal{K}|)$ represents the bar $[a_i, a_j]$. Therefore, the classes of cycles initiating closed-closed bar in the level set persistence module $H_1(\mathcal{L}(z, |\mathcal{K}|))$ generate the summand $\bar{B}_1(z, |\mathcal{K}|)$ of $H_1(|\mathcal{K}|)$.

We compute the cycles initiating the closed-closed bars as follows. A bar with a closed end results from a split that occurs during updating $\vec{G}_{s_{i-1}}$ to \vec{G}_{a_i} . A new primary cycle C is born in both cases of the split, which in turn initiates a new edge, say e in the barcode graph R . We can keep the cycle C associated with e in R . After we extract all bars from the barcode graph, we can determine the closed-closed bars and determine the cycles associated with their initiating edges. The drawback of this approach is that we may store many unnecessary cycles that initiate closed-open bars. To avoid this, we do not store the entire cycle beforehand initiating a bar from a

split. Instead, we store one directed edge d in the cycle C associated with the edge $e \in R$. We also remember the vertex of \mathcal{K} where the split has occurred. After we extract a closed-closed bar from the barcode graph, we obtain its associated directed edge d and the vertex $v \in \mathcal{K}$, and trace out the cycle containing d in the level set graph $\vec{G}_{z(v)}$. Taking into account the time to create R and the time to extract the cycles, this process cannot take more than $O(n \log n + k)$ time for all cycles to be output where k is their total size.

To compute the generating cycles for the summand $\mathring{B}_0(z, |\mathcal{K}|)$ of $H_1(|\mathcal{K}|)$, we use the second equivalence in Proposition 8. It implies that if a cycle basis for the Reeb graph is mapped injectively to a sub-basis of $H_1(|\mathcal{K}|)$, then that sub-basis indeed generate the summand $\mathring{B}_0(z, |\mathcal{K}|)$.

A cycle basis for $H_1(Rb_z(|\mathcal{K}|))$ can be computed in linear time by computing a spanning tree of $Rb_z(|\mathcal{K}|)$ treating it as a graph and then generating a cycle for each additional edge not in the spanning tree. The pre-image of these basis cycles w.r.t. the surjective map $\phi : |\mathcal{K}| \rightarrow Rb_z(|\mathcal{K}|)$ can be computed again in time linear in the total size of the basis cycles and the Reeb graph. These pre-images can also be deformed with a homotopy to the 1-skeleton of \mathcal{K} . The pre-images thus constructed form a cycle basis of the so called *vertical homology* group which is a summand group of $H_1(|\mathcal{K}|)$ [9]. Therefore, the pre-images form a cycle basis of $\mathring{B}_0(z, |\mathcal{K}|)$ which can be computed in $O(n \log n + k)$ time where k is the total size of all such cycles.

Computing H_2 -generators. We know $H_2(|\mathcal{K}|) \cong \mathring{B}_1(z, |\mathcal{K}|) \oplus \bar{B}_2(z, |\mathcal{K}|)$. Since $|\mathcal{K}| \subset \mathbb{R}^3$, $\bar{B}_2(z, |\mathcal{K}|)$ is trivial because there are no 2-cycles in any level set. Therefore, a set of independent 2-cycles in $|\mathcal{K}|$ that map bijectively to a basis of $\mathring{B}_1(z, |\mathcal{K}|)$ form a set of basis cycles for $H_2(|\mathcal{K}|)$.

Let C be any primary cycle that initiates an open-open bar extracted from the barcode graph R . As we already explained, the component Z of R providing the open-open bar in this case corresponds to a 2-cycle in $|\mathcal{K}|$. We can think R as a 1-complex linearly embedded in \mathbb{R}^3 with a vertex $v \in R$ having height $z(v)$. Then, there is a continuous surjective map $\xi : E \rightarrow R$ where $E \subseteq \mathbb{R}^3 \setminus |\mathcal{K}|$ is the union of all faces bounded by primary cycles over all levels $r \in \mathbb{R}$. In fact, R is the Reeb graph of the height function $z : E \rightarrow \mathbb{R}$. An open-open bar in $H_1(\mathcal{L}(z, |\mathcal{K}|))$ signifies a non-trivial class of $H_2(|\mathcal{K}|)$ by Theorem 1. The boundary of the inverse image $\xi^{-1}(Z)$ is a 2-cycle in \mathcal{K} , and it can be argued that it is independent of all such cycles. We can extract this 2-cycle by taking any complex triangle $t \in \mathcal{K}$ where $d \cdot \text{tri} = t$ for a directed edge $d \in C$, and then collecting all triangles that bound the void whose boundary includes t . This can be done by a simple depth-first walk in the adjacency data structure of \mathcal{K} . In total, after $O(n)$ -time walk, we collect all 2-cycles generating a basis for $H_2(|\mathcal{K}|)$.

