Swap, Shift and Trim to Edge Collapse a Filtration

Marc Glisse ✉ 🏠 ⓘ
Université Paris-Saclay, CNRS, Inria, Laboratoire de Mathématiques d'Orsay, 91405, Orsay, France

Siddharth Pritam ✉ ⓘ
School of Engineering, Department of Computer Science, Shiv Nadar University, Delhi NCR, India

— Abstract —

Boissonnat and Pritam introduced an algorithm to reduce a filtration of flag (or clique) complexes, which can in particular speed up the computation of its persistent homology. They used so-called *edge collapse* to reduce the input flag filtration and their reduction method required only the 1-skeleton of the filtration. In this paper we revisit the use of edge collapse for efficient computation of persistent homology. We first give a simple and intuitive explanation of the principles underlying that algorithm. This in turn allows us to propose various extensions including a zigzag filtration simplification algorithm. We finally show some experiments to better understand how it behaves.

1 Introduction

Efficient computation of persistent homology has been a central quest in Topological Data Analysis (TDA) since the early days of the field about 20 years ago. Given a filtration (a nested sequence of simplicial complexes), computation of persistent homology involves reduction of a boundary matrix, whose rows and columns are the simplices of the input filtration. Traditionally, there are two complementary lines of research that have been explored to improve the computation of persistent homology. The first approach led to improvement of the persistence algorithm (the boundary matrix reduction algorithm) and of its analysis, to efficient implementations and optimizations, and to a new generation of software [16, 4, 3, 18, 22, 26, 1]. The second and complementary approach is to reduce (or simplify) the input filtration to a smaller filtration through various geometric or topological techniques in an exact or approximate way and then compute the persistent homology of the smaller reduced filtration. This research direction has been intensively explored as well [21, 14, 10, 7, 27, 19, 11, 13].

Flag complexes and, in particular, the Vietoris-Rips complexes are an important class of simplicial complexes that are extensively used in TDA. Flag complexes are fully characterized by their graph (or 1-skeleton) and can thus be stored in a very compact way. Therefore, they are of great practical importance and are well studied theoretically. Various efficient codes and reduction techniques have been developed for those complexes [3, 27, 26, 1]. However, further progress have been made only recently by the work of Boissonnat and Pritam [5, 6]. Both works [5, 6] put forward preprocessing techniques, which reduce an input flag filtration (nested sequence of flag complexes) to a smaller flag filtration using only the 1-skeleton. The work in [5] uses a special type of collapse called strong collapse (removal of special vertices called dominated vertices), introduced by J. Barmak and E. Miniam [2]. In [6] they extend
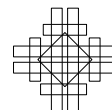
the notion of strong collapse to edge collapse (removal of special edges, called dominated edges) and use it for further filtration simplification which improves the performance by several orders of magnitude.

In this paper, we revisit the usage of edge collapse for efficient computation of persistent homology. We first give a simple and intuitive explanation of the principles underlying the algorithm proposed in [6]. We identify that an algorithm to edge collapse a filtration can be deconstructed as three fundamental operations: 1. *Swap* two edges having same filtration value, 2. *Shift* a dominated edge forward in the filtration and 3. *Trim* the very last dominated edge. This new approach allows us to propose various extensions, which we list below.

- **Backward:** We propose a backward reduction algorithm, which processes the edges of a flag filtration with decreasing filtration values. The algorithm in [6] processes edges one by one with increasing filtration values, i.e. in the forward direction. The backward processing results (shown experimentally) in faster reduction of the edges as it allows various operations like domination checks, computing the neighbourhood of an edge etc to be performed fewer times than in the forward algorithm of [6].
- **Parallel:** We propose a divide and conquer heuristic to further improve and semi-parallelize our backward reduction algorithm. Our approach is to subdivide the input filtration into two smaller sub-sequences (consisting of consecutive edges), we process these smaller sub-sequences in parallel and then merge the solutions of two sequences to form the solution of the complete sequence. The two sub-sequences can be further sub-divided and processed recursively in parallel.
- **Approximate:** With this simplified perspective a simple tweak in the backward algorithm allows us to have an approximate version of the reduction algorithm. There are two goals in mind behind an approximate version, first to speed up the algorithm, and second to obtain a smaller reduced sequence. We perform certain experiments to show how the approximate version performs on these two parameters.
- **Zigzag:** We provide a (parallelizable) reduction algorithm for a zigzag flag filtration, which is a sequence of flag complexes linked through inclusion maps in both forward and backward directions.

We note that we don't assume that all the vertices appear in the beginning of the filtration. That is the filtration values of vertices can be arbitrary as well.

2   Background

We briefly recall the basic notions like simplicial complexes, flag complexes, persistent homology and edge collapse. For more details on these topics please refer to [15, 17, 23].

Simplicial complex and simplicial map.   An **abstract simplicial complex** K is a collection of subsets of a non-empty finite set X, such that for every subset A in K, all the subsets of A are in K. We call an *abstract simplicial complex* simply a *simplicial complex* or just a *complex*. An element of K is called a **simplex**. An element of cardinality $k+1$ is called a k-simplex and k is called its **dimension**. Given a simplicial complex K, we denote its geometric realization as $|K|$. A simplex is called **maximal** if it is not a proper subset of any other simplex in K. A sub-collection L of K is called a **subcomplex** if it is a simplicial complex itself. An inclusion $\psi : K \xhookrightarrow{\sigma} K \cup \sigma$ of a single simplex σ is called **elementary**, otherwise, it's called **non-elementary**. An inclusion $\psi : K \hookrightarrow L$ between two complexes K and L induces a continuous map $|\psi| : |K| \to |L|$ between the underlying geometric realizations.

Flag complex and neighborhood. A complex K is a **flag** or a **clique** complex if, when a subset of its vertices forms a clique (i.e. any pair of vertices is joined by an edge), they span a simplex. It follows that the full structure of K is determined by its 1-skeleton (or graph) we denote by G. For a vertex v in G, the **open neighborhood** $N_G(v)$ of v in G is defined as $N_G(v) := \{u \in G \mid [uv] \in E\}$, where E is the set of edges of G. The **closed neighborhood** $N_G[v]$ is $N_G[v] := N_G(v) \cup \{v\}$. Similarly we define the closed and open neighborhood of an edge $[xy] \in E$, $N_G[xy]$ and $N_G(xy)$ as $N_G[xy] := N_G[x] \cap N_G[y]$ and $N_G(xy) := N_G(x) \cap N_G(y)$, respectively.

