Computing discrete Morse complexes from simplicial complexes

Ulderico Fugacci[a], Federico Iuricich[b], Leila De Floriani[c]

[a] *Graz University of Technology, Graz, Austria*
[b] *Clemson University, Clemson, SC, USA*
[c] *University of Maryland, College Park, MD, USA*

Abstract

We consider the problem of efficiently computing a discrete Morse complex on simplicial complexes of arbitrary dimension and very large size. Based on a common graph-based formalism, we analyze existing data structures for simplicial complexes, and we define an efficient encoding for the discrete Morse gradient on the most compact of such representations. We theoretically compare methods based on reductions and coreductions for computing a discrete Morse gradient, proving that the combination of reductions and coreductions produces new mutually equivalent approaches. We design and implement a new algorithm for computing a discrete Morse complex on simplicial complexes. We show that our approach scales very well with the size and the dimension of the simplicial complex also through comparisons with the only existing public-domain algorithm for discrete Morse complex computation. We discuss applications to the computation of multi-parameter persistent homology and of extrema graphs for visualization of time-varying 3D scalar fields.

Keywords: Shape analysis, topological data analysis, discrete Morse theory, homology, persistent homology, shape understanding, scientific data visualization.

1. Introduction

In recent years, computational topology has become a fundamental tool for the analysis and visualization of scientific data. In particular, the efficient development of software tools for extracting topological features from data has led to an increasing number of applications of topology-based approaches in shape analysis and understanding, and in particular in the analysis of sensor [1] and social [2] networks, in chemistry [3], in astrophysics [4], in medicine [5]. Several mathematical tools have been studied for computing a compact, topologically-equivalent object starting from a simplicial complex of large size. Examples of these tools are the discrete Morse complex [6], the size graph [7], and the tidy set [8].

Discrete Morse Theory (DMT) [6] is a powerful theory defined in a completely combinatorial setting, that aims at the construction of a discrete representation of a given simplicial complex, based on a *discrete Morse gradient* (also called *Forman gradient* or *discrete gradient field*) from which a homology-equivalent chain complex, the *discrete Morse complex* is built. The Forman gradient and the associated discrete Morse complex have been used both for the analysis and visualization of scalar fields [9], and for computing standard and persistent homology [10, 11, 12].

In very recent research areas, like the analysis of higher dimensional scalar fields [13] or in the analysis of shapes based on multi-parameter persistent homology [14, 15], there is a need for efficient methods capable of encoding a Forman gradient on higher dimensional simplicial complexes.

In this work, we introduce the first complete study for implementing a Forman gradient on high dimensional simplicial complexes. We start from a theoretical evaluation of the various methods used for building a Forman gradient, which are generally called reduction-based or coreduction-based.

We describe a third method initially formulated in [16], obtained by interleaving reductions and coreductions, and we prove the equivalence of all three techniques. This equivalence will provide us the freedom to implement the method that best fits any given data structure.

To this aim, we undertake a theoretical and experimental evaluation of the three most common data structures for encoding simplicial complexes. Here, we focused on data structures with available public-domain implementations. Our experiments clearly show that the Generalized Indexed data structure with Adjacencies (IA^*) [17], a data structure encoding only the vertices and a subset of the simplices of the complex, is the only one that can suitably scale to higher dimensions without being affected by the exponential growth in the number of simplices. We propose a solution to compactly encode a Forman gradient attached to the IA^* data structure.

Based on the latter encoding, we have defined and implemented an efficient, dimension-independent, algorithm for computing a Forman gradient and for retrieving the discrete Morse complex defined by it, which is fundamental for computing, among others, homology and persistent homology. We compare our approach to the one developed in [11] and implemented in the software library *Perseus* [18] which computes a discrete Morse complex using a data structure implementing the Hasse diagram of the complex, the *Incidence Graph*. Our experiments show that our approach is more efficient and it is also easy to parallelize.

The remainder of the paper is organized as follows. Section 2 introduces some preliminary notions about simplicial complexes, simplicial and persistent homology, and discrete Morse theory. Section 3 reviews some classical topological data structures for simplicial complexes as well as algorithms for computing a Forman gradient and a discrete Morse complex. In Section 4, we introduce, evaluate and compare the data structures for compactly encoding a simplicial complex and we discuss a new compact encoding for the Forman gradient. In Section 5, we present the reduction and coreduction-based algorithms used for computing a Forman gradient. Section 6 is devoted to the formal proof of the theoretical equivalence of these approaches, while, in Section 7, we introduce a new approach based on the interleaving of the two. In Section 8, we describe a coreduction-based algorithm for building a discrete Morse complex based on the IA^* data structure and on the compact representation of the Forman gradient. In Section 9, we evaluate the performances of our algorithm on a variety of input complexes. Finally, in Section 10, we draw some concluding remarks and discuss applications of our approach to single and multi-parameter persistent homology computation and to the analysis and visualization of time-varying 3D scalar fields.

2. Background

In this section, we introduce some notions which are at the basis of our work. We briefly define and discuss simplicial complexes, simplicial homology and persistent simplicial homology, as well as discrete Morse theory.

2.1. Simplicial complexes

A *k-simplex* σ is the convex hull of $k+1$ affinely independent points in the Euclidean space. For instance, a 0-simplex is a single point, a 1-simplex an edge, a 2-simplex a triangle, and a 3-simplex a tetrahedron. Given a k-simplex σ, the *dimension* of σ is defined to be k, and denoted as $dim(\sigma)$. Any simplex σ', which is the convex hull of a non-empty subset of the points generating σ, is called a *face* of σ. Conversely, σ is called a *coface* of σ'.

A *simplicial complex* Σ is a finite set of simplices such that:

- each face of a simplex in Σ belongs to Σ;
- each non-empty intersection of any two simplices in Σ is a face of both.

We define the *dimension* of a simplicial complex Σ, denoted as $dim(\Sigma)$, as the largest dimension of its simplices. Given a simplex σ of Σ, we define the *star* of σ as the set of the cofaces of σ in Σ. A simplex σ is called a *top simplex* if its star consists only of σ itself. Given a simplex σ face/coface of σ', σ and σ' are said to be *incident*. For $k > 0$, two k-simplices in Σ are said to be *adjacent* if they share a face of dimension $k-1$, while two 0-simplices u and v in Σ are called *adjacent* if they are both faces of the same 1-simplex.

Queries on a simplicial complex are often expressed in terms of the topological relations defined by the adjacencies and incidences among its simplices.

- *Boundary relations:* given a q-simplex τ and a k-simplex σ with $q > k$, we say that σ is in *boundary* (q,k)*-relation* with τ if σ is a face of τ. We denote as $bd_{q,k}(\tau)$ the set of simplices in boundary (q,k)-relation with τ.

- *Coboundary relations:* given a q-simplex τ and a k-simplex σ with $q > k$, we say that τ is in *coboundary* (k,q)*-relation* with σ if τ is a coface of σ. We denote as $cbd_{k,q}(\sigma)$ the set of simplices in coboundary (k,q)-relation with σ.

- *Adjacency relations:* given two k-simplices σ and σ', we say that σ is in *adjacency* (k,k)*-relation* with σ' if σ is adjacent to σ'. We denote as $adj_{k,k}(\sigma)$ (or, simply $adj(\sigma)$) the set of simplices in adjacency (k,k)-relation with σ.

In the following, we will call *immediate* boundary and coboundary relations those boundary and coboundary relations involving simplices of consecutive dimensions. In the following, we will often refer to them as $bd(\cdot)$ and $cbd(\cdot)$ or, when we need to explicit the complex Σ with respect to these relations are considered, as $bd_\Sigma(\cdot)$ and $cbd_\Sigma(\cdot)$. Figure 1 illustrates the topological relations of a 1-simplex (edge) σ_0 in a simplicial complex Σ. Simplices in the immediate boundary, immediate coboundary and adjacency relations are depicted in blue, red, and green, respectively. Specifically, $bd_{1,0}(\sigma_0) = \{v_1, v_3\}$, $cbd_{1,2}(\sigma_0) = \{\tau\}$, and $adj_{1,1}(\sigma_0) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$.

Simplicial complexes are a subclass of the more general class of cell complexes [19]. They are extensively used because of their combinatorial properties and of the possibility of representing collections of unorganized sets of points, usually called *point clouds*. *Alpha-shapes* [20], *Delaunay triangulations* [21], *Čech complexes* [22], *Vietoris-Rips complexes* [23], *witness complexes* [24, 25, 26] and *graph-induced complexes* [27] are different ways for endowing a point cloud with a simplicial structure. *Čech complexes* are the most classical way to build a simplicial complex starting from a point cloud,

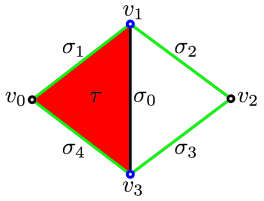


Figure 1: Topological relations of edge σ_0. Immediate boundary relation $bd_{1,0}(\sigma_0)$ consists of the two blue vertices v_1, v_3. Immediate coboundary relation $cbd_{1,2}(\sigma_0)$ consists of the red triangle τ. Adjacency relation $adj_{1,1}(\sigma_0)$ consists of the four green edges σ_1, σ_2, σ_3, σ_4.

but their construction requires exponential time in the number of the input points [23].

Vietoris-Rips (VR) complexes [23] represent a compromise between Čech complexes and the approximations based on subsampling adopted by witness [24, 25, 26] and graph-induced [27] complexes. Let $G = (N, A)$ be a graph, a *clique* in G is defined as a complete subgraph of G. The *flag complex* of G, denoted as $Flag(G)$, is the simplicial complex whose simplices correspond to the cliques of G. Given a finite set of points P in a metric space (such as the Euclidean space) and a positive real number ϵ, the *Vietoris-Rips (VR) complex* is the flag complex of the graph whose set of nodes coincides with P and having an arc for each pair of points in P whose distance is at most ϵ. Figure 2(a) shows, for each point of a set P, the neighboring points at a distance less or equal to ϵ. Figure 2(b) shows the edges connecting points in P whose mutual distance is less or equal ϵ. Figure 2(c) shows the cliques computed on graph G and the resulting VR complex Σ.

2.2. Simplicial and persistent homology

Simplicial homology provides invariants for shape description and characterization. Given a simplicial complex Σ, we define the *chain complex* associated with Σ as the pair $C_*(\Sigma) := (C_k(\Sigma), \partial_k)_{k\in\mathbb{Z}}$, where:

- $C_k(\Sigma)$ is the free Abelian group whose elements, called k*-chains*, are linear combinations with integer coefficients of the k-simplices of Σ;

- $\partial_k : C_k(\Sigma) \to C_{k-1}(\Sigma)$ is the homomorphism encoding the boundary relations between the k-simplices and those $(k-1)$-simplices of Σ such that $\partial^2 = 0$.

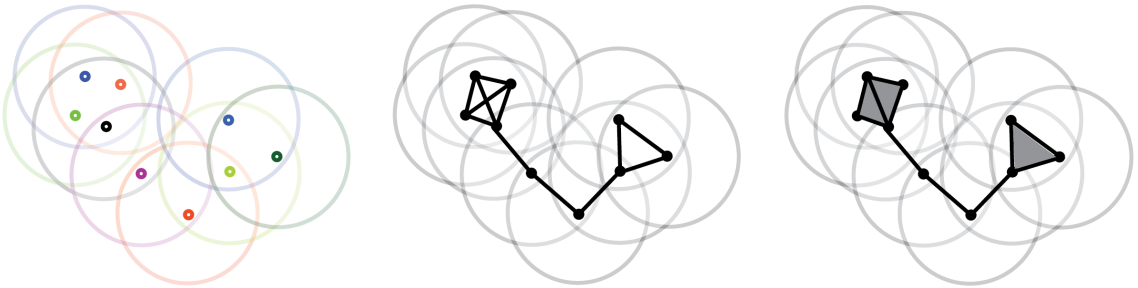


Figure 2: Construction of a VR complex Σ: given a finite set of points P, the disks of radius ϵ are computed (a); an edge is created for each pair of points at distance less than ϵ (b); the VR complex is retrieved by adding a simplex for each clique of the obtained graph (c).

Given $C_*(\Sigma)$, we denote as $Z_k(\Sigma) := \ker \partial_k$ the group of the k-cycles of Σ, and as $B_k(\Sigma) := \mathrm{Im}\, \partial_{k+1}$ the group of the k-boundaries of Σ. The k^{th} *homology group* of Σ is defined as $H_k(\Sigma) := H_k(C_*(\Sigma)) = Z_k(\Sigma)/B_k(\Sigma)$. Intuitively, homology groups reveal the presence of "holes" in a simplicial complex Σ. The non-null elements of each homology group are cycles, which do not represent the boundary of any collection of simplices of Σ. The rank β_k of the k^{th} homology group of a simplicial complex Σ is called the k^{th} *Betti number* of Σ. In particular, β_0 counts the number of connected components of Σ, β_1 its tunnels and holes, and β_2 the shells surrounding voids or cavities.