Remark 6.1. *We observe that all of the computations that we described can be adapted to the case when \mathcal{K} is not necessarily pure. During the level set updates, we do not let those cycles $C_{\vec{F}}$ generate nodes and edges in the barcode graph R where the face F (on right) lies on the intersection of the level set with tetrahedra. This is because C_F cannot be a cycle basis element of the level set graph containing C_F . The edges and vertices of \mathcal{K} that are not adjacent to any triangle do not affect the computation of the level set zigzag persistence because they do not contribute to any primary cycle. However, they appear in the Reeb graph computation and may contribute to the open-open bars and hence infinite bars for the first homology.*

Remark 6.2. *Because of Proposition 5, if we run our level set update algorithm with the roles of primary and secondary cycles switched, we obtain the Reeb graph $Rb_z(|\mathcal{K}|)$ as the barcode graph. This gives an alternate $O(n \log n)$ -time algorithm for computing Reeb graphs.*

Proof of Theorem 2: We have already presented an $O(n \log n)$ time algorithm for computing $H_1(\mathcal{L}(z, |\mathcal{K}|))$. Notice that since level sets reside in planes, $H_i(\mathcal{L}(z, |\mathcal{K}|))$ is trivial for every $i > 1$. To

compute $H_0(\mathcal{L}(z, |\mathcal{K}|))$, we need to compute the connected components for level sets and track them. The barcode graph in this case is exactly the Reeb graph which can be computed in $O(n \log n)$ time by the algorithm of Parsa [16]. The barcode from this graph can be extracted as before. The basis cycles for $H_1(\mathcal{K})$ and $H_2(\mathcal{K})$ can be computed in $O(n \log n + k)$ time as described before. The case for $H_0(\mathcal{K})$ is trivial because connected components of \mathcal{K} can be determined in linear time by a depth first search in the 1-skeleton of \mathcal{K} .

Proof of Theorem 3: Because of Theorem 1, all bars of $H_1(\mathcal{SL}(z, |\mathcal{K}|))$ except the infinite bars that correspond to the open-open bars of $H_0(\mathcal{L}(z, |\mathcal{K}|))$ can be obtained by computing $H_1(\mathcal{L}(z, |\mathcal{K}|))$. We can use the first equivalence in Proposition 8 to derive the infinite bars of $H_1(\mathcal{SL}(z, |\mathcal{K}|))$ that correspond to the open-open bars of $H_0(\mathcal{L}(z, |\mathcal{K}|))$. We compute the Reeb graph $Rb_z(|\mathcal{K}|)$ in $O(n \log n)$ time using the algorithm by Parsa [16] and then extract the open-open bars of $H_0(z, Rb_z(|\mathcal{K}|))$ by running the $O(n \log n)$ -time extended persistence algorithm of Agarwal et al. [1] on it.

Since the level sets of z on $|\mathcal{K}|$ resides on planes, $H_2(\mathcal{L}(z, |\mathcal{K}|))$ is trivial. Hence, there is no finite bars in the standard persistence $H_2(\mathcal{SL}(z, |\mathcal{K}|))$. Also, for the same reason, the infinite bars of $H_2(\mathcal{SL}(z, |\mathcal{K}|))$ correspond only to the open-open bars of $H_1(\mathcal{L}(z, |\mathcal{K}|))$ which can be computed in $O(n \log n)$ time as we have described already. For $H_0(\mathcal{SL}(z, |\mathcal{K}|))$, we can compute the closed-open bars and closed-closed bars of $H_0(\mathcal{L}(z, |\mathcal{K}|))$ to obtain the finite and infinite bars respectively. We have already mentioned that this can be done by computing the Reeb graph and extracting bars from it in $O(n \log n)$ time.

