Strong collapse and persistent homology

Jean-Daniel Boissonnat, Siddharth Pritam, Divyansh Pareek

Strong Collapse and Persistent Homology*

Jean-Daniel Boissonnat
Université Côte d'Azur, INRIA, France
Jean-Daniel.Boissonnat@inria.fr

Siddharth Pritam
Université Côte d'Azur, INRIA, France
siddharth.pritam@inria.fr

Divyansh Pareek
Indian Institute of Technology Bombay, India
divyansh@cse.iitb.ac.in

Abstract

In this paper, we introduce a fast and memory efficient approach to compute the Persistent Homology (PH) of a sequence of simplicial complexes. The basic idea is to simplify the complexes of the input sequence by using strong collapses, as introduced by J. Barmak and E. Miniam [DCG (2012)], and to compute the PH of an induced sequence of reduced simplicial complexes that has the same PH as the initial one. Our approach has several salient features that distinguishes it from previous work. It is not limited to filtrations (i.e. sequences of nested simplicial subcomplexes) but works for other types of sequences like towers and zigzags. To strong collapse a simplicial complex, we only need to store the maximal simplices of the complex, not the full set of all its simplices, which saves a lot of space and time. Moreover, the complexes in the sequence can be strong collapsed independently and in parallel.

We also focus on the problem of computing persistent homology of a flag tower, i.e. a sequence of flag complexes connected by simplicial maps. We show that if we restrict the class of simplicial complexes to flag complexes, we can achieve decisive improvement in terms of time and space complexities with respect to previous work. Moreover we can strong collapse a flag complex knowing only its 1-skeleton and the resulting complex is also a flag complex. When we strong collapse the complexes in a flag tower, we obtain a reduced sequence that is also a flag tower we call the core flag tower. We then convert the core flag tower to an equivalent filtration to compute its PH. Here again, we only use the 1-skeletons of the complexes. The resulting method is simple and extremely efficient. As a result and as demonstrated by numerous experiments on publicly available data sets, our approach is extremely fast and memory efficient in practice. Finally, we can compromize between precision and time by choosing the number of simplicial complexes of the sequence we strong collapse.

2012 ACM Subject Classification Mathematics of computing, Topological Data Analysis, Computational geometry

Keywords and phrases Computational Topology, Topological Data Analysis, Strong Collapse, Persistent homology

Digital Object Identifier 10.4230/LIPIcs...

* This research has received funding from the European Research Council (ERC) under the European Union's Seventh Framework Programme (FP/2007- 2013) / ERC Grant Agreement No. 339025 GUDHI (Algorithmic Foundations of Geometry Understanding in Higher Dimensions).

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this article, we address the problem of computing the Persistent Homology (PH) of a given sequence of simplicial complexes (defined precisely in Section 4) in an efficient way. It is known that computing persistence can be done in $O(n^\omega)$ time, where n is the total number of simplices and $\omega \leq 2.4$ is the matrix multiplication exponent [25, 20]. In practice, when dealing with massive and high-dimensional datasets, n can be very large (of order of billions) and computing PH is then very slow and memory intensive. This especially occurs for flag complexes since the simplices of a flag complex are the cliques of the 1-skeleton of the complex. Improving the performance of PH computation has therefore become an important research topic in Computational Topology and Topological Data Analysis.

Much progress has been accomplished in recent years along several directions. Regarding PH computation itself, a number of clever implementations and optimizations have led to a new generation of software [21, 4, 3, 27]. A second way to improve the overall process is to reduce the size of the complexes in the input sequence while preserving (or approximating in a controlled way) the persistent homology of the sequence. Examples of exact reductions can be found in the work of Mischaikow and Nanda [26] who use Morse theory to reduce the size of a filtration, and in the work of Dłotko and Wagner who use simple collapses [18]. More efficient reductions can be obtained if one allows to approximate the PH of the input sequence. Several algorithms with theoretical guarantees have been proposed recently using a variety of ideas and techniques, e.g. interleaving smaller and easily computable simplicial complexes, or sub-sampling the point sample [12, 9, 29, 24, 13, 17, 16].

In this paper, we introduce a new approach to simplify the complexes of the input sequence which uses the notion of strong collapse introduced by J. Barmak and E. Miniam [2]. Specifically, our approach can be summarized as follows. Given a sequence $\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$ of simplicial complexes K_i connected through simplicial maps $\{\xrightarrow{f_i}$ or $\xleftarrow{g_j}\}$, we independently strong collapse the complexes of the sequence to reach a sequence $\mathcal{Z}^c : \{K_1^c \xrightarrow{f_1^c} K_2^c \xleftarrow{g_2^c} K_3^c \xrightarrow{f_3^c} \cdots \xrightarrow{f_{(n-1)}^c} K_n^c\}$, with *induced simplicial maps* $\{\xrightarrow{f_i^c}$ or $\xleftarrow{g_j^c}\}$ (defined in Section 4). The complex K_i^c is called the **core** of the complex K_i and we call the sequence $\mathcal{Z}^c$ the **core sequence** of $\mathcal{Z}$. We show that one can compute the PH of the sequence $\mathcal{Z}$ by computing the PH of the core sequence $\mathcal{Z}^c$, which is usually of much smaller size.

Our method has some similarity with the work of Wilkerson et. al. [31, 32] who also use strong collapses to reduce PH computation but it differs in three essential aspects: it is not limited to filtrations (i.e. sequences of nested simplicial subcomplexes) but works for other types of sequences like towers and zigzags. It also differs in the way strong collapses are computed and in the manner PH is computed.

A first central observation is that to strong collapse a simplicial complex K, we only need to store its maximal simplices (i.e. those simplices that have no coface). The number m of maximal simplices is smaller than the total number of simplices by a factor that is exponential in the dimension d of the complex and it is linear in the number n_v of vertices for a variety of complexes [5]. Working only with maximal simplices dramatically reduces the time and space complexities compared to the algorithm of [32]. We prove that the complexity of our algorithm is $\mathcal{O}(n_v^2 d\Gamma_0^2)$. Here Γ_0 is the largest number of maximal simplices incident to a vertex. As observed in [6, 5], Γ_0 is usually small and appears to be almost constant along a filtration (see Section 3.1 for a discussion). As a consequence, the time to collapse

the i-th simplicial complex K_i in the sequence is almost independent of i, which is a clear advantage over methods (for filtrations) that use a full representation of the complexes and suffer an increasing cost as i increases.

Once a reduced sequence has been computed as described above, we cannot compute its PH directly. Indeed, all PH algorithms require as input a full representation of the complexes. We thus have to convert the representation by maximal simplices used to efficiently perform the strong collapses into a full representation of the complexes. This takes time exponential in the dimension (of the collapsed complexes). However, this exponential burden is to be expected since it is known that computing PH is NP-hard when the complexes are represented by their maximal faces [1]. Nevertheless, we demonstrate in this paper that strong collapses combined with known persistence algorithms lead to major improvements over previous methods to compute the PH of a sequence.

Most of the previous methods are for general simplicial complexes except [3, 29] which focus on the Vietoris-Rips complex, an example of a flag complex. In this paper, we show that further decisive progress can be obtained if one restricts the family of simplicial complexes to flag complexes. Flag complexes are fully characterized by their graph (or 1-skeleton), the other faces being obtained by computing the cliques of the graph. Hence, a flag complex can be represented by its 1-skeleton, which is a very compact representation. Flag complexes are very popular and, in particular, Vietoris-Rips complexes are widely used in Topological Data Analysis.

In the case of the Vietoris-Rips complex, the maximal simplices are the maximal cliques of the 1-skeleton of the complex and their computational time can be very large (exponential in the dimension of the complex). As a result, a naive application of our method would devote most of the time to compute the maximal faces of the complexes prior to their strong collapse. We show that we can avoid computing maximal cliques and we can strong collapse any flag complex using only the *1-skeleton* or graph of the complex. The strong collapses of a flag complex can be computed in time $O(n_v^2 k^2)$ where n_v is the number of vertices of the complex and k the maximal degree of its graph. Another crucial observation is that the reduced complex obtained by strong collapsing a flag complex is itself a flag complex.

Furthermore, if we consider a sequence of flag complexes connected by simplicial maps (a flag tower), we can strong collapse all the complexes of the sequence. The obtained core sequence is also a flag tower with smaller complexes that are connected by simplicial maps that are induced from the maps of the original sequence and the strong collapses. In the general case where the core sequence is not a filtration (which usually happens even if the original sequence is a filtration), we need to convert the flag tower to an equivalent filtration to compute its PH using known algorithms. To do so, we build on the work of [16, 23] that we restrict to flag complexes and strong collapses. This allows us to convert a flag tower to an equivalent flag filtration using again only the 1-skeleton.

The major advantages of our approach are:

– The collapses of the complexes in the sequence can be performed independently and in parallel. This is due to the fact that strong collapses can be expressed as simplicial maps, unlike simple collapses [30].

– Instead of computing the exact PH, we can compute an approximate PH which is substantially faster at a very minimal cost. We will explain more about the approximation scheme in Sections 6.

Further advantages for flag complexes are:

– We can compute PH for large complexes of high dimensions as we don't need to compute the maximal simplices.

– The dimension of the original sequence is in fact irrelevant and what matters is the dimension of the core sequence, which is usually quite small.

As a result, our approach is extremely fast and memory efficient in practice as demonstrated by numerous experiments on publicly available data sets. The algorithm described in this paper has been implemented. Numerous experiments show that the computation of the persistent homology of flag complexes can be obtained much faster than with previous methods, e.g. Ripser [3]. The code will be soon released in the Gudhi library [21].

The current version of the paper is combination of two papers [8] and [7].

An outline of this paper is as follows. Section 2 recalls the basic ideas and constructions related to simplicial complexes and strong collapses. We describe our core algorithm for general simplicial complexes and flag complexes in Section 3. In Section 4, we prove that zigzag modules are preserved under strong collapse. In Section 5, we provide an algorithm to construct an equivalent flag filtration from a flag tower. In Section 6, we provide an approximation scheme and some experimental results and, in Section 7, we end with a short discussion.