Persistent homology [28, 29, 30] aims at overcoming intrinsic limitations of standard homology by allowing for a multi-scale approach defined through a filtration. Let Σ be a simplicial complex, a *filtration* F of Σ is a finite sequence of subcomplexes $\{\Sigma^m \,|\, 0 \leq m \leq M\}$ of Σ such that $\emptyset = \Sigma^0 \subseteq \Sigma^1 \subseteq \cdots \subseteq \Sigma^M = \Sigma$. The *p-persistent* k^{th} *homology group* $H_k^p(\Sigma^m)$ of Σ^m consists of the k-cycles included from $C_k(\Sigma^m)$ into $C_k(\Sigma^{m+p})$ modulo boundaries.

Figure 3 shows an example of a filtration of a simplicial complex Σ. Persistent homology detects the changes in the homology of Σ and it allows distinguishing between relevant homology classes, such as the 1-cycle in Σ^1 which is born at step (a) and persists until the end of the filtration, and negligible homology classes like, for instance, the 1-cycle which is born at step (b) and immediately dies at step (c).

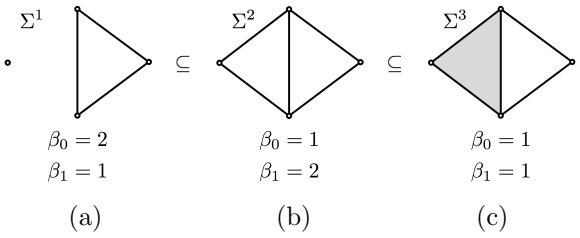


Figure 3: A filtration of a simplicial complex Σ. In (a), Σ^1 consists of two different connected components and one non-boundary 1-cycle (a); in (b), Σ^2 gains a non-boundary 1-cycle while it becomes connected; finally, in (c), the 1-cycle created at step (b) becomes the boundary of the unique triangle in Σ^3 and its contribution in homology vanishes.

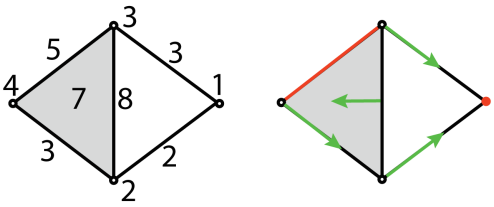
2.3. Discrete Morse theory

Discrete Morse theory due to Forman [6, 31] provides a powerful tool for analyzing the topology of an object. It has been defined for cell complexes but, for the sake of simplicity, we will review discrete Morse theory in the context of simplicial complexes.

A simplicial complex Σ is endowed with a function $f : \Sigma \to \mathbb{R}$, called a *discrete Morse function* if, for every simplex σ in Σ,

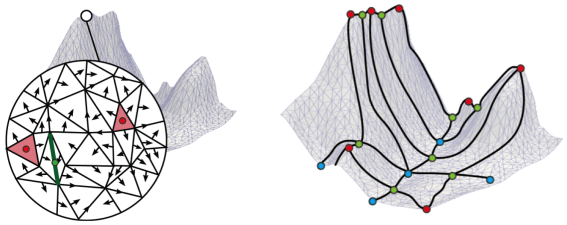
- $c^+(\sigma) := \#\{\tau \in cbd(\sigma) \,|\, f(\tau) \leq f(\sigma)\} \leq 1$,
- $c^-(\sigma) := \#\{\rho \in bd(\sigma) \,|\, f(\rho) \geq f(\sigma)\} \leq 1$.

It is easy to show (see [6], Lemma 2.5) that, for a discrete Morse function, $c^+(\sigma)$ and $c^-(\sigma)$ cannot be simultaneously equal to 1. A k-simplex σ in Σ



(a) (b)

Figure 4: (a) A discrete Morse function on a simplicial complex and (b) the corresponding Forman gradient (red simplices are critical simplices).



(a) (b)

Figure 5: (a) A Forman gradient computed on a simplicial complex and (b) the graph structure formed by the gradient paths (boundary maps $\tilde{\partial}_k$) connecting the critical simplices (groups $\mathcal{M}_k$).

is called *critical simplex of index k* (or, *k-saddle*) if $c^+(\sigma) = c^-(\sigma) = 0$. A critical simplex of index 0 is called a *minimum*, while a critical simplex of index $d = dim(\Sigma)$ a *maximum*. Figure 4(a) shows a discrete Morse function f defined on a simplicial complex. Each simplex is labeled with the corresponding value of function f. Vertex 1 is critical (minimum), since f has a higher value on all edges incident to it. Edge 5 is critical (saddle), since f has a higher value on the incident triangle 7, and lower values on its vertices.

A *discrete vector field* V on Σ is a collection of pairs of simplices $(\sigma, \tau) \in \Sigma \times \Sigma$ such that $\sigma \in bd(\tau)$ and each simplex of Σ is in at most one pair of V. A discrete Morse function $f : \Sigma \to \mathbb{R}$ induces a discrete vector field $V = \{(\sigma, \tau) \in \Sigma \times \Sigma \,|\, \sigma \in bd(\tau) \text{ and } f(\sigma) \geq f(\tau)\}$, called a *Forman gradient* (or, equivalently, *gradient vector field*) of f on Σ. A pair $(\sigma, \tau) \in V$ can be depicted as an arrow from σ to τ. Given a discrete vector field V, a *V-path* (or, equivalently, a *gradient path*) is a sequence $[(\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots, (\sigma_r, \tau_r)]$ of pairs of k-simplices σ_i and $(k+1)$-simplices τ_i, such that $(\sigma_i, \tau_i) \in V$, σ_{i+1} is a face of τ_i, and $\sigma_i \neq \sigma_{i+1}$. A V-path is a *closed path* if σ_1 is a face of τ_r different from σ_r. It has been proven that a discrete vector field V is the Forman gradient of a discrete Morse function if and only if V is free of closed paths [6].

Given a Forman gradient V on a simplicial complex Σ, the *discrete Morse complex* associated with Σ is a chain complex $\mathcal{M}_* := (\mathcal{M}_k, \tilde{\partial}_k)_{k \in \mathbb{Z}}$, where:

- groups $\mathcal{M}_k$ are generated by the critical k-simplices;
- the boundary maps $\tilde{\partial}_k$ are obtained by following the gradient paths of V (see Subsection 8.2 for a detailed description).

A discrete Morse complex $\mathcal{M}_*$, associated with a simplicial complex Σ, provides a homologically equivalent representation of Σ (see [6], Theorem. 8.2). If we consider a simplicial complex Σ, and we compute a Forman gradient V on it (see Figure 5(a)), we obtain a discrete Morse complex $\mathcal{M}_*$ having its cells in one-to-one correspondence with the critical simplices of V. $\mathcal{M}_*$ can be described as a graph having nodes in correspondence of the critical simplices of V, and having the arcs in one-to-one correspondence with the gradient paths connecting such nodes (see Figure 5(b)).

Since Σ and $\mathcal{M}_*$ are homologically equivalent, computing the homology on $\mathcal{M}_*$ is preferable due to the fact that the cells in $\mathcal{M}_*$ are generally fewer than the simplices in Σ. As shown in [32], the homological equivalence between a simplicial complex Σ and a discrete Morse complex $\mathcal{M}_*$ associated with Σ can be generalized to persistent homology by requiring that the Forman gradient V is filtered with respect to the filtration F considered. Formally, given a filtration $F = \{\Sigma^m \,|\, 0 \leq m \leq M\}$ of a simplicial complex Σ, a Forman gradient V of Σ is *filtered* with respect to F if, for each pair $(\sigma, \tau) \in V$, there exists $m \in \{1, \ldots, M\}$ such that $\sigma, \tau \in \Sigma^m$ and $\sigma, \tau \notin \Sigma^{m-1}$.

3. Related work

In this section, we review the state-of-the-art on data structures for encoding simplicial complexes and on algorithms for computing a discrete Morse complex.

3.1. Topological data structures for simplicial complexes

Several topological data structures for encoding a simplicial complex have been proposed in the literature, mainly for simplicial complexes in low dimensions, and focusing on triangle and tetrahedral meshes (see [33] for a survey). We consider here data structures specific for simplicial complexes in arbitrary dimensions.

The most general dimension-independent data structure for cell and simplicial complexes is the Incidence Graph. An *Incidence Graph (IG)* [34] is a topological incidence-based representation of a simplicial complex which encodes all the simplices as nodes of a graph and their immediate boundary and coboundary relations as its arcs. The *Simplified Incidence Graph (SIG)* [35] and the *Incidence Simplicial (IS)* data structures [36] are simplified representations of the IG. A comparison among IG, SIG and IS is presented in [37], while an implementation of all these data structures is included in the *Mangrove Topological Data Structure* library available in the public domain [38].

In the case of triangle and tetrahedral meshes, adjacency-based data structures are the most widely used thanks to their compactness and efficiency. The *Generalized Indexed data structure with Adjacencies (IA^*)* [17] generalizes such representations and is capable of encoding non-manifold simplicial complexes of any dimension. Recently, a topological data structure has been proposed for simplicial complexes embedded in the Euclidean space in [39], where topological relations can be efficiently extracted in parallel on different portions of the domain.

In recent years, new data structures have been developed well suited to perform specific tasks. The *Simplex Tree (ST)* [40] has been defined to efficiently extract boundary relations for computing persistent homology. The Simplex Tree encodes all simplices in the complex and tends to be more verbose than the IA^* data structure, as shown in [39]. An implementation of the ST is available in the *Gudhi* public domain library [41]. The *Maximal Simplex Tree (MST)* and the *Simplex Array List (SAL)* [42] are optimized versions of the ST. To the extent of our knowledge, there are no implementations of these data structures. The *skeleton blocker* data structure [43] has been created specifically to perform edge contraction on a simplicial complex, but it can be efficiently initialized only when working with flag complexes. An implementation of the latter is provided in the *Gudhi* library [41].

3.2. Computing a discrete Morse complex

The process of building a discrete Morse complex from a simplicial complex typically consists of two steps: (i) computing the Forman gradient and identifying the critical simplices, and (ii) extracting the boundary maps. We can classify algorithms for computing a Forman gradient as: *unconstrained* [44, 45, 46, 47, 11, 48] and *constrained algorithms* [49, 50, 51, 10, 52, 53, 54, 55]. *Unconstrained algorithms* compute a Forman gradient on a cell/simplicial complex when no scalar value is provided. The aim is to create a homologically equivalent representation of the input complex having as few critical cells as possible. *Constrained algorithms* start from a cell/simplicial complex endowed with a scalar function F_0 defined on its vertices, and aim at constructing a Forman gradient that best fits F_0 [50, 51, 10, 52]. The discrete Morse complex is used in the analysis and visualization of scalar fields as a compact representation of the field behavior. The aim is to obtain a decomposition of the dataset in regions of influence for each critical simplex. Ascending and descending traversal techniques [55, 56, 57] for the V-paths of the Forman gradient have been developed for reconstructing the ascending and descending Morse cells, respectively. We refer to [9] for an in-depth description of these methods. When computing persistent homology on a simplicial complex Σ [58, 59, 60], the aim is to obtain a complex which is a compact version of Σ and has the same persistent homology [10, 32, 61]. To this aim, the gradient V-paths need to be visited by starting from the critical simplices and by traversing the paths in a descending manner. A detailed description of this process is provided in Subsection 8.2.

4. Encoding a simplicial complex endowed with a Forman gradient

In this section, we consider the problem of encoding a simplicial complex endowed with a discrete vector field in a compact way. We start with

an analysis of existing data structures for simplicial complexes. Then, driven by the need to identify the most efficient data structure to adopt in our algorithm, we perform an experimental comparison among them. Finally, we show how we can encode a Forman gradient efficiently using a compact data structure which represents only vertices and top simplices. This is particularly challenging since a representation for a Forman gradient V on a complex Σ requires encoding the pairings between two simplices of consecutive dimension for all simplices in Σ.

4.1. Encoding a simplicial complex

We analyze here three data structures for encoding a simplicial complex, namely the Incidence Graph (IG) [34], the Simplex Tree (ST) [40, 62], and the Generalized Indexed data structure with Adjacencies (IA^* data structure) [17]. The IG is the most widely-used data structure for simplicial complexes, the ST has been used in topological data analysis applications, being implemented in the *Gudhi* library, the IA^* data structure is a compact representation for simplicial complexes encoding only vertices and top simplices. Implementations in the public domain exist for all of them, and on such implementations we base our experimental comparisons.

