Comput. Geom. 89 (2020) 101623

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Computing multiparameter persistent homology through a discrete Morse-based approach

Sara Scaramuccia [a], Federico Iuricich [b,*], Leila De Floriani [c], Claudia Landi [d]

[a] University of Genova, Genova, Italy
[b] Clemson University, Clemson (SC), USA
[c] University of Maryland, College Park (MD), USA
[d] University of Modena and Reggio Emilia, Italy

ARTICLE INFO

Article history:
Received 23 March 2019
Received in revised form 28 January 2020
Accepted 6 February 2020
Available online 12 February 2020

Keywords:
Multiparameter persistent homology
Topological data analysis
Discrete Morse theory
Morse reductions
Homotopy expansion

ABSTRACT

Persistent homology allows for tracking topological features, like loops, holes and their higher-dimensional analogues, along a single-parameter family of nested shapes. Computing descriptors for complex data characterized by multiple parameters is becoming a major challenging task in several applications, including physics, chemistry, medicine, and geography. *Multiparameter persistent homology* generalizes persistent homology to allow for the exploration and analysis of shapes endowed with multiple filtering functions. Still, computational constraints prevent multiparameter persistent homology to be a feasible tool for analyzing large size data sets. We consider *discrete Morse theory* as a strategy to reduce the computation of multiparameter persistent homology by working on a reduced dataset. We propose a new preprocessing algorithm, well suited for parallel and distributed implementations, and we provide the first evaluation of the impact of multiparameter persistent homology on computations.

The increasing amount of data available has led to the development of information handling techniques beyond machine learning approaches. Topological Data Analysis in particular provides a set of new tools for retrieving, organizing and analyzing complex data by focusing on qualitative information about their shape. Recent applications of topological data analysis in neuroscience [8,41], image processing [16,1,63] and astrophysics [61], to name a few, have proven its strength and versatility.

Homology [43] is one of the most relevant tools used in topology but suffers from the drawback of being scarcely descriptive. *Persistent homology* [32] overcomes this issue by allowing for multiresolution analysis of homology by means of *filtrations*. It is used in data analysis to study the evolution of qualitative features of data and it is appreciated for its robustness to noise, and dimension independence.

So far, many optimization methods for computing persistent homology have been proposed. Those more tightly related to this work refer to another relevant tool for topological data analysis, namely *discrete Morse theory* [37]. Discrete Morse theory provides an important preprocessing tool for homology computation. By defining a *discrete gradient vector field* (also called *discrete gradient*) over the input datum, the size of the input space can be reduced to the critical parts, generally few. The discrete gradient can also be built so as to preserve the filtration structure, thus enhancing also persistent homology

* Corresponding author.
E-mail addresses: sara.scaramuccia@dibris.unige.it (S. Scaramuccia), fiurici@clemson.edu (F. Iuricich), deflo@umiacs.umd.edu (L. De Floriani), claudia.landi@unimore.it (C. Landi).

https://doi.org/10.1016/j.comgeo.2020.101623
0925-7721/

computations via a reduction procedure. Although other persistent homology optimizations outperform this Morse-based preprocessing, these no longer apply to the generalization of persistent homology, called *multiparameter persistent homology*.

Multiparameter persistent homology is an extension of persistent homology motivated by the fact that data analysis and comparisons often involve studying properties naturally described by multiple parameters. However, high computational costs and scalability problems prevent it to be applicable over large data. A Morse-based preprocessing solution, generalized to the multiparameter case, has been proposed in [3,5]. This can have, in theory, a valuable impact on multiparameter persistent homology computations. However, that preprocessing still presents limitations in scalability with real data.

In [45] we have proposed the first algorithm capable of computing a discrete gradient on simplicial real-sized multifiltered shapes and images. We have integrated the discrete gradient into a visualization tool for studying regions of correlation in a multifield dataset, i.e., a regular grid with a vector-valued function defined on its vertices. In this work, we extend the algorithm presented in [45] to the computation of multiparameter persistent homology. Taking the work by Allili et al. [5] as the state-of-the-art for computing a discrete gradient for multiparameter persistent homology computation, our contributions consist of:

- a new efficient and parallel algorithm for computing a discrete gradient on multiparameter filtrations;
- a detailed analysis of the algorithm's complexity and a proof of its equivalence to [5];
- a comparison of complexity and computational performances of our algorithm with respect to [5];
- an evaluation of the advantages of using our algorithm as a preprocessing step for multiparameter persistent homology computation.

Our approach is well suited to be used with both simplicial complexes and regular grids and is well suited for a parallel implementation. Moreover, we show that the use of the discrete gradient provides an improvement of at least one order of magnitude in the computation of multipersistent homology.

The remainder of this paper is organized as follows. In Section 1, we introduce the notions at the basis of our work. Related work is reviewed in Section 2. The new preprocessing algorithm is described in Section 3, where we also present a detailed complexity analysis. In Section 4, we present the proof of correctness as well as a theoretical and experimental comparison of our approach with the one presented by Allili et al. [5]. The results obtained by computing multiparameter persistent homology with our approach are discussed in Section 5. Finally, in Section 6, we draw concluding remarks and we discuss future developments.

1. Background

In this section, we introduce the notions at the base of our work. After simplicial complexes we discuss homology [43] and persistent homology [32]. Then, we describe multiparameter persistent homology [18]. We conclude the section by discussing discrete Morse theory [37].

1.1. Simplicial complexes

A simplicial complex is a discrete topological structure made of simple elements, called simplices. A *k-dimensional simplex* σ, or k-simplex for short, is the convex hull of $k+1$ affinely independent points. Often, we will write σ^k to indicate a k-simplex. A *face* τ of σ is the convex hull of any subset of points generating σ. If the dimensions of τ and σ differ by one we call τ a *facet* of σ. Dually, σ is a *coface* of τ and a *cofacet* when the two dimensions differ by one.

A *simplicial complex* S is a finite collection of simplices such that:

- every face of a simplex in S is also in S,
- the intersection of any two simplices in S is either empty or a single simplex in S (*intersection property*).

We will denote by S_k the set of k-simplices in S. An element in S_0 is also called a *vertex*. A simplicial complex S of dimension d is a simplicial complex in which the maximum dimension of the simplices of S is d.

1.2. Persistent homology

Homology is a topological invariant used in data analysis to qualitatively describe shapes. The homology of a simplicial complex S detects independent k-dimensional cycles of S, i.e., connected components (0-cycles), tunnels (1-cycles), voids (2-cycles), and so on. Cycles are formally captured by linear combinations of simplices whose boundary vanishes. In this work, we focus on linear combinations over $\mathbb{F}_2$, i.e., the field with only the two elements 0 and 1.

The *chain complex* $\mathcal{C}(S) = (\mathcal{C}_*(S), \partial_*)$ associated with a simplicial complex S consists of the family $\mathcal{C}_*(S) = \{\mathcal{C}_k(S)\}_{k\in\mathbb{Z}}$ of $\mathbb{F}_2$-vector spaces along with the collection of linear maps $\partial_* = \{\partial_k : \mathcal{C}_k(S) \longrightarrow \mathcal{C}_{k-1}(S)\}_{k\in\mathbb{Z}}$ defined as follows:

- $\mathcal{C}_k(S)$ is the $\mathbb{F}_2$-vector space generated by S_k, and its elements are called *k-chains*,

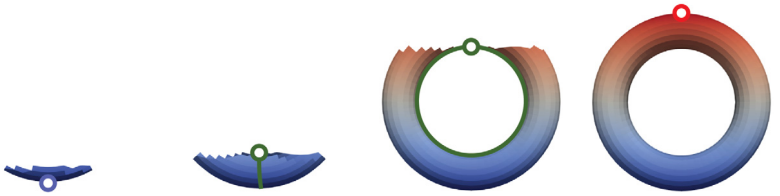

Fig. 1. A filtering function defined on a torus. Dots indicate the presence of a new component (a), new tunnels (b-c), and a new void (d), respectively. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

- ∂_k is called *boundary map* and defined by linear extension from the images of each k-simplex τ:

$$\partial_k(\tau) = \sum_{\sigma \in S} \kappa(\tau, \sigma)\sigma,$$

with $\kappa(\tau, \sigma) = 1$ if and only if σ is a facet of τ. Elements in the kernel of ∂_k are called *k-cycles*, while elements in the image of ∂_{k+1} are called *k-boundaries*. The *k-homology* of a simplicial complex S is defined as the quotient vector space of k-cycles over k-boundaries:

$$H_k(S) = \ker \partial_k / \operatorname{im} \partial_{k+1}.$$

Persistent homology describes the homological changes occurring along an increasing sequence of simplicial complexes, called a *filtration*. We consider $\mathbb{R}$ with its usual order, and we call its elements *grades*. A *filtration* $\mathcal{S}$ of a simplicial complex S is a finite collection of simplicial subcomplexes S^u in S, indexed by $u \in \mathbb{R}$, such that for all grades $u \leqslant v$, S^u is a simplicial subcomplex in S^v. Here, we are interested in filtrations derived by the sublevel sets of functions defined on S: a *filtering function* is a function $f : S \longrightarrow \mathbb{R}$ such that, if σ is a face of τ, then $f(\sigma) \leqslant f(\tau)$.

Given a filtering function f, $\mathcal{S}(f)$ defined by setting $S^u = \{\sigma \in S | f(\sigma) \leqslant u\}$ is a filtration of S. For each pair of grades $u \leqslant v$ of the filtration $\mathcal{S}(f)$, we have that $S^u(f)$ is closed in $S^v(f)$. This means that for all $\tau \in S^u(f)$, if condition $\kappa(\tau, \sigma) \neq 0$ holds for some $\sigma \in S^v(f)$, then $\sigma \in S^u(f)$.

In Fig. 1 we show a torus filtered by using the height function as filtering function. Each image illustrates a change in the homology of the sublevel sets. In Fig. 1(a) a new component is introduced. Two loops are created, in Fig. 1(b) and (c), respectively. A void is created in Fig. 1(d). For each homology class, we have a representative k-cycle appearing in the filtration. The marked blue dot is the representative 0-cycle for the new component. The two green 1-cycles are representative for the two tunnels. The set of triangles forming the entire torus surface corresponds to a 2-cycle.

The inclusion of simplicial complexes in a filtration preserves cycles and boundaries, that is, for all grades $u \leqslant v$, there exists a linear map $\iota_k^{u,v} : H_k(S^u) \longrightarrow H_k(S^v)$ induced at homology level, not necessarily injective, since cycles can possibly become boundaries by adding cells.

The *persistent* kth-*homology* relative to the grades $u \leqslant v$ is the image of $\iota_k^{u,v}$ as a subspace in $H_k(S^v)$, that is the space of all the homology classes of $H_k(S^u)$ which persist in $H_k(S^v)$. The global persistent homology information, for all possible grades $u \leqslant v$, is encoded in the *persistence module*.

The kth *persistence module* $H_k(\mathcal{S})$ of the filtered complex $\mathcal{S}$ consists of:

- the collection of $\mathbb{F}_2$-vector spaces $H_k(S^u)$ varying the grade $u \in \mathbb{R}$,
- the collection of all inclusion-induced linear maps $\iota_k^{u,v} : H_k(S^u) \longrightarrow H_k(S^v)$, varying the pairs of grades satisfying $u \leqslant v$ in $\mathbb{R}$.

In this work, we are interested in a generalization of persistent homology obtained by considering multiple filtrations at once. Such tool is called multiparameter persistent homology. So, from now on, we refer to classic persistent homology as one-parameter persistent homology.

1.3. Multiparameter persistent homology

Multiparameter persistent homology analyzes the changes in homology for a *multiparameter filtration* (or *multifiltration*, for short). Instead of a total order on $\mathbb{R}$, we consider a partial ordered set $(\mathbb{R}^n, \preceq)$ such that, for any $u = (u_1, \ldots, u_n), v = (v_1, \ldots, v_n) \in \mathbb{R}^n$, $u \preceq v$ if and only if $u_i \leqslant v_i$, for all $i \in \{1, \ldots, n\}$. We call *grades* the elements of the partial order set. If $u \preceq v$ and $u \neq v$, we write $u \precneqq v$ for short.