2 Preliminaries

In this section, we provide a brief review of the notions of simplicial complex and strong collapse as introduced in [2]. We assume some familiarity with basic concepts like homotopic maps, homotopy type, homology groups and other algebraic topological notions. Readers can refer to [22] for a comprehensive introduction of these topics.

Simplex, 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. From now on we will 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**. 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. L is a **full subcomplex** if it contains all the simplices of K that are spanned by the vertices (0-simplices) of the subcomplex L.

A vertex to vertex map $\psi : K \to L$ between two simplicial complexes is called a **simplicial map**, if the images of the vertices of a simplex always span a simplex. Simplicial maps are thus determined by the images of the vertices. In particular, there is a finite number of simplicial maps between two given finite simplicial complexes. Simplicial maps induce continuous maps between the underlying *geometric realisations* of the simplicial complexes. Two simplicial maps $\phi : K \to L$ and $\psi : K \to L$ are **contiguous** if, for all $\sigma \in K$, $\phi(\sigma) \cup \psi(\sigma) \in L$. We denote two contiguous maps as $\phi \sim_c \psi$. Two contiguous maps are known to be homotopic [28, Theorem 12.5]. Two maps ϕ_1 and ϕ_2 are in the same **contiguity class** if there exists a sequence of simplicial maps $\phi_1 = \psi_0, \psi_1, ..., \psi_k = \phi_2$ such that the consecutive maps ψ_i and ψ_{i+1} are contiguous for all $i = 0...k$. We denote two maps in the same contiguity class as $\phi_1 \sim \phi_2$. Two maps in the same contiguity class are homotopic.

Flag complex: A complex K is a flag or a clique complex if, when a subset of its vertices has pairwise edges between them, they span a simplex. It follows that the full structure of K is determined by its 1-skeleton that we denote by G. In particular, its maximal faces are the maximal cliques of G. For a vertex v in G, the **open neighborhood** $N_G(v)$ of v in G is defined as $N_G(v) := \{u \in G \mid [uv] \in E\}$, here E is the set of edges of G. The **closed neighborhood** $N_G[v]$ is $N_G[v] := N_G(v) \cup \{v\}$. We further define the relative closed neighborhood of u by v in G as the set of vertices in $N_G[u]$ that are not in $N_G[v]$. We denote it by $N_G[u \setminus v]$.

Dominated vertex: Let σ be a simplex of a simplicial complex K, the **closed star** of σ in K, $st_K(\sigma)$ is a subcomplex of K which is defined as follows, $st_K(\sigma) := \{\tau \in K |\ \tau \cup \sigma \in K\}$. The **link** of σ in K, $lk_K(\sigma)$ is defined as the set of simplices in $st_K(\sigma)$ which do not intersect with σ, $lk_K(\sigma) := \{\tau \in st_K(\sigma) | \tau \cap \sigma = \emptyset\}$.

Taking a join with a vertex transforms a simplicial complex into a **simplicial cone**. Formally if L is a simplicial complex and a is a vertex not in L then the simplicial cone aL is defined as $aL := \{a, \tau \mid \tau \in L \ \textit{or} \ \tau = \sigma \cup a; \ \textit{where} \ \sigma \in L\}$. A vertex v in K is called a **dominated vertex** if the link of v in K, $lk_K(v)$ is a simplicial cone, that is, there exists a vertex $v' \neq v$ and a subcomplex L in K, such that $lk_K(v) = v'L$. We say that the vertex v' is *dominating* v and v is *dominated* by v'. The symbol $K \setminus v$ (deletion of v from K) refers to the subcomplex of K which has all simplices of K except the ones containing v. Below is an important remark from [2, Remark 2.2], which proposes an alternative definition of dominated vertices.

▶ Remark 1. A vertex $v \in K$ is dominated by another vertex $v' \in K$, *if and only if* all the maximal simplices of K that contain v also contain v' [2].

Strong collapse: An **elementary strong collapse** is the deletion of a dominated vertex v from K, which we denote with $K \searrow\!\searrow^{e} K \setminus v$. Figure 1 illustrates an easy case of an elementary strong collapse. There is a **strong collapse** from a simplicial complex K to its subcomplex L, if there exists a series of elementary strong collapses from K to L, denoted as $K \searrow\!\searrow L$. The inverse of a strong collapse is called a **strong expansion**. If there exists a combination of strong collapses and/or strong expansion from K to L. K and L are said to have the same **strong homotopy type**.

The notion of strong homotopy type is stronger than the notion of simple homotopy type in the sense that if K and L have the same strong homotopy type, then they have the same simple homotopy type, and therefore the same homotopy type [2]. The converse is not necessarily true: there are examples of contractible or simply collapsible simplicial complexes that are not strong collapsible.

A complex without any dominated vertex will be called a **minimal complex**. A **core** of a complex K is a minimal subcomplex $K^c \subseteq K$, such that $K \searrow\!\searrow K^c$. *Every simplicial complex has a* ***unique core*** *up to isomorphism. The core decides the strong homotopy type of the complex*, and two simplicial complexes have the same strong homotopy type *if and only if* they have isomorphic cores [2, Theorem 2.11].

Retraction map: If a vertex $v \in K$ is dominated by another vertex $v' \in K$, the vertex map $r : K \to K \setminus v$ defined as: $r(w) = w$ if $w \neq v$ and $r(v) = v'$, induces a simplical map that is a **retraction** map. The homotopy between r and the identity $i_{K \setminus v}$ over $K \setminus v$ is in fact a

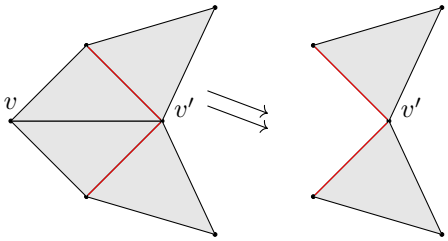

Figure 1 Illustration of an *elementary strong collapse*. In the complex on the left, v is dominated by v'. The link of v is highlighted in red. Removing v leads to the complex on the right.

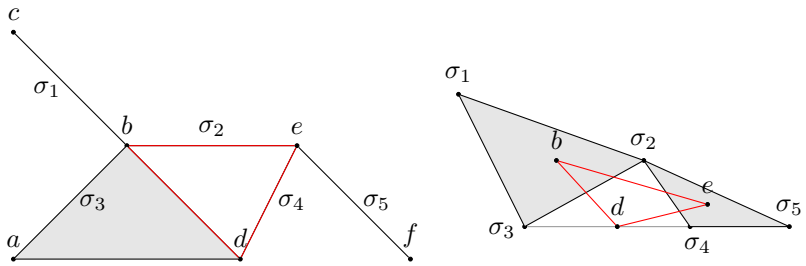

Figure 2 Left: K (in grey), Right: $\mathcal{N}(K)$ (in grey) and $\mathcal{N}^2(K)$ (in red). $\mathcal{N}^2(K)$ is isomorphic to a full-subcomplex of K highlighted in red on the left.

strong deformation retraction. Furthermore, the composition $(i_{K\setminus v})r$ is contiguous to the identity i_K over K [2, Proposition 2.9].

Nerve of a simplicial complex: A closed **cover** $\mathcal{U}$ of a topological space $\mathcal{X}$ is a set of closed sets of $\mathcal{X}$ such that $\mathcal{X}$ is a subset of their union. The **nerve** of a cover $\mathcal{U}$ is an abstract simplicial complex, defined as the set of all non-empty intersections of the elements of $\mathcal{U}$. The nerve is a well known construction that transforms a continuous space to a combinatorial space preserving its homotopy type. The nerve $\mathcal{N}(K)$ of a simplicial complex K is defined as the nerve of the set of maximal simplices of the complex K (considered as a cover of the complex). Hence all the maximal simplices of K will be the vertices of $\mathcal{N}(K)$ and their non-empty intersection will form the simplices of $\mathcal{N}(K)$. For $j \geq 2$ the iterative construction is defined as $\mathcal{N}^j(K) = \mathcal{N}(\mathcal{N}^{j-1}(K))$. This definition of nerve preserves the homotopy type, $K \simeq \mathcal{N}(K)$[2]. A remarkable property of this nerve construction is its connection with strong collapses.

Taking the nerve of any simplicial complex K twice corresponds to a strong collapse.

▶ **Theorem 2.** *[2, Proposition 3.4] For a simplicial complex K, there exists a subcomplex L isomorphic to $\mathcal{N}^2(K)$, such that $K \searrow\searrow L$.*

An easy consequence of this theorem is that a complex K is *minimal* if and only if it is isomorphic to $\mathcal{N}^2(K)$ [2, Lemma 3.6]. This means that we can keep collapsing our complex K by applying iteratively $\mathcal{N}^2(.)$ until we reach the core of the complex K. The sequence $K, \mathcal{N}^2(K), ..., \mathcal{N}^{2p}(K)$ is a decreasing sequence in terms of number of simplices.

3 Strong collapse of a simplicial complex

In this section, we first describe an algorithm to strong collapse a general simplicial complex and then we present an algorithm for the case of a flag complex. The former needs to represent the *maximal simplices* of the complex while the latter only requires to represent its *1-skeleton*. In both cases, we provide implementation details and complexity analyses.

3.1 Strong collapse of a general simplicial complex

Data structure: Basically, we represent K as the adjacency matrix M between the vertices and the maximal simplices of K. We will simply call M the adjacency matrix of K. The rows of M represent the vertices and the columns represent the maximal simplices of K. For convenience, we will identify a row (resp. column) and the vertex (resp. maximal simplex) it represents. An entry $M[v_i][\sigma_j]$ associated with a vertex v_i and a maximal simplex σ_j is set to 1 if $v_i \in \sigma_j$, and to 0 otherwise. For example, the matrix M in the left of the Table 1 corresponds to the leftmost simplicial complex K in Figure 2. Usually, M is very sparse. Indeed, each column contains at most $d+1$ non-zero elements since the simplices of a d-dimensional complex have at most $d+1$ vertices, and each line contains at most Γ_0 non-zero elements where Γ_0 is the maximum number of maximal simplices that are incident to a given vertex. As already mentioned, in many practical situations, Γ_0 is a small fraction of the number of maximal simplices. It is therefore beneficial to store M as a list of vertices and a list of maximal simplices. Each vertex v in the list of vertices points to the maximal simplices that contain v, and each simplex in the list of maximal simplices points to its vertices. This data structure is similar to the SAL data structure of [6].

