Computing Zigzag Persistence on Graphs in Near-Linear Time

Tamal K. Dey
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Tao Hou
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Abstract

Graphs model real-world circumstances in many applications where they may constantly change to capture the dynamic behavior of the phenomena. Topological persistence which provides a set of birth and death pairs for the topological features is one instrument for analyzing such changing graph data. However, standard persistent homology defined over a growing space cannot always capture such a dynamic process unless shrinking with deletions is also allowed. Hence, *zigzag persistence* which incorporates both insertions and deletions of simplices is more appropriate in such a setting. Unlike standard persistence which admits nearly linear-time algorithms for graphs, such results for the zigzag version improving the general $O(m^\omega)$ time complexity are not known, where $\omega < 2.37286$ is the matrix multiplication exponent. In this paper, we propose algorithms for zigzag persistence on graphs which run in near-linear time. Specifically, given a filtration with m additions and deletions on a graph with n vertices and edges, the algorithm for 0-dimension runs in $O(m\log^2 n + m\log m)$ time and the algorithm for 1-dimension runs in $O(m\log^4 n)$ time. The algorithm for 0-dimension draws upon another algorithm designed originally for pairing critical points of Morse functions on 2-manifolds. The algorithm for 1-dimension pairs a negative edge with the *earliest* positive edge so that a 1-cycle containing both edges resides in all intermediate graphs. Both algorithms achieve the claimed time complexity via dynamic graph data structures proposed by Holm et al. In the end, using Alexander duality, we extend the algorithm for 0-dimension to compute the $(p-1)$-dimensional zigzag persistence for $\mathbb{R}^p$-embedded complexes in $O(m\log^2 n + m\log m + n\log n)$ time.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Algebraic topology

Keywords and phrases persistent homology, zigzag persistence, graph filtration, dynamic networks

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.30

Related Version *Full Version*: `https://arxiv.org/abs/2103.07353` [7]

Funding This research is supported by NSF grants CCF 1839252 and 2049010.

1 Introduction

Graphs appear in many applications as abstraction of real-world phenomena, where vertices represent certain objects and edges represent their relations. Rather than being stationary, graph data obtained in applications usually change with respect to some parameter such as time. A summary of these changes in a quantifiable manner can help gain insight into the data. Persistent homology [3, 10] is a suitable tool for this goal because it quantifies the life span of topological features as the graph changes. One drawback of using standard non-zigzag persistence [10] is that it only allows addition of vertices and edges during the change, whereas deletion may also happen in practice. For example, many complex systems such as social networks, food webs, or disease spreading are modeled by the so-called "dynamic networks" [13, 14, 19], where vertices and edges can appear and disappear at different time. A variant of the standard persistence called *zigzag persistence* [3] is thus a more natural tool in such scenarios because simplices can be both added and deleted. Given

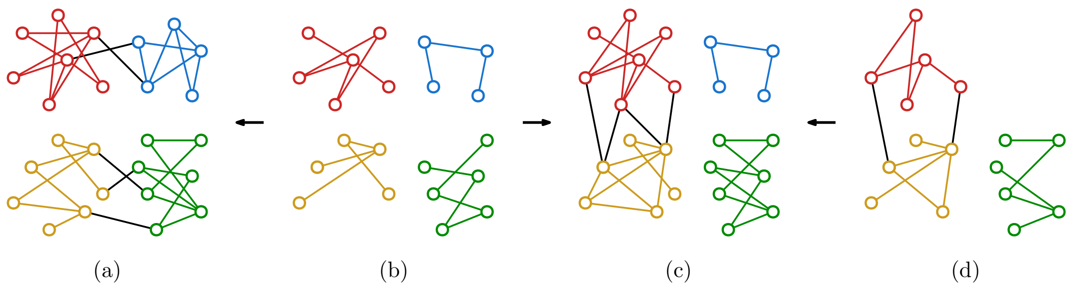


Figure 1 A sequence of graphs with four prominent clusters each colored differently. Black edges connect different clusters and forward (resp. backward) arrows indicate additions (resp. deletions) of vertices and edges. From (a) to (b), two clusters split; from (b) to (c), two clusters merge; from (c) to (d), one cluster disappears.

a sequence of graphs possibly with additions and deletions (formally called a *zigzag filtration*), zigzag persistence produces a set of intervals termed as *zigzag barcode* in which each interval registers the birth and death time of a homological feature. Figure 1 gives an example of a graph sequence in which clusters may split (birth of 0-dimensional features) or vanish/merge (death of 0-dimensional features). Moreover, addition of edges within the clusters creates 1-dimensional cycles and deletion of edges makes some cycles disappear. These births and deaths are captured by zigzag persistence.

Algorithms for both zigzag and non-zigzag persistence have a general-case time complexity of $O(m^\omega)$ [4, 10, 15, 16], where m is the length of the input filtration and $\omega < 2.37286$ is the matrix multiplication exponent [2]. For the special case of graph filtrations, it is well known that non-zigzag persistence can be computed in $O(m\,\alpha(m))$ time, where $\alpha(m)$ is the inverse Ackermann's function that is almost constant for all practical purposes [5]. However, analogous faster algorithms for zigzag persistence on graphs are not known. In this paper, we present algorithms for zigzag persistence on graphs with near-linear time complexity. In particular, given a zigzag filtration of length m for a graph with n vertices and edges, our algorithm for 0-dimension runs in $O(m\log^2 n + m\log m)$ time, and our algorithm for 1-dimension runs in $O(m\log^4 n)$ time. Observe that the algorithm for 0-dimension works for arbitrary complexes by restricting to the 1-skeletons.

The difficulty in designing faster zigzag persistence algorithms for the special case of graphs lies in the deletion of vertices and edges. For example, besides merging into bigger ones, connected components can also split into smaller ones because of edge deletion. Therefore, one cannot simply kill the younger component during merging as in standard persistence [10], but rather has to pair the *merge* and *departure* events with the *split* and *entrance* events (see Sections 3 for details). Similarly, in dimension one, deletion of edges may kill 1-cycles so that one has to properly pair the creation and destruction of 1-cycles, instead of simply treating all 1-dimensional intervals as infinite ones.

Our solutions are as follows: in dimension zero, we find that the $O(n\log n)$ algorithm by Agarwal et al. [1] originally designed for pairing critical points of Morse functions on 2-manifolds can be utilized in our scenario. We formally prove the correctness of applying the algorithm and use a *dynamic connectivity* data structure [12] to achieve the claimed complexity. In dimension one, we observe that a positive and a negative edge can be paired by finding the *earliest* 1-cycle containing both edges which resides in all intermediate graphs. We further reduce the pairing to finding the *max edge-weight* of a path in a minimum spanning forest. Utilizing a data structure for *dynamic minimum spanning forest* [12], we achieve the claimed time complexity. Section 4 details this algorithm.

Using Alexander duality, we also extend the algorithm for 0-dimension to compute $(p-1)$-dimensional zigzag for $\mathbb{R}^p$-embedded complexes. The connection between these two cases for non-zigzag persistence is well known [9, 18], and the challenge comes in adopting this duality to the zigzag setting while maintaining an efficient time budget. With the help of a *dual filtration* and an observation about faster void boundary reconstruction for $(p-1)$-*connected* complexes [8], we achieve a time complexity of $O(m\log^2 n + m\log m + n\log n)$.

All omitted proofs in this version appear in the full version [7].