A *multiparameter filtration* $\mathcal{S}$ of a simplicial complex S is a finite collection of simplicial subcomplexes S^u indexed by $u \in \mathbb{R}^n$ such that, for all grades $u \preceq v$, S^u is a simplicial subcomplex in S^v. Multiparameter filtrations can be induced by vector-valued functions. Any function $f : S \longrightarrow \mathbb{R}^n$ satisfying $f(\sigma) \preceq f(\tau)$, for all faces σ of τ, induces a multifiltration $S(f)$ by setting S^u as the set of all simplices σ satisfying $f(\sigma) \preceq u$. In this case, f is called a *(multi)filtering function* on S, and S^u is called the *sublevel set* with respect to the *(multi)grade* u.

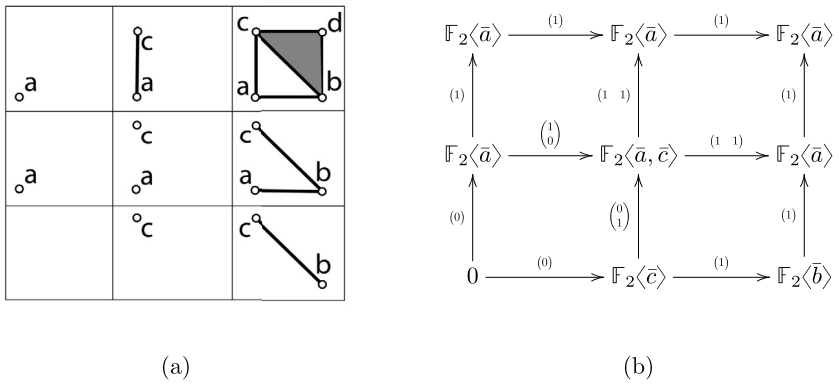

Fig. 2. (a) A bifiltration of a 2-dimensional simplicial complex. (b) A representation of the corresponding persistence module in degree 0 (connected components). For each $\mathbb{F}_2$-vector space in the persistence module, the corresponding reference basis is made explicit, e.g., $\bar{a}$ is the homology class of the vertex a. Each matrix expresses the linear maps with respect to such bases.

Given a multiparameter filtration of a simplicial complex S, homology construction $H_k(\cdot)$ can be applied to each sublevel set in the multifiltration. Each inclusion $S^u \subseteq S^v$ between multifiltration sublevel sets induces a linear map $\iota_k^{u,v} : H_k(S^u) \longrightarrow H_k(S^v)$.

A homology class is *persistent from grade u to grade v* if it is not trivial in $H_k(S^u)$ and still non-trivial in $H_k(S^v)$. For each homology degree k, the kth *persistence module* of a multifiltration of S is the family of vector spaces $H_k(S^u)$ with $u \in \mathbb{R}^n$ along with all linear maps $\iota_k^{u,v}$ with $u \preceq v$. In Fig. 2(b), we show the persistence module associated with the bifiltration depicted in Fig. 2(a).

1.4. Discrete Morse theory

The algorithm we propose retrieves a combinatorial object, called a *discrete gradient*, compatible with the multiparameter filtration over a simplicial complex S. The relevance of this output has to be considered within the framework of Forman's *discrete Morse theory* [37]. A *(discrete) vector* is a pair of simplices (σ, τ) such that σ is a facet of τ. A *discrete vector field* is any collection of vectors V such that each simplex is a component of at most one vector in V. A *V-path* is a sequence of vectors (σ_i, τ_i) belonging to V, for $i = 0, \ldots, r$, such that, for all indexes $0 \leqslant i \leqslant r-1$, σ_{i+1} is a facet of τ_i and $\sigma_i \neq \sigma_{i+1}$. A V-path is said to be *closed* if $\sigma_0 = \sigma_r$, and *trivial*, if $r = 0$. A discrete vector field V is a *discrete gradient* if all of its closed V-paths are trivial. Simplices that do not belong to any vector are said to be *critical*. Given a discrete gradient V, a *separatrix* from a critical cell τ^{k+1} to a critical cell σ^k is a V-path from any facet of τ^{k+1} to any cofacet of σ^k.

There are many ways to construct a discrete gradient on a simplicial complex. In this work, we are interested in a specific class of discrete gradients, those consistent with a multiparameter filtration. Given a multifiltration $\mathcal{S}$ of a simplicial complex S, a discrete gradient V over S is called *compatible with $\mathcal{S}$* if, for each $(\sigma, \tau) \in V$ and each filtration grade $u \in \mathbb{R}^n$, it holds that

$$\sigma \in S^u \quad \Rightarrow \quad \tau \in S^u$$

A discrete gradient implicitly represents a cell complex called a *Morse complex*, and generated by navigating the gradient V-paths. The cells of a Morse complex M of V are in one-to-one correspondence with the critical simplices of V. For any two critical simplices, the incidence between the corresponding cells in M is defined based on the following *incidence function*: given σ, τ in M, $\kappa_M(\sigma, \tau) = 1$ if and only if the number of separatrices from σ to τ is odd and $\kappa_M(\sigma, \tau) = 0$ otherwise. Analogously to the simplicial case, κ_M allows us to define a boundary map and, hence, the homology of the Morse complex.

Theorem 4.3 in [51] proves that, for one-parameter filtrations, the persistent homology of the Morse complex of V is equivalent to that of the original simplicial complex. Corollary 3.2 in [3] proves the same result for multiparameter filtrations. From a computational point of view, a key advantage is the size of M with respect to S: M contains fewer cells than S, making the computation of the persistence module faster.

2. Related work

In this section, we review related work on the computation of both persistent and multipersistent homology.

2.1. Computing persistent homology

In the one-parameter case, computing the persistence module of a complex, consists of reducing the *boundary matrix* of the complex [32]. The standard algorithm by Edelsbrunner et al. [32] has a cubic time complexity in the worst case but

new approaches have been studied, with an improved efficiency. We classify such approaches into three groups: *integrated optimizations*, *annotation-based*, and *preprocessing algorithms*.

Integrated optimizations aim at improving the efficiency of the standard approach by either reducing the number of steps required for matrix reduction, or by progressively removing columns during computation. Examples of approaches based on integrated optimizations are: the *twist* algorithm [23], the *row* algorithm [24], the approach based on *sparsity* presented in [50], the one based on *spectral sequences* [30], and the *chunk* algorithm [7].

Annotation-based algorithms take advantage of an efficient data structure, the annotation matrix, to compute the persistent co-homology of a complex. An *annotation* [13] is a map that assigns a binary vector to each simplex of a simplicial complex. Each annotation provides the coordinate vectors that are used to efficiently identify the homology classes. The notion of annotation was originally introduced in [13] for computing localized homology. In [25,12] the approach has been adapted and improved for persistent homology computation.

Preprocessing algorithms reduce the size of the input filtration while preserving its persistent homology. In [28], homology-preserving techniques, such as reductions, coreductions [52,54,27] and acyclic subspaces [53], are adapted to the case of persistent homology. Approaches rooted in discrete Morse Theory [37] compute a discrete gradient V compatible with the input filtration. The theoretical results in [51] guarantee that the Morse complex constructed from V has the same persistence module as the input complex. Many algorithms have been developed for computing a discrete gradient from a function sampled at the vertices of a cell complex. The algorithm described in [46] is the first one to introduce a divide-and-conquer approach for computing a Forman gradient on real data. However, it suffers from the drawback of introducing many spurious critical simplices. Two parallel approaches have been defined in [59,60] for 2D and 3D images, respectively. They provide a substantial speedup in computing the discrete gradient, but they still create spurious critical simplices. In [57], a dimension-agnostic algorithm is proposed that processes each vertex locally. It has been proved that, up to the 3D case, the critical cells identified are in one-to-one correspondence with the topological changes in the sub-level sets, i.e. no spurious critical simplices are created. An efficient implementation of [57] on regular grids is discussed in [42]. Similar approaches have been developed for triangle [36] and tetrahedral meshes [62], and for dimension independent simplicial complexes [38].

2.2. Computing multiparameter persistent homology

The first issue when computing multiparameter persistent homology is that no complete descriptor exists for the persistence module [18]. The first algorithm for retrieving the persistence module is proposed in [17], where the three tasks of computing the k-boundaries, k-cycles and their quotients at each multigrade u are translated into *submodule membership problems* in computational commutative algebra. The algorithm introduces an artifact dependency on the chosen basis, and has a time complexity of $O(|S|^4 n^3)$, where $|S|$ is the number of simplices in the complex and n is the number of independent parameters in the multifiltration. The algorithm in [40] acts on the multifiltration at the chain level rather than at homology level. First, k-cycles and k-boundaries are expressed in terms of the same basis. Then, the Smith Normal Form reduction [55,2] is applied at each multigrade u in the multifiltration leading to a worst time complexity of $O(|S|^3 \bar{\mu}^n)$, with $\bar{\mu} := \max_{i=0,\dots,n} \mu_i$, where μ_i is the number of multigrades in the multifiltration along the i^{th}-axis. The algorithm has been implemented in the *Topcat* library [56]. Recently, a new algorithm based on a generalized matrix reduction technique has been published on Arxiv [26] with a worst-case time complexity of $O(|S|^4)$ in the 2-parameter case.

An alternative to the persistence module is computing invariants providing partial information. A non-complete descriptor for multiparameter persistent homology is the *rank invariant*, introduced in [18]. For each pair of multigrades $u \preceq v$ the rank invariant is the rank of the corresponding inclusion-induced map, which is the number of homology classes from multigrade u still persistent at multigrade v. The rank invariant value over a single pair (u, v) can be easily derived from the persistence module representation. However, computing the full rank invariant means computing its value for each possible pair of multigrades (u, v) satisfying $u \preceq v$, which is $\frac{1}{2}\mu^2$, where μ is the cardinality of all multigrades considered in the multifiltration (typically very large).

The *persistence space* [21,22] is equivalent to the rank invariant but it avoids computing the persistence module. The persistence space can be computed based on the *foliation method* [10]. With such approach, the persistence space is constructed incrementally by slicing the space of the input multiparameter filtrations and by constructing a number of one-parameter filtrations on which classic persistence homology is computed. The persistence pairs obtained on each slice form the persistence space. The first approach for computing the persistence space has been limited to the case of 0^{th}-homology [10]. An approximate version of the persistence space is proposed in [9] for two-parameter filtrations, also called *bifiltrations*: a selection of slices is performed to guarantee a fixed tolerance for the matching distance [14] among persistence spaces. This method has found applications in shape comparison in the PHOG library [11], where the authors use the approximate persistence space to deal with photometric attributes of a 3D shape.

Limitedly to bifiltrations, a visualization tool for the persistence space is RIVET [49]. RIVET uses bigraded Betti numbers [47,34] to locate multigrades where homology classes born or die. This procedure requires time $O(\mu^3\lambda)$, where λ is the product of $\lambda_x\lambda_y$ with λ_x and λ_y the number of x- and y-coordinates of such multigrades. RIVET computes an arrangement of lines such that all the filtrations within the same cell of the arrangement, have the same *barcode template*. A barcode template is constructed in $O(\mu^3\lambda + (\mu + \log\lambda)\lambda^2)$ time.

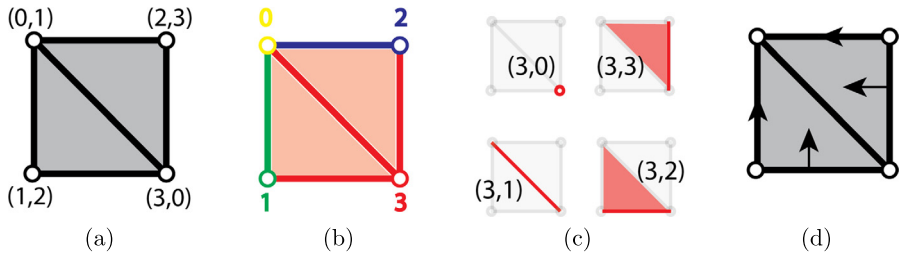

Fig. 3. (a) A multiparameter filtration (S, f) where S is a triangle mesh. (b) Value of the indexing I for each simplex of S. (c) Within the index-based lower star of $\mathrm{Low}_I(3)$, simplices are subdivided and paired based on their value of f. (d) The final discrete gradient V.