Core algorithm: Given the adjacency matrix M of K, we compute the adjacency matrix C of the *core* K^c. It turns out that using basic row and column removal operations, we can easily compute C from M. Loosely speaking, our algorithm recursively computes $\mathcal{N}^2(K)$ until it reaches K^c.

The columns of M (which represent the maximal simplices of K) correspond to the vertices of $\mathcal{N}(K)$. Also, the columns of M that have a non-zero value in a particular row v correspond to the maximal simplices of K that share the vertex associated with row v. Therefore, each row of M represents a simplex of the nerve $\mathcal{N}(K)$. Not all simplices of $\mathcal{N}(K)$ are associated with rows of M but all maximal simplices are since they correspond to subsets of maximal simplices with a common vertex. In order to only store the maximal simplices of $\mathcal{N}(K)$, we *remove* all the rows of M that correspond to non-maximal simplices of $\mathcal{N}(K)$. This results in a new smaller matrix M whose transpose, noted $\mathcal{N}(M)$, is the adjacency matrix of the nerve $\mathcal{N}(K)$. We then exchange the roles of rows and columns (which is the same as taking the transpose) and run the very same procedure as before so as to obtain the adjacency matrix $\mathcal{N}^2(M)$ of $\mathcal{N}^2(K)$.

The process is iterated as long as the matrix can be reduced. Upon termination, we output the reduced matrix $C := \mathcal{N}^{2p}(M)$, for some $p \geq 1$, which is the adjacency matrix of the core K^c of K. Removing a row or column is the most basic operation of our algorithm. We will discuss it in more detail later in the paragraph *Domination test*.

Example: As mentioned before, the matrix M in the left of Table 1 represents the simplicial complex K in the left of Figure 2. We go through the rows first, rows a and c are subsets

	σ_1	σ_2	σ_3	σ_4	σ_5
a	0	0	1	0	0
b	1	1	1	0	0
c	1	0	0	0	0
d	0	0	1	1	0
e	0	1	0	1	1
f	0	0	0	0	1

	b	d	e
σ_1	1	0	0
σ_2	1	0	1
σ_3	1	1	0
σ_4	0	1	1
σ_5	0	0	1

	σ_2	σ_3	σ_4
b	1	1	0
d	0	1	1
e	1	0	1

Table 1 From left to right M, $\mathcal{N}(M)$ and $\mathcal{N}^2(M)$.

of row b and row f is a subset of e. Removing rows a, c and f and transposing M yields the adjacency matrix $\mathcal{N}(M)$ of $\mathcal{N}(K)$ in the middle. Now, row σ_1 is a subset of σ_2 and of σ_3, and σ_5 is a subset of σ_2 and of σ_4. We remove these two rows of $\mathcal{N}(M)$ and transpose $\mathcal{N}(M)$ so as to get $\mathcal{N}^2(M)$. The rightmost matrix in Table 1 is associated to the core (red triangle) on the right of Figure 2. .

Domination test: Now we explain in more detail how to detect the rows that need to be removed. Let v be a row of M and σ_v be the associated simplex in $\mathcal{N}(K)$. If σ_v is not a maximal simplex of $\mathcal{N}(K)$, it is a proper face of some maximal simplex $\sigma_{v'}$ of $\mathcal{N}(K)$. Equivalently, the row v' of M that is associated with $\sigma_{v'}$ contains row v in the sense that the non zero elements of v appear in the same columns as the non zero elements of v'. We will say that row v is dominated by row v' and determining if a row is dominated by another one will be called the row domination test. Notice that when a row v is dominated by a row v', the same is true for the associated vertices since all the maximal simplices that contain vertex v also contain vertex v', which is the criterion to determine if v is dominated by v' (See Remark 1). The algorithm removes all dominated rows and therefore all dominated vertices of K.

After removing rows, the algorithm removes the columns that are no longer maximal in K, which might happen since we removed some rows. Removing a column may lead in turn to new dominated vertices and therefore new rows to be removed. When the algorithm stops, there are no rows to be removed and we have obtained the core K^c of the complex K. Note that the algorithm provides a constructive proof of Theorem 2.

Removing columns is done in very much the same way: we just exchange the roles of rows and columns.

Computing the retraction map r: The algorithm also provides a direct way to compute the retraction map r defined in Section 2. The retraction map corresponding to the strong collapses executed by the algorithm can be constructed as follows. A row r being removed in M corresponds to a dominated vertex in K and the row which *contains* r corresponds to a dominating vertex. Therefore we map the dominated vertex to the dominating vertex and compose all such maps to get the final retraction map from K to its core K^c. The final map is simplicial as well, as it is a composition of simplicial maps.

Reducing the number of domination tests: We first observe that, when one wants to determine if a row v is dominated by some other row, we don't need to test v with all other

rows but with at most d of them. Indeed, at most $d+1$ rows can intersect a given column since a simplex has at most $d+1$ vertices. For example, in Table 1 (Left), to check if row e (highlighted in brown) is dominated by another row, we pick the first non-zero column σ_2 (highlighted in Gray) and compare e with the non-zero entries {b} of σ_2.

A second observation is that we don't need to test all rows and columns for domination, but only the so-called candidate rows and columns. We define a row r to be a **candidate row** for the next iteration if at least one column containing one of the non-zero elements of r has been removed in the previous column removal iteration. Similarly, by exchanging the roles of rows and columns, we define the **candidate columns**. Candidate rows and columns are the only rows or columns that need to be considered in the *domination* tests of the algorithm. Indeed, a column τ of M whose non-zero elements all belong to rows that are present from the previous *iteration* cannot be dominated by another column τ' of M, since τ was not dominated at the previous iteration and no new non-zero elements have ever been added by the algorithm. The same argument follows for the candidate rows.

We maintain two *queues*, one for the candidate columns (colQueue) and one for the candidate rows (rowQueue). These queues are implemented as First in First out (FIFO) queues. At each iteration, we *pop out* a candidate row or column from its respective queue and test whether it is dominated or not. After each successful domination test, we *push* the candidate columns or rows in their appropriate queue in preparation for the subsequent iteration. In the first iteration, we *push* all the rows in rowQueue and then alternatively use colQueue and rowQueue. Algorithm 2 gives the pseudo code of our algorithm.

Time Complexity: The most basic operation in our algorithm is to determine if a row is dominated by another given row, and similarly for columns. In our implementation, the rows (columns) of the matrix that are considered by the algorithm are stored as sorted lists. Checking if one sorted list is a subset of another sorted list can be done in time $\mathcal{O}(l)$, where l is the size of the longer list. Note that the length of a row list is at most Γ_0 where Γ_0 denotes the largest number of maximal simplices incident to a vertex. The length of a column list is at most $d+1$ where d is the dimension of the complex. Hence checking if a row is dominated by another row (Line 8) takes $\mathcal{O}(\Gamma_0)$ time and checking if a column is dominated by another column (Line 19) takes $\mathcal{O}(d)$ time.

Next consider one execution of the while loop on rows (Lines 4-14). We will loop at most n_v times since *rowQueue* contains at most n_v elements, where n_v is the number of vertices of the complex K. Since simplex σ has at most $d+1$ vertices w and since testing if v is a subset of w (Line 8) takes at most Γ_0 time as shown above, a row domination test (Lines 7-13) takes time $O(d\Gamma_0)$. Hence, executing one while loop on rows takes $n_v d\Gamma_0$ time. If, during the execution of the while loop, we never execute Lines 9-10, there is no dominated vertex, *colQueue* remains empty and the algorithm stops. Otherwise, we enter the while loop on columns (Lines 15-25).

We will loop at most m times since *colQueue* contains at most m elements where m is the number of maximal simplices in K. Vertex v is incident to at most Γ_0 maximal simplices and, as shown above, testing if τ is a subset of σ (Line 19) takes $O(d)$ time. It follows that executing one while loop on columns takes $\mathcal{O}(md\Gamma_0)$ time.

It remains to bound the number of times we execute the while loops or, equivalently, the number of times we execute Line 27. This number is at most n_v. Indeed, each time we execute Line 27, the algorithm has removed at least one row from M since otherwise it stops. It follows that the total complexity of the algorithm is $\mathcal{O}(n_v d\Gamma_0\,(n_v+m))$. Noticing that

Algorithm 1 Core algorithm

```
1: procedure CORE(M)                                   ▷ M is the adjacency matrix of K.
2:     rowQueue ← push all rows of M (all vertices of K)
3:     colQueue ← empty
4:     while rowQueue is not empty do
5:         v ← pop(rowQueue)
6:         σ ← the first non-zero column of v
7:         for non-zero rows w in σ do
8:             if v is a subset of w then
9:                 Remove v from M
10:                push all non-zero columns τ of v to colQueue if not pushed before
11:                break
12:            end if
13:        end for
14:    end while
15:    while colQueue is not empty do
16:        τ ← pop(colQueue)
17:        v ← the first non-zero row of τ
18:        for non-zero columns σ in v do
19:            if τ is subset of σ then
20:                Remove τ from M
21:                push all non-zero rows w of τ to rowQueue if not pushed before
22:                break
23:            end if
24:        end for
25:    end while
26:    if rowQueue is not empty then
27:        GOTO 4
28:    end if
29:    return M                                    ▷ M is now the adjacency matrix of the core of K.
30: end procedure
```

$n_v\Gamma_0 \geq m$, the complexity can be simply written as $\mathcal{O}(n_v^2 d\Gamma_0^2)$.