2 Preliminaries

A *zigzag module* (or *module* for short) is a sequence of vector spaces

$$\mathcal{M} : V_0 \xleftrightarrow{\psi_0} V_1 \xleftrightarrow{\psi_1} \cdots \xleftrightarrow{\psi_{m-1}} V_m$$

in which each ψ_i is either a forward linear map $\psi_i : V_i \to V_{i+1}$ or a backward linear map $\psi_i : V_i \leftarrow V_{i+1}$. We assume vector spaces are over field $\mathbb{Z}_2$ in this paper. A module $\mathcal{S}$ of the form

$$\mathcal{S} : W_0 \xleftrightarrow{\phi_0} W_1 \xleftrightarrow{\phi_1} \cdots \xleftrightarrow{\phi_{m-1}} W_m$$

is called a *submodule* of $\mathcal{M}$ if each W_i is a subspace of V_i and each ϕ_i is the restriction of ψ_i. For an interval $[b,d] \subseteq [0,m]$, $\mathcal{S}$ is called an *interval submodule* of $\mathcal{M}$ over $[b,d]$ if W_i is one-dimensional for $i \in [b,d]$ and is trivial for $i \notin [b,d]$, and ϕ_i is an isomorphism for $i \in [b,d-1]$. It is well known [3] that $\mathcal{M}$ admits an *interval decomposition* $\mathcal{M} = \bigoplus_{\alpha\in\mathcal{A}} \mathcal{I}^{[b_\alpha,d_\alpha]}$ which is a direct sum of interval submodules of $\mathcal{M}$. The (multi-)set of intervals $\{[b_\alpha,d_\alpha] \mid \alpha \in \mathcal{A}\}$ is called the *zigzag barcode* (or *barcode* for short) of $\mathcal{M}$ and is denoted as $\mathsf{Pers}(\mathcal{M})$. Each interval in a zigzag barcode is called a *persistence interval*.

In this paper, we mainly focus on a special type of zigzag modules:

▶ **Definition 1** (Elementary zigzag module). *A zigzag module is called* ***elementary*** *if it starts with the trivial vector space and all linear maps in the module are of the three forms:* (i) *an isomorphism;* (ii) *an injection with rank 1 cokernel;* (iii) *a surjection with rank 1 kernel.*

A *zigzag filtration* (or *filtration* for short) is a sequence of simplicial complexes

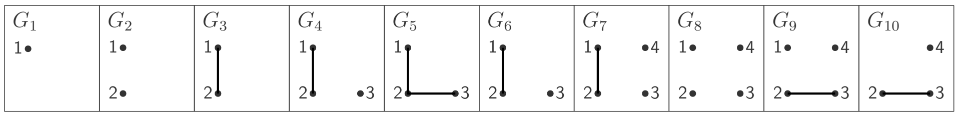
$$\mathcal{F} : K_0 \xleftrightarrow{\sigma_0} K_1 \xleftrightarrow{\sigma_1} \cdots \xleftrightarrow{\sigma_{m-1}} K_m$$

in which each $K_i \xleftrightarrow{\sigma_i} K_{i+1}$ is either a forward inclusion $K_i \hookrightarrow K_{i+1}$ with a single simplex σ_i added, or a backward inclusion $K_i \hookleftarrow K_{i+1}$ with a single σ_i deleted. When the σ_i's are not explicitly used, we drop them and simply denote $\mathcal{F}$ as $\mathcal{F} : K_0 \leftrightarrow K_1 \leftrightarrow \cdots \leftrightarrow K_m$. For computational purposes, we sometimes assume that a filtration starts with the empty complex, i.e., $K_0 = \varnothing$ in $\mathcal{F}$. Throughout the paper, we also assume that each K_i in $\mathcal{F}$ is a subcomplex of a fixed complex K; such a K, when not given, can be constructed by taking the union of every K_i in $\mathcal{F}$. In this case, we call $\mathcal{F}$ a filtration *of* K.

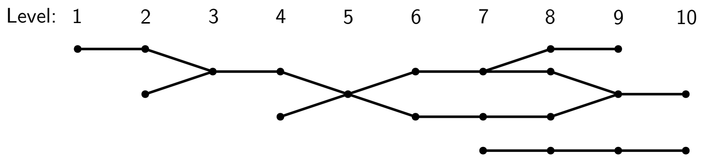
Applying the p-th homology with $\mathbb{Z}_2$ coefficients on $\mathcal{F}$, we derive the *p-th zigzag module of $\mathcal{F}$*

$$\mathsf{H}_p(\mathcal{F}) : \mathsf{H}_p(K_0) \xleftrightarrow{\varphi_0^p} \mathsf{H}_p(K_1) \xleftrightarrow{\varphi_1^p} \cdots \xleftrightarrow{\varphi_{m-1}^p} \mathsf{H}_p(K_m)$$

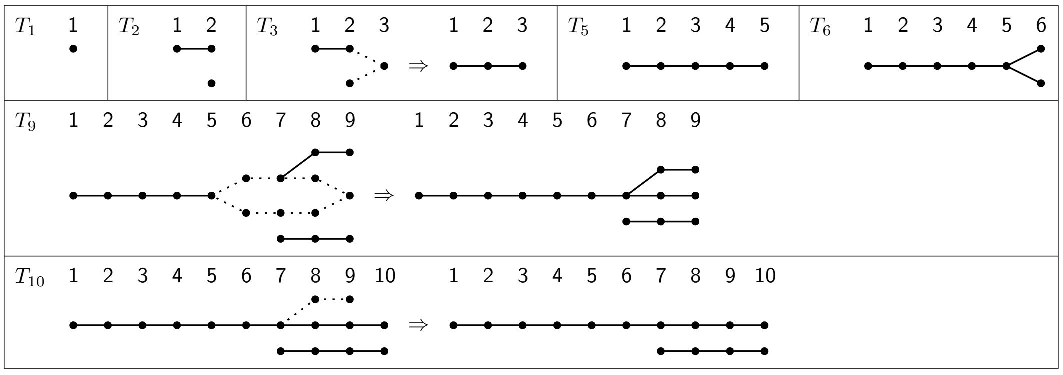
in which each φ_i^p is the linear map induced by the inclusion. In this paper, whenever $\mathcal{F}$ is used to denote a filtration, we use φ_i^p to denote a linear map in the module $\mathsf{H}_p(\mathcal{F})$. Note that $\mathsf{H}_p(\mathcal{F})$ is an elementary module if $\mathcal{F}$ starts with an empty complex. Specifically, we call $\mathsf{Pers}(\mathsf{H}_p(\mathcal{F}))$ the *p-th zigzag barcode* of $\mathcal{F}$.



(a) A zigzag filtration of graphs with 0-th barcode $\{[2, 2], [4, 4], [6, 8], [8, 9], [7, 10], [1, 10]\}$.



(b) The barcode graph for the filtration shown in Figure 2a.



(c) Barcode forests constructed in Algorithm 1 for the barcode graph in Figure 2b. For brevity, some forests are skipped. The horizontally arranged labels indicate the levels.

Figure 2 Examples of a zigzag filtration, a barcode graph, and barcode forests.

3 Zero-dimensional zigzag persistence

We present our algorithm for 0-th zigzag persistence[1] in this section. The input is assumed to be on graphs but note that our algorithm can be applied to any complex by restricting to its 1-skeleton. We first define the barcode graph of a zigzag filtration which is a construct that our algorithm implicitly works on. In a barcode graph, nodes correspond to connected components of graphs in the filtration and edges encode the mapping between the components:

▶ **Definition 2** (Barcode graph). *For a graph G and a zigzag filtration $\mathcal{F} : G_0 \leftrightarrow G_1 \leftrightarrow \cdots \leftrightarrow G_m$ of G, the **barcode graph** $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ of $\mathcal{F}$ is a graph whose vertices (preferably called **nodes**) are associated with a **level** and whose edges connect nodes only at adjacent levels. The graph $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ is constructively described as follows:*

- *For each G_i in $\mathcal{F}$ and each connected component of G_i, there is a node in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ at level i corresponding to this component; this node is also called a **level-i node**.*
- *For each inclusion $G_i \leftrightarrow G_{i+1}$ in $\mathcal{F}$, if it is forward, then there is an edge connecting a level-i node v_i to a level-$(i+1)$ node v_{i+1} if and only if the component of v_i maps to the component of v_{i+1} by the inclusion. Similarly, if the inclusion is backward, then v_i connects to v_{i+1} by an edge iff the component of v_{i+1} maps to the component of v_i.*

*For two nodes at different levels in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$, the node at the higher (resp. lower) level is said to be **higher** (resp. **lower**) than the other.*

[1] For brevity, henceforth we call p-dimensional zigzag persistence as p-*th* zigzag persistence.