Most optimization methods developed for classic persistent homology have not yet found a counterpart in the multiparameter case. Limitedly to the study of 0^{th}-homology the algorithm proposed in [20] is the first approach capable of reducing the size of an input complex S without affecting its persistence module.

The approach proposed in [3] can be seen as a Morse-based method generalizing to the multiparameter case the one proposed in [46]. The algorithm computes a discrete gradient field having the same persistence module as the input complex. Like its one-parameter counterpart [46], it suffers from introducing many spurious critical simplices. In a successive paper by Allili et al. [5], authors propose to construct the discrete gradient locally inside the lower star of the simplices of S. The resulting discrete gradient is proven to induce a Morse complex with the same persistence module as the original multifiltration. However, the algorithm requires a global ordering of all the simplices of S and does not scale well with the size of the data. In Section 4, we will discuss this issue further when analyzing our approach.

A new reduction algorithm has been proposed by Fugacci et al. [39] inspired by the chunk algorithm [7]. The latter works on a matrix-based representation of the simplicial complex S called boundary matrix. Each column of the *boundary matrix* represents a simplex σ in S by storing the facets of σ. The method in [39] and the Morse-based approach discussed in this paper have been experimentally compared showing that the one in [39] is an order of magnitude faster than ours on triangle meshes. On the other hand, the Morse-based approach has other valuable characteristics. Above all, with the Morse-based approach, we are able to maintain a complete mapping between the original simplicial complex and the obtained Morse complex. This means that, once we have computed multiparameter persistent homology on the Morse complex, we are able to project back the results on the simplicial complex for studying, for example, the distribution of the critical simplices on the original complex.

3. Local computation of a discrete gradient over a multiparameter filtration

In this section, we present a new algorithm for computing a discrete gradient vector field compatible with a multiparameter filtration. To make the exposition easier, we describe the algorithm by focusing on simplicial complexes, although the approach is valid for any cell complex satisfying the intersection property, such as *cubical complexes*.

In Section 3.1 we provide a high-level description of the algorithm workflow. A detailed description of the auxiliary functions is given in Section 3.2, while in Section 3.3 we discuss the algorithm complexity.

3.1. Overview

The proposed algorithm receives a multiparameter filtration as input and produces a compatible discrete gradient. In the following, we describe the input multiparameter filtration as a pair (S, f), where S is a d-dimensional simplicial complex and $f : S_0 \longrightarrow \mathbb{R}^n$ is a vector-valued function defined on the vertices of S. We indicate with $f_i(v)$ the i-th component of the vector associated with a vertex v. f_i is extended to any simplex σ of S by $f_i(\sigma) = \max_{v \in \sigma} f_i(v)$, for each $0 \leqslant i \leqslant n$, and induces the multifiltration that we denote as $S(f)$, as described in Section 1.3.

Without loss of generality, we require function f to be component-wise injective on the vertices, i.e., it is injective on the vertices for each component. In applications, any function can be transformed into a component-wise injective one by means of simulation of simplicity [33]. The output of the algorithm is a pair (V, M), where V is the set of paired simplices of S and M is the set of the critical (unpaired) simplices. When proving correctness, we will show that V is a discrete gradient compatible with the filtration, and that M contains the cells of its Morse complex.

The main strategy of the algorithm consists of decomposing S according to an indexing I defined on its vertices. The use of I allows partitioning S into disjoint sets and then run the pairing algorithm in parallel on each set. In this way we avoid the global ordering used in the algorithm by Allili et al. [5]. A full comparison of the two approaches is provided in Section 4. Notice that the final result is independent of the choice of I (Lemma 2 in Section 4).

The algorithm consists of three main steps: *vertex-based decomposition*, *multigrade grouping*, and *pairing*. A running example is depicted in Fig. 3.

In the first step we decompose S to obtain a partition of the simplices in S. In this step, we only require that simplices belonging to the same multigrade also belong to the same group in the decomposition. This is performed by algorithm `ComputeDiscreteGradient` (Algorithm 1) as follows:

Algorithm 1 ComputeDiscreteGradient(S, f).

Input: S, a simplicial complex; $f : S_0 \longrightarrow \mathbb{R}^n$, a component-wise injective function;
Output: V, list of simplices pairs; M, list of critical simplices;
1: define V and M as two empty lists
2: $I \leftarrow$ `ComputeIndexing`(S_0, f)
3: **for all** v in S_0 **do**
4: $L_I \leftarrow$ `ComputeIndexLowerStar` (v, I, S)
5: **for all** L_u in `SplitIndexLowerStar`(f, L_I) **do**
6: $(V_u, M_u) \leftarrow$ `HomotopyExpansion`(S, I, L_u)
7: add V_u to V
8: add M_u to M
9: **return** (V, M)

- an indexing $I : S_0 \longrightarrow \mathbb{R}$ is computed for the vertices of S (line 2). The indexing is extended to the other simplices $\sigma \in S$ by setting $I(\sigma) := \max_{v \in \sigma} I(v)$. The indexing used must be *well-extensible*, i.e., I must satisfy, for all simplices $\sigma, \tau \in S$, the following property:

$$f(\sigma) \preceq f(\tau) \qquad \Rightarrow \qquad I(\sigma) \leqslant I(\tau). \tag{1}$$

- S is subdivided into lower stars according to I (line 4). We recall that the star of a simplex σ, denoted by $Star(\sigma)$, is the set of its cofaces. Then, the *index-based lower star* of a simplex σ, denoted as $\mathrm{Low}_I(\sigma)$, is the set of its cofaces having a value of I lower or equal to σ. Formally,

$$\mathrm{Low}_I(\sigma) := \{\tau \in \mathrm{Star}(\sigma) \mid I(\tau) \leqslant I(\sigma)\}.$$

In Fig. 3(b) we indicate for each simplex the corresponding value of I. The well-extensible indexing I and the index-based lower star Low_I satisfy two fundamental properties:

- each simplex σ belongs to the index-based lower star of exactly one vertex (see Lemma 1 in Section 4);
- if two simplices have the same value of f, then they belong to the index-based lower star of the same vertex (see Lemma 2 in Section 4).

As a consequence, a well-extensible indexing and the index-based lower stars implicitly provide a partition of S. Thanks to these properties we can guarantee that, by processing the vertices independently, we are identifying valid pairings (see Section 4 for a formal proof).

The second step requires grouping the simplices in the set $L_I = \mathrm{Low}_I(v)$, with $v \in S_0$, having the same multigrade (line 5). The latter is performed by `SplitIndexLowerStar`, which organizes the simplices in a collection of sets, where each set L_u contains simplices with the same multigrade u, i.e., $L_u = \{\sigma \in \mathrm{Low}_I(v) | f(\sigma) = u\}$, $u \in \mathbb{R}^n$. In Fig. 3(c), simplices belonging to the index based lower star $\mathrm{Low}_I(3)$ are partitioned based on their value of f.

In the third step (line 6), each level set L_u is independently processed by `HomotopyExpansion` and the gradient pairs are computed. Paired and critical simplices found in the level set L_u will contribute to the final discrete gradient. Fig. 3(d) shows the final discrete gradient V computed by our algorithm on the multiparameter filtration depicted in Fig. 3(a). Since simplices are subdivided based on their multigrade, each simplex in S appears in exactly one level set and it will be classified, as either paired or critical, only once. This makes the approach easy to parallelize.

3.2. Detailed description

This section provides additional information about the auxiliary functions used in Algorithm 1.

`ComputeIndexing` computes a well-extensible indexing on the vertices of S. There are many ways to obtain a well-extensible indexing I, here we have chosen to sort all the vertices of S according to the values of the first component of f. The total order obtained naturally generates an indexing which is guaranteed to be well-extensible, since, for each pair of simplices σ and τ, $f(\sigma) \preceq f(\tau)$ implies $f_1(\sigma) \leq f_1(\tau)$. Thus, a vertex $v \in \tau$ exists such that $f_1(v) \geqslant f_1(w)$ for every vertex $w \in \sigma$. This implies $I(v) \geqslant I(w)$, for every vertex $w \in \sigma$ and we conclude that $I(\sigma) \leqslant I(\tau)$.

`ComputeIndexLowerStar` computes the index-based lower star $\mathrm{Low}_I(v)$ of a vertex v with respect to indexing I. For each vertex $v \in S_0$, the function extracts the set of simplices of $\mathrm{Low}_I(v)$ incident into v. To do so, we assume that each k-simplex σ is represented by the list of its $k+1$ vertices $[v_0, v_1, \ldots, v_k]$ stored in decreasing order of I, i.e., $I(v_0) > I(v_1) > \cdots > I(v_k)$. Thus, it is sufficient to collect those simplices whose first vertex is v.

Each index-based lower star $\mathrm{Low}_I(v)$ is then partitioned into level sets by `SplitIndexLowerStar` according to multigrades. This function creates an associative array, by cycling on the simplices of $\mathrm{Low}_I(v)$, that maps each multigrade u (represented as a vector of floats) to the set of simplices sharing the same multigrade.

Function `HomotopyExpansion` classifies simplices with the same multigrade. We present its pseudocode in Algorithm 2. The algorithm extends the one in [57]. A k-simplex σ and a $(k+1)$-simplex τ are considered *pairable* only when

Algorithm 2 HomotopyExpansion(S, I, L_u).

Input: S, a simplicial complex; I, a well-extensible indexing; L_u, a list of cells in S forming a level set u w.r.t. f;
Output: V_u, list of discrete vectors; M_u, list of critical simplices;
1: define V_u and M_u two empty lists
2: define `Ord0` and `Ord1` two empty ordered sets
3: define `declared` an array of length $|L_u|$ with Boolean values equal to `false`
4: **for all** τ in L_u **do**
5: **if** `num_undeclared_facets`$(\tau, L_u) = 0$ **then**
6: `insert`$(\tau$,`Ord0`,$I)$
7: **else if** `num_undeclared_facets`$(\tau, L_u) = 1$ **then**
8: `insert`$(\tau$,`Ord1`,$I)$
9: **while** `Ord1`$\neq \varnothing$ or `Ord0`$\neq \varnothing$ **do**
10: **while** `Ord1`$\neq \varnothing$ **do**
11: $\tau \leftarrow$ `delete`(`Ord1`)
12: **if** `num_undeclared_facets`$(\tau, L_u) = 0$ **then**
13: `insert`$(\tau$,`Ord0`,$I)$
14: **else**
15: $\rho \leftarrow$ `unpaired_facet`(τ, L_u)
16: `append`$((\rho, \tau)$,$V_u)$
17: `declared`$[\rho] \leftarrow$`true` , `declared`$[\tau] \leftarrow$`true`
18: `add_cofacets`$(\rho$,L_u,I,`Ord1`$)$
19: `add_cofacets`$(\tau, L_u$,I,`Ord1`$)$
20: **if** `Ord0`$\neq \varnothing$ **then**
21: $\tau \leftarrow$ `delete`(`Ord0`)
22: `append`$(\tau$,$M_u)$
23: `declared`$[\tau] \leftarrow$ `true`
24: `add_cofacets`$(\tau, L_u$,I,`Ord1`$)$
25: **return** (V_u, M_u)

σ is the only unclassified facet of τ. The main objective of `HomotopyExpansion` is that of pairing as many simplices as possible and to classify them as critical only when no pairable simplices are available.

Two ordered sets `Ord0` and `Ord1` are used to keep track of those simplices that have exactly zero unpaired facets and one unpaired facet, respectively. Intuitively, simplices in `Ord0` are candidates to be classified as critical or as tails of arrows in the discrete vector, since they have no face to be paired with. Simplices in `Ord1` are the candidates to be heads of arrows in a discrete vector. Within the two sets `Ord0` and `Ord1`, a simplex σ is represented by the sequence, in decreasing order, of the values of I on its vertices. The two sets are organized based on the lexicographic ordering of such sequences. Auxiliary function `insert` is used to compute the corresponding sequence for any simplex σ and to insert σ in a set. Auxiliary function `delete` is used to extract the first simplex σ in the set, according to the lexycographic order. The two sets are initialized by cycling on the simplices in the input set (lines 4 to 8 of Algorithm 2). Auxiliary function `num_undeclared_facets` is used to count the number of unclassified facets for each simplex. Array `declared` keeps track of the simplices already classified (i.e., either paired or declared critical). At the beginning, all entries of `declared` are set to `false`.