In practice, m is much smaller than n, the total number of simplices, and Γ_0 is much smaller than Γ, the maximum number of simplices incident on a vertex. Typically Γ grows exponentially with d while Γ_0 remains almost constant as d increases. See Table 5 in [6], related results in [5], and the plots in experimental section of [8].

3.2 Strong collapse of a flag complex

In the previous subsection, K was represented by its maximal simplices. When K is a flag complex, the maximal simplices are the maximal cliques of its 1-skeleton and K is entirely characterized by its 1-skeleton. We will show that the core of K is itself a flag complex whose graph is called the *core graph* of K and the core graph can be computed from the 1-skeleton G of K in time $\mathcal{O}(n_v^2 k^2)$, where n_v is the number of vertices of K and k is an upper bound on the degree of G (i.e. the number of edges that are incident on a vertex in K). The gain in the time complexity wrt the general algorithm of the previous subsection is substantial as the time to compute all the maximal simplices of a flag complex from its graph is exponential in the number of its vertices. This will be of crucial importance in practice.

In the following lemma, we describe a condition in terms of the closed neighborhood $N_G[v]$ of a vertex v of a flag complex K under which v will be dominated by another vertex v' of K. This result already appeared in another context in [19, Lemma 4.1]. We give a proof for completeness.

▶ **Lemma 3.** *Let K be a flag complex. A vertex $v \in K$ is dominated by v' iff $N_G[v] \subseteq N_G[v']$.*

Proof. If v is dominated by v', then, according to Remark 1, the set of maximal simplices that contain v is a subset of the set of maximal simplices that contain v'. It follows that $N_G[v] \subseteq N_G[v']$.

Now we prove the other direction. Let σ be a maximal simplex of K containing v. Any other vertex x of σ is joined to v by an edge $[x, v] \in \sigma$. Moreover, since $N_G[v] \subseteq N_G[v']$, $[v, v']$ and $[x, v']$ are in K. It follows that every vertex in σ has an edge with both v and v' and, since K is a flag complex and σ is maximal, v' must be in σ. This implies that all the maximal simplices that contain v also contain v'. Hence v is dominated by v'. ◀

As mentioned before, an elementary strong collapse consists in removing a dominated vertex, and it can be easily observed that removing a vertex does not affect the 'flagness' of the residual complex $K \setminus v$. In other words, if σ is a maximal clique with vertex v, the resultant clique $\sigma \setminus v$ is still a maximal clique in $K \setminus v$. Moreover, all the other cliques that do not contain v still span the complete simplices. This implies that the core K^c of a flag complex K with graph G is a flag complex of a subgraph G^c of G.

In what follows next, we describe an algorithm to compute the core graph $G^c \subseteq G$ whose flag complex is the core K^c of K.

Data structure: We represent G with its adjacency matrix M, where the rows and the columns of M represent the vertices of G. An entry $M[v_i][v_j]$ associated with vertices v_i and v_j is set to 1 if either the edge $[v_i, v_j] \in G$ or $i = j$, and to 0 otherwise. Note that we set $M[v_i][v_j] = 1$ for $i = j$ to be able to consider closed neighborhood. We will say that a row v is contained in another row v' if the set of column indices of the non-zero entries of v is a subset of the indices of the non-zero entries of v'. It is clear that if a row v is contained

in another row v', we have $N_G[v] \subseteq N_G[v']$ and therefore the vertex v is dominated by the vertex v'

Core graph algorithm: Given the adjacency matrix M of G, we compute the adjacency matrix C of the core graph G^c. In view of Lemma 3, we can easily compute C from M using basic row removal operations. Loosely speaking, we remove the rows of M that are contained in another row. When removing the row associated to v, we also remove the column associated to v. The process is iterated as long as the matrix can be reduced. Upon termination, we output the reduced matrix C, which is the adjacency matrix of the core graph G^c of K. Since the core of a complex is always unique, the order in which vertices are removed does not matter [2].

Retraction map computation: We can easily compute the retraction map r defined in Section 2 using the above core graph algorithm. A row v being removed in M corresponds to a dominated vertex in K and the row which contains v corresponds to a dominating vertex. Therefore we map the dominated vertex to the dominating vertex.

Domination tests optimization: Let us observe that, to check if a row v is dominated by some other row v', it is sufficient to compare v with its neighbors, which are at most k in number, if k denotes the maximum degree of the vertices in G.

We define a row v to be a **candidate row** for the next iteration if at least one of its neighbors has been removed in a previous row removal iteration. We observe that the candidate rows are the only rows that need to be considered in the domination tests of the algorithm. Indeed, a row w of M whose set of neighbors has not been modified at the previous *iteration* cannot be dominated by another row v' of M, as w was not dominated in the previous iteration and all other vertices can only loose neighbors. This ensures that w will still remain un-dominated.

We maintain a *queue*, for the candidate rows (rowQueue) which is implemented as a First in First out (FIFO) queue. At each iteration, we *pop out* a candidate row from rowQueue for domination test. After each successful domination test, we push the new candidate rows in the queue in preparation for the subsequent iteration. In the first iteration, we push all the rows in rowQueue. Algorithm 2 gives the pseudo code of our algorithm.

Time Complexity: Let us start by analyzing the most basic operation in our algorithm which is to determine if a row is dominated by another row. We store the rows of the matrix as sorted lists. Deciding if a sorted list is included in another sorted list (Line 8) can be done in time $\mathcal{O}(l)$, where l is the size of the longer list. In our case, the length of a row list is at most $k+1$ where k denotes as before the maximal degree of any vertex. Hence Lines 8-12 takes $\mathcal{O}(k)$ time.

As explained in the paragraph *Domination tests optimization*, each row is checked against at most k other rows. Hence the for loop (Lines 7-13) is executed at most k times. Moreover, since at each iteration we ought to remove at least one row, the total number of iterations on the rows, i.e. the number of times the while loop is executed, is at most $O(n_v^2)$, where n_v is the total number of vertices of the complex K. It follows that the worst-case time complexity of our algorithm is $\mathcal{O}(n_v^2 k^2)$.

Algorithm 2 Core graph algorithm

```
1: procedure CORE(M)
2:     input : the adjacency matrix M of the graph of a flag complex K
3:     rowQueue ← push all rows of M (all vertices of K)
4:     while rowQueue is not empty do
5:         v ← pop(rowQueue)
6:         N_G[v] ← the non-zero columns of v
7:         for w in N_G[v] do
8:             if N_G[v] ⊆ N_G[w] then
9:                 Remove from M the column and the row associated to v
10:                push all the entries of N_G(v) to rowQueue if not pushed before
11:                break
12:            end if
13:        end for
14:    end while
15:    return M                    ▷ M is now the adjacency matrix of the core of K
16: end procedure
```

4 Strong collapse of a sequence of simplicial complexes

In this section, we prove that the persistence homology of a sequence of simplicial complexes is preserved under strong collapse. We give our result for the most general case of a zigzag sequence of simplicial complexes. We begin with some brief background on zigzag persistence. Readers interested in more details can refer to [10, 11, 15].

A *sequence of simplicial complexes* $\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$ is a sequence of complexes K_is connected through simplicial maps $\xrightarrow{f_i}$s and $\xleftarrow{g_j}$s. In the most general case, the maps are in both directions $\rightarrow, \leftarrow$ and the sequence is called a **zigzag sequence**. If furthermore all the maps are forward, i.e. it consists only of the f_is, the sequence is called a simplicial **tower**. If all the maps are inclusions, we have a sequence of nested simplicial complexes called a **filtration**. Our results apply to all types of sequences and will be described for zigzag sequences.

Once we compute the homology classes of all K_is, we get the sequence $\mathcal{P}(\mathcal{Z}) : \{H_p(K_1) \xrightarrow{f_1^*} H_p(K_2) \xleftarrow{g_2^*} H_p(K_3) \xrightarrow{f_3^*} \cdots \xrightarrow{f_{(n-1)}^*} H_p(K_n)\}$. Here $H_p(-)$ denotes the homology class of dimension p with coefficients from a field $\mathbb{F}$ and $*$ denotes an induced homomorphism. $\mathcal{P}(\mathcal{Z})$ is a sequence of vector spaces connected through homomorphisms, called a **zigzag module**. More formally, a *zigzag module* $\mathbb{V}$ is a sequence of vector spaces $\{V_1 \rightarrow V_2 \leftarrow V_3 \rightarrow \cdots \rightarrow V_n\}$ connected with homomorphisms $\{\rightarrow, \leftarrow\}$ between them. A zigzag module arising from a sequence of simplicial complexes captures the evolution of the topology of the sequence.

For two integers b and d, $1 \leq b \leq d \leq n$, we can define an **interval module** $\mathbb{I}[b, d]$ by assigning V_i to $\mathbb{F}$ when $i \in [b, d]$, and null spaces otherwise, the maps between any two $\mathbb{F}$ vector spaces is identity and is zero otherwise. For example $\mathbb{I}[2, 4] : \{0 \xrightarrow{0} \mathbb{F} \xleftarrow{I} \mathbb{F} \xrightarrow{I} \mathbb{F} \xleftarrow{0} 0 \xrightarrow{0} 0\}$, here $n = 6$. Any zigzag module can be *decomposed* as the direct sum of *finitely* many interval modules, which is unique upto the permutations of the interval modules [10]. The multiset of all the intervals $[b_j, d_j]$ corresponding to the interval module decomposition of any zigzag module is called a **zigzag (persistence) diagram**. The zigzag diagram completely

characterizes the zigzag module, that is, there is bijective correspondence between them [10, 33].