Persistent homology. A **sequence** of simplicial complexes $\mathcal{F} : \{K_1 \hookrightarrow K_2 \hookrightarrow \cdots \hookrightarrow K_m\}$ connected through inclusion maps is called a **filtration**. A filtration is a **flag filtration** if all the simplicial complexes K_i are flag complexes.

If we compute the homology groups of all the K_i, we get the sequence $\mathcal{P}(\mathcal{F}) : \{H_p(K_1) \overset{*}{\hookrightarrow} H_p(K_2) \overset{*}{\hookrightarrow} \cdots \overset{*}{\hookrightarrow} H_p(K_m)\}$. Here $H_p()$ denotes the homology group of dimension p with coefficients from a field $\mathbb{F}$ and $\overset{*}{\hookrightarrow}$ is the homomorphism induced by the inclusion map. $\mathcal{P}(\mathcal{F})$ is a sequence of vector spaces connected through the homomorphisms and it is called a **persistence module**. More formally, a *persistence module* $\mathbb{V}$ is a sequence of vector spaces $\{V_1 \to V_2 \to V_3 \to \cdots \to V_m\}$ connected with homomorphisms $\{\to\}$. A persistence module arising from a filtration captures the evolution of the topology of the sequence.

Any persistence module can be *decomposed* into a collection of intervals of the form $[i, j)$ [9]. The multiset of all the intervals $[i, j)$ in this decomposition is called the **persistence diagram** (PD) of the persistence module. An interval of the form $[i, j)$ corresponds to a homological feature (a "cycle") which appeared at i and disappeared at j. The PD completely characterizes the persistence module, that is, there is a bijective correspondence between the PD and the equivalence class of the persistence module [15, 28].

Two different persistence modules $\mathbb{V} : \{V_1 \to V_2 \to \cdots \to V_m\}$ and $\mathbb{W} : \{W_1 \to W_2 \to \cdots \to W_m\}$, connected through a set of homomorphisms $\phi_i : V_i \to W_i$ are **equivalent** if the ϕ_i are isomorphisms and the following diagram commutes [15, 12]. Equivalent persistence modules have the same interval decomposition, hence the same diagram.

$$
\begin{array}{ccccccccc}
V_1 & \longrightarrow & V_2 & \longrightarrow & \cdots & \longrightarrow & V_{m-1} & \longrightarrow & V_m \\
\downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2} & & & & \downarrow{\scriptstyle \phi_{m-1}} & & \downarrow{\scriptstyle \phi_m} \\
W_1 & \longrightarrow & W_2 & \longrightarrow & \cdots & \longrightarrow & W_{m-1} & \longrightarrow & W_m
\end{array}
$$

Edge collapse of a flag complex. In a flag complex K, we say that an edge $e = [ab]$, connecting vertices a and b, is **dominated** by a vertex v (different from a and b) if $N_G[e] \subseteq N_G[v]$. Removing e and all its cofaces from K defines a smaller flag complex K'. It has been proven in [6] that when e is dominated in K, the inclusion $K' \subset K$ induces an isomorphism between the homology groups of K' and K. This removal is called an **edge collapse**.

3   Swapping, shifting and trimming

In this Section, we show three simple and fundamental operations that preserve the persistence diagram of a flag filtration: 1. Swapping two edges with the same filtration value, 2. Shifting a dominated edge, and 3. Trimming a dominated edge at the end of the filtration. These operations can be combined to simplify a flag filtration.

Before we proceed, we will fix some notations. Let $\{t_1, t_2, \cdots, t_n\}$ be a finite index set where $t_i \in \mathbb{R}$ and $t_i < t_j$ for $i < j$. For convenience, we may consider $t_{n+1} = \infty$. With each t_i (called the *filtration value* or *grade*) we associate a graph G_{t_i} such that $G_{t_i} \hookrightarrow G_{t_{i+1}}$ is

an *inclusion*, (not necessarily elementary) of edges. The flag complex of G_{t_i} is denoted as $\overline{G}_{t_i}$ and we consider the associated flag filtration $\mathcal{F}: \overline{G}_{t_1} \hookrightarrow \overline{G}_{t_2} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_n}$. The edges in the set $E := \{e_1, e_2, \cdots e_m\}$ $(m \geq n)$ are thus indexed with an order compatible with the filtration values.

Swapping.   Inserting several edges at the same filtration value can be done in any order. We state this basic observation as the following lemma.

▶ **Lemma 1** (Swapping Lemma). *Given a flag filtration* $\{\overline{G}_{t_1} \hookrightarrow \overline{G}_{t_2} \cdots \hookrightarrow \overline{G}_{t_n}\}$, *such that* $G_{t_i} \hookrightarrow G_{t_{i+1}}$ *is a non-elementary inclusion. Then, the indices of the edges* $G_{t_{i+1}} \backslash G_{t_i}$ *could be assigned interchangeably. That is, swapping their order of insertion preserves the persistence diagram.*

Shifting.   In a filtration, insertion of a dominated edge does not bring immediate topological change. Therefore, its insertion can be shifted until the next grade and possibly even further.

▶ **Lemma 2** (Shifting Lemma). *Let* e *be a dominated edge in* G_{t_i} *inserted at grade* t_i. *Then, the insertion of* e *can be shifted by one grade to* t_{i+1} *without changing the persistence diagram. In other words, the persistence diagrams of the original flag filtration* $\mathcal{F} := \{\overline{G}_{t_1} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_i} \hookrightarrow \overline{G}_{t_{i+1}} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_n}\}$ *and the shifted filtration* $\{\overline{G}_{t_1} \hookrightarrow \cdots \hookrightarrow \overline{G_{t_i} \backslash e} \hookrightarrow \overline{G}_{t_{i+1}} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_n}\}$ *are equivalent.*

Proof. The proof follows from the commutativity of the following diagram, where all maps are induced by inclusions, and the fact that all vertical maps are isomorphisms.

$$
\begin{array}{ccccc}
H_p(\overline{G}_{t_{i-1}}) & \longrightarrow & H_p(\overline{G}_{t_i}) & \longrightarrow & H_p(\overline{G}_{t_{i+1}}) \\
\| & & \updownarrow{\scriptstyle |r_i|*} & & \| \\
H_p(\overline{G}_{t_{i-1}}) & \longrightarrow & H_p(\overline{G_{t_i} \backslash e}) & \longrightarrow & H_p(\overline{G}_{t_{i+1}})
\end{array}
$$

This implies that the persistence diagrams of the sequences $\{\overline{G}_{t_1} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_i} \hookrightarrow \overline{G}_{t_{i+1}} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_n}\}$ and $\{\overline{G}_{t_1} \hookrightarrow \cdots \hookrightarrow \overline{G_{t_i} \backslash e} \hookrightarrow \overline{G}_{t_{i+1}} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_n}\}$ are equivalent, see [6, Theorem 4] for more details. Here, $|r_i|*$ is the isomorphism between the homology groups induced by the retraction map (on the geometric realizations of the complexes) associated to the edge collapse.   ◀

After an edge is shifted to grade t_{i+1}, it can leap frog the edges inserted at grade t_{i+1} using the swapping lemma (Lemma 1) and can be checked for shifting to the next grade.