▶ Remark 3. Note that some works [6, 14] also have used similar notions of barcode graphs.

Figure 2a and 2b give an example of a zigzag filtration and its barcode graph. Note that a barcode graph is of size $O(mn)$, where m is the length of $\mathcal{F}$ and n is the number of vertices and edges of G. Although we present our algorithm (Algorithm 1) by first building the barcode graph, the implementation does not do so explicitly, allowing us to achieve the claimed time complexity; see Section 3.1 for the implementation details. Introducing barcode graphs helps us justify the algorithm, and more importantly, points to the fact that the algorithm can be applied whenever such a barcode graph can be built.

Algorithm 1 Algorithm for 0-th zigzag persistence.

Given a graph G and a zigzag filtration $\mathcal{F} : \varnothing = G_0 \leftrightarrow G_1 \leftrightarrow \cdots \leftrightarrow G_m$ of G, we first build the barcode graph $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$, and then apply the pairing algorithm described in [1] on $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ to compute $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$. For a better understanding, we rephrase this algorithm which originally works on Reeb graphs:
The algorithm iterates for $i = 0, \ldots, m-1$ and maintains a **barcode forest** T_i, whose leaves have a one-to-one correspondence to level-i nodes of $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$. Like the barcode graph, each tree node in a barcode forest is associated with a level and each tree edge connects nodes at adjacent levels. For each tree in a barcode forest, the lowest node is the root. Initially, T_0 is empty; then, the algorithm builds T_{i+1} from T_i in the i-th iteration. Intervals for $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$ are produced while updating the barcode forest. (Figure 2c illustrates such updates.)
Specifically, the i-th iteration proceeds as follows: first, T_{i+1} is formed by copying the level-$(i+1)$ nodes of $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ and their connections to the level-i nodes, into T_i; the copying is possible because leaves of T_i and level-i nodes of $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ have a one-to-one correspondence; see transitions from T_5 to T_6 and from T_9 to T_{10} in Figure 2c. We further change T_{i+1} under the following events:

Entrance: One level-$(i+1)$ node in T_{i+1}, said to be **entering**, does not connect to any level-i node.

Split: One level-i node in T_{i+1}, said to be **splitting**, connects to two different level-$(i+1)$ nodes. For the two events so far, no changes need to be made on T_{i+1}.

Departure: One level-i node u in T_{i+1}, said to be **departing**, does not connect to any level-$(i+1)$ node. If u has splitting ancestors (i.e., ancestors which are also splitting nodes), add an interval $[j+1, i]$ to $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$, where j is the level of the highest splitting ancestor v of u; otherwise, add an interval $[j, i]$ to $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$, where j is the level of the root v of u. We then delete the path from v to u in T_{i+1}.

Merge: Two different level-i nodes u_1, u_2 in T_{i+1} connect to the same level-$(i+1)$ node. Tentatively, T_{i+1} may now contain a loop and is not a tree. If u_1, u_2 are in different trees in T_i, add an interval $[j, i]$ to $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$, where j is the level of the higher root of u_1, u_2 in T_i; otherwise, add an interval $[j+1, i]$ to $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$, where j is the level of the highest common ancestor of u_1, u_2 in T_i. We then glue the two paths from u_1 and u_2 to their level-j ancestors in T_{i+1}, after which T_{i+1} is guaranteed to be a tree.

No-change: If none of the above events happen, no changes are made on T_{i+1}.

At the end, for each root in T_m at a level j, add an interval $[j, m]$ to $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$, and for each splitting node in T_m at a level j, add an interval $[j+1, m]$ to $\mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$.

▶ Remark 4. The justification of Algorithm 1 is given in Section 3.2.

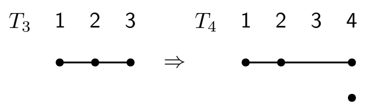


Figure 3 For the example in Figure 2, to form T_4, our implementation only adds a level-4 entering node, whereas the leaf in T_3 is not touched. Since the level of a leaf always equals the index of the barcode forest, the leaf at level 3 in T_3 automatically becomes a leaf at level 4 in T_4.

Figure 2c gives examples of barcode forests constructed by Algorithm 1 for the barcode graph shown in Figure 2b, where T_1 and T_2 introduce entering nodes, T_6 introduces a splitting node, and T_{10} introduces a departing node. In T_{10}, the departure event happens and the dotted path is deleted, producing an interval $[8, 9]$. In T_3 and T_9, the merge event happens and the dotted paths are glued together, producing intervals $[2, 2]$ and $[6, 8]$. Note that the glued level-i nodes are in different trees in T_3 and are in the same tree in T_9.

3.1 Implementation

Inserting (resp. deleting) a vertex in $\mathcal{F}$ simply corresponds to the entrance (resp. departure) event, whereas inserting (resp. deleting) an edge corresponds to the merge (resp. split) event only when connected components in the graph merge (resp. split). To keep track of the connectivity of vertices for each G_i, we use a *dynamic connectivity* data structure by Holm et al. [12], which we denote as $\mathbb{D}$. The data structure $\mathbb{D}$ dynamically updates the spanning forest of G_i when edges are inserted or deleted, and the connected component of a vertex of G_i can be identified by the tree in the spanning forest containing the vertex. The total time for querying and updating $\mathbb{D}$ is $O(m \log^2 n)$ [12], where m is the length of $\mathcal{F}$ and n is the number of vertices and edges of G. As mentioned, we do not explicitly build a barcode graph, but instead maintain a key-value map from connected components of G_i (i.e., identifiers of trees in $\mathbb{D}$'s spanning forest) to leaves of T_i. We update T_i to form T_{i+1} according to the changes of the connected components from G_i to G_{i+1}. We also only record the level of a non-leaf node in a barcode forest T_i because the level of a leaf always equals i. Note that at iteration i, a leaf in T_i may uniquely connect to a single leaf in T_{i+1}. In this case, we simply let the leaf in T_i automatically become a leaf in T_{i+1}; see Figure 3. The size of a barcode forest is then $O(m)$. As in [1], using the *mergeable trees* data structure [11], the traversal and updates of the barcode forest can be done in $O(m \log m)$ time. Therefore, the time complexity of Algorithm 1 is $O(m \log^2 n + m \log m)$. See the full version of the paper [7] for more details on the implementation of Algorithm 1.

3.2 Justification

In this subsection, we justify the correctness of Algorithm 1. For each entering node u in a T_i of Algorithm 1, there must be a single node in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ at the level of u with the same property. So we also have entering nodes in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$. Splitting and departing nodes in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ can be similarly defined.

We first prepare some standard notions and facts in zigzag persistence (Definition 5 and 7, Proposition 9) that help with our proofs. Some notions also appear in previous works in different forms; see, e.g., [15].