Two different zigzag modules $\mathbb{V} : \{V_1 \to V_2 \leftarrow V_3 \to \cdots \to V_n\}$ and $\mathbb{W} : \{W_1 \to W_2 \leftarrow W_3 \to \cdots \to W_n\}$, connected through a set of homomorphisms $\phi_i : V_i \to W_i$ are **equivalent** if the ϕ_is are isomorphisms and the following diagram commutes [10, 15].

$$\begin{array}{ccccccccc} V_1 & \longrightarrow & V_2 & \longleftarrow & V_3 & \cdots & \longrightarrow V_{n-1} & \longrightarrow & V_n \\ \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2} & & \downarrow{\scriptstyle \phi_3} & & \downarrow{\scriptstyle \phi_{n-1}} & & \downarrow{\scriptstyle \phi_n} \\ W_1 & \longrightarrow & W_2 & \longleftarrow & W_3 & \cdots & \longrightarrow W_{n-1} & \longrightarrow & W_n \end{array}$$

Note that the *length* of the modules and the directions of the arrows should be consistent. Two equivalent zigzag modules will have the same interval decomposition, therefore the same zigzag diagram.

Strong collapse of a zigzag module: Given a zigzag sequence $\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$, we define the **core sequence** $\mathcal{Z}^c$ of $\mathcal{Z}$ as $\mathcal{Z}^c : \{K_1^c \xrightarrow{f_1^c} K_2^c \xleftarrow{g_2^c} K_3^c \xrightarrow{f_3^c} \cdots \xrightarrow{f_{(n-1)}^c} K_n^c\}$, where K_i^c is the core of K_i. The forward maps are defined as, $f_j^c := r_{j+1} f_j i_j$; and the backward maps are defined as $g_j^c := r_j g_j i_{j+1}$. The maps $i_j : K_j^c \hookrightarrow K_j$ and $r_j : K_j \to K_j^c$ are the composed inclusions and the retractions maps defined in Section 2 respectively. We call the procedure of forming the core sequence using the cores and the induced simplicial maps as the **core-assembly**.

▶ **Theorem 4.** *The zigzag modules $\mathcal{P}(\mathcal{Z})$ and $\mathcal{P}(\mathcal{Z}^c)$ are equivalent.*

Proof. Consider the following diagram

$$\begin{array}{ccccccccc} K_1 & \xrightarrow{f_1} & K_2 & \xleftarrow[g_2]{} & K_3 & \cdots & \longrightarrow K_{n-1} & \xrightarrow{f_{n-1}} & K_n \\ \downarrow{\scriptstyle r_1} & & \downarrow{\scriptstyle r_2} & & \downarrow{\scriptstyle r_3} & & \downarrow{\scriptstyle r_{n-1}} & & \downarrow{\scriptstyle r_n} \\ K_1^c & \xrightarrow{f_1^c} & K_2^c & \xleftarrow[g_2^c]{} & K_3^c & \cdots & \longrightarrow K_{n-1}^c & \xrightarrow{f_{n-1}^c} & K_n^c \end{array}$$

and the associated diagram after computing the p-th homology groups

$$\begin{array}{ccccccccc} H_p(K_1) & \xrightarrow{f_1^*} & H_p(K_2) & \xleftarrow[g_2^*]{} & H_p(K_3) & \cdots & \longrightarrow H_p(K_{n-1}) & \xrightarrow{f_{n-1}^*} & H_p(K_n) \\ \downarrow{\scriptstyle r_1^*} & & \downarrow{\scriptstyle r_2^*} & & \downarrow{\scriptstyle r_3^*} & & \downarrow{\scriptstyle r_{n-1}^*} & & \downarrow{\scriptstyle r_n^*} \\ H_p(K_1^c) & \xrightarrow{(f_1^c)^*} & H_p(K_2^c) & \xleftarrow[(g_2^c)^*]{} & H_p(K_3^c) & \cdots & \longrightarrow H_p(K_{n-1}^c) & \xrightarrow{(f_{n-1}^c)^*} & H_p(K_n^c) \end{array}$$

Since there exists a strong deformation retraction between r_j and i_j, the induced homomorphisms r_j^* and i_j^* are isomorphisms [22, Corollary 2.11]. We claim that the composed map $i_j r_j$ is in the same contiguity class as the identity on K_j. Indeed, i_j can be decomposed into a series of inclusions $i_j^1 i_j^2 ... i_j^s$ that correspond to elementary strong collapses and similarly r_j can be decomposed into $r_j^s r_j^{s-1} ... r_j^1$ for some s. Using [2, Proposition 2.9] and the fact that contiguity is preserved under composition, we have,

$$i_j r_j = i_j^1 i_j^2 ... i_j^s \circ r_j^s r_j^{s-1} ... r_j^1 \;\sim_c\; i_j^1 i_j^2 ... i_j^{s-1} \circ r_j^{s-1} ... r_j^1 \;\sim_c\; \ldots \;\sim_c\; i_j^1 \circ r_j^1 \;\sim_c \mathrm{id}$$

.

This is a sequence of contiguous maps from $i_j r_j$ to the identity on K_j, which implies that $i_j r_j$ is in the same contiguity class as the identity on K_j.

It then follows that $f_j^c r_j = r_{j+1} f_j i_j r_j$ is in the same contiguity class as $r_{j+1} f_j$, and similarly that $g_j^c r_{j+1}$ is in the same contiguity class as $r_j g_j$. Now, since maps in the same contiguity class are homotopic at the level of geometric realizations and since homotopic maps induce the same homomorphism, we have $(f_j^c r_j)^* = (r_{j+1} f_j)^*$ and thus $(f_j^c)^* r_j^* = r_{j+1}^* f_j^*$ and similarly $(g_j^c)^* r_{j+1}^* = r_j^* g_j^*$, see [22, Proposition (1) page 111]. Therefore all the squares in the lower diagram commute and the set of maps r_j^*s are isomorphisms. Hence $\mathcal{P}(\mathcal{Z})$ and $\mathcal{P}(\mathcal{Z}^c)$ are *equivalent* and their zigzag diagrams are identical. ◀

The algorithms to compute persistence are for sequences of simplicial complexes connected via inclusions only : inclusions can be in both directions, in which case we call them zigzag filtrations, or in a single direction, in which case we call them filtrations. To compute the persistent homology of more general sequences, one usually converts them to an equivalent inclusion-only sequence. Zigzag sequences will thus be transformed in zigzag filtrations and towers will be transformed in filtrations respectively, as described now. Given a zigzag sequence of general simplicial complexes, one can individually strongly collapse each of the complexes using the algorithm described in Section 3 and then assemble the cores using the maps prescribed in this section and compute the persistence of the core sequence. The core sequence of a zigzag filtration is usually a general zigzag sequence rather than a zigzag filtration. To compute the persistent homology of such a core sequence, one can expand it so as to obtain a zigzag *filtration* using the approach described in [16] and [23]. In the special case of a flag filtration or of a flag tower, the core sequence is a flag tower. Again, to compute the persistence of a flag tower, one needs to expand it to get a filtration as will be described in the next section.

▶ Remark 5. The above results can be extended to multidimensional persistence using the more general notion of quiver representation [15].

5 From a flag tower to a flag filtration

In Section 4, we have seen that the core sequence of a flag tower is a flag tower and that the core sequence of a flag filtration is usually a flag tower. Hence, in both cases, if one wants to use algorithms to compute the PH of filtrations, we need to transform the core flag tower into a flag filtration. Therefore the complete pipeline is to strong collapse a flag filtration or a flag tower to a reduced flag tower and then convert the reduced flag tower to another equivalent flag filtration, which is usually smaller than the original flag filtration we started with, as verified experimentally in Section 6.

The main result of this section is that transforming a flag tower into an equivalent flag filtration can be done using only edge inclusions over the 1-skeletons of the complexes.

Previous work: It is known that any general simplicial map can be decomposed into elementary inclusions and elementary contractions. Hence, if we replace an elementary contraction $\{\{u, v\} \mapsto u\}$ by an equivalent (not necessarily elementary) inclusion, we transform a tower into an equivalent filtration. This was the philosophy introduced by Dey et al. [16] and further refined by Kerber and Schreiber [23] who introduced a slightly different approach based on coning and provided theoretical bounds on the size and time to construct the final equivalent filtration. Specifically, they proved that the size of the equivalent filtration is

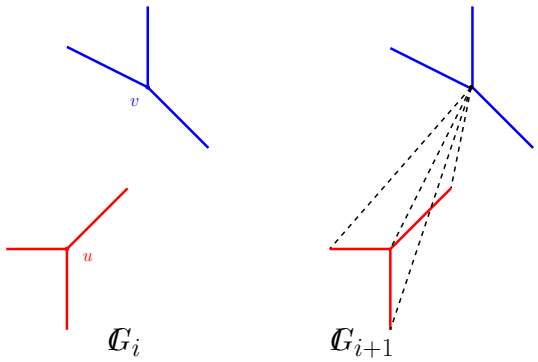

Figure 3 Demonstration of strong expansion: On the left we have the active neighborhoods of the vertex u (in red) and of v (in blue) in the graph $\mathbb{G}_i$. The graphs $\mathbb{G}_i$ might have other vertices and edges, here we have zoomed into the local active neighborhoods of u and v. On the right, we have the local neighborhoods of u and v in $\mathbb{G}_{i+1}$ after conning v to the active neighborhood of u.

$\mathcal{O}(dn \log n_0)$, where d is the maximal dimension of the complexes in the input tower and n (resp. n_0) the total number of elementary inclusions (resp. vertex inclusions) in the input tower [23, Theorem 2].

A new construction: We now present an algorithm that turns a flag tower into a flag filtration with the same PH. Our work builds upon the above mentioned previous works [16, 23]. The difference is that we use strong expansion which is the inverse operation of a strong collapse. The main advantage of strong expansions is that, when the input is a flag tower, we can use the domination criterion of Lemma 3. This leads to a simple algorithm that only deals with edges. The output filtration is a flag filtration, which can be represented very compactly. Moreover, since a strong expansion is a coning, we will be able to use the theoretical results of [23]. Now we describe our construction.

Let K_i be a flag complex and G_i be its 1-skeleton. We associate to K_i an augmented complex $\mathbb{K}_i \supseteq K_i$. As will be seen below, $\mathbb{K}_i$ is also a flag complex whose 1-skeleton will be denoted by $\mathbb{G}_i$. Following the terminology of [23], we call a vertex $v \in \mathbb{K}_i$ to be **active** if it is currently *not dominated*. The active closed neighborhood $ActN_{G_i}[v]$ is then defined as the set of all active vertices in $N_{G_i}[v]$. Similarly, $ActN_{G_i}[v \setminus u]$ denotes the set of active vertices in the closed neighborhood $N_{G_i}[v]$ of v that are not in $N_{G_i}[u]$. Finally, let $\{[u, ActN_{G_i}[v \setminus u]]\}$ denote the set of edges between u and $ActN_{G_i}[v \setminus u]$.