Simplices are classified within the two nested while loops (lines 9 to 24). If `Ord1` is not empty, we extract the first simplex τ in `Ord1` and we verify if the number of unclassified facets of τ has not changed (line 12). Notice that the number of unpaired facets can only decrease. If this number is now zero (i.e., its facet has been classified), we insert τ into `Ord0`. Otherwise, we retrieve its unique unclassified facet σ (line 15), we use function `append` to add (σ,τ) to the set of pairs V_u, and we update the array `declared` accordingly. After classifying σ and τ, `add_cofacets` adds all their unclassified cofacets to either `Ord1` or `Ord0`, if they have one or zero unclassified facets, respectively (lines 18 and 19).

When no pairable simplex is available (i.e., `Ord1` is empty) the first simplex σ in `Ord0` is extracted and declared critical by adding σ to the set of critical simplices M_u (lines 21 to 23). Each cofacet of σ is added to `Ord1` if it has exactly one unclassified facet. The algorithm terminates when both lists are empty.

3.3. Complexity

In this section, we discuss the computational complexity of `ComputeDiscreteGradient` and its auxiliary functions. The parameters involved in the analysis are expressed in terms of cardinality $|\cdot|$ of sets. We indicate with $\mathrm{Star}(\sigma)$ the star of a simplex $\sigma \in S$, and simply with Star any star with maximal cardinality in the simplicial complex S. Notice that, in a d-dimensional simplicial complex, $|\mathrm{Star}|$ is not bounded from above by a constant and is possibly as large as $|S|$, this is not the case for more regular cell complexes like, for example, cubical complexes.

To simplify the runtime analysis and the exposition we make a few assumptions:

- for each simplex $\sigma \in S$, we assume $\mathrm{Star}(\sigma)$ to be computed and stored in the data structure encoding S. Computing $\mathrm{Star}(\sigma)$ on the fly would require $O(|\mathrm{Star}(\sigma)|)$ time [15].
- `Ord0`, `Ord1` are implemented as balanced binary search trees. Inserting, or removing an element from any such tree has a logarithmic cost in its size.

- For each k-simplex $\sigma \in S$, $f(\sigma)$ can be retrieved in $O(k)$ time by retrieving the filtration values of the vertices of σ. We will overestimate k by considering the dimension d of the simplicial complex S.

These assumptions are consistent with the implementation of `ComputeDiscreteGradient` used in our experimental evaluation (see Section 4.2.1).

3.3.1. Analysis of the auxiliary functions

Here, we present the time and storage costs of the auxiliary functions introduced in Section 3.2.

For creating the well-extensible indexing with `ComputeIndexing` we sort the vertices according to a single component of the input function. This requires $O(|S_0|\log|S_0|)$ time and $O(|S_0|)$ extra space for storing the new sorted list of vertices, where S_0 is the set of vertices of S.

The lower star of each vertex v is extracted by `ComputeIndexLowerStar`. The lower star of a vertex is extracted from the precomputed star $\mathrm{Star}(v)$ by selecting those simplices having v as the first vertex. This requires $O(|\mathrm{Star}(v)|)$ time.

Once a lower star is extracted, the level sets are created by means of `SplitIndexLowerStar`. Given a simplex σ, computing the value of each component by $f_i(\sigma) = \max_{v\in\sigma} f_i(v)$ takes linear time in the number of vertices of σ. Then, the filtration value of σ is computed in $O(dn)$ time, by overestimating the dimension of σ with the dimension d of the complex S. Searching for the set of simplices associated with a specific level set takes at most $O(\log|\mathrm{Low}_I(v)|)$ time. Thus, the overall time complexity of `SplitIndexLowerStar` is $C_{LS}(v) = O(|\mathrm{Low}_I(v)|(dn + \log|\mathrm{Low}_I(v)|))$.

In the third step, `HomotopyExpansion` classifies the simplices in each level set L_u. For each simplex σ in L_u, `num_undeclared_facets` requires visiting its facets whose number is limited from above by a constant and inserting each simplex in the set requires $O(\log|L_u|)$ time. Then, preparing `Ord1` and `Ord0` requires $O(|L_u|\log|L_u|)$ time.

Within the two while loops, each simplex is added to a list at most once and it is also classified once. Then, for each simplex σ:

- retrieving its facets (`num_undeclared_facets` or `unpaired_facets`) requires a constant number of operations,
- retrieving its cofacets (`add_cofacets`) takes at most $O(|L_u|)$ time as the number of cofacets is not limited from above by any constant number,
- adding the simplex σ to L_u takes $O(\log|L_u|)$ time.

Overall the time complexity of `HomotopyExpansion` is $C_{HE}(L_u) = O(|L_u|^2 + 2|L_u|\log(|L_u|)) = O(|L_u|^2)$.

3.3.2. Analysis of algorithm `ComputeDiscreteGradient`

By analyzing the complexity of all auxiliary functions we obtain a worst-case time complexity for algorithm `ComputeDiscreteGradient` of:

$$O\left(|S_0|\log|S_0| + \sum_{v\in S_0}\left(|\mathrm{Star}(v)| + C_{LS}(v) + \sum_{L_u\subseteq \mathrm{Low}_I(v)} C_{HE}(L_u)\right)\right).$$

In the internal summation, L_u can be as large as the entire index-based lower star. Thus, for any vertex v, we can estimate $O\left(\sum_{L_u\subseteq\mathrm{Low}_I(v)} C_{HE}(L_u)\right) = O(|\mathrm{Low}_I(v)|^2)$ time. Each k-simplex appears in the star of its $k+1$ vertices. Thus, we write $O\left(\sum_{v\in S_0}|\mathrm{Star}(v)|\right) = O(|S|(d+1))$ again by overestimating the dimension k of each simplex with the dimension d of the complex S.

Each simplex appears in exactly one index-based lower star. Thus, we can estimate the worst-case time complexity of `SplitIndexLowerStar` by $O\left(\sum_{v\in S_0} C_{LS}(v)\right) = O(|S|(d + \log(\max_{v\in S_0}|\mathrm{Low}_I(v)|)))$ time. For the same reason, we can estimate the worst-case time complexity for `HomotopyExpansion` by $O\left(\sum_{v\in S_0} C_{HE}(v)\right) = O(\max_{v\in S_0}|\mathrm{Low}_I(v)|^2)$. Moreover, we notice that $\max_{v\in S_0}|\mathrm{Low}_I(v)| \leqslant |\mathrm{Star}|$.

Based on these observations we can rewrite the overall worst-case time complexity as

$$O(|S_0|\log|S_0| + |S|(d + |\mathrm{Star}|^2)).$$

In general, the cardinality of a vertex star is not bounded from above. In practice its size becomes negligible with respect to the total number of simplices in S, i.e., when working with low dimensional simplicial complexes or 2D and 3D images. In those cases, we can consider d and $|\mathrm{Star}|$ to be constant factors which lead to a worst-case time complexity of $O(|S_0|\log|S_0| + |S|)$.

4. Comparison with the `Matching` algorithm and proof of correctness

The `Matching` algorithm introduced by Allili et al. [4] computes a discrete gradient on a multifiltration by using a global queue. In this Section, we compare algorithm `ComputeDiscreteGradient` to algorithm `Matching`. In Subsection 4.1,

we first describe algorithm `Matching`. In Subsection 4.2, we provide a formal comparison of the two approaches and, in Subsection 4.2.2, we prove their equivalence.

4.1. Globally computing a discrete gradient for multiparameters

The `Matching` algorithm [4,5] acts on a simplicial complex S and a function $f : S_0 \longrightarrow \mathbb{R}^n$ required to be component-wise injective on vertices, and extended to higher dimensional simplices, as defined in Section 3. The pair (S, f) defines a multifiltration of S obtained by sublevel sets. Additionally, the `Matching` algorithm requires an *indexing* J on S, i.e., an injective map $J : S \longrightarrow \mathbb{R}$. The indexing J has to satisfy the following property for every $\sigma \neq \tau \in S$,

$$\sigma \text{ is a face of } \tau \text{ or } f(\sigma) \lneq f(\tau) \quad \Rightarrow \quad J(\sigma) < J(\tau). \tag{2}$$

That is, J has to be compatible both with the coface partial order among simplices and with the value ordering under f.

The algorithm cycles on all simplices in S relying on a global queue. Simplices are processed according to increasing values of J. This makes unfeasible implementing the approach in parallel.

An auxiliary Boolean vector of length $|S|$, called `classified`, is initialized with all entries set to `false`. For each simplex σ, algorithm `Matching` checks whether σ is classified so that only non-classified simplices are processed. We denote by P the set of all simplices $\sigma \in S$, also called *primary simplices*, that are still unclassified when `Matching` starts processing their lower star. A non-classified simplex σ is passed to an auxiliary function extracting the lower star of σ with respect to f

$$\mathrm{Low}_f(\sigma) := \{\tau \in \mathrm{Star}(\sigma) \mid f(\tau) \preceq f(\sigma)\}.$$

Afterwards, `HomotopyExpansion` (pseudocode reported in Algorithm 2) is run with input $(\mathrm{Low}_f(\sigma), J)$ and returns a pair of lists $(V_{\mathrm{Low}_f(\sigma)}, M_{\mathrm{Low}_f(\sigma)})$. All entries in the auxiliary vector `classified` corresponding to the simplices in the two lists are set to `true`. The global output is obtained with the independent contributions of all pairs $(V_{\mathrm{Low}_f(\sigma)}, M_{\mathrm{Low}_f(\sigma)})$.

Theorem 9 in [5] proves that the union of all the discrete gradients, computed locally to the lower star $\mathrm{Low}_f(\sigma)$ of each simplex σ, forms a discrete gradient. Moreover, Proposition 8 in [5] proves that the same gradient is compatible with the input multifiltration (S, f). Specifically, Proposition 15 in [5] directly implies that for primary simplices $\sigma \in P$, lower stars $\mathrm{Low}_f(\sigma)$ form a partition of S.

4.2. Comparison of algorithms `Matching` and `ComputeDiscreteGradient`

In this section, we compare `ComputeDiscreteGradient` to `Matching` from three different standpoints: complexity, performance, and quality of output. We recall that P indicates the primary simplices in `Matching`, i.e., those simplices in S that were still unclassified when `Matching` processed their lower stars. Both `ComputeDiscreteGradient` and `Matching` build a discrete gradient by running `HomotopyExpansion` on a partition of the input complex S:

- `Matching` finds the discrete gradient over each lower star $\mathrm{Low}_f(\sigma)$ with σ a primary simplex of P,
- `ComputeDiscreteGradient` finds the discrete gradient independently over each level set L_u in a index-based lower star $\mathrm{Low}_I(v)$ with $v \in S_0$.

The two algorithms differ in the number of stars they compute and visit. `ComputeDiscreteGradient` computes a lower star for each vertex in the input complex. This means that, in our case, the exact number of stars visited by `ComputeIndexLowerStar` is $|S_0|$.

On the contrary, `Matching` computes a lower star for each primary simplex in P. In the best case, the primary simplices P are exactly the vertices of S, i.e., $|P| = |S_0|$. Then, the two algorithms compute the same number of lower stars. In the worst case, the number of primary simplices $|P|$ is equal to the total number of simplices $|S|$, which greatly affects the performances of `Matching`, as shown in Subsection 4.2.1.

4.2.1. Performance comparison

We recall that the input of both algorithms is described as a pair (S, f), where S is a simplicial complex and $f : S_0 \longrightarrow \mathbb{R}^n$ is a component-wise injective function. Here, we focus on the case where S is a triangle mesh embedded in the Euclidean 3D space and f assigns to each vertex its x and y coordinates (i.e., for $v = (x, y, z)$, $f(v) = (x, y)$). The algorithm in [5] is defined to work in combination with a data structure that encodes all the simplices of a simplicial complex S. This approach does not scale well when the size and the dimension of S increase. The only way to guarantee scalability is that of encoding S with an indexed-based representation based on its top simplices [38]. For a fair comparison, both algorithms have been implemented by using the *FG_Multi library* [35] which provides a compact encoding for the triangle mesh as well as for the discrete gradient.