The *Incidence Graph (IG)* for complex Σ describes its Hasse diagram [63], i.e., the graphical representation of the partially ordered set generated by all the simplices of Σ and their incidence relations. The IG can be viewed as a directed graph $G_{IG} = (N_{IG}, B_{IG} \cup C_{IG})$ in which:

- the nodes in N_{IG} are in one-to-one correspondence with the simplices of Σ; with abuse of notation, we will indicate with σ both a node, and its corresponding simplex;
- a directed arc in B_{IG} (*boundary arc*) connects two nodes (τ, σ) in N_{IG} with $dim(\tau) = dim(\sigma) + 1$ if σ is a face of τ;
- a directed arc (*coboundary arc*) in C_{IG} connects two nodes (σ, τ) in N_{IG} with $dim(\tau) = dim(\sigma) + 1$ if τ is a coface of σ.

In Figure 6(b), the IG representing the simplicial complex Σ depicted in Figure 6(a) is shown. Nodes are colored according to the dimension of the simplex they represent. Note that, for simplicity, we have shown only one undirected arc for each pair of mutual incident nodes, since if a directed arc exists from τ to σ in B_{IG}, arc (σ, τ) must exist in C_{IG} By storing the incidence relations between simplices of consecutive dimension, the IG is efficient in the retrieval of topological relations, but the large amount of information encoded makes it unsuitable for complexes of large size and of high dimensions [37].

The *Simplex Tree (ST)* [40] encodes also all the simplices of a simplicial complex Σ as the IG, but only a subset of the incidence relations encoded in the IG. The ST is based on a total order selected on the vertices of Σ. Let $I(v)$ be the position in the total order of a vertex $v \in \Sigma$. Given a k-simplex $\sigma = \{v_0, ..., v_k\}$ in Σ, $max_v(\sigma) = max(I(v_i))$ is the latest vertex of σ in the total order. The ST can be viewed as a graph $G_{ST} = (N_{ST}, A_{ST})$ in which:

- the nodes in N_{ST} are in one-to-one correspondence with the simplices of Σ, and a node $\sigma \in N_{ST}$ is labeled with $I(max_v(\sigma))$;
- a directed arc $(\sigma, \tau) \in A_{ST}$ connects two nodes in N_{ST}, if σ is in the immediate boundary of τ, and $I(max_v(\tau)) > I(max_v(\sigma))$.

Nodes corresponding to the vertices of Σ are connected to the root of the Simplex Tree. If we select a path from the root to a node $\sigma = \{v_0, ..., v_k\}$, we have that: (i) labels $\{l_0, ..., l_k\}$ are encountered sorted by increasing order along the path and each label appears exactly once; (ii) each label corresponds to a vertex of σ, more precisely $l_i = I(v_i)$, for each $i = 0, ..., k$.

In Figure 6(c), we show the Simplex Tree representation of the simplicial complex depicted in Figure 6(a). The order of the vertices is indicated by the numbers depicted in blue, while the remaining numbers indicate the labels of the nodes corresponding to the k-simplices, with $k > 0$. For the sake of clarity, we are not showing the connections between the vertices and the root. Note that the Simplex Tree is order dependent, in the sense that we can have different STs for the same complex. For example, Figure 6(d) shows the ST obtained for Σ by using a different order for its vertices.

From the two graph representations, we see that $N_{IG} = N_{ST}$ and that $A_{ST} \subset A_{CB}$, since it contains

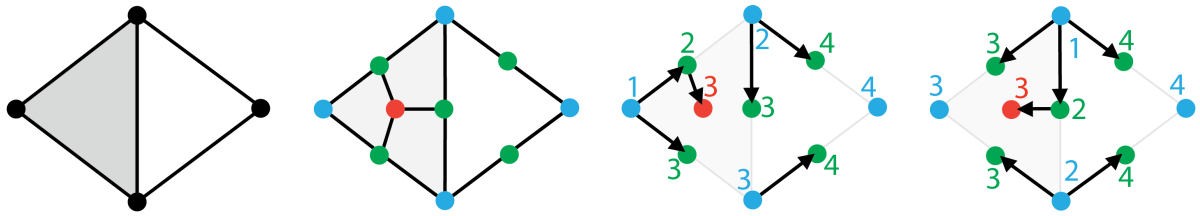


Figure 6: A simplicial complex Σ (a) and its representation through an Incidence Graph (b) and through a Simplex Tree (c); a Simplex Tree using a different ordering for the vertices (d). Blue dots are associated with the vertices of Σ, green dots with its edges and red dots with its triangles.

all those arcs $(\sigma, \tau) \in A_{CB}$ for which $I(max_v(\tau)) > I(max_v(\sigma))$. The Simplex Tree has been designed with the task of efficiently performing only boundary queries. In order to be able to perform also coboundary queries, an extended version of the Simplex Tree has been proposed in [40]. This extended version contains a circular list linking all the nodes having the same label and the same dimension and an arc from a node to its parent. This version is not implemented in the Simplex Tree in the public domain library *Gudhi* [41]. In [42], two compressed optimization of the latter have been presented, namely the *Maximal Simplex Tree* and the *Simplex Array List*, sharing the same functionalities but reducing the number of nodes encoded. To the best of our knowledge, no implementations are provided for these latter.

As mentioned before, more compact representations for a simplicial complex can be obtained by encoding only the vertices and top simplices. To be able to extract boundary, coboundary and adjacency relations efficiently, the simplest representation would encode: (i) for each top k-simplex σ, its boundary defined by the references to its $k+1$ vertices, and its adjacencies defined by references to the simplices adjacent to σ along a $(k-1)$-face; (ii) for each vertex v, its star, defined by the the list of all top simplices incident in v. It can be noticed that storing the entire star of a vertex v is not necessary, since the star can be efficiently reconstructed by navigating the top simplices incident in v through the encoded adjacencies. This constitutes the basis for the Generalized Indexed data structure with Adjacencies (IA^*) [17].

We can describe the IA^* data structure as a graph $G_{IA} = (N_{IA}, A_{IA})$ in which $N_{IA} = N_0 \cup N_{top}$, with set N_0 corresponding to the vertices of Σ, and set N_{top} corresponding to the top simplices of Σ. The set of arcs in A_{IA} is the disjoint union of three subsets $A_{(t,0)}$,$A_{(t,t)}$, $A_{(0,t)}$ defined as follows:

- $A_{(t,0)}$ (*boundary arcs*): a directed arc (σ, v), where σ is in N_{top} and v in N_0, belongs to $A_{(t,0)}$ if v is a vertex of σ;
- $A_{(t,t)}$ (*adjacency arcs*): an undirected arc (σ, τ), where σ and τ are k-simplices in N_{top}, belongs to $A_{(t,t)}$ if σ and τ share a $(k-1)$-face;
- $A_{(0,t)}$ (*coboundary arcs*): a subset of the arcs (v, σ), where v in N_0 and σ is in N_{top}, such that v is on the coboundary of σ, as defined below.

Given a vertex v, we consider the subgraph $G_{IA}(v) = (N_{IA}(v), A_{IA}(v))$ of G_{IA} where:

- $N_{IA}(v)$ consists of all nodes $\sigma \in N_{top}$ such that v is a vertex of σ;
- $A_{IA}(v)$ consists of all arcs in $A_{(t,t)}$ connecting pair of nodes in $N_{IA}(v)$.

Thus, an oriented arc (v, σ) is encoded in $A_{(0,t)}$ for each connected component in $G_{IA}(v)$, where σ is any top simplex in $N_{IA}(v)$ belonging to such component.

In Figure 7, we show the nodes and the arcs encoded in the IA^* data structure (see Figure 7(b)) for the simplicial complex in Figure 7(a). Blue nodes denote vertices, while green and red nodes denote top edge and triangles, respectively. Undirected arcs represent adjacency relations among top simplices, i.e., arcs (τ_1, τ_2) and (σ_1, σ_2). Boundary

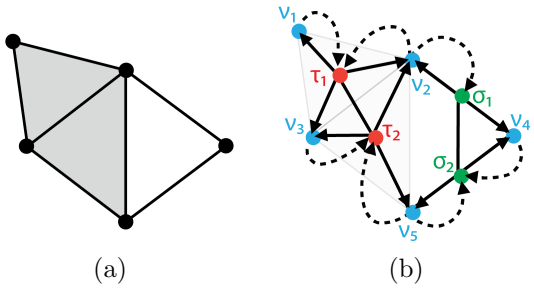


Figure 7: A simplicial complex Σ (a) and its representation through the IA^* data structure (b). Blue dots correspond to vertices, green dots correspond to top edges and red dots to top triangles.

arcs are denoted by arrows, while coboundary arcs by dotted arrows.

The space required by the IA^* data structure depends on the structure on the complex, i.e., the number of arcs in $A_{(t,t)}$ and in $A_{(0,t)}$ depends on the connectivity of the top simplices. If we restrict our consideration to an important subclass of simplicial complexes, that of simplicial pseudomanifolds, we can get some insights for comparing the space required by the IA^* data structure to that of the IG. Recall that a *simplicial d-pseudomanifold* is a $(d-1)$-connected simplicial d-complex such that any $(d-1)$-simplex is on the boundary of either one or two d-simplices.

If Σ is a d-pseudomanifold, we have that the number of arcs in $A_{(t,0)}$ originating from a top k-simplex σ is equal to $k+1$. The number of arcs in $A_{(t,t)}$ originating from a top k-simplex σ is also equal to $k+1$. Thus, $|A_{(t,t)}|$ is equal to $|A_{(t,0)}|$ and, thus, the total cost of storing the boundary and the adjacency arcs in the IA^* data structure is equal to $2|A_{(t,0)}|$. We can observe that $c = 2|A_{(t,0)}|$ is exactly the cost of storing in the IG all the boundary arcs connecting a d-simplex to a $(d-1)$-simplex plus all the dual coboundary arcs connecting a $(d-1)$-simplex to a d-simplex. In the IA^* data structure, besides c, we have the cost c_{st} of storing some top simplices in the star of the vertices. For each vertex v, c_{st} is equal to the number of connected components of $G_{IA}(v)$. In the worst case this might be equal to the number of d-simplices having v on their boundary. However, in the IG we need to take into account the cost of encoding the other boundary and coboundary arcs which connect k- and $(k-1)$-simplices (with $k < d$), which will be clearly much higher than c_{st}.

4.2. Experimental evaluation

This subsection provides an experimental comparison among the IG, the ST and the IA^* data structure. In our experiments, we have used three kinds of data sets. The first data sets are volume data that have been tetrahedralized. Each vertex of the dataset has an associated scalar value. The DTI-SCAN is a Diffusion Tensor MRI Scan of a human brain, the VISMALE dataset is a CT-scan of a man's head and the ACKLEY dataset is a synthetic function discretizing Ackley's function [64]. The datasets in the second group are networks obtained from real data on which cliques have been computed. Two of these datasets (AMAZON1, AMAZON2) are graphs representing the "Customers Who Bought This Item Also Bought" feature of the Amazon website. If a product i is frequently co-purchased with product j, the graph contains a directed edge from i to j (notice, we are considering the graph undirected). The third graph represents a road network in California where intersections and endpoints are described by nodes and the roads connecting these intersections or road endpoints are described by undirected edges (ROADNET). The datasets in the third group are point clouds extracted from a 2-sphere on which a Vietoris-Rips complex has been computed (datasets S1.0, S1.2, S1.3).

In our comparisons, we use the Simplex Tree (ST) implementation in the *Gudhi* library [41], the Incidence Graph (IG) implemented in *Perseus* [18], which is a public domain tool for computing the discrete Morse complex, and the IA^* data structure implemented in [65]. Table 1 summarizes the characteristics of the datasets we used and their storage costs using the three data structures. For each dataset, we provide the dimension of the resulting simplicial complex (column d), the number of its vertices (column $|\Sigma_0|$) and of its top simplices (column $|\Sigma_{top}|$), the size of the complex (column $|\Sigma|$), and the storage cost required by the three data structures, expressed in gigabytes.

We can observe that the storage cost of the IG and of the ST increases based on the total number of simplices. The IG implemented in the *Perseus* library often runs out of memory, while the ST has much higher limits. The storage cost of the IA^* data structure depends on the number

Dataset	d	$\lvert\Sigma_0\rvert$	$\lvert\Sigma_{top}\rvert$	$\lvert\Sigma\rvert$	Storage Cost IA^*	IG	ST
DTI-SCAN	3	0.9M	5.5M	24M	0.97	11.9	2.4
VISMALE	3	4.6M	26M	118M	4.7	-	9.7
ACKLEY4	4	1.5M	32M	204M	6.8	-	12.8
AMAZON01	6	0.2M	0.4M	2.2M	0.12	1.6	0.3
AMAZON02	7	0.4M	1.0M	18.4M	0.28	9.8	1.5
ROADNET	3	1.9M	2.5M	4.8M	0.8	3.3	1.0
SPHERE-1.0	16	100	224	0.6M	0.003	0.9	0.04
SPHERE-1.2	21	100	285	26M	0.0032	-	1.5
SPHERE-1.3	23	100	382	197M	0.0034	-	11.01

Table 1: Datasets used in the experiments and storage costs for encoding the corresponding simplicial complex with the three data structures IA^*, IG and with the ST. The storage costs are expressed in gigabytes.