Trimming.   If the very last edge in the filtration is dominated then we can omit its inclusion. This is a special case of the shifting operation (Lemma 2) assuming that there is a graph G_∞ at infinity.

▶ **Lemma 3** (Trimming Lemma). *Let* $e \notin G_{t_{n-1}}$ *be a dominated edge in the graph* G_{t_n}. *Then, the persistence diagrams of the original sequence* $\mathcal{F} := \{\overline{G}_{t_1} \hookrightarrow \overline{G}_{t_2} \hookrightarrow \cdots \hookrightarrow \overline{G}_{t_n}\}$ *and the trimmed sequence* $\{\overline{G}_{t_1} \hookrightarrow \overline{G}_{t_2} \hookrightarrow \cdots \hookrightarrow \overline{G_{t_n} \backslash e}\}$ *are equivalent.*

Note that when shifting or trimming produces a sequence with identical consecutive graphs $G_{t_i} = G_{t_{i+1}}$, we can just drop index t_{i+1}.

▶ **Lemma 4** (Adjacency). *Let* e *be an edge in a graph* G *and let* e' *be a new edge with* $G' := G \cup e'$. *If* $N_G(e) = N_{G'}(e)$ *and* e *is dominated in* G, *then* e *is also dominated in* G'.

This is in particular the case if e and e' are not boundary edges of a common triangle in $\overline{G'}$. The above lemma is not strictly necessary, but it is useful to speed up algorithms.

4 Persistence simplification

In this Section, we will describe our new approach to use edge collapse to speed up the persistence computation. As mentioned before, the simplification process will be seen as a combination of the basic operations described in Section 3. This new perspective simplifies the design process and correctness proof of the algorithm. Along with this we achieve a significant improvement in the run-time efficiency as shown in Section 8. We first briefly look at the forward algorithm of [6] with this new point of view and then present the new approach called the *backward algorithm* [Algorithm 1]. Both algorithms take as input a flag filtration $\mathcal{F}$ represented as a sorted array E of edges (pairs of vertices) with their filtration value, and output a similar array E^c, sorted in the case the Forward Algorithm and unsorted for the Backward Algorithm, that represents a reduced filtration $\mathcal{F}^c$ that has the same persistence diagram as $\mathcal{F}$.

Forward algorithm. In the forward algorithm (the original one from [6]), the edges are processed in the order of the filtration in a streaming fashion. If a new edge is dominated, we skip its insertion and consider the next edge. If the next edge is dominated as well its insertion is skipped as well. Intuitively, the sequence of such dominated edges forms a train of dominated edges that we are moving to the right. When a new edge e is non-dominated (called *critical*), we output it, and also check what part of the train of dominated edges is allowed to continue to the right (shifted forward) and what part has to stop right there. For all the previously dominated edges (actually only those that are *adjacent* to e), we check if they are still dominated after adding the edge e. If an edge e' becomes critical, we output it with the same filtration value as e, and the following edges now have to cross both e and e' to remain in the train. We stop after processing the last edge, and the edges that are still part of the dominated train are discarded (trimmed).

Backward algorithm. The backward algorithm (Algorithm 1) considers edges in order of decreasing filtration value. Each edge e is considered once, delayed (shifted) as much as possible, then never touched again. We always implicitly swap edges so that while e is the edge considered, it is the last one inserted at its current filtration value, and compute its domination status there. If the edge is dominated, we shift it to the next filtration value, and iterate, swapping and checking for domination again at this new filtration value. If there is no next filtration value, we remove the edge (trimming). Once the edge is not dominated, we update its filtration value and output it. As an optimization, instead of moving the edge one grade at a time, we may jump directly to the filtration value of the next adjacent edge, since we know that moving across the other edges will preserve the domination (Lemma 4).

The main datastructure used here is a neighborhood map N. For each vertex u, it provides another map $N[u]$ from the adjacent vertices v_i to the filtration value $N[u][v_i]$ of edge uv_i. The two main uses of this map are computing the neighborhood of an edge uv at a time t (i.e. in the graph G_t) as $N_t[uv] = N_t[u] \cap N_t[v]$ (filtering out the edges of filtration value larger than t), and checking if such an edge neighborhood is included in the neighborhood of a vertex w at time t. While computing $N_t[uv]$, we also get as a side product the list of the future neighbors $F_t[uv]$, which we sort by filtration value. These operations can be done efficiently by keeping the maps sorted, or using hashtables. The information in N is symmetric, any operation we mention on $N[u][v]$ (removal or updating t) will also implicitly be done on $N[v][u]$. In this Section, we denote $t(e)$ the filtration value of $e \in E$, which is stored as $N[u][v]$ if $e = uv$. Note that even though E is sorted, since several edges may have the same filtration value, $G_{t(e)}$ may contain some edges that appear after e.

We now explain the precise computation of the reduced sequence of edges E^c. See Algorithm 1 for the pseudo-code. The main **for** loop on line 4 (called the backward loop) iterates over the edges in the sequence E by decreasing filtration values, i.e. in the *backward direction*, and checks whether or not the current edge e is dominated in the graph $G_{t(e)}$. If *not*, we insert e in E^c and keep its original filtration value $t(e)$. Else, e is dominated in $G_{t(e)}$, and we increase $t(e)$ to the smallest value $t' > t(e)$ where $N_{t(e)}[e] \subsetneq N_{t'}[e]$. We can then iterate (**goto** on line 12), check if the edge is still dominated at its new filtration value t', etc. When the edge stops being dominated, we insert it in E^c with its new $t(e)$ and update $t(e)$ in the neighborhood map N. If the smallest value $t' > t(e)$ does not actually exist, we remove the edge from the neighborhood map and do *not* insert it in E^c.

■ **Algorithm 1** Core flag filtration backward algorithm.