Table 1
Datasets used for the experiments. For each of them, we indicate the number of independent parameters in the multifiltration (column *Parameters*), the number of simplices in the original dataset (column *Original*), number of critical simplices retrieved by `ComputeDiscreteGradient` and `Macthing` (column *Critical*) and the compression factor (column *Original/Critical*).

Dataset	Parameters	Simplices		Compression factor
		Original	Critical	Original/Critical
Torus	2	1.3M	0.035M	37.9
		5.3M	0.11M	45.3
		21.5M	0.77M	27.7
		2.9M	0.28M	10.2
Sphere	2	11.7M	0.11M	10.2
		47.1M	0.46M	10.1
		3.8M	0.4M	9.5
Gorilla	2	15.2M	1.6M	9.4
		60.9M	6.4M	9.4

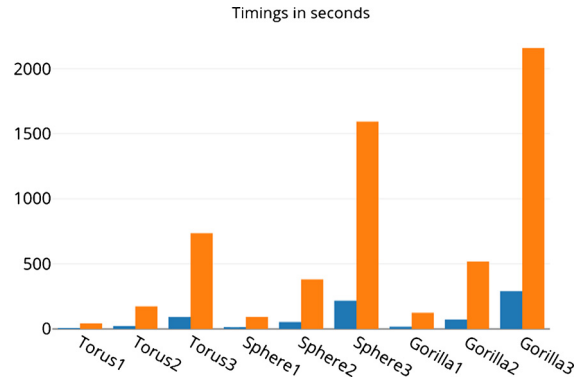

Fig. 4. Timings (in seconds) required by `ComputeDiscreteGradient` (in blue) and `Matching` (in orange).

Simplicial complex representation A triangle mesh S is a simplicial complex of dimension two formed by vertices, edges, and triangles. The *FG_Multi library* implements an incidence-based data structure for compactly encoding the relations among these simplices. Vertices and triangles are the only simplices that are explicitly encoded for a total of $|S_0| + |S_2|$ entities. Each vertex encodes the list of triangles incident in it, while each triangle encodes a reference to its three vertices. Thus, since each triangle references three vertices, the triangle-vertex relation costs $3|S_2|$ while encoding also the vertex-triangle relation doubles this cost leading to a total of $7|S_2| + |S_0|$. The filtering function f is stored with each vertex encoding a vector of floating point values, one value for each component of f.

Discrete gradient representation The discrete gradient V is encoded by adopting the representation described in [36]. The latter focuses on encoding all the gradient pairs locally to each triangle. The encoding uses the following rationale. Since each triangle can be paired with at most three edges, and each edge can be paired with two vertices, we have 9 possible pairs for each triangle. If we consider also the pairs between an edge of σ and an adjacent triangle, we get 12 possible gradient pairs and, thus, $2^{12} = 4096$ possible combinations. However, a discrete gradient imposes certain restrictions, i.e., each simplex can be involved in at most one pairing. As a consequence, we have only 97 valid cases for a triangle. These cases can be encoded using only one byte per triangle and, thus, encoding the gradient only requires $|S_2|$ bytes. This approach has been generalized to tetrahedral meshes [62] and to d-dimensional simplicial complexes [38].

The datasets used in this comparison are originated by three triangle meshes. The experiments have been performed on a dual Intel Xeon E5-2630 v4 CPU at 2.20 GHz with 64 GB of RAM. For each mesh, we have produced two refined versions at increased resolution by recursively applying Catmull-Clark algorithm [19]. The nine triangle meshes are presented in Table 1. Column *Original* indicates the number of simplices composing the mesh. Column *Critical* indicates the number of unpaired (critical) simplices identified by both reduction approaches, while the resulting compression factor is reported in column *Original/Critical*. Timings are shown in Fig. 4. `ComputeDiscreteGradient` takes between 0.89 seconds and 4.8 minutes depending on the dataset and it is generally 7 times faster than our efficient implementation of the `Matching` algorithm. Time performances show the practical efficiency of the local approach compared with the global one. As discussed in Section 3.3, both algorithms have linear time complexity when the number of simplices within each star is negligible. In Fig. 5(left), we show the trends as the number of simplices increases. This confirms that the number of stars to be retrieved and visited has a direct consequence on the algorithm performance. In `ComputeDiscreteGradient` we need to process each vertex star only once, while `Matching` requires processing a star for each grade.

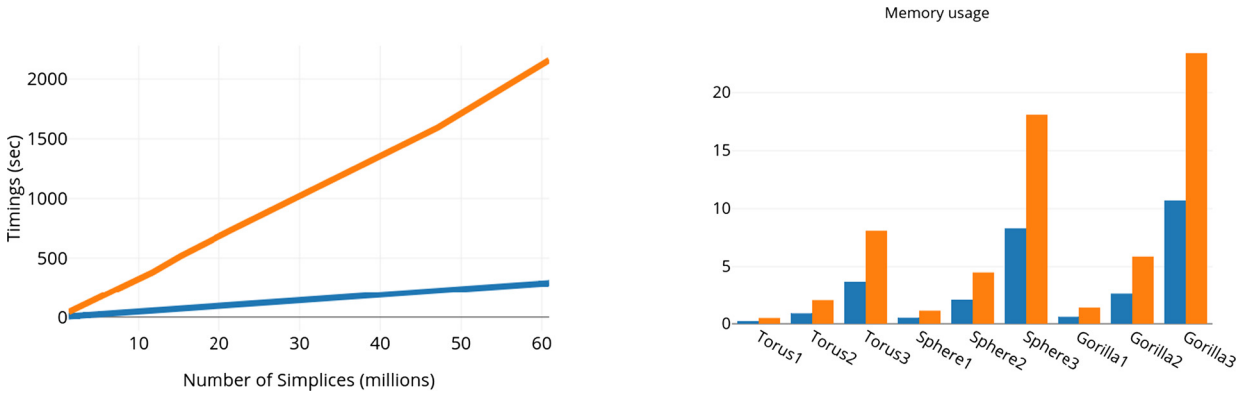

Fig. 5. On the left, timings (in seconds) required by `ComputeDiscreteGradient` (in blue) and `Matching` (orange). On the right, maximum peaks of memory (in gigabytes) required by `ComputeDiscreteGradient` (in blue) and `Matching` (in orange).

Other than time efficiency, our strategy also requires less memory. In Fig. 5(right), we report the maximum peak of memory used by the two algorithms. `ComputeDiscreteGradient` uses up to 10 gigabytes of memory versus more than 20 gigabytes used by `Matching`. The storage cost difference between the two implementations grows linearly with the number of simplices in the datasets. The advantage is due to the different memory consumption at runtime. Specifically, `Matching` stores a global queue over all the simplices in the dataset and needs to track all classified simplices. `ComputeDiscreteGradient` performs these steps over each level set.

4.2.2. Equivalence of the outputs

We now show that `ComputeDiscreteGradient` and `Matching` produce the same gradient when the indexing I used in `ComputeDiscreteGradient` is well-extensible and the filtering function f is componentwise injective on the vertices. These assumptions will be maintained throughout this section. Here, we first state the results required to draw such conclusions, while the detailed proofs can be found in Appendix A.

The proof consists of two parts. First, we show that the two algorithms apply `HomotopyExpansion` to the same partition of the input simplicial complex. Then, given for one algorithm any valid order for processing the simplices in `HomotopyExpansion` we prove that a valid order for the other algorithm exists such that the two provide the same output. We start by proving that the partitioning strategies of `ComputeDiscreteGradient` and `Matching` are the same. In `ComputeDiscreteGradient` we partition S according to the level sets L_u of f in $\mathrm{Low}_I(v)$, for all vertices v and grades u. In `Matching` S is partitioned according to $\mathrm{Low}_f(\sigma)$, with σ varying in the set of primary simplices P, i.e. simplices that do not belong to the lower star with respect to f of any other simplex.

We start showing that the partition provided by sets $\mathrm{Low}_f(\sigma)$, with σ in P, is a refinement of the one provided by the sets $\mathrm{Low}_I(v)$ with $v \in S_0$.

Lemma 1. *For every $\sigma \in S$, and, hence, in particular for $\sigma \in P$, there is exactly one vertex $v \in S_0$ such that $\mathrm{Low}_f(\sigma) \subseteq \mathrm{Low}_I(v)$.*

Next, we show that, for every vertex v, any level set L_u of the filtering function f restricted to $\mathrm{Low}_I(v)$ coincides with the lower star $\mathrm{Low}_f(\sigma)$ of some primary simplex σ.

Lemma 2. *Let L_u be a non-empty level set of f in $\mathrm{Low}_I(v)$ for some grade $u \in \mathbb{R}$ and some vertex $v \in S_0$. There exists a unique simplex $\sigma \in L_u$ such that $\mathrm{Low}_f(\sigma) = L_u$. Moreover, $\sigma \in P$.*

Lemmas 1 and 2 allow us to conclude that both `ComputeDiscreteGradient` and `Matching` run `HomotopyExpansion` on the same subsets $L_u = \mathrm{Low}_f(\sigma)$. Now we need to show that `HomotopyExpansion` does so by processing the simplices of these subsets in the same order.

Proposition 3. *For every valid input (S, f) for `ComputeDiscreteGradient`, there exists an indexing $J : S \longrightarrow \mathbb{R}$ valid for `Matching` such that the output of `Matching`(S, f, J) and `ComputeDiscreteGradient`(S, f) coincide.*

As a consequence of this result, from the correctness of `Matching`, we obtain the correctness of `ComputeDiscreteGradient`.

Corollary 4. *Algorithm `ComputeDiscreteGradient` with input (S, f) returns a discrete gradient compatible with the multifiltration induced by f on S.*

Table 2
Timings (in seconds) and storage costs (in gigabytes) for the persistence module retrieval over the original triangulation (columns *Original*) and the corresponding Morse complex (columns *Morse*). Columns *Simplices* and *Grades* report the number of simplices in the dataset and the number of level sets along each parameter, respectively. Missing entries indicate where the Topcat library runs out of memory. Column *Time* explicits the timings (in seconds) for obtaining the Morse complex using `ComputeDiscreteGradient`.

Dataset	Original				Morse				Time
	Simplices	Grades	Persistence module		Critical simplices	Grades	Persistence module		
			Time	Memory			Time	Memory	
Sphere	38	8×8	0.3	0.24	4	5×5	0.18	0.1	0.0264
	242	42×42	4.4	0.86	20	10×10	0.28	0.2	0.0244
	2882	482×482	–	–	278	92×89	24.3	1.5	0.0473
Torus	96	16×16	0.5	0.1	8	9×9	0.25	0.2	0.0255
	4608	768×768	–	–	128	65×66	7.96	2.4	0.0643
	7200	1200×1200	–	–	156	70×80	12.05	3.0	0.0815

We recall from Section 1.4 that a discrete gradient V implicitly represents a Morse complex consisting only of the critical simplices of V and with the incidence relations given by the separatrices originating and having destination in a pair of critical simplices. Then, we have proved that our algorithm implicitly computes a Morse complex that has the same multiparameter persistent homology of the original complex.

5. Computing multiparameter persistence homology

In this section, we evaluate the impact of our algorithm as a preprocessing method for the computation of the persistence module (see Subsection 5.1) and of the persistence space (see Subsection 5.2). Before that, we describe how to compute the Morse complex represented by the discrete gradient.

Computing the Morse complex We process all the critical simplices of V and, for each critical simplex σ of dimension k, a breadth-first traversal is performed as follows. For each $(k-1)$-simplex τ_1, facet of σ, we extract its paired k-simplex σ_1, if any. We apply the same rationale to σ_1 to continue the visit. As soon as we encounter a $(k-1)$-simplex τ which is unpaired (critical), we classify the two simplices σ and τ as mutually incident.