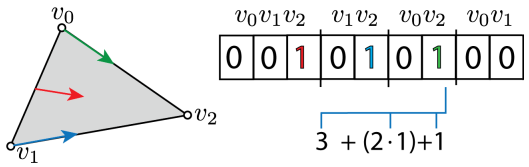


Figure 8: Gradient pairs encoded in a triangle. Pair between vertex v_0 and edge (v_0, v_2) is identified by moving on the first bit reserved for the 1-simplices (3 positions). We move forward of one position for each edge preceding (v_0, v_2) on the triangle (2 positions). We do not have to move forward since v_0 has position 0 on the edge.

of top simplices. This means that simplicial complexes in low dimensions (like ROADNET or the volumetric datasets) may require comparatively more memory than, for example, SPHERE-1.3 (being a 23-simplicial complex composed by less than 400 top simplices). It is clear that the IA^* data structure is always more compact than the ST. The ratio between the storage costs of the two data structures roughly depends on the ratio between the number of top simplices and the size of the complex. The worst-case scenario occurs for (ROADNET dataset) where the IA^* data structure requires 20% less memory than the ST, while in the case of SPHERE-1.3 the storage cost for the IA^* data structure is negligible with respect to the 11 gigabytes required by the ST.

4.3. Encoding a Forman gradient

In this subsection, we describe how to encode a discrete gradient field, like the Forman gradient V, on the data structures encoding a simplicial complex Σ.

If we consider the Incidence Graph G representing Σ, we see that the arcs of G describe all the possible pairings that can be defined on Σ by considering two simplices of consecutive dimension. A Forman gradient can be encoded on the IG by adding one bit flag to each arc a in C_{IG} indicating whether the nodes incident in a are also a valid pair in V. Because of this reason, the IG has been selected in the *Perseus* tool [18]. This encoding cannot be extended to the Simplex Tree since this latter encodes only a subset of the coboundary arcs of the IG.

We describe here a new representation which allows for a compact encoding of a Forman gradient on the IA^* data structure and, in general, for any data structure which encodes vertices plus top simplices. In this case, the encoding for the gradient pairs needs to be attached to the top simplices only. The representation that we have defined encodes, for each top k-simplex τ, a bit-vector of length $\sum_{i=1}^{k}\binom{k+1}{i+1}(i+1)$ representing all the possible pairings on its boundary. The first $k+1$ bits encode the pairing between τ and one of its $(k-1)$-faces. Then, recursively, for each i-face of τ, $i+1$ bits are stored until, for each 1-face, 2 bits are encoded storing the pairings with one of its vertices. For example, considering a 2-simplex (triangle), 3 bits are reserved for encoding the pairings with the boundary edges. Then, for each of them, 2 bits are reserved for encoding the pairings with the boundary vertices (see Figure 8).

If two paired simplices ρ and σ are both on the boundary of τ, the resulting pair will be encoded in the bit-vector of τ. Let j and l (with $j+1=l$) be the dimensions of ρ and σ, respectively, we check the bit associated with the corresponding pair computing:

- the position $\sum_{i=l+1}^{k}\binom{k+1}{i+1}(i+1)$ of the first bit reserved for l-simplices in τ;
- the position of σ on the boundary of τ obtained enumerating the faces of τ;
- the position of the vertex in σ that is not in ρ.

For example, in Figure 8, we consider the pairing between the 0-simplex v_0 and the 1-simplex v_0v_2. The bits reserved for the 1-simplices start at position 3. The position of v_0v_2 on the boundary of the triangle is 1, so we discard the first two bits. Vertex

v_2 is missing in v_0 and its position is 1. Then, the bit representing their pairing relation is at position $3 + (2 \cdot 1) + 1$.

We have implemented a prototype of the gradient encoding based on the *dynamic_bitset* provided by the Boost C++ library. With such encoding, we have been able to represent the gradient frame representation up to 40-dimensional simplicial complexes. Using more involved libraries and architectures could overcome the current limitations, but it might greatly affect computation times.

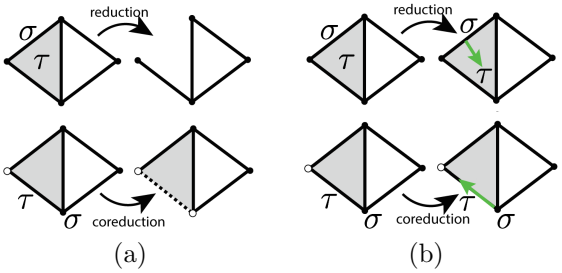


Figure 9: (a) Removal of the reduction/coreduction pair (σ, τ), and (b) corresponding pairing of simplices σ and τ in the gradient.

5. Reductions and coreductions for discrete Morse complexes

Reduction and *coreduction* operators [45] are two homology-preserving operators used for reducing the size of a simplicial complex without affecting its homology. For this reason, reduction and coreduction operators can be used in a preprocessing approach to compute homology, or persistent homology of a simplicial complex [32, 45, 46, 47]. Reduction and coreduction pairs can be fruitfully used also in the context of discrete Morse theory in order to define a Forman gradient. In this section, we present the two methods based on such operators, and we propose a new strategy, while providing also a theoretical comparison of all these techniques.

A *reduction* on a simplicial complex Σ corresponds to a deformation retraction of a simplex which is the face of only one other simplex in the complex. The problem is that, in most situations, available reductions are quickly exhausted. In order to overcome this issue, coreductions have been introduced [45], where a coreduction can be viewed as the dual operation with respect to a reduction. A coreduction is not feasible on a simplicial complex, while it is available in the context of *S-complexes* [45]. For the sake of simplicity, we consider an S-complex as a simplicial complex in which some simplices may be not present even if their cofaces are in the complex. For instance, all the complexes depicted in Figure 9 are S-complexes. In particular, the complexes obtained after performing a coreduction operator are examples of S-complexes which are not simplicial complexes.

Given an S-complex Σ, a pair (σ, τ) of elements of Σ, such that the coefficient of σ in $\partial\tau$ is ± 1, is called a *reduction pair* if $cbd_\Sigma(\sigma) = \{\tau\}$, a *coreduction pair* if $bd_\Sigma(\tau) = \{\sigma\}$.

When simplifying a simplicial complex Σ, the effect of a reduction/coreduction is that of changing the structure of Σ, by removing a pair of simplices without affecting its homology (see Figure 9(a)). When building a Forman gradient V, the same pair is not removed from Σ, but added as a pair to V (see Figure 9(b)).

A *coreduction-based* algorithm builds a Forman gradient using coreduction pairs and free simplices [11], where a *free simplex* is a simplex with an empty boundary. The algorithm works on two sets of simplices: the set of paired simplices V, initialized as empty, and the set of non-excised simplices Σ', initialized as Σ. While Σ' admits a coreduction pair, the algorithm excises a coreduction pair (σ, τ) from Σ' and adds it to V. When no more coreduction is feasible, a free simplex is excised from the complex and labeled as critical. The algorithm repeats these steps until Σ' is empty. Since no simplicial complex admits a coreduction pair, any coreduction-based algorithm performs as its first step the excision of an arbitrary vertex v, which is a free simplex by definition, and declares it as critical. The removal of v turns Σ' into an S-complex and unlocks the possibility of pairing through a coreduction any vertex u adjacent to v.

A *reduction-based* approach performs reductions and removals of top simplices [48]. We recall that a *top simplex* is a simplex with an empty coboundary. The algorithm works on two sets of simplices: the set of paired simplices V, initialized as empty, and the set of non-excised simplices Σ', initialized as Σ. While the set of non-excised simplices Σ'

admits a reduction pair, the algorithm excises a reduction pair from Σ' and adds it to V. When no more reduction is feasible, a top simplex is excised from the complex and labeled as critical. The algorithm stops when Σ' is empty. Differently from a coreduction-based algorithm, whose first step is necessarily the removal of a vertex, the initial step in a reduction-based approach can involve the excision of a feasible reduction pair or the removal of a top simplex. Similarly to the previous case, if no reduction pair is available, the approach has to label an arbitrary top simplex as critical and to remove it from Σ'. After such a removal, the situation is analogous to the starting one and, so, the same strategy can be applied.

In order to minimize the size of the discrete Morse complex, in both approaches the creation of a critical simplex is performed only if no more coreduction, or reduction is feasible. Actually, even if this condition is not satisfied, the acyclicity of the gradient paths is still guaranteed. In the following, we refer to this two approaches, also in the case in which critical simplices can be created when it is not strictly necessary, as *coreduction-based algorithm* and *reduction-based algorithm*, respectively.

6. Equivalence of reduction and coreduction sequences

In this section, we prove the equivalence between the use of reduction and coreduction operators in the construction of a (filtered) Forman gradient and we introduce another class of methods which could operate reductions and coreductions in an interleaved way. The equivalence among these three methods will give us the freedom to choose the one that best fits our data structure.

In order to better understand how the removal of a coreduction, or of a reduction pair affects the coboundary and the boundary of the simplices of a simplicial complex, we first discuss some preliminary results.

Remark 1. *Let τ be a simplex and let σ be one of its faces, then there exists $dim(\tau) - dim(\sigma)$ faces of τ in $cbd_\tau(\sigma)$.*

Lemma 1. *In a coreduction-based algorithm, each removal operation does not modify the coboundary of the remaining simplices.*

Proof. Let Σ be a simplicial complex on which the coreduction-based algorithm is executed. Clearly, the removal of a free simplex does not modify the coboundary of any remaining simplex. Let us consider only removals of coreduction pairs. Let (σ, τ) be a feasible coreduction pair in the set of non-removed simplices Σ'. The only simplices whose coboundary can be modified by the coreduction pair are those belonging to $bd_{\Sigma'}(\tau)$ and to $bd_{\Sigma'}(\sigma)$. Since, for the feasible coreduction pair (σ, τ), $bd_{\Sigma'}(\tau) = \{\sigma\}$, the thesis is obtained by proving that, before performing the coreduction, $bd_{\Sigma'}(\sigma) = \emptyset$. Suppose that there exists $\nu \in bd_{\Sigma'}(\sigma)$. By Remark 1, there exists in Σ a simplex $\sigma' \neq \sigma$ such that $\sigma' \in bd_\Sigma(\tau)$ and $\nu \in bd_\Sigma(\sigma')$. Since (σ, τ) is a feasible coreduction pair in Σ', simplex σ' must have been already removed, i.e., $\sigma' \notin \Sigma'$. Let us proceed by induction. If (σ, τ) is the first coreduction pair performed in the coreduction-based algorithm on complex Σ, then σ' has been removed as a free simplex, but, since $\nu \in bd_\Sigma(\sigma')$ and $\nu \in \Sigma'$, this leads to a contradiction.

Assume now that, for any removal of a coreduction pair performed before (σ, τ), the simplex of smaller dimension of the pair is free. Since $\nu \in bd_\Sigma(\sigma')$ and $\nu \in \Sigma'$, σ' cannot be removed as a free simplex, or by a coreduction pair removal of the kind (ν', σ'). So, σ' has been removed by operating a coreduction pair removal of the kind (σ', τ'), which leads to a contradiction of the inductive hypothesis. □

Lemma 2. *In a reduction-based algorithm, each removal operation does not modify the boundary of the remaining simplices.*

Proof. Let Σ be a simplicial complex on which the reduction-based algorithm is executed. Clearly, the removal of a top simplex does not modify the boundary of any remaining simplex. Let us consider only removals of reduction pairs. Let (σ, τ) be a feasible reduction pair in the set of non-removed simplices Σ'. Similarly to Lemma 1, proving that, before performing the coreduction, $cbd_{\Sigma'}(\tau) = \emptyset$ is sufficient. If there exists $\nu \in cbd_{\Sigma'}(\tau)$, then, by Remark 1, there exist $dim(\nu) - dim(\sigma) \geq 2$ faces of ν in $cbd_{\Sigma'}(\sigma)$. But this leads to a contradiction, because (σ, τ) is a reduction and, thus, $\#cbd_{\Sigma'}(\sigma) = 1$. □

We are now ready to formalize and to prove

the equivalence between the coreduction-based and reduction-based algorithms.