▶ **Definition 5** (Representatives). *Let $\mathcal{M} : V_0 \xleftrightarrow{\psi_0} \cdots \xleftrightarrow{\psi_{m-1}} V_m$ be an elementary zigzag module and $[s, t] \subseteq [1, m]$ be an interval. An indexed set $\{\alpha_i \in V_i \mid i \in [s, t]\}$ is called a set of* ***partial representatives*** *for $[s, t]$ if for each $i \in [s, t-1]$, $\alpha_i \mapsto \alpha_{i+1}$ or $\alpha_i \leftarrow\!\shortmid \alpha_{i+1}$ by ψ_i; it is called a set of* ***representatives*** *for $[s, t]$ if the following additional conditions are satisfied:*

1. *If $\psi_{s-1} : V_{s-1} \to V_s$ is forward with non-trivial cokernel, then α_s is not in $\mathsf{img}(\psi_{s-1})$; if $\psi_{s-1} : V_{s-1} \leftarrow V_s$ is backward with non-trivial kernel, then α_s is the non-zero element in $\mathsf{ker}(\psi_{s-1})$.*
2. *If $t < m$ and $\psi_t : V_t \leftarrow V_{t+1}$ is backward with non-trivial cokernel, then α_t is not in $\mathsf{img}(\psi_t)$; if $t < m$ and $\psi_t : V_t \to V_{t+1}$ is forward with non-trivial kernel, then α_t is the non-zero element in $\mathsf{ker}(\psi_t)$.*

*Specifically, when $\mathcal{M} := \mathsf{H}_p(\mathcal{F})$ for a zigzag filtration $\mathcal{F}$, we use terms **p-representatives** and **partial p-representatives** to emphasize the dimension p.*

▶ Remark 6. Let $\mathcal{F}$ be the filtration given in Figure 2a, and let α_8, α_9 be the sum of the component containing vertex 1 and the component containing vertex 2 in G_8 and G_9. Then, $\{\alpha_8, \alpha_9\}$ is a set of 0-representatives for the interval $[8, 9] \in \mathsf{Pers}(\mathsf{H}_0(\mathcal{F}))$.

▶ **Definition 7** (Positive/negative indices). *Let $\mathcal{M} : V_0 \xleftrightarrow{\psi_0} \cdots \xleftrightarrow{\psi_{m-1}} V_m$ be an elementary zigzag module. The set of **positive indices** of $\mathcal{M}$, denoted $\mathsf{P}(\mathcal{M})$, and the set of **negative indices** of $\mathcal{M}$, denoted $\mathsf{N}(\mathcal{M})$, are constructed as follows: for each forward $\psi_i : V_i \to V_{i+1}$, if ψ_i is an injection with non-trivial cokernel, add $i+1$ to $\mathsf{P}(\mathcal{M})$; if ψ_i is a surjection with non-trivial kernel, add i to $\mathsf{N}(\mathcal{M})$. Furthermore, for each backward $\psi_i : V_i \leftarrow V_{i+1}$, if ψ_i is an injection with non-trivial cokernel, add i to $\mathsf{N}(\mathcal{M})$; if ψ_i is a surjection with non-trivial kernel, add $i+1$ to $\mathsf{P}(\mathcal{M})$. Finally, add $\mathsf{rank}\, V_m$ copies of m to $\mathsf{N}(\mathcal{M})$.*

▶ Remark 8. For each $\psi_i : V_i \leftrightarrow V_{i+1}$ in Definition 7, if $i+1 \in \mathsf{P}(\mathcal{M})$, then $i \notin \mathsf{N}(\mathcal{M})$; similarly, if $i \in \mathsf{N}(\mathcal{M})$, then $i+1 \notin \mathsf{P}(\mathcal{M})$. Furthermore, if ψ_i is an isomorphism, then $i \notin \mathsf{N}(\mathcal{M})$ and $i+1 \notin \mathsf{P}(\mathcal{M})$.

Note that $\mathsf{N}(\mathcal{M})$ in Definition 7 is in fact a multi-set; calling it a set should not cause any confusion in this paper though. Also note that $|\mathsf{P}(\mathcal{M})| = |\mathsf{N}(\mathcal{M})|$, and every index in $\mathsf{P}(\mathcal{M})$ (resp. $\mathsf{N}(\mathcal{M})$) is the start (resp. end) of an interval in $\mathsf{Pers}(\mathcal{M})$. This explains why we add $\mathsf{rank}\, V_m$ copies of m to $\mathsf{N}(\mathcal{M})$ because there are always $\mathsf{rank}\, V_m$ number of intervals ending with m in $\mathsf{Pers}(\mathcal{M})$; see the example in Figure 2a where $\mathsf{rank}\, \mathsf{H}_0(G_{10}) = 2$.

▶ **Proposition 9.** *Let $\mathcal{M}$ be an elementary zigzag module and $\pi : \mathsf{P}(\mathcal{M}) \to \mathsf{N}(\mathcal{M})$ be a bijection. If every $b \in \mathsf{P}(\mathcal{M})$ satisfies that $b \le \pi(b)$ and the interval $[b, \pi(b)]$ has a set of representatives, then $\mathsf{Pers}(\mathcal{M}) = \{[b, \pi(b)] \mid b \in \mathsf{P}(\mathcal{M})\}$.*

Now we present several propositions leading to our conclusion (Theorem 14). Specifically, Proposition 10 states that a certain path in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ induces a set of partial 0-representatives. Proposition 11 lists some invariants of Algorithm 1. Proposition 10 and 11 support the proof of Proposition 13, which together with Proposition 9 implies Theorem 14.

From now on, G and $\mathcal{F}$ always denote the input to Algorithm 1. Since each node in a barcode graph represents a connected component, we also interpret nodes in a barcode graph as 0-th homology classes throughout the paper. Moreover, a path in a barcode graph from a node v to a node u is said to be *within level j and i* if for each node on the path, its level ℓ satisfies $j \le \ell \le i$; we denote such a path as $(v \rightsquigarrow u)_{[j,i]}$.

▶ **Proposition 10.** *Let v be a level-j node and u be a level-i node in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ such that $j < i$ and there is a path $(v \rightsquigarrow u)_{[j,i]}$ in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$. Then, there is a set of partial 0-representatives $\{\alpha_k \in \mathsf{H}_0(G_k) \mid k \in [j, i]\}$ for the interval $[j, i]$ with $\alpha_j = v$ and $\alpha_i = u$.*

Proof. We can assume that $(v \rightsquigarrow u)_{[j,i]}$ is a simple path because if it were not we could always find one. For each $k \in [j+1, i-1]$, let $w_1, \dots, w_r$ be all the level-k nodes on $(v \rightsquigarrow u)_{[j,i]}$ whose adjacent nodes on $(v \rightsquigarrow u)_{[j,i]}$ are at different levels. Then, let $\alpha_k = \sum_{\ell=1}^{r} w_\ell$. Also, let

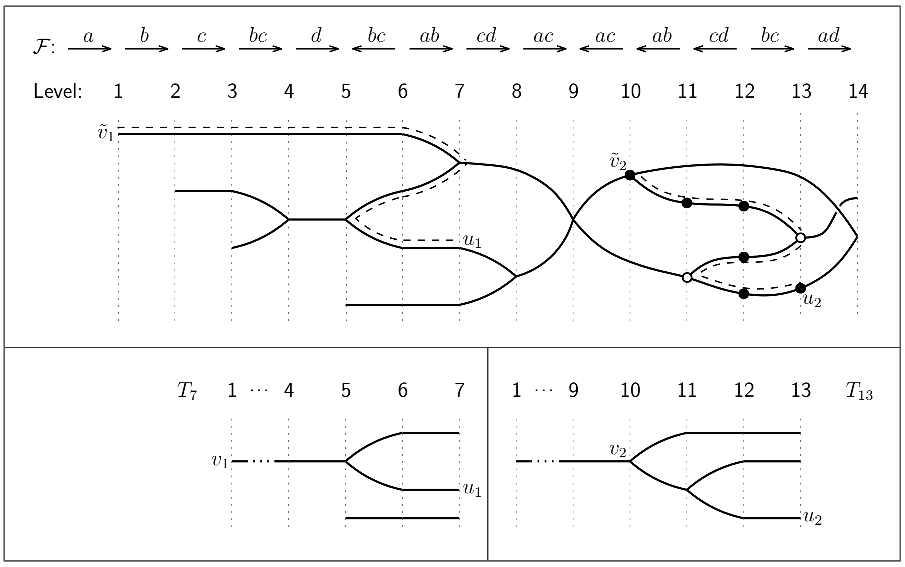


Figure 4 Illustration of invariants of Proposition 11. The top part contains a barcode graph with its filtration given (a, b, c, and d are vertices of the complex). The bottom contains two barcode forests.