In the worst case, computing the incidences for a single critical simplex of dimension k requires visiting all k-simplices multiple times, in the order of $O(|S_k|^2)$, where S_k is the set of k-simplices in S. If the number of critical simplices is of the same order as $|S_k|$ the total worst-case complexity would be cubical in the number of simplices. In practical cases, however, the extraction of the Morse complex is very efficient as each k-simplex belongs to a very limited set of gradient paths, possibly zero.

5.1. Computing the persistence module

The persistence module is computed by means of the open-source library *Topcat* [56]. Due to its strong limitations in terms of time and memory costs, we used simplified datasets for our experiments. We use six triangle meshes of limited size, three representing a torus and three representing a sphere. For each mesh, we use a bifiltration defined by the x and y coordinates of its vertices. Table 2 presents a description of the input datasets. The number of simplices (column *Simplices*) and number of multigrades (column *Grades*) are reported for each simplicial complex (column *Original*) and each Morse complex (column *Morse*). Notice that for a Morse complex M, column *Critical Simplices* indicates the number of critical simplices in the discrete gradient V which is also equivalent to the number of cells in the corresponding Morse complex M. The *Topcat* library uses the boundary matrices of the complex to compute the persistence module. Since it was designed to accept only multifiltrations defined on simplicial complexes, we have modified the library for working on multifiltrations defined over more general cell complexes, like the Morse complex.

We compute the persistence module, for each homology grade, of both the original simplicial complex and the Morse complex generated by our algorithm, and we measure time and storage consumption of the *Topcat* library. We report the results obtained in Table 2, columns *Time* and *Memory*. These represent the timings (in seconds) and the memory (in Gigabytes) required for computing the persistence module. When the *Topcat* library runs out of memory, no result is reported. Where a comparison is possible, computing the persistence module on the Morse complex takes approximately half of the time than computing it on the original simplicial complex.

The memory consumption is the main bottleneck of the *Topcat* library as it is mainly affected by the number of input cells and by the number of multigrades. Using the Morse complex helps to deal with this problem by reducing both quantities. In our experiments, all successful executions have used a limited amount of memory, significantly below the machine limit of 64 GB. This suggests a dramatic increase in memory usage in the ones where the computations have failed. For instance, the failure of the test over the Sphere dataset with 2882 cells and 482×482 multifiltration multigrades suggests that computing the persistence module on a Morse complex of the same size would fail as well. We should stress the fact that the objective of our experiment is that of evaluating the gain in performances when using our reduction approach and

Table 3
Timings (in seconds) required for computing the persistence pairs on 100 uniformly sampled slices. Datasets are reported by rows. For each triangle mesh, the first row is for the original dataset and the second one for the Morse complex considered over the same 100 slices. Column *Simplices* reports the number of simplices in the multifiltration. Column *Pairs* reports the average number of persistence pairs found per slice.

Dataset	Simplices	Pairs	Morse time	Line extraction	Foliations time			
					Building pers. input	Computing persistence	Reindexing pers. output	Foliations total
Shark	9491	4744	0.11	0.86	9.04	1.91	1.45	12.42
	1111	554			1.15	0.21	0.84	2.22
Turtle	10861	5426	0.12	0.63	10.21	2.11	1.53	13.87
	1197	594			1.22	0.22	0.84	2.29
Gun	27826	13873	0.28	0.65	27.49	5.65	2.69	35.85
	3144	1532			3.18	0.60	0.99	4.77
Piano	119081	59349	1.14	0.85	118.14	26.56	10.33	155.91
	10955	5286			11.09	2.26	1.65	15.01

not that of overcoming the limitations of *Topcat*. Column *Morse Time* reports the partial timings required for computing the Morse complex. These include the contribution of running `ComputeDiscreteGradient` together with the retrieval of the boundary matrix. We point out that the reduction timings, ranging from 0.0244 to 0.0815 seconds, are negligible with respect to the time required for computing the persistence module.

5.2. Computing the persistence space

In this subsection, we evaluate the impact of the reduction method on the computation of the *persistence space* [22].

The foliation method The persistence space of an n-parameter filtration is a subset of $\mathbb{R}^n \times \mathbb{R}^n$ that can be computed via the *foliation method* [10]. This consists of considering all possible lines ℓ in $\mathbb{R}^n$ through two grades $u \prec v$. By restricting the n-parameter filtration to the grades belonging to ℓ, we obtain a one-parameter filtration, called a *slice*. On each slice, any technique for one-parameter persistent homology computation can be applied to obtain the corresponding persistence diagram [29]. A *persistence diagram* is a representation, on the Cartesian plane, of a one-parameter persistence module. Each point in the persistence diagram represents a discontinuity in the rank invariant created by either a new-born or a vanishing homology class. The union of the persistence diagrams for all possible lines ℓ mapped back to $\mathbb{R}^n \times \mathbb{R}^n$ gives the persistence space. In practice, it is possible to compute only a sampling of the persistence space by selecting a finite number of lines. The number of lines to consider varies based on the application at hand. As a rule of thumb, the more slices we consider, the more accurate is the approximation of the resulting persistence space. Next, we describe our implementation of the foliation method.

We applied the foliation method on bifiltrations induced by some function $f : S \to \mathbb{R}^2$. The first step consists in selecting ω^2 lines ℓ with a non-negative slope in $\mathbb{R}^2$. Since each line ℓ with positive slope in $\mathbb{R}^2$ is determined by the angle λ of the line with the x axis, and a base point b on ℓ, we have selected ω values for both λ and b. Values of λ are uniformly taken from interval $[0, \frac{\pi}{2}]$. Points b are uniformly taken from the segment whose endpoints are the projections of points (c_1, C_2) and (C_1, c_2), with $C_i := \max_{x \in S} f_i(x)$ and $c_i := \min_{x \in S} f_i(x)$ for $i = 1, 2$, onto the bisector of the second and fourth quadrant along the direction $m = (\cos(\lambda), \sin(\lambda))$. For each choice of λ and b, we thus determine the line ℓ passing through b with direction $m = (\cos(\lambda), \sin(\lambda))$. By varying the values of λ and b, we obtained the desired ω^2 possible lines. The second step creates a new one-parameter filtration on S for each line ℓ. If ℓ has parameters λ, b, for each simplex $\sigma \in S$, the grade of σ in the new filtration is given by $\Phi^\ell(\sigma) := \min_{i=1,2} m_i \cdot \max_{i=1,2} \frac{f_i(\sigma) - b_i}{m_i}$. The last step consists of computing classic persistent homology for the obtained one-parameter filtration by sublevel sets of Φ^ℓ corresponding to each choice of λ and b.

After choosing how to sample slices, the foliation method still requires choosing the number of slices. In what follows, we will present results providing insights on both choices, either by varying the number of slices (between 2 and 100), or by varying the method for computing persistent homology (taken from the PHAT library [6]).

Here we present results for evaluating the impact of our reduction approach when computing the persistence space by using a variable number of slices (between 2 and 100). The meshes considered are from the Princeton Shape Benchmark [58]. Table 3 describes the meshes and the results obtained when computing the persistence space by using 100 slices and by using the standard algorithm implemented in PHAT [6]. For each mesh reported in Table 3, the first row reports data regarding the original mesh, while the second row describes the corresponding Morse complex computed by using our reduction method. For each input complex, we show the number of simplices (column *Simplices*) and the average number of persistence pairs found per slice (column *Pairs*). Timings are reported separately for the computation of the Morse complex (column *Morse time*), for the extraction of slices (column *Line Extraction*) and for the actual computation of the persistence space (column *Foliation Time*). The latter accounts for the construction of the boundary matrix (column *Building Pers. input*), for the computation of persistent homology (column *Computing Persistence*), and for reindexing the persistence pairs according to the multifiltration (column *Reindexing Pers. output*). Column *Foliation Total* shows the sum of the partial timings.

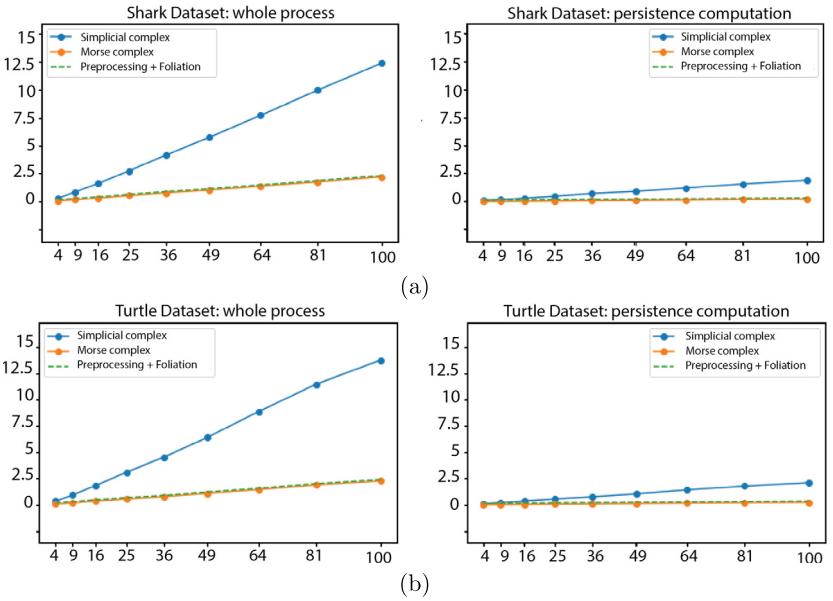

Fig. 6. Time performances (in seconds) plotted with respect to a number of slices varying from 4 to 100 over the same dataset. Datasets considered are triangle meshes: (a) Shark and (b) Turtle. In all the figures performances are indicated in blue for the original dataset and in orange for the corresponding reduced dataset. On the left we show timings required by the foliation method. On the right, we show timings for computing persistence homology over a single slice.

We notice that, by reducing the number of simplices of approximately one order of magnitude, we get a one-order reduction on all timings. Looking at column *Line Extraction* we notice that the time required for extracting the lines is almost the same for all the meshes. This happens because we are always considering the same number of slices. For the partial timings, the highest contribution is shown in column *Building Pers. Input*. This is the part where the cells are sorted by increasing values under Φ^{ℓ} and reindexed according to these values. Both this phase and the following one (i.e., the actual computation of persistent homology) are affected by the number of input simplices, and the results for the Morse complex reflect the one-order reduction in the number of simplices. Results are shown in column *Reducing Pers. Output*. The difference in the results obtained with the original triangle mesh and the corresponding Morse complex suggests that our reduction step lets us avoid computation on many spurious persistence pairs. Column *Foliation Total* indicates timings for computing the persistence space as a whole. The total timings required by a Morse complex range from a minimum of 2.22 seconds (Shark triangle mesh) to a maximum of 15.01 seconds (Piano triangle mesh), whereas, the original datasets require from 12.42 (Shark triangle mesh) to 155.91 seconds (Piano triangle mesh).

In Figs. 6 and 7, we compare the time performances of the foliation method when using a number of slices ranging from 4 to 100. Over each slice, one-parameter persistence is computed by the standard algorithm implemented in PHAT. For each dataset, we show, on the left, the global timings for the foliation phase and, on the right, the partial timings required by the computation of persistent homology.

Blue lines indicate results obtained for the triangle meshes, green dotted lines present results obtained with the Morse complexes accounting for both the reduction algorithm and the foliation step. Orange lines indicate results obtained with the Morse complexes exclusively for the foliation phase. As we can see, orange and green lines almost overlap indicating that the time used for computing the Morse complex is almost negligible with respect to that required to compute the persistence space.

We also notice the linear dependency on the number of slices. For Morse complexes (orange line), the slope coefficient is smaller than for the original simplicial complex (blue line). This is more evident for global timings suggesting that a preprocessing reduction is preferable independently of the number of slices considered. Notice that, the blue line is below the green dashed line only when we are using 4 slices. This is the only case when the preprocessing step could be avoided.

Our tests confirm that the complexity of the foliation method primarily depends on the number of slices considered. Our reduction approach impacts the performances by simply reducing the number of cells to be processed. Moreover, our tests show that the proposed preprocessing is effective also for a small number of slices.