Proposition 1. *Given a simplicial complex Σ and the Forman gradient V produced by a reduction-based algorithm, it is always possible to obtain the same Forman gradient through a coreduction-based algorithm. The reverse is also true.*

Proof. For the sake of brevity, we only prove that the Forman gradient produced by a reduction-based algorithm on Σ can be obtained with a coreduction-based algorithm. The proof of the reverse is entirely similar (by using Lemma 1). Let Σ be a simplicial complex and let

$$R_1^1, R_2^1, \ldots, R_{i_1}^1, R_1^2, R_2^2, \ldots, R_{i_2}^2, \ldots, R_1^n, R_2^n, \ldots, R_{i_n}^n \quad (1)$$

be the ordered sequence of reduction pairs and top simplices removed during the execution of a reduction-based algorithm, where, for $1 \leq l \leq n$ and $1 \leq j \leq i_l - 1$, R_j^l represents a reduction pair and, for each $1 \leq l \leq n$, $R_{i_l}^l$ represents a top simplex.

According to the notation adopted in (1), Figure 10(a) depicts the ordered sequence of reduction pairs and top simplices removed during the execution of a reduction-based algorithm. We want to prove that, by using the same removals, it is possible to obtain a sequence of coreduction pairs and free simplices compatible with a coreduction-based algorithm producing the same Forman gradient. Figure 10(b), for example, shows a sequence of coreduction pairs and free simplices compatible with a coreduction-based algorithm obtained by reversing the reduction-based sequence depicted in Figure 10(a) and producing the same Forman gradient.

We consider the following sequence obtained taking sequence (1) in reverse order:

$$R_{i_n}^n, R_{i_n-1}^n, \ldots, R_1^n, R_{i_{n-1}}^{n-1}, \ldots, R_{i_1}^1, \ldots, R_2^1, R_1^1 \quad (2)$$

Consider (2) as an ordered list of removal operations performed on Σ. The following properties hold:

1. For each $1 \leq l \leq n$ and $1 \leq j \leq i_l - 1$, R_j^l is a feasible coreduction pair.
2. For each $1 \leq l \leq n$, $R_{i_l}^l$ is a free simplex.

To prove the two properties, we denote with:

- Σ_j^l the simplicial complex obtained in (1) after performing all the removal operations up to R_j^l included;
- S_j^l the S-complex obtained in (2) after performing all the removal operations up to R_j^l excluded.

We have that, for each value of l and j,

$$\Sigma_j^l \sqcup S_j^l = \Sigma \quad (3)$$

1. Let $R_j^l = (\sigma, \tau)$ with $1 \leq l \leq n$ and $1 \leq j \leq i_l - 1$. We have to prove that it represents a coreduction in the sequence (2), i.e., $bd_{S_j^l}(\tau) = \{\sigma\}$. By Lemma 2, in (1), τ cannot be removed before the simplices in $bd_\Sigma(\tau)$. So, all the simplices in $bd_\Sigma(\tau) \setminus \{\sigma\}$ belong to Σ_j^l. Then, by (3), $bd_{S_j^l}(\tau) = \{\sigma\}$ and, thus, (σ, τ) is a feasible coreduction in S_j^l.
2. Let $R_{i_l}^l$ be the simplex σ. We have to prove that it represents a free simplex in the sequence (2), i.e., $bd_{S_{i_l}^l}(\sigma) = \emptyset$. Analogously to 1., by Lemma 2, in (1), all the simplices belonging to $bd_\Sigma(\sigma)$ are in $\Sigma_{i_l}^l$. Then, by (3), $bd_{S_{i_l}^l}(\sigma) = \emptyset$ and, thus, σ is a free simplex in $S_{i_l}^l$.

Sequence (2) satisfies properties 1. and 2. So, it represents a sequence of removals compatible with a coreduction-based algorithm producing on Σ the same Forman gradient of (1). □

It is interesting to understand if the equivalence between reduction-based and coreduction-based algorithms still holds with the further condition that allows for the introduction of a critical simplex only if no reduction [coreduction] pair is available. Proposition 1 ensures that, given a reduction [coreduction] sequence produced on a simplicial complex Σ by an algorithm requiring such a condition, it is always possible to find a coreduction [reduction] sequence inducing the Forman gradient on Σ. In spite of this, Proposition 1 does not guarantee that a sequence produced by an algorithm satisfying the condition mentioned above exists. Figure 11 shows that, in general, this does not hold. The Forman gradient depicted in Figure 11 can be considered as produced by a reduction-based algorithm starting

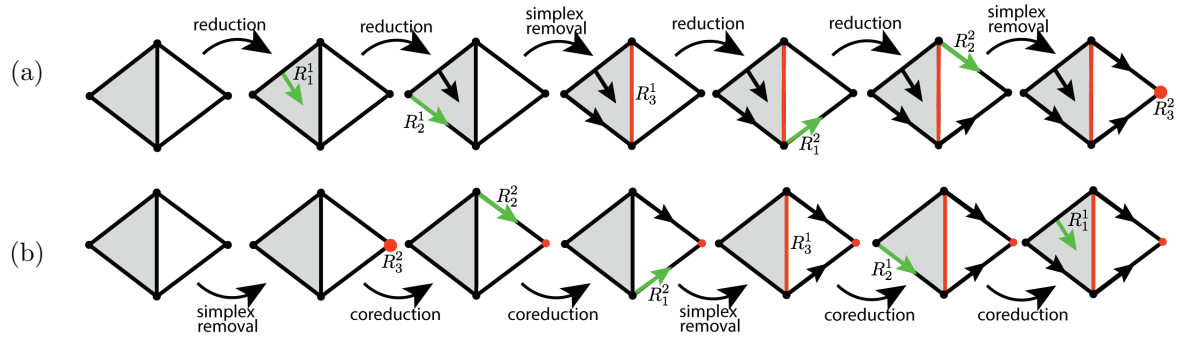


Figure 10: (a) A sequence of reduction pairs (green arrows) and top simplex removals (red simplices) produced by a reduction-based algorithm on a simplicial complex and (b) the sequence of coreduction pairs resulting in the same gradient than (a).

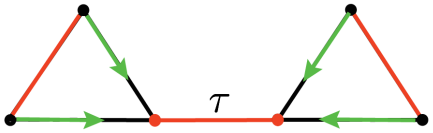


Figure 11: A Forman gradient on a simplicial complex that cannot be produced by a coreduction-based algorithm in which critical simplices are introduced only when no more coreduction pair is feasible.

with the removal of the top simplex τ and introducing critical simplices only when it is strictly necessary. This Forman gradient cannot be produced by a coreduction-based algorithm in which critical simplices are introduced only when no more coreduction pair is feasible because such an algorithm applied to this simplicial complex necessarily produces a Forman gradient with just one critical simplex of dimension 0 and two critical simplices of dimension 1.

7. Interleaving reductions and coreductions

A new method to build a gradient field V on a simplicial complex is to execute removals of reduction and coreduction pairs in an interleaved way. We denote as *interleaved-based algorithm* an algorithm producing a discrete vector field by using removals of reduction and coreduction pairs, of top simplices and of free simplices. Given a simplicial complex Σ, pairs of simplices are excised from Σ by arbitrarily choosing between reduction or coreduction pairs. When no more pairs can be removed, a free simplex or a top simplex is excised from the complex and labeled as critical. The algorithm repeats these steps until Σ is empty.

Here, we prove that such an algorithm actually produces a Forman gradient and that all interleaved methods are equivalent.

Proposition 2. *Given a simplicial complex Σ, the discrete vector field V produced by any interleaved-based algorithm is a Forman gradient.*

Proof. Given two pairs (σ, τ), (σ', τ') in V, we define $(\sigma, \tau) \leq (\sigma', \tau')$ if there exists a V-path starting with (σ, τ) and ending with (σ', τ'). In order to prove the thesis, i.e., that V is free of closed V-path, it is enough to prove that $\leq$ define a partial order on V. Consider set V as built in any intermediate step of the proposed algorithm and let (σ, τ) be the last pair inserted in V. The following properties allow to achieve the thesis:

1. (σ, τ) is a minimal element with respect to the elements already inserted in V originating from a coreduction pair;

2. (σ, τ) is a maximal element with respect to the elements already inserted in V originating from a reduction pair.

Suppose that condition 1 does not hold. Then, there must exist an already performed coreduction pair (σ', τ') such that $\sigma \in bd(\tau')$. This implies that, at the step in which (σ', τ') has been performed, $\sigma, \sigma' \in bd(\tau')$. But this is impossible, otherwise the coreduction pair (σ', τ') could not have been performed.

Suppose that condition 2 does not hold. Then, there must exist an already performed reduction pair (σ', τ') such that $\sigma' \in bd(\tau)$ and this implies that, at the step in which (σ', τ') has been performed, $\tau, \tau' \in cbd(\sigma')$. But this is impossible, otherwise the reduction pair (σ', τ') could not have been performed. □

Having proven that any possible interleaved method leads to a Forman gradient, we are now interested in understanding if these different approaches could produce equivalent results or not. As an immediate consequence of Lemma 1 and Lemma 2, we can claim the following result.

Remark 2. *In each interleaved-based algorithm, each coreduction pair and free simplex removal cannot make a reduction pair feasible; each reduction pair and top simplex removal cannot make a coreduction pair feasible.*

Finally, we can prove that all interleaved methods are equivalent.

Proposition 3. *Given a simplicial complex Σ and the Forman gradient V on it produced by an interleaved-based algorithm, it is always possible to obtain the same Forman gradient with a reduction-based algorithm or, equivalently, with a coreduction-based algorithm.*

Proof. We prove that the sequence of removals produced by an interleaved-based algorithm on a simplicial complex can be also obtained with a sequence of coreduction pairs and free simplex removals. By Remark 2, we can suitably order such a sequence, moving all the coreduction pairs and the free simplices at the beginning, thus creating a new sequence equivalent to the previous one. We apply to the last part, composed only of reduction pairs and top simplices, of this new sequence the same sorting strategy proposed in Proposition 1 to transform a reduction-based sequence to a coreduction-based sequence, and in this way, we obtain the thesis. □

From both an application and a theoretical point of view, it is interesting to find a method to build a Forman gradient which minimizes the number of critical simplices. It is known that, in general, this problem is NP-hard [66]. The previous results show that, from a theoretical point of view, the use of different simplification operators (such as reduction and coreduction pairs), or the combination of more than one, does not actually affect the number of resulting critical simplices.

For the sake of completeness, let us note that the results proven in this section still hold when the above-described approaches are applied to build a filtered Forman gradient. This is due to the fact that the satisfaction of the condition required to guarantee that V is a filtered Forman gradient with respect to a filtration F does not take into account if the pairs of V have been created thanks to a reduction or a coreduction operator.

8. A coreduction-based algorithm for computing a discrete Morse complex

In this section, we describe an algorithm based on the IA^* data structure for computing a discrete Morse complex. The algorithm consists of two steps: (i) computation of a (filtered) Forman gradient through a coreduction-based approach, and (ii) extraction of the boundary maps defining the discrete Morse complex.

8.1. Construction of a (filtered) Forman gradient

The theoretical equivalences proven in Section 6 and Section 7 tell us that there is no preferable homology-preserving operator for computing a Forman gradient. Here, we introduce a new dimension-independent algorithm, that can also runs in parallel, which uses a representation of the simplicial complex as an IA^* data structure and the encoding of the Forman gradient discussed in Subsection 4.

The basic underlying approach is the coreduction-based algorithm, introduced in [11] and implemented there only for regular grids. We summarize it for simplicial complexes. When considering simplicial complexes, the coreduction-based algorithm computes a Forman gradient by using coreduction pairs starting from the simplices of lowest dimension. The set of k-simplices of complex Σ is considered by increasing values of k, starting from $k = 0$. As long as a coreduction pair exists between a k-simplex σ and a $(k+1)$-simplex τ, pair (σ, τ) is added to the Forman gradient V. When no k-simplex can be paired, one simplex is randomly chosen and declared critical. When all k-simplices have been paired or denoted as critical,

the working dimension k is increased by one. Since no coreduction pair is feasible on a simplicial complex, at the first step, an arbitrary vertex v is denoted as critical in V to trigger coreductions. In [32], the coreduction-based approach is used for persistent homology computation, and thus by considering a filtration of the original complex. If each simplex is paired only with another simplex belonging to the same filtration value, the resulting discrete Morse complex will have the same persistent homology of the original complex.

The dimension-independent coreduction-based algorithm proposed here, unlike previous ones, uses a local approach that allows us to work on the stars of the vertices independently, which makes it particularly suitable for a parallel implementation. We define an indexing on the vertices of the input simplicial complex Σ, and we extend the indexing to all the simplices in such a way that each simplex in Σ has an index equal to the maximum of the indexes of its vertices. With such indexing, the coreduction pairs can be computed locally to the lower star of each vertex. Given a vertex v, a simplex σ belongs to the *lower star* of v (denoted as $St^{-}(v)$) if: (i) σ is a coface of v, and (ii) v has lowest index value among the vertices of σ. *Algorithm* 1 illustrates the process for computing a Forman gradient on a simplicial complex Σ having an indexing F_0 defined on its vertices.