Using the notions defined above, we now explain how to inductively construct a filtration associated to a given flag tower. The construction operates in a streaming fashion on the 1-skeleton. We distinguish two kinds of inputs: elementary inclusions (of a vertex or an edge) and elementary contractions. For $i = 0$, we set $\mathbb{G}_0 = \emptyset$. We then define $\mathbb{G}_i$ as follows.

1. if $G_i \xrightarrow{\cup\sigma} G_{i+1}$ is an elementary *inclusion* where σ is either a vertex or an edge, we set $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \sigma$.
2. if $G_i \xrightarrow{\{u,v\}\mapsto u} G_{i+1}$ is an elementary *contraction*
   - 2.1. if $|ActN_{\mathbb{G}_i}[v \setminus u]| \leq |ActN_{\mathbb{G}_i}[u \setminus v]|$, we set $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \{[u, ActN_{\mathbb{G}_i}[v \setminus u]\}$ and v as contracted
   - 2.2. otherwise, we set $\mathbb{G}_{i+1} := \mathbb{G}_i \cup \{[v, ActN_{\mathbb{G}_i}[u \setminus v]\}$ and u as contracted.

Note that $\mathbb{G}_i \subseteq \mathbb{G}_{i+1}$ and thus $\mathbb{K}_i \subseteq \mathbb{K}_{i+1}$. We continue the construction until the end of our input tower.

Complexity Analysis: The vertices marked *contracted* during our construction are exactly the same as the inactive vertices defined in [23]. By construction, any contracted vertex (in the tower) will be dominated permanently in the final filtration. Indeed such a vertex stops existing in the tower (G) later on, and its neighborhood stays the same (in $\mathbb{G}$) and the vertex remains dominated. Therefore, at any point in our construction, the number of active vertices is less than the number of active vertices that are used in [23]. Moreover, since a strong expansion is a coning, the size of the final filtration in our construction is at most that obtained by the construction prescribed in [23]. Moreover, since we are working with 1-skeletons only, the space and time complexity of our method is much lower than that of [23].

Let us now analyze the time complexity of the algorithm. Notice that there are two basic operations on the 1-skeleton, elementary inclusions in Line 1, and the computation and comparison of $ActN_{\mathbb{G}_i}[v \setminus u]$ and $ActN_{\mathbb{G}_i}[u \setminus v]$ in Lines 2.1 and 2.2. Elementary inclusions can be performed in constant time $\mathcal{O}(1)$. To compute $ActN_{\mathbb{G}_i}[v \setminus u]$ we need to compute $N_{\mathbb{G}_i}[v]$, $N_{\mathbb{G}_i}[u]$ and the subset of vertices that are dominated (inactive) in $N_{\mathbb{G}_i}[v \setminus u]$. We can access $N_{\mathbb{G}_i}[v]$ and $N_{\mathbb{G}_i}[u]$ in constant time $\mathcal{O}(1)$ and compute the set-difference $N_{\mathbb{G}_i}[v \setminus u]$ in $\mathcal{O}(k \log k)$ time, where k is the maximum degree of a vertex. Finally computing the dominated vertices in $N_{\mathbb{G}_i}[v \setminus u]$ can be done in $\mathcal{O}(k^3)$ time as $|N_{\mathbb{G}_i}[v \setminus u]| \leq k$. Therefore computing $ActN_{\mathbb{G}_i}[v \setminus u]$ and $ActN_{\mathbb{G}_i}[u \setminus v]$ takes $\mathcal{O}(k^3)$ time. Since the size of active relative closed neighborhoods are bounded by k, for each elementary contraction, we include at most k edges in $\mathcal{O}(k)$ time. It follows that the worst-case time complexity for each elementary contraction is $\mathcal{O}(k^3)$.

We conclude that the time complexity of the algorithm is $\mathcal{O}(|\mathbb{G}_m| + n_c * k^3)$ where n_c is the number of elementary contractions in the input tower and $|\mathbb{G}_m|$ is the size of the 1-skeleton of the output flag filtration. The space complexity of our construction is $\mathcal{O}(n_0 * k)$ which is the size of a sparse adjacency matrix of a flag complex with n_0 vertices.

Correctness: Finally we prove a few lemmas and state our main result, Theorem 10, to certify the correctness of the output of our construction.

▶ **Lemma 6.** *Let $f_i : K_i \xrightarrow{\{u,v\}\mapsto u} K_{i+1}$ be the first elementary contraction in the tower $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{m-1}} K_m$. Then there exists a complex $K'_{i+1} \subseteq K_{i+1}$ such that $K_{i+1} \searrow\searrow K'_{i+1}$ and $\mathbb{K}_{i+1} \searrow\searrow K'_{i+1}$.*

Proof. Since f_i is the first contraction $\mathbb{K}_i = K_i$ and $\mathbb{G}_i = G_i$. Let $\mathbb{G}_{i+1} := G_i \cup \{[u, ActN_{G_i}[v \setminus u]\}$ be the graph defined in the construction Line 2.1. By construction, contracting v to u in both graphs G_i and $\mathbb{G}_{i+1}$ yields the same graph G_{i+1}.

Let $x' \in ActN_{G_i}[v \setminus u]$. We observe that adding the edge $[ux']$ to G_i does not change the fact that all $x \in \{N_{G_i}[v \setminus u] \setminus ActN_{G_i}[v \setminus u]\}$ are still dominated in $\mathbb{G}_{i+1}$ since the addition of $[ux']$ only adds neighbors to $N_{G_i}[x']$ and $N_{G_i}[u]$. Also, x is still dominated in G_{i+1} as well. Since the only vertex whose domination can change is the one that was dominated by v and the contraction $\{u, v\} \mapsto u$ transfers the neighborhood of v to u and therefore x would now be dominated by u.

Removing all the dominated vertices in $N_{G_i}[v \setminus u]$ thus provides a sequence of elementary

strong collapses both in K_{i+1} and $\mathbb{K}_{i+1}$. By performing all such elementary strong collapses, $\mathbb{K}_{i+1}$ is eventually transformed into a complex K^0_{i+1}, $\mathbb{K}_{i+1} \searrow\searrow K^0_{i+1}$ and K_{i+1} to K'_{i+1}, $K_{i+1} \searrow\searrow K'_{i+1}$. See Figure 4 for an example of all the graphs associated to the complexes defined here.

By doing so, in $\mathbb{G}_{i+1}$ we have removed all the dominated vertices from $N_{G_i}[v \setminus u]$ and added edges between u and the non dominated vertices that are in $N_{G_i}[v \setminus u]$. This implies that v is dominated by u in K^0_{i+1}. The elementary strong collapse of v onto u, implies $K^0_{i+1} \searrow\searrow K'_{i+1}$ and therefore $\mathbb{K}_{i+1} \searrow\searrow K'_{i+1}$. ◀

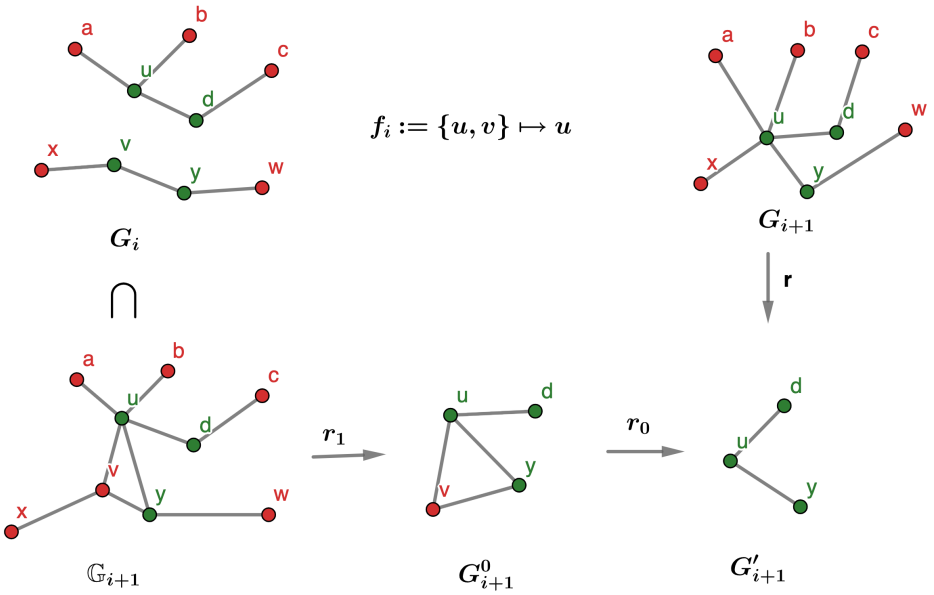

Figure 4 An example of the graphs associated to the complexes defined in the proof of Lemma 6. Vertices in red are dominated (inactive) and in green are active.

▶ **Lemma 7.** *Let $f_i : K_i \xrightarrow{\{u,v\}\mapsto u} K_{i+1}$ be the first elementary contraction in the tower $\mathbb{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{m-1}} K_m$. Then the following diagram commutes*

$$\begin{array}{ccc} H_p(K_i) & \xrightarrow{f_i^*} & H_p(K_{i+1}) \\ & \searrow^{i_k^*} & \downarrow{\scriptstyle \phi^*_{i+1}} \\ & & H_p(\mathbb{K}_{i+1}) \end{array}$$

and ϕ^*_{i+1} is an isomorphism.

Here $\phi_{i+1} := i' \circ r : K_{i+1} \xrightarrow{r} K'_{i+1} \xhookrightarrow{i'} \mathbb{K}_{i+1}$ is the composition of the retraction r and the inclusion i' associated to corresponding strong collapses as defined in Lemma 6. i_k^*, f_i^* and ϕ^*_{i+1} are homomorphisms induced by the corresponding simplicial maps.