$\alpha_j = v$ and $\alpha_i = u$. It can be verified that $\{\alpha_k \mid k \in [j,i]\}$ is a set of partial 0-representatives for $[j,i]$. See Figure 4 for an example of a simple path $(\tilde{v}_2 \rightsquigarrow u_2)_{[10,13]}$ (the dashed one) in a barcode graph, where the solid nodes contribute to the induced partial 0-representatives and the hollow nodes are excluded. ◀

For an i with $0 \leq i \leq m$, we define the *prefix* $\mathcal{F}^i$ of $\mathcal{F}$ as the filtration $\mathcal{F}^i : G_0 \leftrightarrow \cdots \leftrightarrow G_i$ and observe that $\mathbb{G}_{\mathrm{B}}(\mathcal{F}^i)$ is the subgraph of $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ induced by nodes at levels less than or equal to i. We call level-i nodes of $\mathbb{G}_{\mathrm{B}}(\mathcal{F}^i)$ as *leaves* and do not distinguish leaves in T_i and $\mathbb{G}_{\mathrm{B}}(\mathcal{F}^i)$ because they bijectively map to each other. It should be clear from the context though which graph or forest a particular leaf is in.

▶ **Proposition 11.** *For each $i = 0, \ldots, m$, Algorithm 1 maintains the following invariants:*

1. *There is a bijection η from trees in T_i to connected components in $\mathbb{G}_{\mathrm{B}}(\mathcal{F}^i)$ containing leaves such that a leaf u is in a tree Υ of T_i if and only if u is in $\eta(\Upsilon)$.*
2. *For each leaf u in T_i and each ancestor of u at a level j, there is a path $(\tilde{v} \rightsquigarrow u)_{[j,i]}$ in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$ where $\tilde{v}$ is a level-j node.*
3. *For each leaf u in T_i and each splitting ancestor of u at a level j, let $\tilde{v}$ be the unique level-j splitting node in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$. Then, there is a path $(\tilde{v} \rightsquigarrow u)_{[j,i]}$ in $\mathbb{G}_{\mathrm{B}}(\mathcal{F})$.*

▶ Remark 12. See Figure 4 for examples of invariant 2 and 3. In the figure, v_1 is a level-1 non-splitting ancestor of u_1 in T_7 and $\tilde{v}_1$ is a level-1 node in the barcode graph; v_2 is a level-10 splitting ancestor of u_2 in T_{13} and $\tilde{v}_2$ is the unique level-10 splitting node in the barcode graph. The paths $(\tilde{v}_1 \rightsquigarrow u_1)_{[1,7]}$ and $(\tilde{v}_2 \rightsquigarrow u_2)_{[10,13]}$ are marked with dashes.

▶ **Proposition 13.** *Each interval produced by Algorithm 1 admits a set of 0-representatives.*

Proof. Suppose that an interval is produced by the merge event at iteration i. We have the following situations:

- If the nodes u_1, u_2 in this event (see Algorithm 1) are in the same tree in T_i, let v be the highest common ancestor of u_1, u_2 and note that v is a splitting node at level j. Also note that u_1, u_2 are actually leaves in T_i and hence can also be considered as level-i

nodes in $\mathbb{G}_\mathrm{B}(\mathcal{F})$. Let $\tilde{v}$ be the unique level-j splitting node in $\mathbb{G}_\mathrm{B}(\mathcal{F})$. By invariant 3 of Proposition 11 along with Proposition 10, there are two sets of partial 0-representatives $\{\alpha_k \,|\, k \in [j,i]\}, \{\beta_k \,|\, k \in [j,i]\}$ for $[j,i]$ with $\alpha_j = \tilde{v}$, $\alpha_i = u_1$, $\beta_j = \tilde{v}$, and $\beta_i = u_2$. We claim that $\{\alpha_k + \beta_k \,|\, k \in [j+1,i]\}$ is a set of 0-representatives for the interval $[j+1,i]$. To prove this, we first note the following obvious facts: (i) $\{\alpha_k + \beta_k \,|\, k \in [j+1,i]\}$ is a set of partial 0-representatives; (ii) $\alpha_{j+1} + \beta_{j+1} \in \mathsf{ker}(\varphi_j^0)$; (iii) $\alpha_i + \beta_i$ is the non-zero element in $\mathsf{ker}(\varphi_i^0)$. So we only need to show that $\alpha_{j+1} + \beta_{j+1} \neq 0$. Let v_1, v_2 be the two level-$(j+1)$ nodes in $\mathbb{G}_\mathrm{B}(\mathcal{F})$ connecting to $\tilde{v}$. Then, α_{j+1} equals v_1 or v_2 and the same for β_{j+1}. To see this, we first show that α_{j+1} can only contain v_1, v_2. For contradiction, suppose instead that α_{j+1} contains a level-$(j+1)$ node x with $x \neq v_1$, $x \neq v_2$. Let $(\tilde{v} \rightsquigarrow u_1)_{[j,i]}$ be the simple path that induces $\{\alpha_k \,|\, k \in [j,i]\}$ as in Proposition 10 and its proof. Then, x is on the path $(\tilde{v} \rightsquigarrow u_1)_{[j,i]}$ and the two adjacent nodes of x on $(\tilde{v} \rightsquigarrow u_1)_{[j,i]}$ are at level j and $j+2$, in which we let y be the one at level j. Note that $y \neq \tilde{v}$ because x is not equal to v_1 or v_2. Since $(\tilde{v} \rightsquigarrow u_1)_{[j,i]}$ is within level j and i, y must be adjacent to another level-$(j+1)$ node distinct from x on $(\tilde{v} \rightsquigarrow u_1)_{[j,i]}$. Now we have that y is a level-j splitting node with $y \neq \tilde{v}$, contradicting the fact that $\mathbb{G}_\mathrm{B}(\mathcal{F})$ has only one level-j splitting node. The fact that α_{j+1} contains v_1 or v_2 but not both can be similarly verified. To see that $\alpha_{j+1} + \beta_{j+1} \neq 0$, suppose instead that $\alpha_{j+1} + \beta_{j+1} = 0$, i.e., $\alpha_{j+1} = \beta_{j+1}$, and without loss of generality they both equal v_1. Note that we can consider T_i as derived by contracting nodes of $\mathbb{G}_\mathrm{B}(\mathcal{F}^i)$ at the same level[2]. The fact that $\alpha_{j+1} = \beta_{j+1} = v_1$ implies that u_1, u_2 are descendants of the same child of v in T_i, contradicting the fact that v is the highest common ancestor of u_1, u_2. So we have that $\alpha_{j+1} + \beta_{j+1} \neq 0$.