The algorithm iterates on the vertices of Σ, extracting first the top simplices in the lower star of v, denoted as LT_v, which are encoded in a list. For each vertex v, the algorithm iterates on the dimension of the simplices in the lower star of v. The algorithm works, for each dimension, with two sets of simplices: the set of k-simplices that can be declared critical, denoted as CR_v (row 12), and the set of $(k+1)$-simplices to pair (row 14), denoted as ST_v. CR_v and ST_v have a maximum size equal to the maximum, by varying k, of the number of k-simplices in the lower star of a vertex in Σ, and they are encoded as balanced binary search trees. A candidate simplex is extracted from set ST_v (row 16) and paired with its unique unpaired face (row 18). Recall that a simplex τ can be paired with another simplex σ by coreduction if σ is the only unpaired face of τ. If there are no coreductions available (row 21), a new critical simplex is taken from CR_v. Every time a simplex is paired or set as critical, it is also removed from ST_v or CR_v, respectively. When set CR_v is empty, the working dimension is increased. The algorithm terminates when all the simplices in the lower star of each vertex v have been paired, or set as critical.

Algorithm 1 - FormanGradient(Σ,F_0)

```
1:  INPUT: Σ, d-dimensional simplicial complex
2:  INPUT: F_0, indexing of vertices of Σ
3:  OUTPUT: V, Forman gradient; C, set of critical simplices
4:  Σ_0 := vertices of Σ
5:  V := ∅
6:  C := ∅
7:  for v ∈ Σ_0 do
8:    k := 0
9:    ST_v := {v}
10:   LT_v := LowerTop(v, Σ, F_0)
11:   while k <= d do
12:     CR_v := ST_v
13:     k := k + 1
14:     ST_v := LowerStar(v, Σ, F_0, LT_v, k)
15:     while CR_v ≠ ∅ do
16:       (σ, τ) := getNextPair(v, Σ, ST_v, CR_v)
17:       if (σ, τ) ≠ ∅ then
18:         addPair(σ, τ, V)
19:         Remove(τ, ST_v)
20:         Remove(σ, CR_v)
21:       else
22:         σ = getFirstCritical(CR_v)
23:         addCritical(σ, C)
24:         remove(σ, CR_v)
25:       end if
26:     end while
27:   end while
28: end for
```

The procedures and the functions, on which Algorithm 1 is based, are:

- Function $LowerTop(v, \Sigma, F_0)$: computes all the top simplices of Σ belonging to the lower star of v and encodes such simplices in list LT_v. This is performed by navigating the star of vertex v through the adjacency arcs in the IA^* data structure. Thus, it works in time $O(t_v)$, where t_v denotes the number of top simplices in the star of v.

- Function $LowerStar(v, \Sigma, F_0, LT_v, k)$: extracts all the k-simplices belonging to the lower

star of v from LT_v and encodes such simplices in ST_v. This operation is performed by cycling on the elements of LT_v and collecting the k-faces of each top simplex that are also incident in v. The extraction of the k-simplices of a top simplex of dimension i is performed in $O(\binom{i+1}{k+1})$. If we denote as $t_{v,i}$ the number of top simplices of dimension i incident in v, the total number N_k of simplices extracted is $N_k = \sum_{i=1}^{d} t_{v,i}\binom{i+1}{k+1})$ in the worst case, since some simplices are contained within the boundary of more than one top simplex. Since each of such simplices is inserted in ST_v, $LowerStar(v, \Sigma, F_0, LT_v, k)$ may require $O(N_k \log N_k)$ time in the worst case.

- Procedure $addPair(\sigma, \tau, V)$: adds a new pair to V. Since the gradient pairs are encoded on the top simplices only, we have to find the top simplices incident in both σ and τ. This is done by examining all the top simplices in the star of a vertex of σ and detecting all those having τ on their boundary. For each of these latter, we update the corresponding bit-vector. The operation requires $O(t_w)$, where t_w denotes the number of top simplices incident in a vertex w of σ.

- Function $getNextPair(v, \Sigma, ST_v, CR_v)$: iterates on the set of unpaired simplices ST_v selecting the first simplex available for a coreduction. For each simplex τ in ST_v, the simplices on the boundary of τ containing v are extracted, and then, for each of such boundary simplices σ, the membership of σ to CR_v is checked. In the worst case, we will need to check the membership of all the elements in CR_v. This leads to a worst-case complexity of $O(k|ST_v||CR_v| \log |CR_v|)$.

- Function $getFirstCritical(CR_v)$: returns the first simplex in the set of candidate critical simplices CR_v. Since CR_v is implemented as a balanced binary search tree, the worst-case time complexity is $O(\log |CR_v|)$.

- Procedure $Remove(\sigma, CR_v)$: eliminates a simplex from CR_v (or ST_v). Since both CR_v and ST_v are implemented as a balanced binary search tree, the worst-case time complexity is $O(\log |CR_v|)$.

For each dimension, the computation cost is dominated by the cost of executing Function $getNextPair(ST_v, CR_v, \Sigma)$. If we denote as cr_m and as st_m the maximum size of CR_v and ST_v, respectively over all dimensions, the time complexity for a single vertex v is $O((d-1)st_m cr_m \log(cr_m))$. Note that both cr_m and st_m can be of the order of the number of k-simplices incident in v. Since the algorithm computes the Forman gradient locally to the lower star of each vertex, the approach is easy to parallelize by running Algorithm 1 on multiple vertices at a time. Results are shown in Section 9.

We prove the correctness of Algorithm 1 by showing that it is a coreduction-based algorithm ensuring that the generated discrete vector field V is a filtered Forman gradient.

Proposition 4. *Let Σ be a simplicial complex, $F_0 : \Sigma_0 \to \mathbb{R}$ be an injective function and F be the filtration of Σ naturally induced by F_0. Given Σ and F_0 as input, Algorithm 1 returns a filtered Forman gradient with respect to F.*

Proof. Algorithm 1 processes the lower stars of the vertices of Σ independently. Without loss of generality, we can assume that the lower stars are processed in a sequence ordered by ascending values of function F_0. In this way, we obtain an ordered sequence of simplices added to the gradient V and to the set of critical simplices C. We prove that this sequence, denoted as S, actually represents a feasible sequence of coreduction pairs and free simplices for Σ. Let us consider a pair of simplices (σ, τ) declared as a pair of V during the processing of the lower star $St^-(v)$ of v. Let σ' be a simplex in $\mathrm{bd}_\Sigma \tau$ different from σ. If $\sigma' \in St^-(v)$, then σ' has to be already added to V or to C. Otherwise, if $\sigma' \notin St^-(v)$, then there exists a vertex w of Σ such that $\sigma' \in St^-(w)$ and $F_0(w) < F_0(v)$. So, σ' has to be already added to V or to C during the processing of $St^-(w)$. In both cases, (σ, τ) can be considered as a feasible coreduction pair in the sequence S. Similarly, any simplex σ added to C during the processing of a lower star can be considered as a free simplex in the sequence S. So, Algorithm 1 is a coreduction-based algorithm and then, thanks to Proposition 2, it returns a Forman gradient. Moreover, since Algorithm 1 pairs only simplices belonging to the same lower star and, by the definition of F, these simplices have the same

filtration value. Thus, the returned Forman gradient V is necessarily filtered with respect to F. □

8.2. Extracting the discrete Morse complex

The discrete Morse complex $\mathcal{M}_*$ associated with a (filtered) Forman gradient V on Σ is retrieved by navigating the paths of V. The output consists of the boundary maps $\tilde{\partial}_k : \mathcal{M}_k \to \mathcal{M}_{k-1}$. These latter can be seen as the arcs of a graph in which the nodes correspond to the critical simplices and each arc has a multiplicity which corresponds to a gradient path between two critical simplices.

Extracting the boundary maps by visiting the paths of V may cause simplices to be visited more than once, as discussed in [10]. In the worst case, a critical k-simplex may be connected by V-paths to all the k-simplices of Σ (this set is denoted as Σ_k). Moreover, each k-simplex of this set can be visited, via multiple V-paths, more than once; in the worst case each simplex will be visited $O(|\Sigma_k|)$ times. The resulting worst-case complexity for retrieving the boundary maps of a single critical k-simplex can be quadratic in the number $|\Sigma_k|$ of k-simplices of Σ.

Even if this is a very rare case, some solutions have been proposed to guarantee lower complexity bounds by either using a Boolean function for marking the visited simplices [67, 57], or by using a priority queue [55] for limiting the number of simplices visited more than once. Both approaches have limitations however. The approach in [67] is useful for reconstructing a combinatorial representation for the connectivity of the critical simplices, but it does not visit all the possibile paths, which is necessary for retrieving the correct boundary maps in $\mathbb{Z}$. The approach in [55] can successfully retrieve the correct boundary maps, but it requires a input scalar function to be defined all over the simplices of Σ.

The algorithm presented here is based on the general approach outlined in [10].

Algorithm 2 illustrates the steps required for traversing the gradient paths in a descending fashion. Starting from a critical k-simplex τ, a breadth-first traversal is performed by navigating from τ to its adjacent k-simplices passing through their shared $(k-1)$-simplices. The breadth-first traversal is supported by a queue Q. Given a k-simplex τ_0 extracted from the queue Q (row 8), we examine all the $(k-1)$-simplices σ in the boundary of τ_0

Algorithm 2 - BoundaryMaps(Σ,τ,V)

```
1:  INPUT: Σ, d-dimensional simplicial complex
2:  INPUT: τ, critical k-simplex
3:  INPUT: V, Forman gradient
4:  OUTPUT: M, boundary maps as collections of
    arcs
5:  Q := ∅
6:  Q.enqueue(τ)
7:  while Q ≠ ∅ do
8:    τ0 := Q.dequeue()
9:    for σ1 ∈ getBoundary(τ0, Σ) do
10:     if isPaired(σ1, V, τ1) then
11:       Q.enqueue(τ1)
12:     else
13:       Add(M, τ1, σ1)
14:     end if
15:   end for
16: end while
```

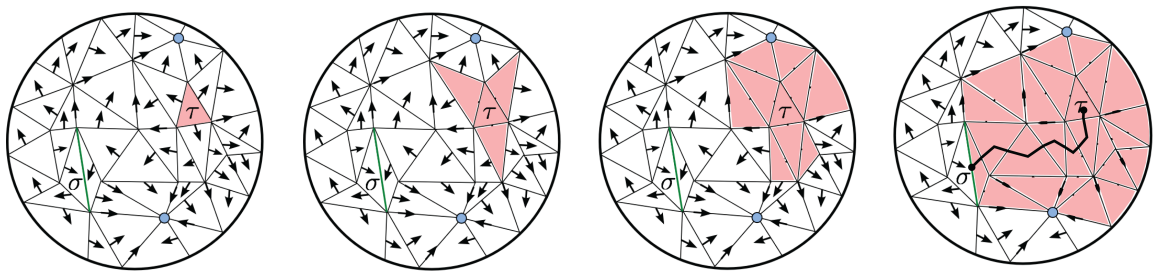
(row 11). For each $(k-1)$-simplex σ, if σ is paired with a k-simplex τ_1 (row 12), τ_1 is added to the queue (rows 15 and 16). If σ is a critical simplex, then σ is stored as on the boundary of τ.

In Figure 12(a), we show an example of the descending traversal performed by starting from critical triangle τ. For each edge on the boundary of τ, the paired triangle is visited and enqueued (indicated in red in Figure 12(b)). The process continues recursively for each new triangle (Figure 12(c)) until the entire region associated with τ has been covered. When a critical edge σ is encountered, the relation with τ is stored in the boundary maps (Figure 12(d)).

The procedures and the functions, on which Algorithm 2 is based, are:

- Function $getBoundary(\tau, \Sigma)$: returns the immediate boundary of k-simplex τ, i.e., its $(k-1)$-faces. Extracting the immediate boundary is performed by taking all the combinations of the k vertices of τ, and it is a linear process in the number of vertices of τ.

- Function $isPaired(\sigma, V, \tau_1)$: returns the value *True* if $(k-1)$-simplex σ is paired with a k-simplex in V and the value *False* otherwise. In the former case it returns the paired simplex τ_1. This is done by considering all the top simplices in the star of a vertex w of σ and visiting



(a) (b) (c) (d)

Figure 12: Descending traversal starting from τ. Expanding the gradient V-paths the critical edge σ is encountered and stored as connected to τ.

the gradient encoding of those top simplices which are incident in σ. The time complexity is $O(t_w)$ in the worst case, where t_w is the number of top simplices incident in vertex w.