 1: **procedure** CORE-FLAG-FILTRATION(E)
 2:     **input :** set of edges E sorted by filtration value
 3:     $E^c \leftarrow \emptyset$
 4:     **for** $e \in E$ **do**   ▷ In non-increasing order of $t(e)$
 5:         Compute $N_{t(e)}(e)$ and $F_{t(e)}(e)$
 6:         **for** $w \in N_{t(e)}(e)$ **do**
 7:             Test if w dominates e at $t(e)$
 8:         **end for**
 9:         **if** e is dominated in $G_{t(e)}$ **then**
10:             **if** $F_{t(e)}(e)$ is empty **then**
11:                 Remove $N[u][v]$ ▷ Trimming.
12:                 **go to** 23 (next edge)
13:             **else**   ▷ Shift and Swap.
14:                 $t' \leftarrow$ filtration of the first element of $F_{t(e)}(e)$
15:                 Move from $F_{t(e)}(e)$ to $N_{t(e)}(e)$ the vertices that become neighbors of e at t'
16:                 $N[u][v] = t(e) \leftarrow t'$
17:                 **go to** 6
18:             **end if**
19:         **else**
20:             Insert $\{e, t(e)\}$ in E^c
21:             **go to** 23 (next edge)
22:         **end if**
23:     **end for**
24:     **return** E^c ▷ E^c is the 1-skeleton of the core flag filtration.
25: **end procedure**

▶ **Theorem 5** (Correctness). *Let $\mathcal{F}$ be a flag filtration, and $\mathcal{F}^c$ the reduced filtration produced by Algorithm 1. $\mathcal{F}$ and $\mathcal{F}^c$ have the same persistence diagram.*

Proof. The proof is based on the observation that the algorithm inductively performs the elementary operations from Section 3: either it trims the very last edge of the current sequence (Line 11) or shifts and swaps a dominated edge forward to get a new sequence. Then the result follows using Lemmas 1–3 inductively. The only subtlety is around Line 15, where instead of simply performing one shift to the next filtration value, we perform a whole sequence of operations. We first shift e to the next filtration value t' (and implicitly swap e with the other edges of filtration value t'). As long as we have not reached the first element of

$F_{t(e)}(e)$, we know that shifting has not changed the neighborhood of e and thus by Lemma 4 the fact that e is dominated. We can then safely keep shifting (and swapping) until we reach that first element of $F_{t(e)}(e)$.   ◄

Complexity.   We write n_e for the total number of edges and k for the maximum degree of a vertex in G_{t_n}. The main loop of the procedure, Line 4 of Algorithm 1, is executed n_e times. Nested, we loop (in the form of go to 6) on the elements of $F_{t(e)}(e)$, of which there are at most k. For each of those elements, on Line 6, we iterate on $N_{t(e)}(e)$, which has size at most k. Finally, testing if a specific vertex dominates a specific edge amounts to checking if one set is included in another, which takes linear time in k for sorted sets or hash tables. The other operations are comparatively of negligible cost. Sorting $F_{t(e)}(e)$ on Line 5 takes time $k \log k = o(k^2)$. Line 15 may take time $k \log k$ depending on the datastructure, $O(k^2)$ in any case. This yields a complexity of $O(n_e k^3)$.

5   Parallelisation

Delaying the insertion of an edge until the next grade, and possibly swapping it, is a very local operation. As such, there is no problem doing several of them in parallel as long as they are in disjoint intervals of filtration values. We exploit this observation and further optimize our algorithm by parallelizing a significant part of the computation using a divide and conquer approach.

To describe the parallel approach, let us use the same notations t_i, G_{t_i}, $\mathcal{F}$, $G_{\mathcal{F}}$ and E as in Section 3. To make things simpler, we assume that all edges have distinct filtration values. We subdivide the given input edge set $E := \{e_1, e_2, \cdots e_n\}$ of size n into two smaller halves: the left half $E_l := \{e_1, e_2, \cdots e_{n/2}\}$ and the right half $E_r := \{e_{n/2+1}, e_{n/2+2}, \cdots e_n\}$ of roughly the same size. We will describe a version of the algorithm based on the backward algorithm, but the same could be done with the forward algorithm, or a mixture of both.

We first apply the backward algorithm to E_l normally (*left call*), which produces a reduced E_l^c. We also remember the list of all edges that were removed in this procedure: $E_l^r := E_l \setminus E_l^c$. Independently (in parallel), we apply the backward algorithm to E (*right call*), but stop after processing all the edges of E_r on Line 4 of Algorithm 1. In a final sequential merging step, we resume the right call, processing only the edges of E_l^r, as if they all had the same initial filtration value $t_{n/2+1}$. The subdivision can obviously be applied recursively to increase the parallelism.

▶ **Lemma 6.** *The parallel algorithm produces exactly the same output as the sequential algorithm, and is thus correct.*

Proof. The right call and the sequential algorithm start by handling the edges of E_r in exactly the same way. When we reach the edges of E_l, for each edge e, there are two cases. Either the sequential algorithm shifts e no further than $t_{n/2}$, in which case the left call does the same. Or the sequential algorithms shifts e further (possibly all the way to removing it), then shifting to $t_{n/2+1}$ is handled by the left call, while the rest of the shift happens in the merging step.   ◄

6   Approximation

Another interesting extension is an approximate version that gives a diagram within bottleneck distance ϵ of the true diagram (or some other similar criterion). Since the Rips filtration is often used as an approximation of the Čech filtration, an additional error is often acceptable.

If an edge is non-dominated for a short range of filtration values and becomes dominated again afterwards, it is tempting to skip the non-dominated region and keep delaying this edge. However, if we are not careful, the errors caused by these skips may add up and result in a diagram that is far from the original. The simplest idea would be to round all filtration values to the nearest multiple of ϵ before running the backward algorithm (as in [5]). However, we can do a little better.

We describe here one safe approximation algorithm, based on the backward algorithm. When considering a new edge e, instead of checking if it is dominated at its original position $t(e)$, we start checking ϵ later, at filtration value $t(e)+\epsilon$. If it is dominated, we resume normal processing from there. However, if the edge is not dominated ϵ after its original insertion time, we keep it at its original position and avoid uselessly shifting the whole sequence.