6. Concluding remarks

We have proposed a new preprocessing algorithm for multiparameter persistent homology suitable for applications to large data sets. We have highlighted the local characteristics of our approach as opposed to the global characteristics of the

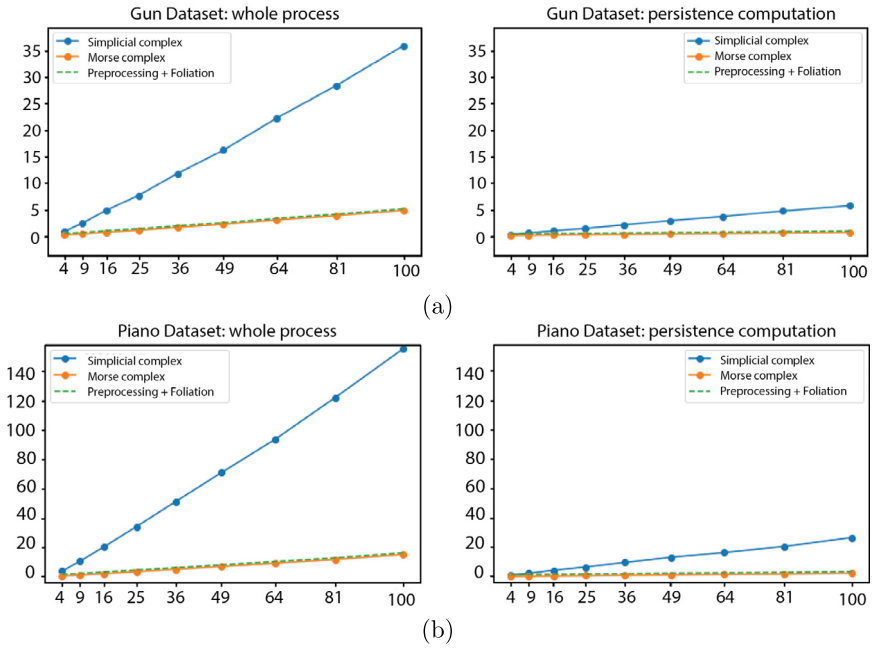

Fig. 7. Time performances (in seconds) plotted with respect to a number of slices varying from 4 to 100 over the same dataset. Datasets considered are triangle meshes: (a) Gun and (b) Piano. In all the figures performances are indicated in blue for the original dataset and in orange for the corresponding reduced dataset. On the left we show timings required by the foliation method. On the right, we show timings for computing persistence homology over a single slice.

equivalent existing approach by Allili et al. [5]. With real data, the use of the Morse complex allowed us to successfully compute multiparameter persistent homology. We increased about 50 times the size of the input complex that can be managed, and up to about 250 times the size of the filtration that can be treated. Our local preprocessing has shown its advantages, especially when computing the persistence space [21] through the foliation method. We found that the Morse complex outperforms the corresponding original filtration, regardless of the number of slices used in the foliation method. By using the Morse complex for computing the persistence module we still have experienced efficiency problems, even with small datasets. This suggests that our preprocessing is not powerful enough to make the persistence module computation feasible and that optimizations of current algorithms require further improvements. As new algorithms for computing the persistent module arise, such as the approach by Dey and Xin [26], our preprocessing approach may prove its importance for computing multipersitent homology.

We think that the idea of a discrete gradient compatible with a multifiltration deserves further investigations. Currently, we are expanding the set of visual features that can be extracted from a discrete gradient for data analysis and visualization by studying the relationships between the critical simplices identified by our method and the *multigraded Betti numbers* computed by RIVET [49]. This may help us constructing a bridge between the discrete notions of critical simplices and the piecewise linear notions of Pareto sets [44] and Jacobi sets [31].

Also, we plan to evaluate the scalability of our approach to higher dimensional simplicial complexes. So far, the alternative approach by Fugacci and Kerber [39] has shown its advantages when working on triangle meshes. However, the algorithm requires a global representation of the facets of each simplex while our approach only requires the encoding of the top simplices only. While this type of encoding has its drawbacks in lower dimensions, as shown by the timings in [39], it has been proven by Fugacci et al. [38] that it provides better scalability when the dimension of the simplicial complex increases.

A current limitation of our approach is that of requiring a componentwise injective function. From this property, the algorithm's correctness (see Section 4) and its time complexity follow (see Section 3.3). Moreover, it guarantees the minimality of the critical simplices this algorithm identifies, proved by Landi and Scaramuccia in [48]. It would be interesting for future work to extend our results to functions that are not componentwise injective or defined on all the simplices of the simplicial complex.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been partially supported by the US National Science Foundation under grant number IIS-1910766. The authors wish to thank Michael Kerber and Ulderico Fugacci for interesting discussion on the results.

Appendix A

In this appendix, we report all the proofs of the results provided in Subsection 4.2.2.

Proof of Lemma 1. Let $\sigma \in S$. Because the sets $\mathrm{Low}_I(v)$, with $v \in S_0$, form a partition of S, there is a unique vertex v of S such that σ belongs to $\mathrm{Low}_I(v)$. By definition of lower star, v is a face of σ, and $I(\sigma) \leqslant I(v)$. Because I is extended from the vertices to other simplices by taking the maximum over all vertices, it also holds that $I(v) \leqslant I(\sigma)$, and hence $I(v) = I(\sigma)$.

Let τ be a simplex in $\mathrm{Low}_f(\sigma)$. Again by definition of lower star and because f is defined by max-extension as well, we similarly get that σ is a face of τ and $f(\tau) = f(\sigma)$. Since I is well-extensible, this implies that $I(\tau) = I(\sigma)$, and hence $I(\tau) = I(v)$. Because $v < \sigma < \tau$ we can thus conclude that $\tau \in \mathrm{Low}_I(v)$. Therefore, $\mathrm{Low}_f(\sigma) \subseteq \mathrm{Low}_I(v)$. □

Proof of Lemma 2. In order to prove uniqueness, assume by contradiction that there are two simplices $\sigma, \sigma' \in S$ such that $\mathrm{Low}_f(\sigma) = L_u = \mathrm{Low}_f(\sigma')$. Hence, $\sigma' \in \mathrm{Low}_f(\sigma)$ and $\sigma \in \mathrm{Low}_f(\sigma')$, implying that σ' is a coface of σ and σ is a coface of σ'. Thus, $\sigma' = \sigma$.

In order to prove existence, we take σ to be the maximal common face of all simplices of L_u. Since $v \in \tau$ for every $\tau \in L_u$, σ is non-empty and belongs to $\mathrm{Low}_I(v)$. In particular, σ belongs to L_u. Indeed, on one hand, for every τ in L_u, $f(\sigma) \preceq f(\tau) = u$, because f does not decrease with the coface relation. On the other hand, since f is obtained by max-extension from the vertices of a component-wise injective function, for every $i = 1, \ldots, n$ there is a unique vertex w_i such that $u_i = f_i(w_i)$. Hence, for all $\tau \in L_u$, it holds that $u_i = f_i(\tau) \geqslant f_i(w_i) = u_i$, implying that w_i is a vertex of τ for all $\tau \in L_u$. By definition of σ, w_i is also a vertex of σ. Therefore, $f_i(\sigma) \geqslant f_i(w_i) = u_i$. Because this holds for every $i = 1, \ldots, n$, we deduce that $u \preceq f(\sigma)$, which together with $f(\sigma) \preceq u$ yields $f(\sigma) = u$. In conclusion, σ belongs to L_u.

We now claim that $\mathrm{Low}_f(\sigma) = L_u$. We easily see that L_u is contained in $\mathrm{Low}_f(\sigma)$ because, by definition of σ, all simplices $\tau \in L_u$ are cofaces of σ, and by definition of level set, $f(\sigma) = f(\tau) = u$. Conversely, to prove that $\mathrm{Low}_f(\sigma) \subseteq L_u$, let τ be a simplex in $\mathrm{Low}_f(\sigma)$. Lemma 1 ensures that $\tau \in \mathrm{Low}_I(v)$, so it is sufficient to prove that $f(\tau) = u$. We have already proved that $f(\sigma) = u$, hence the claim follows by using again the fact that, for all $\tau \in \mathrm{Low}_f(\sigma)$, $f(\tau) = f(\sigma)$.

To conclude the proof, it remains to show that σ is a primary simplex for `Matching`. Recall that a simplex is primary, i.e. $\sigma \in P$, if and only if there is no other simplex τ such that $\sigma \in \mathrm{Low}_f(\tau)$. By contradiction, let us assume such simplex τ exists. Hence $\mathrm{Low}_f(\sigma) \subseteq \mathrm{Low}_f(\tau)$. By Lemma 1, σ and τ belong to the same index-based lower star $\mathrm{Low}_I(v)$. Since $\sigma \in \mathrm{Low}_f(\tau)$, we get $u = f(\sigma) = f(\tau)$. Thus, $L_u = \mathrm{Low}_f(\sigma) \subseteq \mathrm{Low}_f(\tau) \subseteq L_u$, implying that $\mathrm{Low}_f(\sigma) = \mathrm{Low}_f(\tau)$, and hence $\sigma = \tau$. □

Proof of Proposition 3. To prove the claim, we show how to construct an indexing J on S valid as an input of `Matching` such that each call of `HomotopyExpansion` with simplices of $\mathrm{Low}_f(\sigma) = L_u$ taken in the order of J gives the same result as `ComputeDiscreteGradient`.

For each $\sigma \in P$, we indicate by J_σ the indexing on simplices of $\mathrm{Low}_f(\sigma) = L_u$ built in `HomotopyExpansion`(S, I, L_u) when called by `ComputeDiscreteGradient`. By construction, it is increasing with the coface relation, and consistent with the orderings of the lists `Ord0` and `Ord1`.

Let $g : P \longrightarrow \mathbb{R}$ be any injective function compatible with f, i.e., $f(\sigma) \preceq f(\tau)$ implies $g(\sigma) \leqslant g(\tau)$ (obtained, for example, by topological sorting). For every simplex $\tau \in S$, we can consider the map $Q : S \to P$ such that $Q(\tau) = \sigma$, with σ the unique primary simplex $\sigma \in P$ such that $\tau \in \mathrm{Low}_f(\sigma)$. Thus, we can extend g from P to S by taking $G = g \circ Q$. By construction, G is still compatible with f.

Therefore, for any simplex $\tau \in S$, we get a pair of real numbers $(G(\tau), J_{Q(\tau)}(\tau))$. The set of all such pairs can be lexicographically ordered, and we can finally take $J : S \longrightarrow \mathbb{R}$ to be an injective map giving a total order equivalent to such lexicographic order.