Algorithm 2 is executed for each critical simplex in the Forman gradient. For each k-simplex τ popped from Q, the for loop is performed up to k times. For each simplex σ on the boundary of τ we check whether it is paired or not $O(t_w)$. Then, we can conclude that each iteration of the while loop takes $O(kt_m)$, where t_m is the maximum of the number of top simplices t_k considered in $isPaired$ at the varies of σ. The algorithm has $O(qkt_m)$ worst-case time complexity, where q is the number (counted with multiplicity) of k-simplices of Σ inserted in the queue Q.

9. Experimental results

In this section, we evaluate the performances of the coreduction-based algorithm for Forman gradient computation and of the algorithm for computing the boundary maps that give a Morse complex, described in Section 2.3, which are based on the encoding of the original simplicial complex as an IA^* data structure. As described in Section 8, computing the Forman gradient focusing on the lower star of each vertex is an operation well suited for distributed, or parallel implementation. To test the gain in performances of such an approach, we have implemented also a parallel version of our gradient computation algorithm based on OpenMP. We compare our two implementations (sequential and parallel) with the implementation provided by *Perseus* which computes the Morse complex using an IG for encoding the input simplicial complex. To the extent of our knowledge, there are no implementations of the discrete Morse complex on a Simplex Tree.

Dataset	$\lvert\Sigma\rvert$	$\lvert C\rvert$	IA^*	IA^*_p	IG
DTI-SCAN	24M	0.14M (171x)	3.1m	0.7m	77.3h
VISMALE	118M	0.94M (125x)	29.2m	6.5m	-
ACKLEY4	204M	0.01M (10^4x)	1.1h	19.7m	-
AMAZON1	2.2M	0.16M (13.7x)	14.5s	3.7s	20.9h
AMAZON2	18.4M	0.37M (49.7x)	281.9s	68.3s	>200h
ROADNET	4.8M	0.75M (6.4x)	15.8s	6.06s	>200h
S1.0	0.6M	16 (10^5)x	56.8s	22.1s	61.7s
S1.2	26M	12 (10^7)x	4.2h	1.8h	-
S1.3	197M	7 (10^8)x	173h	74.3h	-

Table 2: Compression factor achieved by using the discrete Morse complex instead of the original simplicial complex. Column $|C|$ indicates the number of critical simplices, as opposed to the number of simplices $|\Sigma|$, for each dataset. Columns IA^*, IA^*_p and IG indicate the timings required for computing the discrete Morse complex with our sequential implementation, the multi-thread implementation, and the Perseus tool, respectively.

In our experiments, we consider both real and synthetic datasets. The hardware configuration used is an Intel i7 3930K CPU at 3.20Ghz with 64GB of RAM. The data sets used in our experiments are described in Table 1. There are tetrahedralized volume data sets, and data sets obtained from networks and point clouds. Networks and point clouds have no filtration provided as input.

In Table 2, we show first information about the size of the obtained discrete Morse complex (i.e., the number of cells), with respect to the original simplicial complex. The compression factor depends on the homological changes in the filtration of a dataset and on the dataset. Volumetric datasets benefit from a compression of about

two orders of magnitude, network datasets are compressed by a factor of ten, while higher-dimensional complexes are compressed by five to eight orders of magnitude. This shows the advantage of using the Morse complex instead of the original one for computing homological information.

By comparing the timings, we see that our approach (based on the IA^* data structure) always outperforms Perseus (based on the IG). When the number of simplices is low (dataset SPHERE-1.0), the two implementations require a similar amount of time but, as soon as the number of simplices increases, our approach is faster by two or three orders of magnitude. With the increasing of the dimension of the complex, we see that the complexity of computing the discrete Morse complex reaches its limits taking also 173 hours to complete for dataset SPHERE-1.3. In our multi-threaded implementation, we have been able to use eight threads on our machine configuration, processing 8 vertices at a time. The speed up gained varies between a 2x and a 5x.

Figure 13 shows three evaluations. In the first graph, we are evaluating the memory used for representing the simplicial complex Σ (in blue) and the Forman gradient V (in red). We can notice that for the first three data sets, the complex is the entity requiring the highest amount of memory. For the remaining data sets, we notice that when the dimension increases, the storage cost decreases. For example, when comparing ACKELEY4 and S1.0, the total number of simplices is almost the same (see Table 2, column $|\Sigma|$), while memory consumption is dramatically reduced, being dataset S1.3 stored with less than 3.4MB compared to the 7.9GB required by ACKELEY4. This is again due to the use of the IA* data structure and to the encoding for the Forman gradient based on the top simplices.

While extracting the lower star in Algorithm 1, the k-simplices are recursively extracted from the IA^* data structure and explicitly represented. This operation causes the main increase in the memory consumption at runtime. This is documented in the remaining graphs of Figure 13. We are indicating in green the static overhead required for storing the simplicial complex and the Forman gradient and in purple the amount of memory used at runtime.

As we can notice (column IA^*), the difference between the static overhead and the dynamic overhead is larger when working on datasets in higher dimensions, while it becomes negligible when working in two or three dimensions. This fact is intrinsically related to the dimension d of the original simplicial complex. When d is small, the lower star of each vertex is also small. When working on higher dimensional complexes, the number of simplices in the lower star grows exponentially, since the number of simplices on the boundary of any k-simplex is exponential in k. In the worst-case scenario of our experiments (S1.3), the encoding of the star occupies 1.8GB at runtime, while storing the simplicial complex and the gradient requires less than 100MB.

The implementation in Perseus, based on the IG, does not present a difference between static and dynamic overhead, since all the simplices are already represented at the beginning and progressively simplified during the computation. Thus, the maximum peak is reached before starting the reduction algorithm. As a result, the IG presents serious limitations when the dimension of the complex increases.

If we considering our parallel implementation (column IA^*_p), we see that the maximum peak of memory is higher, since all threads run on the same machine. Looking at the graphs in the first column (DTI-SCAN, VISMALE, ACKLEY4), we recognize that the runtime overhead of this version is still comparable to the one of the single-thread implementation. This is an expected result since these are low dimensional data sets with a fairly small lower star for each vertex. With the increasing in the data set dimension (second column), the overhead required by the parallel implementation starts to be relevant. In the worst case, we have experienced a memory overhead up to 6 times larger than the single thread implementation (third column data set S1.3). These results suggest that the whole framework is promising for a distributed environment, where each process has its dedicated amount of memory.

10. Concluding remarks

We have studied different strategies to endow a simplicial complex with a Forman gradient through the use of homology-preserving operators and to extract the corresponding discrete Morse complex.

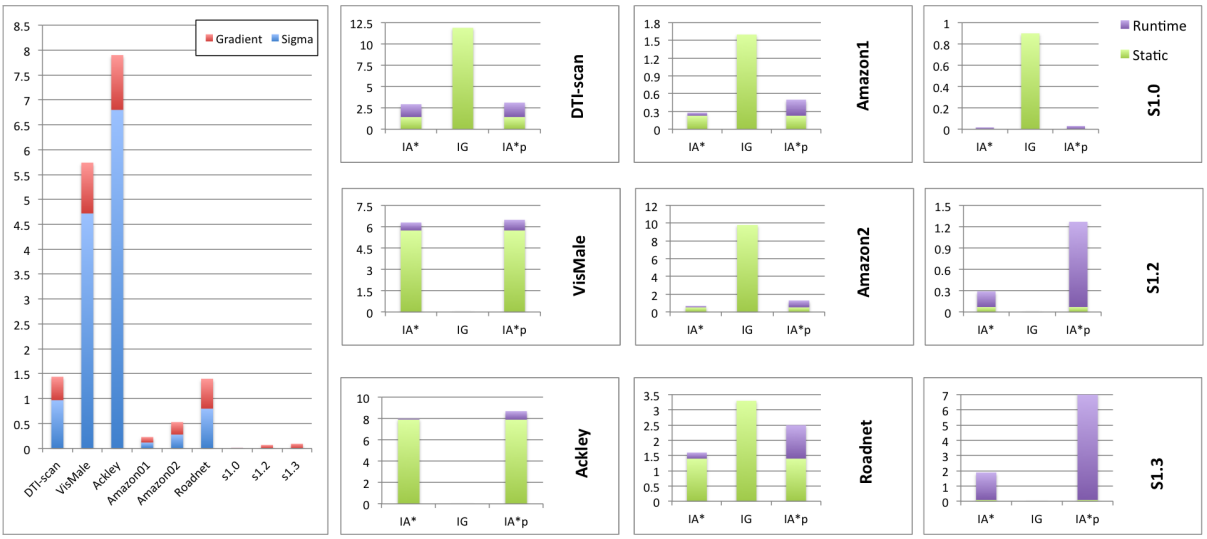


Figure 13: Storage cost required by computing and storing the Forman gradient and the discrete Morse complex. The first graph on the left indicates the amount of memory in GB required for storing the simplicial complex (blue bars) and the Forman gradient (red bars). The remaining graphs indicate, for each dataset, the amount of memory in GB used for storing the complex and the Forman gradient (green bars), and the overhead required at runtime for computing the gradient (purple bars). Results are presented comparing the IA^* data structure, the IG, and the parallel implementation based on the IA^* data structure (indicated as IA^*_p). Missing columns represent experiments that exceeded the maximum amount of memory available.

We have formally proven the theoretical equivalence of such methods which allow for reducing the complexity of the computation through reductions and coreductions. We have developed and implemented algorithms to efficiently build a discrete Morse complex based on coreductions, on a space-efficient representation of the simplicial complex and on a compact encoding of the Forman gradient, also implementing a parallel version of the latter.

Based on the results obtained from the parallel implementation, we are currently working on a distributed version of Algorithm 1. Since the process is localized within the star of each vertex, by distributing the computation on different machines, we expect to get a boost on timings without affecting memory consumption.

We are also considering the application of this work in single-parameter and multi-parameter persistent homology computation, as the basis for tools for shape understanding and retrieval, and in segmentation of time-varying 3D scalar fields in the context of scientific data visualization.

The best implementation currently available in the literature for computing *persistent homology* [28] on high-dimensional complexes is based on annotations and on the Simplex Tree and it represents all the simplices of the simplicial complex explicitly [62]. This is also the case for any persistent homology computation algorithms based on boundary map reduction, because of the need to represent all the simplices explicitly. This puts practical limitations when working on large complexes. In these cases, our approach is particularly useful since the Morse complex is a simpler structure sharing the same persistent homology as the original simplicial complex.

Multi-parameter persistent homology (also called multi-dimensional persistent homology) is an extension of persistent homology for data characterized by multiple parameters, like multi-field data sets. In this case, not a single filtration but multiple filtrations are considered. To date, the approaches proposed in the literature for computing multi-parameter persistent homology are at a pioneering level and are not able to deal with the complexity and the size of real datasets. Recently, an interesting connection between multi-parameter persistent homology and discrete Morse theory has been pointed out in [14]. A formal proof is given

of the equivalence between the multi-parameter persistent homology of the Morse complex defined by a Forman gradient compatible with the multi-filtration and that of the underlying simplicial complex is provided. An algorithm has been proposed by Allili et al. [15] for computing a Forman gradient on a vector-valued function, but its implementation is limited to triangle meshes of very small size. Based on the dimension-independent encoding for the Forman gradient described in this paper, we are planning to develop a new algorithm that works independently of the dimension of the domain (the underlying simplicial complex) and of the codomain (the number of filtrations provided). A parallel implementation will be also at the center of future studies for empowering the computation of multi-parameter persistent homology.

In scientific visualization, *extremum graphs* have been defined as topological tools to understand and visualize the structure of 3D scalar fields, i.e., scalar fields defined at points in the three-dimensional Euclidean space [13]. The *extremum graph* is a subgraph of the graph representing the boundary maps of the Morse complex. We recall that the boundary maps encode all the incidence relations between a critical k-simplex and a critical $(k-1)$-simplex, for $1 \leq k \leq d = dim(\Sigma)$. The extremum graph only represents the boundary maps between critical d-simplices and $(d-1)$-simplices and between critical 1-simplices and 0-simplices. For each pair of critical simplices, it also encodes the chain of simplices that connects the two critical ones. In [13], a visualization technique, called *topological spines*, has been developed specifically for extremum graphs not only of static 3D scalar fields, but also of time-varying fields, which can be regarded as 4D scalar fields. The algorithms described in Section 8 can be suitably adapted to efficiently compute the extremum graphs of a scalar field. Using the scalar function as a filtering function, we can compute the Forman gradient V using Algorithm 1. The gradient paths of V now describe the behavior of the input scalar field. Using Algorithm 2, we can extract the incidence relations between critical simplices by starting the descending traversal from critical d-simplices and from critical 1-simplices. Our approach will make the computation of extremum graphs [68] (and topological spines) feasible for 4D fields, but also for 3D fields defined on a tetrahedral mesh (as needed for complex 3D domains), while the current approach [13] works only works on scalar fields defined on cubic grids.