▶ **Lemma 7.** *The resulting module is ϵ-interleaved[1] with the original one.*

Proof. Consider the set D of edges that are delayed by this algorithm, and C the edges that are kept at their original position. Starting from the original sequence, we can delay all the edges of D by exactly ϵ. The flag filtration defined by this delayed sequence is obviously $(0, \epsilon)$-interleaved with the original. We now run the regular backward algorithm on this sequence, with the difference that the edges in C are handled as if they were never dominated. The output filtration has the same persistence diagram as the delayed sequence, which is at distance at most ϵ from the diagram of the original filtration. The key observation here is that this output filtration is precisely what the approximation algorithm produces.   ◀

7 Zigzag persistence

The filtrations we have discussed so far are increasing sequences of complexes. There exists a more general type of filtration, called zigzag filtration [8, 20] $\mathcal{Z} : K_1 \hookrightarrow K_2 \hookleftarrow K_3 \hookrightarrow \cdots \hookrightarrow K_n$. Here consecutive complexes are still related by an inclusion, but the direction of this inclusion may be different for every consecutive pair. In other words, the complex K_i is obtained by either *inclusion* or *removal* of simplices from the previous complex K_{i-1}. Persistence diagrams can still be defined for these filtrations. Again, in this paper, we are only interested in flag zigzag filtrations, where each complex is a clique complex. For a flag zigzag filtration the underlying graphs are related through inclusion or removal of edges. We show that edge collapse can again be used for simplification of such sequences.

In the case of standard persistence (explained in Section 4) the goal of the simplification process was to shift as many dominated edges as possible towards the end of a filtration and then trim them. For a zigzag flag filtration there are several possible ways to simplify it: 1. If a dominated edge is included and is never removed, then as usual we try to *shift* it towards the end and trim it. 2. If an edge is included and removed both as dominated, then we try to *shift* the inclusion till its removal and then annihilate both operations. 3. If an edge is included as non-dominated but later removed as dominated then we try to shift its removal towards the right till the end or its re-insertion. 4. A zigzag filtration is symmetric and a removal is an inclusion from the opposite direction, therefore, we can shift dominated removals towards the beginning and perform symmetric operations as in 2.

The 3rd method reduces the number of events at the cost of a slightly bigger complex, which may or may not be preferred over a more "zigzaggy" filtration, so we do not use it in the default algorithm.

---

[1] See [9] for a definition of interleaving.

With more ways to simplify, the simplification process of a zigzag flag filtration is more delicate compared to the usual filtration. And it has some subtleties, first, can we shift a dominated edge inclusion across an edge removal? We show that (in Lemma 8), a dominated edge e can be shifted across an edge removal if e is also dominated after the edge removal. Resolving the first issue leads us to the question, how to index (order) inclusions and removals of the same grade? In practice, this situation is not common and two complexes at consecutive grades are linked through either inclusions or removals. Therefore, we adopt the following representation for a zigzag flag filtration.

We will use the same notations t_i, G_{t_i}, $\overline{G}_{t_i}$ and E as in Section 3. We represent a zigzag filtration in slightly more general way as $\mathcal{Z} : \overline{G}_{t_1} \leftarrow \overline{G}_{t'_1} \hookrightarrow \overline{G}_{t_2} \leftarrow \cdots \hookrightarrow \overline{G}_{t_{i-1}} \leftarrow \overline{G}_{t'_{i-1}} \hookrightarrow \overline{G}_{t_i} \leftarrow \overline{G}_{t'_i} \hookrightarrow \overline{G}_{t_{i+1}}, \cdots \hookrightarrow \overline{G}_{t_n}$. Here $G_{t'_i}$ is an intermediate graph at grade t_i. In a usual zigzag, $\overline{G}_{t'_i}$ is equal to either $\overline{G}_{t_i}$ or $\overline{G}_{t_{i+1}}$ depending on the direction of the arrow. Note that the standard zigzag algorithm still applies to this version.

Below, we provide a sufficient condition to shift and swap an inclusion with removal.

▶ **Lemma 8** (Zigzag Shifting-Swapping Lemma). *Let e be an edge inserted at t_i, $e \in G_{t'_i}$ and dominated in both graphs G_{t_i} and $G_{t'_i}$. Then the persistence diagrams of the original zigzag flag filtration $\{ \cdots \leftarrow \overline{G}_{t'_{i-1}} \overset{e}{\hookrightarrow} \overline{G}_{t_i} \leftarrow \overline{G}_{t'_i} \hookrightarrow \overline{G}_{t_{i+1}} \leftarrow \cdots \}$ and the shifted-swapped sequence $\{ \cdots \leftarrow \overline{G}_{t'_{i-1}} \hookrightarrow \overline{G_{t_i} \setminus e} \leftarrow \overline{G_{t'_i} \setminus e} \overset{e}{\hookrightarrow} \overline{G}_{t_{i+1}} \leftarrow \cdots \}$ are equivalent. That is, the grade of e can be shifted to t_{i+1}.*

Proof. The proof follows through a similar argument as Lemma 2. All three squares in the following diagram commute as all the maps are induced by inclusions. Note that the top left and the bottom right horizontal maps can be induced by non-elementary inclusions.

$$
\begin{array}{ccccccc}
H_p(\overline{G}_{t'_{i-1}}) & \overset{|e|^*}{\hookrightarrow} & H_p(\overline{G}_{t_i}) & \longleftarrow & H_p(\overline{G}_{t'_i}) & \longrightarrow & H_p(\overline{G}_{t_{i+1}}) \\
\| & & |e|^* \uparrow \downarrow |rt|* & & |e|^* \uparrow \downarrow |rt|* & & \| \\
H_p(\overline{G}_{t'_{i-1}}) & \longrightarrow & H_p(\overline{G_{t_i} \setminus e}) & \longleftarrow & H_p(\overline{G_{t'_i} \setminus e}) & \overset{|e|^*}{\hookrightarrow} & H_p(\overline{G}_{t_{i+1}})
\end{array}
$$

Since the vertical maps are either equalities or isomorphisms induced by the inclusion of the dominated e ($|rt|$ is the corresponding retraction map associated with the collapse), the result follows immediately. That is, the shift of e to the grade t_{i+1} preserves the diagram. ◀

Note that in the above lemma, the hypothesis that edge e should be dominated in the graph $G_{t'_i}$ is necessary as shown in Figure 1.

Simultaneous insertion and removal of a dominated edge can be canceled.