- If u_1, u_2 are in different trees in T_i, then without loss of generality let u_1 be the one whose root v_1 is at the higher level (i.e., level j). As the root of u_1, the node v_1 must be an entering node, and the connected component of $\mathbb{G}_\mathrm{B}(\mathcal{F}^i)$ containing u_1 must have a single level-j node $\tilde{v}_1$. Then, by invariant 2 of Proposition 11 along with Proposition 10, there are two sets of partial 0-representatives $\{\alpha_k \,|\, k \in [j,i]\}, \{\beta_k \,|\, k \in [j,i]\}$ for $[j,i]$ with $\alpha_j = \tilde{v}_1$, $\alpha_i = u_1$, $\beta_j = \tilde{v}_2$, and $\beta_i = u_2$, where $\tilde{v}_2$ is a level-j node. We claim that $\{\alpha_k + \beta_k \,|\, k \in [j,i]\}$ is a set of 0-representatives for the interval $[j,i]$ and the verification is similar to the previous case where u_1 and u_2 are in the same tree.

For intervals produced by the departure events and at the end of the algorithm, the existence of 0-representatives can be similarly argued. ◀

▶ **Theorem 14.** *Algorithm 1 computes the 0-th zigzag barcode for a given zigzag filtration.*

Proof. First, we have the following facts: every level-j entering node in $\mathbb{G}_\mathrm{B}(\mathcal{F})$ introduces a $j \in \mathsf{P}(\mathsf{H}_0(\mathcal{F}))$ and uniquely corresponds to a level-j root in T_i for some i; every level-j splitting node in $\mathbb{G}_\mathrm{B}(\mathcal{F})$ introduces a $j+1 \in \mathsf{P}(\mathsf{H}_0(\mathcal{F}))$ and uniquely corresponds to a level-j splitting node in T_i for some i. Whenever an interval $[j,i]$ is produced in Algorithm 1, $i \in \mathsf{N}(\mathsf{H}_0(\mathcal{F}))$ and the entering or splitting node in T_i introducing j as a positive index either becomes a *regular* node (i.e., connecting to a single node on both adjacent levels) or is deleted in T_{i+1}. This means that j is never the start of another interval produced. At the end of Algorithm 1, the number of intervals produced which end with m also matches the rank of $\mathsf{H}_0(G_m)$. Therefore, intervals produced by the algorithm induce a bijection $\pi : \mathsf{P}(\mathsf{H}_0(\mathcal{F})) \to \mathsf{N}(\mathsf{H}_0(\mathcal{F}))$. By Proposition 9 and 13, our conclusion follows. ◀

[2] We should further note that this contraction is not done on the entire $\mathbb{G}_\mathrm{B}(\mathcal{F}^i)$ but rather on connected components of $\mathbb{G}_\mathrm{B}(\mathcal{F}^i)$ containing leaves.

4 One-dimensional zigzag persistence

In this section, we present an efficient algorithm for 1-st zigzag persistence on graphs. We assume that the input is a graph G with a zigzag filtration $\mathcal{F} : \varnothing = G_0 \xleftrightarrow{\sigma_0} G_1 \xleftrightarrow{\sigma_1} \cdots \xleftrightarrow{\sigma_{m-1}} G_m$ of G. We first describe the algorithm without giving the full implementation details. The key to the algorithm is a pairing principle for the positive and negative indices. Then, in Section 4.1, we make several observations which reduce the index pairing to finding the *max edge-weight* of a path in a minimum spanning forest, leading to an efficient implementation.

We notice that the following are true for every inclusion $G_i \xleftrightarrow{\sigma_i} G_{i+1}$ of $\mathcal{F}$ (recall that φ_i^1 denotes the corresponding linear map in the induced module $\mathsf{H}_1(\mathcal{F})$):

- If σ_i is an edge being added and vertices of σ_i are connected in G_i, then φ_i^1 is an injection with non-trivial cokernel, which provides $i+1 \in \mathsf{P}(\mathsf{H}_1(\mathcal{F}))$.
- If σ_i is an edge being deleted and vertices of σ_i are connected in G_{i+1}, then φ_i^1 is an injection with non-trivial cokernel, which provides $i \in \mathsf{N}(\mathsf{H}_1(\mathcal{F}))$.
- In all the other cases, φ_i^1 is an isomorphism and $i \notin \mathsf{N}(\mathsf{H}_1(\mathcal{F}))$, $i+1 \notin \mathsf{P}(\mathsf{H}_1(\mathcal{F}))$.

As can be seen from Section 3, computing $\mathsf{Pers}(\mathsf{H}_1(\mathcal{F}))$ boils down to finding a pairing of indices of $\mathsf{P}(\mathsf{H}_1(\mathcal{F}))$ and $\mathsf{N}(\mathsf{H}_1(\mathcal{F}))$. Our algorithm adopts this structure, where $\mathcal{U}_i$ denotes the set of unpaired positive indices at the *beginning* of each iteration i:

Algorithm 2 Algorithm for 1-st zigzag persistence on graphs.

```
$\mathcal{U}_0 := \varnothing$
for $i := 0, \ldots, m-1$:
    if $G_i \xrightarrow{\sigma_i} G_{i+1}$ provides $i+1 \in \mathsf{P}(\mathsf{H}_1(\mathcal{F}))$:
        $\mathcal{U}_{i+1} := \mathcal{U}_i \cup \{i+1\}$
    else if $G_i \xleftarrow{\sigma_i} G_{i+1}$ provides $i \in \mathsf{N}(\mathsf{H}_1(\mathcal{F}))$:
        pair $i$ with a $j_* \in \mathcal{U}_i$ based on the Pairing Principle below
        output an interval $[j_*, i]$ for $\mathsf{Pers}(\mathsf{H}_1(\mathcal{F}))$
        $\mathcal{U}_{i+1} := \mathcal{U}_i \setminus \{j_*\}$
    else:
        $\mathcal{U}_{i+1} := \mathcal{U}_i$
for each $j \in \mathcal{U}_m$:
    output an interval $[j, m]$ for $\mathsf{Pers}(\mathsf{H}_1(\mathcal{F}))$
```

Note that in Algorithm 2, whenever a positive or negative index is produced, σ_i must be an edge. One key piece missing from the algorithm is how we choose a positive index to pair with a negative index:

▶ **Pairing Principle for Algorithm 2.** *In each iteration i where $G_i \xleftarrow{\sigma_i} G_{i+1}$ provides $i \in \mathsf{N}(\mathsf{H}_1(\mathcal{F}))$, let J_i consist of every $j \in \mathcal{U}_i$ such that there exists a 1-cycle z containing both σ_{j-1} and σ_i with $z \subseteq G_k$ for every $k \in [j, i]$. Then, $J_i \neq \varnothing$ and Algorithm 2 pairs i with the* smallest *index j_* in J_i.*

▶ Remark 15. See Proposition 17 for a proof of $J_i \neq \varnothing$ claimed above.

▶ Remark 16. Algorithms for non-zigzag persistence [10, 20] always pair a negative index with the *largest* (i.e., youngest) positive index satisfying a certain condition, while Algorithm 2 pairs with the smallest one. This is due to the difference of zigzag and non-zigzag persistence and our particular condition that 1-cycles can never become boundaries in graphs. See [4, 15] for the pairing when assuming general zigzag filtrations.

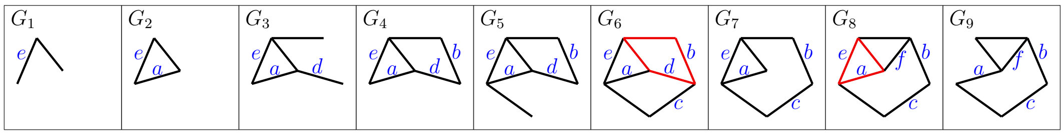


Figure 5 A zigzag filtration with 1-st barcode $\{[4,6],[2,8],[6,9],[8,9]\}$. For brevity, the addition of vertices and some edges are skipped.