Proof. Using the fact that f_i is the first contraction, we have the inclusion $\mathbb{K}_i = K_i \subseteq \mathbb{K}_{i+1}$. Let K^0_{i+1} and K'_{i+1} be the complexes as defined in the proof of Lemma 6. Consider the following diagram of simplicial complexes, and note that $i' = i_1 \circ i_0$ where i_0 and i_1 are both inclusions induced by the respective strong collapses and $r \circ f_i := K_i \xrightarrow{f_i} K_{i+1} \xrightarrow{r} K'_{i+1}$.

$$\begin{array}{ccc} K_i & \xrightarrow{r\circ f_i} & K'_{i+1} \\ \downarrow{\scriptstyle i_k} & & \downarrow{\scriptstyle i_0} \\ \mathbb{K}_{i+1} & \xleftarrow[i_1]{} & K^0_{i+1} \end{array}$$

We claim that the maps $\phi_{i+1} \circ f_i$ and i_k are in the same contiguity class, which we denote $\phi_{i+1} \circ f_i \sim i_k$.

Let r_0 and r_1 be the retraction maps associated with the strong collapses $K^0_{i+1} \searrow\searrow K'_{i+1}$ and $\mathbb{K}_{i+1} \searrow\searrow K^0_{i+1}$ respectively. We have $i_1 \circ i_0 \circ r_0 \circ r_1 \sim 1_{\mathbb{K}_{i+1}}$ [2], where $1_{\mathbb{K}_{i+1}}$ is the identity over $\mathbb{K}_{i+1}$.

Now we observe that the retraction map r_0 is the same as f_i as it is the elementary contraction $\{u, v\} \mapsto u$. And the retraction map $r_1 : \mathbb{K}_{i+1} \to K^0_{i+1}$ is the same as the retraction map $r : K_{i+1} \to K'_{i+1}$ expect for any vertex x that is dominated by v in $\mathbb{K}_{i+1}$. For such x, $r_1(x) = v$ and $r(x) = u$ as x is dominated by u in K_{i+1} and is mapped to u through the map r [see proof of Lemma 6]. This implies that, the composition $r_0 \circ r_1 = r \circ f_i$ for all vertices in K_i or $\mathbb{K}_{i+1}$ (K_i and $\mathbb{K}_{i+1}$ have the same vertex set), see Figure 4 for an illustration.

By replacing $i_1 \circ i_0$ by i' and $r_0 \circ r_1$ by $r \circ f_i$, we have $i_1 \circ i_0 \circ r_0 \circ r_1 = i' \circ r \circ f_i \sim 1_{\mathbb{K}_{i+1}}$. Now since $K_i \subseteq \mathbb{K}_{i+1}$ and K_i and $\mathbb{K}_{i+1}$ have the same vertex set $i_k = 1_{\mathbb{K}_{i+1}}$ also by definition $\phi_{i+1} = i' \circ r$, therefore $\phi_{i+1} \circ f_i \sim i_k$. Since maps in the same contiguity class are homotopic at the level of geometric realizations, the diagram in the lemma commutes.

It follows directly from the definition that ϕ^*_{i+1} is an isomorphism as $\phi^*_{i+1} = (i')^* \circ r^*$ and $(i')^*$ and r^* both are isomorphism since (i') and r are inclusion and retraction maps associated to strong collapse. ◀

Proceeding by induction, Lemma 6 then immediately implies the following result.

▶ **Lemma 8.** *Given a tower $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{m-1}} K_m$. For each $0 \le i \le m$, there exists a map $\phi_i : K_i \to \mathbb{K}_i$ that is an isomorphism, where ϕ_i is a composition of retraction and inclusion associated to strong collapse.*

Again, using an inductive argument along with Lemmas 7 and 8, we can deduce the following result.

▶ **Theorem 9.** *The following diagram commutes and all the vertical maps ϕ^*_i are isomorphisms.*

$$\begin{array}{ccccccc} H_p(K_1) & \xrightarrow{f_1^*} & H_p(K_2) & \xrightarrow{f_2^*} & \cdots & \longrightarrow H_p(K_{m-1}) & \xrightarrow{f_{m-1}^*} H_p(K_m) \\ \downarrow{\scriptstyle \phi_1^*} & & \downarrow{\scriptstyle \phi_2^*} & & & \downarrow{\scriptstyle \phi_{m-1}^*} & \downarrow{\scriptstyle \phi_m^*} \\ H_p(\mathbb{K}_1) & \xrightarrow{*} & H_p(\mathbb{K}_2) & \xrightarrow{*} & \cdots & \longrightarrow H_p(\mathbb{K}_{m-1}) & \xrightarrow{*} H_p(\mathbb{K}_m) \end{array}$$

As a consequence, the tower $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{m-1}} K_m$ and the constructed filtration $\mathcal{F} : \mathbb{K}_0 \hookrightarrow \mathbb{K}_1 \hookrightarrow \ldots \hookrightarrow \mathbb{K}_m$ have the same persistence diagram.

Here ϕ_i is a composition of retraction map and inclusion map associated to strong collapse (as defined in Lemma 6) and therefore an isomorphism for each $i \in \{0, \cdots, m\}$ and $*$ indicates the induced homomorphisms.

We summarize our result in the following theorem. We write $\mathcal{T} : K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} ... \xrightarrow{f_{m-1}} K_m$ for the given flag tower where, w.l.o.g., $K_0 = \emptyset$. and each f_i is either an inclusion or an elementary contraction. The inclusions are not necessarily elementary but corresponds to an elementary inclusion on the graphs G_i. We denote by d the maximal dimension of the K_is in $\mathcal{T}$, and by n the total number of elementary inclusions of simplices in $\mathcal{T}$, by n_c the total number of elementary contractions and by n_0 the number of vertex inclusions in $\mathcal{T}$.

▶ **Theorem 10.** *There exists a filtration $\mathcal{F} : \mathbb{K}_0 \hookrightarrow \mathbb{K}_1 \hookrightarrow ... \hookrightarrow \mathbb{K}_m$, where the inclusions are not necessarily elementary, such that $\mathcal{T}$ and $\mathcal{F}$ have the same persistence diagram and the size of the filtration $|\mathbb{K}_m|$ is at most $\mathcal{O}(d * n * \log n_0)$. Moreover, $\mathcal{F}$ is a flag filtration which can be computed from $\mathcal{T}$ using only the 1-skeletons G_is of the K_is. The time complexity of the algorithm is $O(|\mathbb{G}_m| + n_c * k^3)$ time and its space complexity $\mathcal{O}(n_0 * k)$, where $\mathbb{G}_m$ denotes the 1-skeleton of $\mathbb{K}_m$ and k is an upper bound on the degree of the vertices in $\mathbb{G}_m$.*

▶ Remark 11. The above construction can easily be generalized to any zigzag flag sequence, i.e. one can convert a general zigzag flag sequence to an equivalent zigzag flag filtration. The complete pipeline for a zigzag sequence will be : strong collapse the input zigzag flag sequence to a reduced zigzag flag sequence, and then transform the reduced zigzag sequence to an equivalent zigzag flag filtration whose PH can be computed using known algorithms [11].

6 Approximation of the persistence diagram of filtrations

In this section, we provide a general scheme to approximate any given tower and a more specific one for the special case of Vietoris-Rips filtrations. We then perform some experiments and compute approximate persistence diagrams of Vietoris Rips filtrations to showcase the efficiency of our method.

General approximation scheme for flag towers: Let $\mathcal{T} : \{K_1 \xrightarrow{f_1} K_2 \xrightarrow{f_2} \cdots \xrightarrow{f_{(m-1)}} K_m\}$ be a flag tower of which we want to compute the persistence diagram. We first strong collapse the various complexes K_i, $i = 1, ..., m$ and then compute a *core tower* that connects the K_i^cs through induced simplicial maps as described in Section 4. Since each K_i is a flag complex, the computation of its core can be computed much more efficiently using only its 1-skeleton as discussed in detail in the Section 3.2. Lastly, we compute a flag *filtration* with the same PH as the core tower as described in Section 5. This filtration can then be sent to any algorithm that computes the persistence homology of a flag filtration [27, 3, 4, 21].

It is important to note that the algorithm computes the exact PH of the input flag tower $\mathcal{T}$. However, instead of considering all the K_is and the associated simplicial maps, we can select some of the K_is we rename $K'_1, ..., K'_q$, and compose the original maps $f_1, ..., f_{m-1}$ to obtain simplicial maps $f'_1, ..., f'_{q-1}$ connecting the selected complexes. We thus obtain a subtower $\mathcal{T}'$ and the algorithm will then compute the exact PH of $\mathcal{T}'$. Below we present an application of this idea to Vietoris-Rips filtrations. By rounding the values of the threshold parameter to a given number of snapshots (defined below) of the threshold value, we will be able to approximate in a controlled way the PH of the initial filtration $\mathcal{T}$.

Approximate persistence diagram of a VR filtration: Given a VR filtration, one can choose to collapse the original complexes after each edge inclusion. However, as mentioned before, we can also choose to strong collapse the complexes less often, i.e. after several edge inclusions rather than just one. This will result in a faster algorithm but comes with a cost: the computed PD is then only approximate. We call **snapshots** the values of the scale parameter at which we choose to strong collapse the complex. The difference between two consecutive snapshots is called a **step**. We approximate the filtration value of a simplex as the value of the snapshot at which it first appears. As a consequence, our algorithm will report all persistence pairs that are separated by at least one snapshot. Hence if all steps are equal to some $\epsilon > 0$, we will compute all the persistence pairs whose lengths are at least ϵ. It follows that the l_∞-*bottleneck distance* between the computed PD and the exact one is at most ϵ. If instead the ratio between any two consecutive steps is taken to be a constant $\rho > 1$, the l_∞-bottleneck distance will be at most $\log \rho$ after reparameterizing the filtrations on a log-log-scale [29].