▶ **Lemma 9** (Cancellation Lemma). *Let e be an edge inserted and removed at t_i. If e is dominated in G_{t_i}, then the persistence diagrams of the following two sequences $\{ \cdots \leftarrow \overline{G}_{t'_{i-1}} \overset{e}{\hookrightarrow} \overline{G}_{t_i} \overset{e}{\leftarrow} \overline{G}_{t'_i} \hookrightarrow \overline{G}_{t_{i+1}} \leftarrow \cdots \}$ and $\{ \cdots \leftarrow \overline{G}_{t'_{i-1}} \hookrightarrow \overline{G_{t_i} \setminus e} \leftarrow \overline{G}_{t'_i} \hookrightarrow \overline{G}_{t_{i+1}} \leftarrow \cdots \}$ are the same.*

Algorithm. Algorithm 2 to simplify $\mathcal{Z} : \overline{G}_{t_1} \leftarrow \cdots \overline{G}_{t_i} \leftarrow \overline{G}_{t'_i} \hookrightarrow \overline{G}_{t_{i+1}}, \cdots \hookrightarrow \overline{G}_{t_n}$ is again a combination of swapping, shifting, trimming and cancelling of a dominated edge. For each edge e in $\mathcal{Z}$ there is a list of pairs $< t, inc >$ associated with it, where t is a grade and inc is a Boolean variable to denote whether e is inserted or removed at t. Below, we provide the main steps of the zigzag simplification algorithm. The algorithm first processes all the edge inclusions in decreasing grade order from t_n to t_1 and tries to shift them towards the
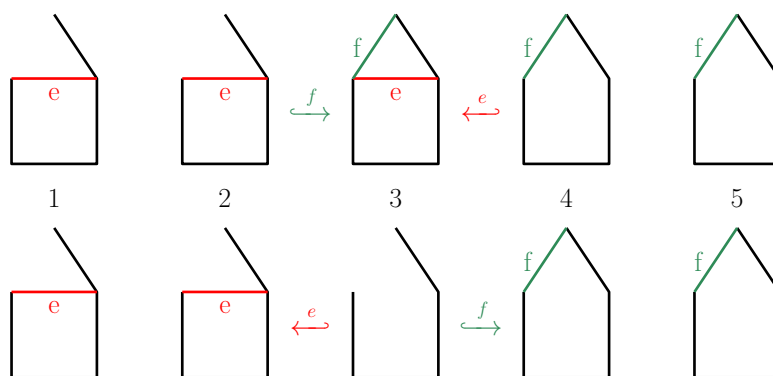
■ **Figure 1** In the top sequence, the green edge f is dominated at grade 3 and non-dominated at grade 4. Shifting and swapping the inclusion of f with the removal of the red edge e results in the bottom sequence. This results in two different one dimensional persistence diagrams of the associated flag complexes. For the top sequence it is $\{[1, 5]\}$ and for the bottom $\{[1, 2], [4, 5]\}$. Note that it is standard to use closed intervals in a zigzag persistence diagram.

end. After processing the first edge inclusion, it processes all the removals in increasing grade order from t_1 to t_n and tries to shift them towards the beginning. This process can be repeated several times until it converges. We use $t(e)$ to denote the current grade of the edge e being considered by the algorithm.

■ **Algorithm 2** Core zigzag flag filtration algorithm.

1: **procedure** CORE-ZIGZAG-FLAG-FILTRATION(E)
2:     **for all** edge *inclusions*, backward (from t_n to t_1) **do**
3:         **if** the current edge e is dominated in the graph $G_{t(e)}$ **then**
4:             **if** $t(e) == t_n$ **then**
5:                 trim e (delete the element $< t(e), inc >$).
6:             **else if** $G_{t(e)} \neq G_{t'(e)}$ **then**   ▷ the next step is a removal $G_{t(e)} \hookleftarrow G_{t'(e)}$.
7:                 **if** $e \notin G_{t'(e)}$ **then**
8:                     delete the inclusion-removal pair of e at $t(e)$.
9:                 **else if** e is dominated in $G_{t'(e)}$. **then**
10:                     set $t(e) = t(e) + 1$ and go-to step 3. ▷ $t(e) + 1$ denotes the next grade.
11:                 **end if**
12:             **else**   ▷ the next step is an inclusion $G_{t(e)} \hookrightarrow G_{t(e)+1}$.
13:                 set $t(e) = t(e) + 1$ and go-to step 3.
14:             **end if**
15:         **end if**
16:     **end for**
17:     Move forward from t_1 to t_n and process edge *removals* symmetric to steps 2-16.
18: **end procedure**

Note that an edge can be inserted and removed multiple times, in this case, the algorithm proceeds by pairing an inclusion with its next removal. Algorithm 2 outlines the essential aspects of the computation but is not optimal. Like Algorithm 1 we can use the Adjacency lemma (Lemma 4) to perform fewer domination checks. We can easily parallelize the zigzag simplification algorithm using the same divide and conquer approach described in Section 5.

8 Experiments

Complete graph. Starting from a complete graph on 700 vertices where all edges appear at the same time, the size of the graph after applying the algorithm several times decreases as 244650 (initial), 5340, 3086, 1307, 788 and finally 699. It stops decreasing after the 5th round since 699 edges is obviously minimal. This example demonstrates that one round of the algorithm is far from producing a fully reduced sequence. However, it removed a large number of edges, which makes subsequent rounds much faster, and may have already reduced the complex enough to compute (persistent) homology.

Torus: distribution of filtration values. We use a dataset with 1307 points on a torus embedded in $\mathbb{R}^3$. Figure 2 (left) shows the distribution of the edge lengths. Originally, there are 853471 edges and the longest has size 2.6. We apply repeatedly the backward algorithm until the process converges. In the end, we are left with 65053 edges, and a maximal filtration value of 1.427.
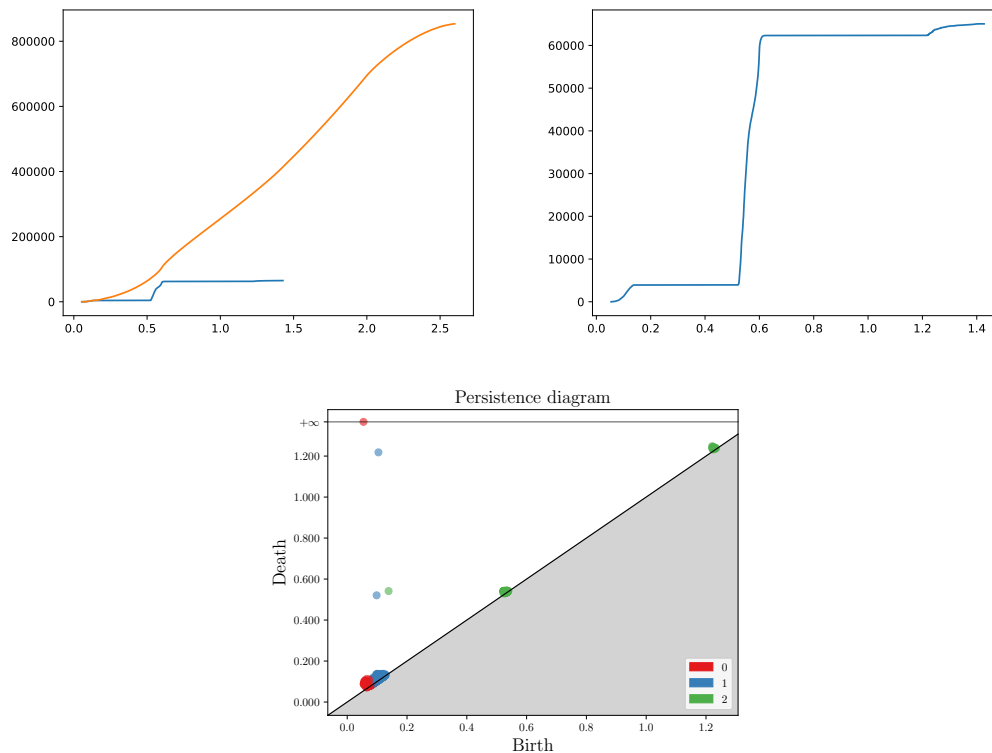
Figure 2 Filtration value of edges for a torus (top). Orange is for original edges and blue after collapse. Top right: enlarged blue graph. Bottom: persistence diagram.