Acknowledgments

This work has been partially supported by the US National Science Foundation under grant number IIS-1116747. The authors wish to thank Davide Bolognini, Emanuela De Negri and Maria Evelina Rossi for their helpful comments and suggestions.

References

[1] V. De Silva, R. Ghrist, Homological sensor networks, Notices of the American Mathematical Society 54.

[2] R. Fellegara, U. Fugacci, F. Iuricich, L. De Floriani, Analysis of geolocalized social networks based on simplicial complexes, in: 9th ACM SIGSPATIAL International Workshop on Location-Based Social Networks (LSBN), ACM, 2016.

[3] S. Martin, A. Thompson, E. A. Coutsias, J.-P. Watson, Topology of cyclo-octane energy landscape, Journal of Chemical Physics 132 (23) (2010) 234115.

[4] R. van de Weygaert, G. Vegter, H. Edelsbrunner, B. J.-T. Jones, P. Pranav, C. Park, W. A. Hellwing, B. Eldering, N. Kruithof, E. Bos, et al., Alpha, Betti and the Megaparsec universe: on the topology of the cosmic web, in: Transactions on Computational Science XIV, Springer, 2011, pp. 60–101.

[5] M. K. Chung, P. Bubenik, P. T. Kim, Persistence diagrams of cortical surface data, in: Information Processing in Medical Imaging, Springer, 2009, pp. 386–397.

[6] R. Forman, Morse theory for cell complexes, Advances in Mathematics 134 (1) (1998) 90–145.

[7] P. Frosini, M. Pittore, New methods for reducing size graphs, International Journal of Computer Mathematics 70 (3) (1999) 505–517.

[8] A. J. Zomorodian, The tidy set: a minimal simplicial set for computing homology of clique complexes, in: Proceedings of the 2010 Annual Symposium on Computational Geometry, ACM, 2010, pp. 257–266.

[9] L. De Floriani, U. Fugacci, F. Iuricich, P. Magillo, Morse complexes for shape segmentation and homological analysis: discrete models and algorithms, Computer Graphics Forum 34 (2) (2015) 761–785.

[10] V. Robins, P. J. Wood, A. P. Sheppard, Theory and algorithms for constructing discrete Morse complexes from grayscale digital images, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (8) (2011) 1646–1658.

[11] S. Harker, K. Mischaikow, M. Mrozek, V. Nanda, Discrete Morse theoretic algorithms for computing homology of complexes and maps, Foundations of Computational Mathematics 14 (1) (2014) 151–184.

[12] S. Harker, K. Mischaikow, M. Mrozek, V. Nanda, H. Wagner, M. Juda, P. Dłotko, The efficiency of a homology algorithm based on discrete Morse theory and coreductions, in: Proceedings 3rd International Workshop on Computational Topology in Image Context (CTIC 2010). Image A, Vol. 1, 2010, pp. 41–47.

[13] C. Correa, P. Lindstrom, P.-T. Bremer, Topological spines: a structure-preserving visual representation of scalar fields, IEEE Transactions on Visualization and Computer Graphics 17 (12) (2011) 1842–1851.

[14] M. Allili, T. Kaczynski, C. Landi, Reducing complexes in multidimensional persistent homology theory, Journal of Symbolic Computation 78 (2017) 61 – 75.

[15] M. Allili, T. Kaczynski, C. Landi, F. Masoni, Algorithmic construction of acyclic partial matchings for multidimensional persistence, Springer, 2017, pp. 375–387.

[16] U. Fugacci, F. Iuricich, L. De Floriani, Efficient computation of simplicial homology through acyclic matching, in: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2014 16th International Symposium on, 2014, pp. 587–593.

[17] D. Canino, L. De Floriani, K. Weiss, IA^*: an adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions, Computers & Graphics 35 (3) (2011) 747–753.

[18] V. Nanda, The *Perseus* software project for rapid computation of persistent homology. URL `http://www.math.rutgers.edu/~vidit/perseus/index.html`

[19] A. T. Lundell, S. Weingram, The topology of CW complexes, Van Nostrand Reinhold Company, 1969.

[20] H. Edelsbrunner, D. G. Kirkpatrick, R. Seidel, On the shape of a set of points in the plane, IEEE Transactions on Information Theory 29 (4) (1983) 551–559.

[21] E. W. Chambers, V. De Silva, J. Erickson, R. Ghrist, Vietoris-Rips complexes of planar point sets, Discrete & Computational Geometry 44 (1) (2010) 75–90.

[22] A. Hatcher, Algebraic topology, Cambridge University Press, 2002.

[23] A. J. Zomorodian, Fast construction of the Vietoris-Rips complex, Computer and Graphics (2010) 263–271.

[24] V. De Silva, G. Carlsson, Topological estimation using witness complexes, in: Proceedings of the First Eurographics Conference on Point-Based Graphics, 2004, pp. 157–166.

[25] V. De Silva, A weak definition of Delaunay triangulation, arXiv preprint cs/0310031.

[26] L. J. Guibas, S. Y. Oudot, Reconstruction using witness complexes, Discrete & Computational geometry 40 (3) (2008) 325–356.

[27] T. K. Dey, F. Fan, Y. Wang, Graph induced complex on point data, in: Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry, SoCG '13, 2013, pp. 107–116.

[28] H. Edelsbrunner, J. Harer, Persistent homology - a survey, Contemporary Mathematics 453 (2008) 257–282.

[29] A. J. Zomorodian, Topology for computing, Cambridge University Press, 2005.

[30] R. Ghrist, Barcodes: the persistent topology of data, Bulletin of the American Mathematical Society 45 (1) (2008) 61–75.

[31] R. Forman, A user's guide to discrete Morse theory, Séminaire Lotharingien de Combinatoire 48 (2002) 35.

[32] K. Mischaikow, V. Nanda, Morse theory for filtrations and efficient computation of persistent homology, Discrete & Computational Geometry 50 (2) (2013) 330–353.

[33] L. De Floriani, A. Hui, Data structures for simplicial complexes: an analysis and a comparison, in: M. Desbrun, H. Pottmann (Eds.), Proc. 3rd Eurographics Symposium on Geometry Processing, 2005, pp. 119–128.

[34] H. Edelsbrunner, Algorithms in combinatorial geometry, Springer, 1987.

[35] L. De Floriani, D. Greenfieldboyce, A. Hui, A data structure for non-manifold simplicial d-complexes, in: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry processing, ACM, 2004, pp. 83–92.

[36] L. De Floriani, A. Hui, D. Panozzo, D. Canino, A dimension-independent data structure for simplicial complexes, Proceedings of the 19th International Meshing Roundtable (2010) 403–420.

[37] D. Canino, L. De Floriani, Representing simplicial complexes with Mangrove, Proceedings of the 22nd Iinternational Meshing Roundtable (2013) 465–483.

[38] D. Canino, The Mangrove TDS Library: a C++ tool for fast prototyping of topological data structures (2012).
URL `http://mangrovetds.sourceforge.net`

[39] R. Fellegara, K. Weiss, L. De Floriani, The Stellar tree: a compact representation for simplicial complexes and beyond, arXiv preprint arXiv:1707.02211.

[40] J.-D. Boissonnat, C. Maria, The Simplex Tree: an efficient data structure for general simplicial complexes, Algorithmica 70 (3) (2014) 406–427.

[41] C. Maria, J.-D. Boissonnat, M. Glisse, M. Yvinec, The GUDHI library: simplicial complexes and persistent homology, in: H. Hong, C. Yap (Eds.), Mathematical Software ICMS 2014, Springer, 2014, pp. 167–174.

[42] J.-D. Boissonnat, K. C. S., S. Tavenas, Building efficient and compact data structures for simplicial complexes, in: L. Arge, J. Pach (Eds.), 31st International Symposium on Computational Geometry, 2015, pp. 642–656.

[43] D. Attali, A. Lieutier, D. Salinas, Efficient data structure for representing and simplifying simplicial complexes in high dimensions, in: Proceedings of the 27th ACM Symposium on Computational Geometry, 2011, pp. 501–509.

[44] T. Lewiner, H. Lopes, G. Tavares, Optimal discrete Morse functions for 2-manifolds, Computational Geometry 26 (3) (2003) 221 – 233.

[45] M. Mrozek, B. Batko, Coreduction homology algorithm, Discrete & Computational Geometry 41 (1) (2009) 96–118.

[46] M. Mrozek, T. Wanner, Coreduction homology algorithm for inclusions and persistent homology, Comput. Math. Appl. 60 (10) (2010) 2812–2833.

[47] P. Dłotko, T. Kaczynski, M. Mrozek, T. Wanner, Coreduction homology algorithm for regular CW-complexes, Discrete & Computational Geometry 46 (2) (2011) 361–388.

[48] B. Benedetti, F. H. Lutz, Random discrete Morse theory and a new library of triangulations, Experimental Mathematics 23 (1) (2014) 66–94.

[49] F. Cazals, F. Chazal, T. Lewiner, Molecular shape analysis based upon the Morse-Smale complex and the Connolly function, in: Proc. 9th Annual Symposium on Computational Geometry, 2003, pp. 351–360.

[50] H. King, K. Knudson, N. Mramor, Generating discrete Morse functions from point data, Experimental Mathematics 14 (4) (2005) 435–444.

[51] A. Gyulassy, P.-T. Bremer, B. Hamann, V. Pascucci, Practical considerations in Morse-Smale complex computation, in: V. Pascucci, X. Tricoche, H. Hagen, J. Tierny (Eds.), Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications, Mathematics and Visualization, Springer, 2011, pp. 67–78.

[52] A. Gyulassy, P.-T. Bremer, V. Pascucci, Computing Morse-Smale complexes with accurate geometry, IEEE Transactions on Visualization and Computer Graphics 18 (12) (2012) 2014–2022.

[53] D. Günther, J. Reininghaus, H. Wagner, I. Hotz, Efficient computation of 3D Morse-Smale complexes and persistent homology using discrete Morse theory, The Visual Computer 28 (10) (2012) 959–969.

[54] N. Shivashankar, S. Maadasamy, V. Natarajan, Parallel computation of 2D Morse-Smale complexes, IEEE Transactions on Visualization and Computer Graphics 18 (10) (2012) 1757–1770.

[55] N. Shivashankar, V. Natarajan, Parallel computation of 3D Morse-Smale complexes, Computer Graphics Forum 31 (3) (2012) 965–974.

[56] R. Fellegara, F. luricich, L. De Floriani, K. Weiss, Efficient computation and simplification of discrete Morse decompositions on triangulated terrains, in: Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '14, 2014, pp. 223–232.

[57] K. Weiss, F. Iuricich, R. Fellegara, L. De Floriani, A primal/dual representation for discrete Morse complexes on tetrahedral meshes, Computer Graphics Forum 32 (3) (2013) 361–370.

[58] A. J. Zomorodian, G. Carlsson, Computing persistent homology, Discrete & Computational Geometry 33 (2) (2005) 249–274.

[59] U. Bauer, M. Kerber, J. Reininghaus, H. Wagner, PHAT - Persistent Homology Algorithms Toolbox, in: H. Hong, C. Yap (Eds.), Mathematical Software ICMS 2014, Vol. 8592 of Lecture Notes in Computer Science, Springer, 2014, pp. 137–143.

[60] U. Bauer, M. Kerber, J. Reininghaus, Distributed computation of persistent homology, in: Proceedings of the Meeting on Algorithm Engineering & Expermiments, 2014, pp. 31–38.

[61] P. Dłotko, H. Wagner, et al., Simplification of complexes of persistent homology computations, Homology, Homotopy and Applications 16 (1) (2014) 49–63.

[62] J.-D. Boissonnat, T. K. Dey, C. Maria, The compressed annotation matrix: an efficient data structure for computing persistent cohomology, Algorithmica 73 (3) (2015) 607–619.

[63] S. Pemmaraju, S. Skiena, Computational Discrete Mathematics: combinatorics and graph theory with Mathematica, Cambridge University Press, 2003.

[64] D. H. Ackley, A connectionist machine for genetic hillclimbing, Kluwer Academic Publishers, 1987.

[65] F. Iuricich, The IA^*, an indexed-based data structure with adjacencies for encoding simplicial complexes.
URL `https://github.com/IuricichF/IAstar`

[66] M. Joswig, M. E. Pfetsch, Computing optimal Morse matchings, SIAM J. Discret. Math. 20 (1) (2006) 11–25.

[67] D. Günther, J. Reininghaus, I. Hotz, H. Wagner, Memory-efficient computation of persistent homology for 3D images using discrete Morse theory, in: 24th SIBGRAPI Conference

on Graphics, Patterns and Images, 2011, pp. 25–32.

[68] V. Narayanan, D. M. Thomas, V. Natarajan, Distance between extremum graphs, in: 2015 IEEE Pacific Visualization Symposium (PacificVis), 2015, pp. 263–270.