7 Discussions

This work has spawned some interesting questions. The foremost among them is perhaps the question of being able to extend the presented approach toward computing the general persistence for simplicial complexes \mathcal{K} embedded in \mathbb{R}^3 while maintaining an $O(n \log n)$ time complexity. In this case, the level sets are embedded on surfaces of possibly high genus that can themselves change topology as sweep proceeds. It is not clear how to track a basis efficiently in this case. One possibility is to look for functions that can be transformed into height or height-like functions. Our approach applies to functions on $|\mathcal{K}|$ that can be extended continuously to entire \mathbb{R}^3 with level sets being points, planes, or spheres. The height function on $|\mathcal{K}|$ is one such function whose extension to entire \mathbb{R}^3 has such level sets which are merely planes.

With our approach we can compute the homology generators in $O(n \log n + k)$ time. With these generators, is it possible to compute the co-homology generators as well efficiently? This will allow annotating the simplices in the sense of [5] so that the homology class of any given cycle can be determined efficiently. One can compute a cohomology basis or an annotation from scratch in matrix multiplication time [5], but can we leverage the fact that a homology basis is already available?

We have exploited the embedding of a complex to compute the persistence efficiently for a special class of PL functions. Can we do the same for some interesting class of filtrations; or, even for a sequence of embedded complexes connected with simplicial maps? Finally, can our approach be extended to dimensions beyond \mathbb{R}^3 which will imply breaking through the current matrix multiplication time barrier for computing persistence in general?

Acknowledgment. The author acknowledges the support of the NSF grants CCF-1740761, CCF-1526513, and DMS-1547357 for this research.

References

- [1] P. K. Agarwal, H. Edelsbrunner, J. Harer, and Y. Wang. Extreme Elevation on a 2-Manifold. *Discrete Comput. Geom.*, **36** (2006), 553-572.
- [2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. The design and analysis of computer algorithms. Addison-Wesley Pub. Company., 1974.
- [3] P. Bendich, S. Cabello, and H. Edelsbrunner. A point calculus for interlevel set homology. *Pattern Recog. Lett.* **33** (2012), 1436-1444.
- [4] D. Burghilea and T. K. Dey. Topological Persistence for circle valued maps. *Discrete Comput. Geom.* **50** (2013), 69–98.
- [5] O. Busaryev, S. Cabello, C. Chen, T. K. Dey, and Y. Wang. Annotating simplices with a homology basis and its applications *13th Scandinavian Sympos. Workshop Algo. Theory (SWAT 2012)*, LNCS Vol. 7357 (2012), 189–200.
- [6] G. Carlsson, V. de Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. *Proc. 26th Annu. Sympos. Comput. Geom.* (2009), 247–256.
- [7] H. Y. Cheung, T. C. Kwok, and L. C. Lau. Fast matrix rank algorithms and applications. *J. ACM* **60**, (2013), 31:1-25.
- [8] D. Cohen-Steiner, H. Edelsbrunner and J. Harer. Extending persistence using Poincaré and Lefschetz duality. *Found. Comput. Math.* **9** (2009), 79-103.
- [9] T. K. Dey and Y. Wang. Reeb graphs: approximation and persistence. *Discrete Comput. Geom.* **49** (2013), 46–73.
- [10] T. K. Dey, K. Li, C. Luo, P. Ranjan, I. Safa, and Y. Wang. Persistent heat signature for pose-oblivious matching of incomplete models. *Comput. Graphics Forum.* **29**(5) (2010), 1545–1554.
- [11] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.* **28** (2002), 511–533.
- [12] H. Edelsbrunner and S. Parsa. On the computational complexity of Betti numbers: reductions from matrix rank. *Proc. ACM-SIAM Sympos. Discrete Alg.* (2014), 152-160.
- [13] P. Gabriel. Unzerlegbare Darstellungen I. *Manuscripta Mathematica*, **6** (1972), 71–103.
- [14] L. Georgiadis, R.E. Tarjan, and R.F. Werneck. Design of Data Structures for Mergeable Trees. *Proc. ACM-SIAM Sympos. Discrete Alg.* (2006), 349-403.
- [15] N. Milosavljevic, D. Morozov, and P. Skraba. Zigzag Persistent Homology in Matrix Multiplication Time. *Proc. Annu. Sympos. Comput. Geom.* (2011), 216-225.
- [16] S. Parsa. A deterministic $O(m \log m)$ time algorithm for the Reeb graph. *Discrete Comput. Geom.* **49**(4) (2013), 864-878.
- [17] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete Comput. Geom.* **33**(2005), 249–274.