First, note that some implementations (of which the first one is Eirene [18]) of Rips persistence first check at which filtration value the complex becomes a cone (here around 2) and ignore longer edges. In our algorithm, this check is performed implicitly and the long edges are dominated by the apex of the cone and thus get removed (we actually manage to go significantly lower than 2). Still, it remains sensible to avoid those edges when possible.

After collapsing, we notice several regions in the curve. First some short edges are added progressively, until the complex gets the homotopy type of a torus. Then nothing happens for a while, until we have enough edges to kill one of the 1-cycles and fill the cavity, where many edges are inserted at the same time. Then again nothing happens while the complex is equivalent to a circle, until we can kill this last 1-cycle, and the process quickly stops with a contractible complex.

Benchmark backward vs forward.   We benchmark the new backward algorithm with the forward algorithm. For the forward algorithm, we use the code from Giotto-ph [25], which is derived from our implementation in Gudhi but faster by a factor 1.5 to 2. Our bench marking considers two aspects: run-time and reduction size (see Table 1). The datasets are: *uniform* for an i.i.d. sample of points in a square, *sparse* for the same, but using a low threshold on the maximal size of edges, *polygon* for a regular polygon, *circle* for an i.i.d. uniform sample of a circle, *dragon* comes from [24] and *O3* from [3] (the first version uses a threshold of 1.4 on edge lengths).

The backward algorithm comes with an optimization using a *dense* array indexed by vertices. This usually speeds things up nicely, but in cases where the original set of edges is very sparse, this dense array can be an issue, so we also have a version without this array, denoted *sparse*.

■ **Table 1** Run-time and reduction size comparison. Column *before* and *after* contains the number of edges before and after collapsing, and column *time* contains run time in seconds of the collapse.

|	vertices	before	Forward		Backward		
			after	time	after	time dense	time sparse
uniform	1000	499500	2897	2.4	2897	**1.7**	2.4
sparse	50000	389488	125119	0.3	125119	1.9	**0.17**
polygon	300	44850	44701	3.6	44701	**0.5**	1
circle	300	44850	41959	4.8	41959	**0.4**	0.8
complete	900	404550	24540	43	**5980**	**0.4**	0.4
torus	1307	853471	94993	31	94993	**3.2**	5
dragon	2000	1999000	53522	29	53522	**14**	20
O3 (1.4)	4096	4107941	13674	59	13674	**37**	51
O3	1024	523776	519217	200	519217	**12**	23

Table 1 shows a clear advantage for the backward algorithm in cases where few edges can be removed, or when several edges have the same filtration value. Except for *complete* which is a plain complete graph with every edge at the same filtration value, all edges are computed as Euclidean Rips graphs.

When all the input edges have distinct filtration values, both algorithms output exactly the same list of edges. However, this isn't the case anymore when multiple edges have the same filtration value (and in particular if we apply the algorithm several times). The forward algorithm, as presented, relies on the order of the edges and does not take advantage of edges with the same filtration value. The backward algorithm, at its core, checks if an edge is dominated *at a specific filtration value (grade).* As seen in Table 1, for a complete graph on 900 vertices, the backward algorithm outputs 5 times fewer edges than the forward algorithm.

Table 2 Gains with the approximate algorithm, for different interleaving factors.

	original	1 (exact)	1.01	1.1	1.5	2	10	100
uniform	499500	2897	2891	2859	2609	2462	2356	2353
circle	44850	42007	30423	20617	17552	16404	14574	14342
(seconds)		*0.4*	*0.33*	*0.22*	*0.16*	*0.14*	*0.12*	*0.115*
dragon	1999000	53522	52738	52161	45439	40564	36094	35860
O3 (1.4)	4107941	13674	13635	13418	12682	12050	11828	11823

Size gains with approximate version. Table 2 shows the number of remaining edges when we don't require the output to have the same persistence diagram, but only ask that the modules be multiplicatively α-interleaved. Usually, the approximate version gives modest gains over the exact version, for roughly the same running time. However, in some cases that are hard to simplify like the circle, even a small error allows a significant number of collapses.
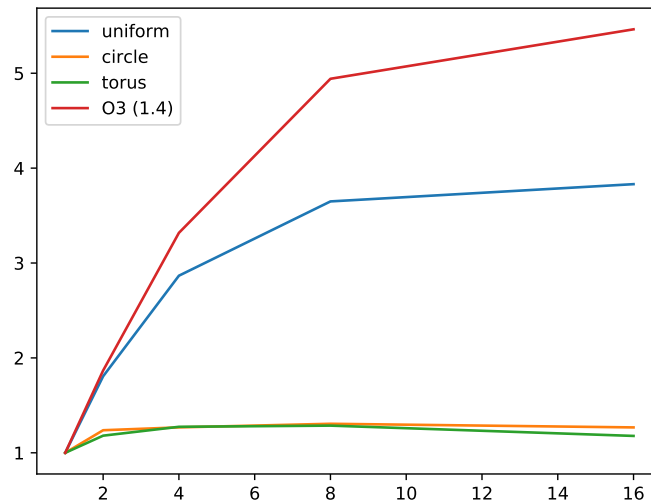
Figure 3 Speed gain in function of the number of threads.