Figure 5 gives an example of the pairing of the indices and their corresponding edges. In the figure, when edge d is deleted from G_6, there are three unpaired positive edges a, b, and c, in which b and c admit 1-cycles as required by the Pairing Principle. As the earlier edge, b is paired with d and an interval $[4,6]$ is produced. The red cycle in G_6 indicates the 1-cycle containing b and d which exists in all the intermediate graphs. Similar situations happen when e is paired with a in G_8, producing the interval $[2,8]$.

For the correctness of Algorithm 2, we first provide Proposition 17 which justifies the Pairing Principle and is a major step leading toward our conclusion (Theorem 18):

▶ **Proposition 17.** *At the beginning of each iteration i in Algorithm 2, for every $j \in \mathcal{U}_i$, there exists a 1-cycle z^i_j containing σ_{j-1} with $z^i_j \subseteq G_k$ for every $k \in [j,i]$. Furthermore, the set $\{z^i_j \mid j \in \mathcal{U}_i\}$ forms a basis of $\mathsf{Z}_1(G_i)$. If the iteration i produces a negative index i, then the above statements imply that there is at least one z^i_j containing σ_i. This z^i_j satisfies the condition that $z^i_j \subseteq G_k$ for every $k \in [j,i]$, $\sigma_{j-1} \in z^i_j$, and $\sigma_i \in z^i_j$, which implies that $J_i \neq \varnothing$ where J_i is as defined in the Pairing Principle.*

▶ **Theorem 18.** *Algorithm 2 computes the 1-st zigzag barcode for a given zigzag filtration on graphs.*

Proof. The claim follows directly from Proposition 9. For each interval $[j_*, i]$ produced from the pairing in Algorithm 2, by the Pairing Principle, there exists a 1-cycle z containing both σ_{j_*-1} and σ_i with $z \subseteq G_k$ for every $k \in [j_*, i]$. The cycle z induces a set of 1-representatives for $[j_*, i]$. For each interval produced at the end, Proposition 17 implies that such an interval admits 1-representatives. ◀

4.1 Efficient implementation

For every i and every $j \leq i$, define Γ^i_j as the graph derived from G_j by deleting every edge σ_k s.t. $j \leq k < i$ and $G_k \xleftarrow{\sigma_k} G_{k+1}$ is backward. For convenience, we also assume that Γ^i_j contains all the vertices of G. We can simplify the Pairing Principle as suggested by the following proposition:

▶ **Proposition 19.** *In each iteration i of Algorithm 2 where $G_i \xleftarrow{\sigma_i} G_{i+1}$ provides $i \in \mathsf{N}(\mathsf{H}_1(\mathcal{F}))$, the set J_i in the Pairing Principle can be alternatively defined as consisting of every $j \in \mathcal{U}_i$ s.t. $\sigma_i \in \Gamma^i_j$ and the vertices of σ_i are connected in Γ^{i+1}_j ($\sigma_i \notin \Gamma^{i+1}_j$ by definition).*

We now turn graphs in $\mathcal{F}$ into weighted ones in the following way: initially, $G_0 = \varnothing$; then, whenever an edge σ_i is added from G_i to G_{i+1}, the weight $w(\sigma_i)$ is set to i. For a path in a weighted graph, let the *max edge-weight* of the path be the maximum weight of its edges. Then, as can be seen from the full version [7] of the paper, the pairing boils down to computing the max edge-weight of the path connecting u, v in the minimum spanning forest (MSF) of G_{i+1}. For this, we utilize the *dynamic-MSF* data structure proposed by Holm et al. [12]. Assuming that n is the number of vertices and edges of G, the dynamic-MSF data structure supports the following operations:

- Return the identifier of a vertex's connected component in $O(\log n)$ time, which can be used to determine whether two vertices are connected.
- Return the max edge-weight of the path connecting any two vertices in the MSF in $O(\log n)$ time.
- Insert or delete an edge from the current graph (maintained by the data structure) and possibly update the MSF in $O(\log^4 n)$ amortized time.

We now present the full details of Algorithm 2:

Algorithm Algorithm 2: details.

Maintain a dynamic-MSF data structure $\mathbb{F}$, which consists of all vertices of G and no edges initially. Also, set $\mathcal{U}_0 = \varnothing$. Then, for each $i = 0, \ldots, m-1$, if σ_i is a vertex, do nothing; otherwise, do the following:

Case $G_i \xrightarrow{\sigma_i} G_{i+1}$**:** Check whether vertices of σ_i are connected in G_i by querying $\mathbb{F}$, and then add σ_i to $\mathbb{F}$. If vertices of σ_i are connected in G_i, then set $\mathcal{U}_{i+1} = \mathcal{U}_i \cup \{i+1\}$; otherwise, set $\mathcal{U}_{i+1} = \mathcal{U}_i$.

Case $G_i \xleftarrow{\sigma_i} G_{i+1}$**:** Delete σ_i from $\mathbb{F}$. If the vertices u, v of σ_i are found to be not connected in G_{i+1} by querying $\mathbb{F}$, then set $\mathcal{U}_{i+1} = \mathcal{U}_i$; otherwise, do the following:
  - Find the max edge-weight w_* of the path connecting u, v in the MSF of G_{i+1} by querying $\mathbb{F}$.
  - Find the smallest index j_* of $\mathcal{U}_i$ greater than $\max\{w_*, w(\sigma_i)\}$. (Note that we can store $\mathcal{U}_i$ as a red-black tree [5], so that finding j_* takes $O(\log n)$ time.)
  - Output an interval $[j_*, i]$ and set $\mathcal{U}_{i+1} = \mathcal{U}_i \setminus \{j_*\}$.

At the end, for each $j \in \mathcal{U}_m$, output an interval $[j, m]$.

Now we can see that Algorithm 2 has time complexity $O(m \log^4 n)$, where each iteration is dominated by the update of $\mathbb{F}$.

5 Codimension-one zigzag persistence of embedded complexes

In this section, we present an efficient algorithm for computing the $(p-1)$-th barcode given a zigzag filtration of an $\mathbb{R}^p$-embedded complex, by extending our algorithm for 0-dimension with the help of Alexander duality [17].

Throughout this section, $p \geq 2$, K is a simplicial complex embedded in $\mathbb{R}^p$, and $\mathcal{F} : \varnothing = K_0 \leftrightarrow \cdots \leftrightarrow K_m$ is a zigzag filtration of K. We call connected components of $\mathbb{R}^p \setminus |K|$ as *voids* of K or K*-voids*, to emphasize that only voids of K are considered in this section. The *dual graph* G of K has the vertices corresponding to the voids as well as the p-simplices of K, and has the edges corresponding to the $(p-1)$-simplices of K. The *dual filtration* $\mathcal{E} : G = G_0 \leftrightarrow G_1 \leftrightarrow \cdots \leftrightarrow G_m$ of $\mathcal{F}$ consists of subgraphs G_i of G such that: (i) all vertices of G dual to a K-void are in G_i; (ii) a vertex of G dual to a p-simplex is in G_i iff the dual p-simplex is *not* in K_i; (iii) an edge of G is in G_i iff its dual $(p-1)$-simplex is *not* in K_i.