Experimental setup : Our algorithm has been implemented for Vietoris-Rips (VR) filtrations as a C++ module named VertexCollapser. Recall that, for VR-filtrations, the filtration value of a simplex is the length of the longest edge of the simplex. VertexCollapser takes as input a VR filtration and returns the reduced flag filtration as shown above. VertexCollapser can be used in two modes: in the exact mode, the output filtration has the same PD as the input filtration while, in the approximate mode as described above, a certified approximation is returned. The output filtration can then be sent to any software that computes the PD of a VR-filtration such as the Gudhi library (the one we chose for our experiments) [21] or Ripser [3]. Therefore VertexCollapser is to be considered as an ad-on to software computing PD. VertexCollapser will be available as an open-source package of a next release of the Gudhi library.

We present results on five datasets **netw-sc**, **senate**, **eleg**, **HIV** and **drag 1** that are publicly available [14]. Each dataset is given as the interpoint distance matrix. The reported time includes the time of VertexCollapser and the time to compute the persistent diagram (PD). The time of VertexCollapser includes: 1. The time taken to compute the largest 1-skeleton associated to the maximum threshold value, 2. The time taken to collapse all the sub-skeletons and assemble their cores. 3. To transform them into an equivalent flag-filtration.

The code has been compiled using the compiler 'clang-900.0.38' and all computations were performed on a '2.8 GHz Intel Core i5' machine with 16 GB of available RAM. We took all steps to be equal. Parameter *Step* controls the quality of the approximation : if $Step = 0$, we obtain the exact PD, otherwise *Step* is an upper bound on the l_∞-*bottleneck distance* between the output diagram and the exact one. VertexCollapser works irrespective of the dimension of the input complexes. However, the size of the complexes in the reduced filtration, even if much smaller than in the original filtration, might exceed the capacities of the PD computation algorithm. For this reason, we introduced a parameter *dim* and restricts PD computation to dimension at most *dim*.

Some experimental results are reported in Table 2. We first observe that the reduction done by VertexCollapser is enormous. The reduced complexes are small and of low dimension (column Size/Dim) compared to the input VR-complexes which are of dimensions respectively 57, 54 and 105 for the first three datasets **netw-sc**, **senate** and **eleg**. We also observe that, while the time taken by VertexCollapser is large for exact persistence computation, very good approximations can be obtained fast. Moreover the computing time mildly increases

with the number of snapshots. This suggests that implementing the collapses in parallel would lead to further substantial improvement.

Data	Pnt	Thrsld	VertexCollapser +PD					
			Size/Dim	*dim*	Pre-Time	Tot-Time	*Step*	Snaps
netw-sc	379	5.5	155/3	∞	7.28	7.38	0.02	263
”	”	”	155/3	∞	13.93	14.03	0.01	531
”	”	”	175/3	∞	366.46	366.56	0	8420
senate	103	0.415	405/4	∞	2.53	2.54	0.001	403
”	”	”	417/4	∞	15.96	15.98	0	2728
eleg	297	0.3	577K/15	∞	11.65	26.02	0.001	284
”	”	”	835K/16	∞	518.36	540.40	0	9850
HIV	1088	1050	127.3M/?	4	660	3,955	4	184
drag1	1000	0.05	478.3M/?	4	687	14,170	0.0002	249

Table 2 The columns are, from left to right: dataset (Data), number of points (Pnt), maximum value of the scale parameter (Thrsld), number of simplices (Size) and dimension of the final filtration (Dim), parameter (*dim*), time (in seconds) taken by VertexCollapser, total time (in seconds) including PD computation (Tot-Time), parameter *Step* and the number of snapshots used (Snaps). The ? sign in column (Dim) means that we could not compute the dimension of the final filtration and restricted to the dimension in column (dim).

Comparison with Ripser: Ripser [3] is the state of the art software to compute persistent diagrams of VR filtrations. It computes the *exact* PD associated to the input filtration up to dimension *dim*. Although VertexCollapser is more complementary to Ripser than a competitor, we run Ripser[1] on the same datasets as in Table 2 to demonstrate the benefit of using VertexCollapser. Results are presented in Table 3. The main observation is that Ripser performs quite well in low dimensions but its ability to handle higher dimensions is limited.

Data	Pnt	Threshold	Val		Val		Val	
			dim	Time	*dim*	Time	*dim*	Time
netw-sc	379	5.5	4	25.3	5	231.2	6	∞
senate	103	0.415	3	0.52	4	5.9	5	52.3
”	”	”	6	406.8	7	∞		
eleg	297	0.3	3	8.9	4	217	5	∞
HIV	1088	1050	2	31.35	3	∞		
drag1	1000	0.05	3	249	4	∞		

Table 3 Time is the total time (in seconds) taken by Ripser. ∞ means that the experiment ran longer than 12 hours or crashed due to memory overload.

[1] We used the command <./ripser inputData –format distances –threshold inputTh –dim inputDim >.

7 Discussion

In this article, we presented a novel approach to compute the persistence homology of a sequence of simplicial complexes. Our approach is based on strong collapses that has been introduced by Barmak and Minian [2]. Our method works very well in pratice and, as shown using publicly available datasets, improves a lot the time and space complexity of PH computation. We believe that the solid mathematical foundations presented in [2], its applicability to all kinds of sequences of simplicial complexes, and the availability of the simple and efficient algorithms developed in this article, will make strong collapses immensely useful to reduce the complexity of many problems in computational topology.

On the theoretical side, this work raises several questions. In particular, it would be nice to have theoretical guarantees on the amount of reduction the algorithm can achieve. We intend to explore this and related issues in future work.

Acknowledgements. We want to thank Marc Glisse for useful discussions and Mathijs Wintraecken for reviewing a draft of the article. We also want to thank Francois Godi and Siargey Kachanovich for their help with Gudhi, and Hannah Schreiber for her help with Sophia.

References

1 M. Adamaszek and J. Stacho. Complexity of simplicial homology and independence complexes of chordal graphs. *Computational Geometry: Theory and Applications*, 57:8–18, 2016.
2 J. A. Barmak and E. G. Minian. Strong homotopy types, nerves and collapses. *Discrete and Computational Geometry*, 47:301–328, 2012.
3 U. Bauer. Ripser. URL: `https://github.com/Ripser/ripser`.
4 U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. PHAT – persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78, 2017.
5 J-D. Boissonnat and C. S. Karthik. An efficient representation for filtrations of simplicial complexes. In *ACM Transactions on Algorithms*, 2018.
6 J-D. Boissonnat, C. S. Karthik, and S. Tavenas. Building efficient and compact data structures for simplicial complexes. *Algorithmica*, 79:530–567, 2017.
7 J-D. Boissonnat and S. Pritam. Computing persistent homology of flag complexes via strong collapses. *International Symposium on Computational Geometry (SoCG)*, pages 55:1–55:15, 2019.
8 J-D. Boissonnat, S.Pritam, and D. Pareek. Strong Collapse for Persistence. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112, pages 67:1–67:13, 2018.
9 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.
10 G. Carlsson and V. de Silva. Zigzag persistence. *Found Comput Math*, 10:367–405, 2010.
11 G. Carlsson, V. de Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. *International Symposium on Computational Geometry (SoCG)*, pages 247–256, 2009.
12 F. Chazal and S. Oudot. Towards persistence-based reconstruction in Euclidean spaces. *International Symposium on Computational Geometry (SoCG)*, 2008.
13 A. Choudhary, M. Kerber, and S. Raghvendra:. In *Polynomial-Sized Topological Approximations Using The Permutahedron*, volume 61, pages 42–80, 2019.
14 Datasets. URL: `https://github.com/n-otter/PH-roadmap/''`.
15 H. Derksen and J. Weyman. Quiver representations. *Notices of the American Mathematical Society*, 52(2):200–206, February 2005.

16 T. K. Dey, F. Fan, and Y. Wang. Computing topological persistence for simplicial maps. In *International Symposium on Computational Geometry (SoCG)*, pages 345–354, 2014.

17 T. K. Dey, D. Shi, and Y. Wang. *SimBa: An efficient tool for approximating Rips-filtration persistence via Simplicial Batch-collapse*. In European Symp. on Algorithms (ESA), pages 35:1–35:16, 2016.

18 P. Dłotko and H. Wagner. Simplification of complexes for persistent homology computations,. *Homology, Homotopy and Applications*, 16:49–63, 2014.

19 E. Fieux and J. Lacaze. Foldings in graphs and relations with simplicial complexes and posets. *Discrete Mathematics*, 312(17):2639 – 2651, 2012.

20 F. Le Gall. Powers of tensors and fast matrix multiplication. *ISSAC '*, 14:296–303, 2014.

21 Gudhi: Geometry understanding in higher dimensions. URL: `http://gudhi.gforge.inria.fr/`.

22 A. Hatcher. *Algebraic Topology*. Univ. Press Cambridge, 2001.

23 M. Kerber and H. Schreiber:. Barcodes of towers and a streaming algorithm for persistent homology. *Discrete and Computational Geometry*, 61:852–879, 2019.

24 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.

25 N. Milosavljevic, D. Morozov, and P. Skraba. Zigzag persistent homology in matrix multiplication time. In *International Symposium on Computational Geometry (SoCG)*, page 216–225, 2011.

26 K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete and Computational Geometry*, 50:330–353, September 2013.

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

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

29 D. Sheehy. Linear-size approximations to the Vietoris–Rips filtration. *Discrete and Computational Geometry*, 49:778–796, 2013.

30 J. H. C Whitehead. Simplicial spaces nuclei and m-groups. *Proc. London Math. Soc*, 45:243–327, 1939.

31 A. C. Wilkerson, H. Chintakunta, and H. Krim. Computing persistent features in big data: A distributed dimension reduction approach. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 11–15, 2014.

32 A. C. Wilkerson, T. J. Moore, A. Swami, and A. H. Krim. Simplifying the homology of networks via strong collapses. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 11–15, 2013.

33 A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete and Computational Geometry*, 33:249–274, 2005.