Parallelism benchmark. We wrote a limited[2] prototype based on `tbb::parallel_reduce` and tested it on an i7-10875H CPU (8 cores, 16 threads) by limiting the number of threads. Figure 3 shows promising results for some datasets, but also that there is room for better parallel algorithms.

Persistence benchmark. In our experience, doing edge collapses before computing persistent homology helps a lot for (homology) dimension 2 or higher. However, it is a terrible idea if we only care about dimension 0. The case of dimension 1 is more mixed, it can help in some cases and hurt in others. By default we would only recommend its use for dimension greater than or equal to 2.

---

[2] This implementation assumes that no two edges have the same filtration value.

For convenience, the persistence computation is done using the version of Ripser [3] found in giotto-ph [25] with $n\_threads = 1$, and with our new backward algorithm. This means that edges after the complex has become a cone are ignored. Table 3 shows the time it takes to compute persistent homology in dimension up to k, either directly, or first collapsing before computing it.

Table 3 Persistent homology computation time in seconds, with or without edge collapse.

	dim 1	collapse & dim 1	dim 2	collapse & dim 2	collapse & dim 3
torus3D	6.2	3.8	75	6.4	47
dragon	3.3	9.2	148	9.7	16.3

References

1 M. Aggarwal and V. Periwal. Dory: Overcoming barriers to computing persistent homology, 2021. `arXiv:2103.05608`.

2 J. A. Barmak and E. G. Minian. Strong homotopy types, nerves and collapses. *Discrete and Computational Geometry*, 47:301–328, 2012. `doi:10.1007/s00454-011-9357-5`.

3 U. Bauer. Ripser: efficient computation of Vietoris-Rips persistence barcodes. *Journal of Applied and Computational Topology*, 5(3):391–423, 2021. `doi:10.1007/s41468-021-00071-5`.

4 U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. PHAT - persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78, 2017. `doi:10.1016/j.jsc.2016.03.008`.

5 J-D. Boissonnat and S. Pritam. Computing persistent homology of flag complexes via strong collapses. *International Symposium on Computational Geometry (SoCG)*, 2019. `doi:10.4230/LIPIcs.SoCG.2019.55`.

6 J-D. Boissonnat and S. Pritam. Edge collapse and persistence of flag complexes. *International Symposium on Computational Geometry (SoCG)*, 2020. `doi:10.4230/LIPIcs.SoCG.2020.19`.

7 M. Botnan and G. Spreemann. Approximating persistent homology in Euclidean space through collapses. *In: Applicable Algebra in Engineering, Communication and Computing*, 26:73–101, 2015. `doi:10.1007/s00200-014-0247-y`.

8 G. Carlsson and V. de Silva. Zigzag persistence. *Found Comput Math*, 10, 2010. `doi:10.1007/s10208-010-9066-0`.

9 F. Chazal, V. de Silva, M. Glisse, and S. Oudot. *The Structure and Stability of Persistence Modules*. SpringerBriefs in Mathematics. Springer, Cham, 2016. `doi:10.1007/978-3-319-42545-0`.

10 F. Chazal and S. Oudot. Towards persistence-based reconstruction in Euclidean spaces. *International Symposium on Computational Geometry (SoCG)*, 2008. `doi:10.1145/1377676.1377719`.

11 A. Choudhary, M. Kerber, and S. Raghvendra:. Polynomial-sized topological approximations using the permutahedron. *Discrete and Computational Geometry*, 61:42–80, 2019. `doi:10.1007/s00454-017-9951-2`.

12 H. Derksen and J. Weyman. Quiver representations. *Notices of the American Mathematical Society*, 52(2):200–206, February 2005. URL: `https://www.ams.org/journals/notices/200502/fea-weyman.pdf`.

13 T. K. Dey, D. Shi, and Y. Wang. Simba: An efficient tool for approximating Rips-filtration persistence via simplicial batch collapse. *ACM J. Exp. Algorithmics*, 24, January 2019. `doi:10.1145/3284360`.

14 P. Dłotko and H. Wagner. Simplification of complexes for persistent homology computations. *Homology, Homotopy and Applications*, 16:49–63, 2014. `doi:10.4310/HHA.2014.v16.n1.a3`.

15 H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

16 Gudhi: Geometry understanding in higher dimensions. URL: `https://gudhi.inria.fr/`.

17 A. Hatcher. *Algebraic Topology*. Univ. Press Cambridge, 2001. URL: `https://pi.math.cornell.edu/~hatcher/AT/ATpage.html`.

18 A. Hylton, G. Henselman-Petrusek, J. Sang, and R. Short. Tuning the performance of a computational persistent homology package. *Software: practice & experience*, 49(5):885–905, May 2019. `doi:10.1002/spe.2678`.

19 M. Kerber and R. Sharathkumar. Approximate Čech complex in low and high dimensions. In *Algorithms and Computation*, pages 666–676. By Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam. Vol. 8283. Lecture Notes in Computer Science, 2013. `doi:10.1007/978-3-642-45030-3_62`.

20 C. Maria and S. Oudot. Zigzag persistence via reflections and transpositions. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 181–199, January 2015. `doi:10.1145/1542362.1542408`.

21 K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete and Computational Geometry*, 50:330–353, September 2013. `doi:10.1007/s00454-013-9529-6`.

22 D. Mozozov. Dionysus. URL: `http://www.mrzv.org/software/dionysus/`.

23 J. Munkres. *Elements of Algebraic Topology*. Perseus Publishing, 1984.

24 N. Otter, M. Porter, U. Tillmann, P. Grindrod, and H. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science, Springer Nature*, 6:17, 2017. `doi:10.1140/epjds/s13688-017-0109-5`.

25 J. B. Pérez, S. Hauke, U. Lupo, M. Caorsi, and A. Dassatti. Giotto-ph: A Python Library for High-Performance Computation of Persistent Homology of Vietoris-Rips Filtrations. *CoRR*, 2021. `arXiv:2107.05412`.

26 M. Xiao S. Zhang and H. Wang. GPU-Accelerated Computation of Vietoris-Rips Persistence Barcodes. *International Symposium on Computational Geometry (SoCG)*, 2020. `doi:10.4230/LIPIcs.SoCG.2020.70`.

27 D. Sheehy. Linear-size approximations to the Vietoris–Rips filtration. *Discrete and Computational Geometry*, 49:778–796, 2013. `doi:10.1007/s00454-013-9513-1`.

28 A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete and Computational Geometry*, 33:249–274, 2005. `doi:10.1007/s00454-004-1146-y`.