One could verify that each G_i is a well-defined subgraph of G. We note the following: (i) inclusion directions in $\mathcal{E}$ are reversed; (ii) $\mathcal{E}$ is not exactly a zigzag filtration (because an arrow may introduce no changes) but can be easily made into one. Figure 6 gives an example in $\mathbb{R}^2$, in which we observe the following: whenever a $(p-1)$-cycle (i.e., 1-cycle) is formed in the primal filtration, a connected component in the dual filtration splits; whenever

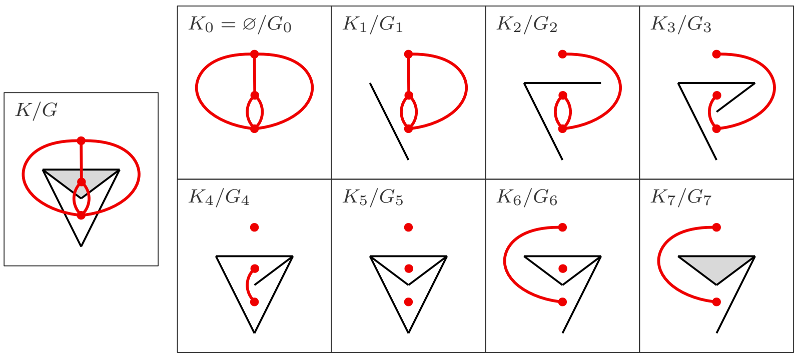


Figure 6 A zigzag filtration of an $\mathbb{R}^2$-embedded complex K and its dual filtration, where the dual graphs are colored red. For brevity, changes of vertices in the primal filtration are ignored.

a $(p-1)$-cycle is killed in the primal filtration, a connected component in the dual filtration vanishes. Intuitively, G_i encodes the connectivity of $\mathbb{R}^p \setminus |K_i|$, and so by Alexander duality, we have the following proposition:

▶ **Proposition 20.** $\mathsf{Pers}(\mathsf{H}_{p-1}(\mathcal{F})) = \mathsf{Pers}(\tilde{\mathsf{H}}_0(\mathcal{E}))$.

Proposition 21 indicates a way to compute $\mathsf{Pers}(\tilde{\mathsf{H}}_0(\mathcal{E}))$:

▶ **Proposition 21.** $\mathsf{Pers}(\tilde{\mathsf{H}}_0(\mathcal{E})) = \mathsf{Pers}(\mathsf{H}_0(\mathcal{E})) \setminus \{[0, m]\}$.

The above two propositions suggest a naive algorithm for computing $\mathsf{Pers}(\mathsf{H}_{p-1}(\mathcal{F}))$ using Algorithm 1. However, building the dual graph G requires reconstructing the void boundaries of K, which is done by a "walking" algorithm obtaining a set of $(p-1)$-cycles and then by a nesting test of these $(p-1)$-cycles [8, Section 4.1]. The running time of this process is $\Omega(n^2)$ where n is the size of K. To achieve the claimed complexity, we first define the following [8]:

▶ **Definition 22.** *In a simplicial complex X, two q-simplices σ, σ' are **q-connected** if there is a sequence $\sigma_0, \dots, \sigma_\ell$ of q-simplices of X such that $\sigma_0 = \sigma$, $\sigma_\ell = \sigma'$, and every σ_i, σ_{i+1} share a $(q-1)$-face. A maximal set of q-connected q-simplices of X is called a **q-connected component** of X, and X is **q-connected** if it has only one q-connected component.*

Based on the fact that void boundaries can be reconstructed without the nesting test for $(p-1)$-connected complexes in $\mathbb{R}^p$ [8], we restrict $\mathcal{F}$ to several $(p-1)$-connected subcomplexes of K and then take the union of the $(p-1)$-th barcodes of these restricted filtrations:

Algorithm 3 Algorithm for $(p-1)$-th zigzag persistence on $\mathbb{R}^p$-embedded complexes.

```
1. Compute the (p−1)-connected components $D^1, \dots, D^r$ of $K$.
2. For each $\ell = 1, \dots, r$, let

       $C^\ell = \mathrm{cls}(D^\ell) \cup \{\tau \in K \mid \tau$ is a $p$-simplex whose $(p-1)$-faces are in $D^\ell\}$

   and let $\mathcal{X}^\ell$ be a filtration of $C^\ell$ defined as

       $\mathcal{X}^\ell : K_0 \cap C^\ell \leftrightarrow K_1 \cap C^\ell \leftrightarrow \cdots \leftrightarrow K_m \cap C^\ell$

   Then, compute $\mathsf{Pers}(\mathsf{H}_{p-1}(\mathcal{X}^\ell))$ for each $\ell$.
3. Return $\bigcup_{\ell=1}^{r} \mathsf{Pers}(\mathsf{H}_{p-1}(\mathcal{X}^\ell))$ as the $(p-1)$-th barcode of $\mathcal{F}$.
```

In Algorithm 3, $\mathrm{cls}(D^\ell)$ denotes the closure of D^ℓ, i.e., the complex consisting of all faces of simplices in D^ℓ. For each ℓ, let n_ℓ be the number of simplices in C^ℓ; then, the dual graph of C^ℓ can be constructed in $O(n_\ell \log n_\ell)$ time because C^ℓ is $(p-1)$-connected [8]. Using Algorithm 1 to compute $\mathsf{Pers}(\mathsf{H}_{p-1}(\mathcal{X}^\ell))$ as suggested by the duality, the running time of Algorithm 3 is $O(m \log^2 n + m \log m + n \log n)$, where m is the length of $\mathcal{F}$ and n is the size of K.

The correctness of Algorithm 3 can be seen from the following proposition:

▶ **Proposition 23.** *The modules* $\mathsf{H}_{p-1}(\mathcal{F})$ *and* $\bigoplus_{\ell=1}^{r} \mathsf{H}_{p-1}(\mathcal{X}^\ell)$ *are isomorphic which implies that* $\mathsf{Pers}(\mathsf{H}_{p-1}(\mathcal{F})) = \bigcup_{\ell=1}^{r} \mathsf{Pers}(\mathsf{H}_{p-1}(\mathcal{X}^\ell))$.

References

1 Pankaj K. Agarwal, Herbert Edelsbrunner, John Harer, and Yusu Wang. Extreme elevation on a 2-manifold. *Discrete & Computational Geometry*, 36(4):553–572, 2006.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

3 Gunnar Carlsson and Vin de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.

4 Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, pages 247–256, 2009.

5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. URL: `http://mitpress.mit.edu/books/introduction-algorithms`.

6 Tamal K. Dey. Computing height persistence and homology generators in $\mathbb{R}^3$ efficiently. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2649–2662. SIAM, 2019.

7 Tamal K. Dey and Tao Hou. Computing zigzag persistence on graphs in near-linear time. *arXiv preprint*, 2021. `arXiv:2103.07353`.

8 Tamal K. Dey, Tao Hou, and Sayan Mandal. Computing minimal persistent cycles: Polynomial and hard cases. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2587–2606. SIAM, 2020.

9 Herbert Edelsbrunner and Michael Kerber. Alexander duality for functions: the persistent behavior of land and water and shore. In *Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry*, pages 249–258, 2012.

10 Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 454–463. IEEE, 2000.

11 Loukas Georgiadis, Haim Kaplan, Nira Shafrir, Robert E. Tarjan, and Renato F. Werneck. Data structures for mergeable trees. *ACM Transactions on Algorithms (TALG)*, 7(2):1–30, 2011.

12 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

13 Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, 2012.

14 Woojin Kim and Facundo Memoli. Stable signatures for dynamic graphs and dynamic metric spaces via zigzag persistence. *arXiv preprint*, 2017. `arXiv:1712.04064`.

15 Clément Maria and Steve Y. Oudot. Zigzag persistence via reflections and transpositions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 181–199. SIAM, 2014.

16 Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the Twenty-Seventh Annual Symposium on Computational Geometry*, pages 216–225, 2011.

17 James R. Munkres. *Elements of Algebraic Topology*. CRC Press, 2018.

18 Benjamin Schweinhart. *Statistical Topology of Embedded Graphs*. PhD thesis, Princeton University Press, 2015.

19 Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey. *arXiv preprint*, 2020. `arXiv:2005.07496`.

20 Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.