We need to show that J is a valid ordering for `Matching`, that is it satisfies condition (2). If $\sigma < \tau$, we have $f(\sigma) \preceq f(\tau)$ because f is defined by max-extension from the vertices. By compatibility of G with f, this implies that $G(\sigma) \leqslant G(\tau)$. In the case $G(\sigma) < G(\tau)$, by the equivalence of J with the lexicographic order on all the pairs $(G(\tau), J_{Q(\tau)}(\tau))$ with $\tau \in S$, we get $J(\sigma) < J(\tau)$. In the case $G(\sigma) = G(\tau)$, by injectivity of g, it follows that $Q(\sigma) = Q(\tau)$. Because $J_{Q(\sigma)}$ is increasing with the coface relation, we get $J(\sigma) < J(\tau)$. Hence, in any case, $\sigma < \tau$ implies $J(\sigma) < J(\tau)$. Let us now assume that $f(\sigma) \precnsim f(\tau)$. From $f(\sigma) = f \circ Q(\sigma)$ and $f(\tau) = f \circ Q(\tau)$, it follows that $f \circ Q(\sigma) \precnsim f \circ Q(\tau)$. Necessarily, $Q(\sigma) \neq Q(\tau)$. Thus, by injectivity of g, $g \circ Q(\sigma) < g \circ Q(\tau)$. Because $G = g \circ Q$, we conclude that $J(\sigma) < J(\tau)$ by equivalence of J with the lexicographic order on the pairs considered above. Therefore, we have proved that J satisfies condition (2) and is a valid ordering for `Matching`. In conclusion, by Lemma 2, `Matching` and `ComputeDiscreteGradient` call `HomotopyExpansion` with the same input and, provided that `Matching` uses the ordering J, they also process simplices in the same order. Hence, `Matching` and `ComputeDiscreteGradient` produce the same output. □

Proof of Corollary 4. By Proposition 3, for any valid input (S, f) for `ComputeDiscreteGradient` there is a valid input (S, f, J) for `Matching` such that `ComputeDiscreteGradient`(S, f) is equal to `Matching`(S, f, J). By Proposition 8 and Theorem 9 in [5], `Matching`(S, f, J) returns a discrete gradient V that is compatible with the multifiltration induced by the pair (S, f). Therefore, so does `ComputeDiscreteGradient`(S, f). □

References

[1] H. Adams, G. Carlsson, On the nonlinear statistics of range image patches, SIAM J. Imaging Sci. 2 (1) (Jan. 2009) 110–117.
[2] M.K. Agoston, Computer Graphics and Geometric Modeling: Mathematics, Springer Verlag London Ltd., 2005.
[3] M. Allili, T. Kaczynski, C. Landi, Reducing complexes in multidimensional persistent homology theory, J. Symb. Comput. 78 (C) (2017) 61–75.
[4] M. Allili, T. Kaczynski, C. Landi, F. Masoni, Algorithmic construction of acyclic partial matchings for multidimensional persistence, in: W. Kropatsch, N. Artner, I. Janusch (Eds.), Discrete Geometry for Computer Imagery, DGCI 2017, in: Lecture Notes in Computer Science, vol. 10502, Springer, Cham, 2017, pp. 375–387.
[5] M. Allili, T. Kaczynski, C. Landi, F. Masoni, Acyclic partial matchings for multidimensional persistence: algorithm and combinatorial interpretation, J. Math. Imaging Vis. 61 (2) (Feb. 2019) 174–192.
[6] U. Bauer, M. Kerber, J. Reininghaus, Persistent Homology Algorithm Toolbox (PHAT), 2013.
[7] U. Bauer, M. Kerber, J. Reininghaus, Clear and compress: computing persistent homology in chunks, in: P.-T. Bremer, I. Hotz, V. Pascucci, R. Peikert (Eds.), Topological Methods in Data Analysis and Visualization III, Springer International Publishing, Cham, 2014, pp. 103–117.
[8] P. Bendich, J.S. Marron, E. Miller, A. Pieloch, S. Skwerer, Persistent homology analysis of brain artery trees, Ann. Appl. Stat. 10 (1) (Mar. 2016) 198–218.
[9] S. Biasotti, A. Cerri, P. Frosini, D. Giorgi, A new algorithm for computing the 2-dimensional matching distance between size functions, Pattern Recognit. Lett. 32 (14) (2011) 1735–1746.
[10] S. Biasotti, A. Cerri, P. Frosini, D. Giorgi, C. Landi, Multidimensional size functions for shape comparison, J. Math. Imaging Vis. 32 (2) (2008) 161–179.
[11] S. Biasotti, A. Cerri, D. Giorgi, M. Spagnuolo, PHOG: photometric and geometric functions for textured shape retrieval, Comput. Graph. Forum 32 (5) (2013) 13–22.
[12] J.D. Boissonnat, T.K. Dey, C. Maria, The compressed annotation matrix: en efficient data structure for computing persistent cohomology, in: Algorithms - ESA 2013, in: Lecture Notes in Computer Science, vol. 8125, Springer, Berlin/Heidelberg, 2013, pp. 695–706.
[13] O. Busaryev, S. Cabello, C. Chen, T.K. Dey, Y. Wang, Annotating simplices with a homology basis and its applications, in: F.V. Fomin, P. Kaski (Eds.), Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 7357, Springer, 2012, pp. 189–200.
[14] F. Cagliari, B. Di Fabio, M. Ferri, One-dimensional reduction of multidimensional persistent homology, Proc. Am. Math. Soc. 138 (08) (2010) 3003–3017.
[15] D. Canino, L.D. Floriani, K. Weiss, IA* an adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions, Comput. Graph. 35 (3) (2011) 747–753.
[16] G. Carlsson, T. Ishkhanov, V. de Silva, A. Zomorodian, On the local behavior of spaces of natural images, Int. J. Comput. Vis. 76 (1) (Jan. 2008) 1–12.
[17] G. Carlsson, G. Singh, A. Zomorodian, Computing multidimensional persistence, in: Y. Dong, D.-Z. Du, O. Ibarra (Eds.), Lecture Notes in Computer Science, vol. 5878, Springer, Berlin/Heidelberg, 2009, pp. 730–739.
[18] G. Carlsson, A. Zomorodian, The theory of multidimensional persistence, in: Proceedings of the Twenty-Third Annual Symposium on Computational Geometry, Gyeongju, South-Korea, vol. 392, ACM, New York, 2007, pp. 184–193.
[19] E. Catmull, J. Clark, Recursively generated b-spline surfaces on arbitrary topological meshes, Comput. Aided Des. 10 (6) (1978) 350–355.
[20] A. Cerri, P. Frosini, C. Landi, A global reduction method for multidimensional size graphs, Electron. Notes Discrete Math. 26 (2006) 21–28.
[21] A. Cerri, C. Landi, The persistence space in multidimensional persistent homology, in: R. Gonzalez-Diaz, M.-J. Jimenez, B. Medrano (Eds.), Discrete Geometry for Computer Imagery, in: Lecture Notes in Computer Science, vol. 7749, Springer, Berlin/Heidelberg, 2013, pp. 180–191.
[22] A. Cerri, C. Landi, Hausdorff stability of persistence spaces, Found. Comput. Math. 16 (2) (2016) 343–367.
[23] C. Chen, M. Kerber, Persistent homology computation with a twist, in: 27th European Workshop on Computational Geometry, vol. 45, 2011, pp. 28–31.
[24] V. de Silva, D. Morozov, M. Vejdemo-Johansson, Dualities in persistent (co)homology, Inverse Probl. 124003 (12) (2011) 16.
[25] T.K. Dey, F. Fan, Y. Wang, Computing topological persistence for simplicial maps, in: Annual Symposium on Computational Geometry, SOCG'14, ACM, 2014, pp. 345–354.
[26] T.K. Dey, C. Xin, Generalized persistence algorithm for decomposing multi-parameter persistence modules, arXiv:1904.03766, 2019.
[27] P. Dłotko, T. Kaczynski, M. Mrozek, T. Wanner, Coreduction homology algorithm for regular CW-complexes, Discrete Comput. Geom. 46 (2) (2011) 361–388.
[28] P. Dłotko, H. Wagner, Simplification of complexes for persistent homology computations, Homol. Homotopy Appl. 16 (1) (2014) 49–63.
[29] H. Edelsbrunner, J. Harer, Persistent homology - a survey, in: Contemporary Mathematics, vol. 453, American Mathematical Society, Providence, RI, 2008, pp. 257–282.
[30] H. Edelsbrunner, J. Harer, Persistent homology—a survey, Contemp. Math. 453 (2008) 257–282.
[31] H. Edelsbrunner, J.L. Harer, Jacobi sets, in: Foundations of Computational Mathematics, Minneapolis, 2002, in: London Mathematical Society Lecture Note Series, vol. 312, Cambridge University Press, 2002, pp. 37–57.
[32] H. Edelsbrunner, D. Letscher, A. Zomorodian, Topological persistence and simplification, Discrete Comput. Geom. 28 (4) (2002) 511–533.
[33] H. Edelsbrunner, E.P. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, ACM Trans. Graph. 9 (1) (Jan 1990) 66–104.
[34] D. Eisenbud, The Geometry of Syzygies: A Second Course in Commutative Algebra and Algebraic Geometry, Springer, New York, NY, 2005.
[35] F. Iuricich, MDG: a C++ library for computing discrete gradients on multivariate data, 2018.
[36] R. Fellegara, F. Iuricich, 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, ACM, 2014, pp. 223–232.
[37] R. Forman, Morse theory for cell complexes, Adv. Math. 134 (1998) 90–145.
[38] U. Fugacci, F. Iuricich, L. De Floriani, Computing discrete Morse complexes from simplicial complexes, Graph. Models 103 (2019) 101023.
[39] U. Fugacci, M. Kerber, Chunk reduction for multi-parameter persistent homology, in: G. Barequet, Y. Wang (Eds.), 35th International Symposium on Computational Geometry, SoCG 2019, Dagstuhl, Germany, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 129, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 37:1–37:14.
[40] O. Gäfvert, Algorithms for Multidimensional Persistence, Master thesis, KTH Royal Institute of Technology, 2016.
[41] C. Giusti, R. Ghrist, D.S. Bassett, Two's company, three (or more) is a simplex: algebraic-topological tools for understanding higher-order structure in neural data, J. Comput. Neurosci. 41 (1) (Aug. 2016) 1–14.
[42] D. Günther, J. Reininghaus, H. Wagner, I. Hotz, Efficient computation of 3d Morse–Smale complexes and persistent homology using discrete Morse theory, Vis. Comput. 28 (10) (2012) 959–969.

[43] A. Hatcher, Algebraic Topology, Cambridge UP, Cambridge, 2002.
[44] L. Huettenberger, C. Heine, C. Garth, Decomposition and simplification of multivariate data using Pareto sets, IEEE Trans. Vis. Comput. Graph. 20 (12) (2014) 2684–2693.
[45] F. Iuricich, S. Scaramuccia, C. Landi, L. De Floriani, A discrete Morse-based approach to multivariate data analysis, in: SIGGRAPH ASIA 2016 Symposium on Visualization, Dec. 1–8, 2016.
[46] H. King, K. Knudson, N. Mramor, Generating discrete Morse functions from point data, Exp. Math. 14 (4) (2005) 435–444.
[47] K.P. Knudson, A refinement of multi-dimensional persistence, Homol. Homotopy Appl. 10 (1) (2008) 259–281.
[48] C. Landi, S. Scaramuccia, Persistence-perfect discrete gradient vector fields and multi-parameter persistence, arXiv:1904.05081, 2019.
[49] M. Lesnick, M. Wright, Interactive visualization of 2-D persistence modules, arXiv:1512.00180, Dec. 2015, pp. 1–75.
[50] N. Milosavljević, D. Morozov, P. Skraba, Zigzag persistent homology in matrix multiplication time, in: Proceedinggs of 27th Annual Symposium of Computational Geometry, SoCG '11, ACM, New York, NY, USA, 2011, pp. 216–225.
[51] K. Mischaikow, V. Nanda, Morse theory for filtrations and efficient computation of persistent homology, Discrete Comput. Geom. 50 (2) (2013) 330–353.
[52] M. Mrozek, B. Batko, Coreduction homology algorithm, Discrete Comput. Geom. 41 (1) (2009) 96–118.
[53] M. Mrozek, P. Pilarczyk, N. Zelazna, Homology algorithm based on acyclic subspace, Comput. Math. Appl. 55 (11) (jun 2008) 2395–2412.
[54] M. Mrozek, T. Wanner, Coreduction homology algorithm for inclusions and persistent homology, Comput. Math. Appl. 60 (10) (2010) 2812–2833.
[55] J.R. Munkres, Elements of Algebraic Topology, Perseus Books, 1984.
[56] O. Gäfvert, TopCat: a Java library for computing invariants on multidimensional persistence modules, 2016.
[57] V. Robins, P.J. Wood, A.P. Sheppard, Theory and algorithms for constructing discrete Morse complexes from grayscale digital images, IEEE Trans. Pattern Anal. Mach. Intell. 33 (8) (2011) 1646–1658.
[58] P. Shilane, P. Min, M. Kazhdan, T. Funkhouser, The Princeton shape benchmark, in: Shape Modeling Applications, 2004, Proceedings, Genova, Italy, 2004, pp. 167–178.
[59] N. Shivashankar, S. Maadasamy, V. Natarajan, Parallel computation of 2d Morse-Smale complexes, IEEE Trans. Vis. Comput. Graph. 18 (10) (2012) 1757–1770.
[60] N. Shivashankar, V. Natarajan, Parallel computation of 3d Morse-Smale complexes, Comput. Graph. Forum 31 (3) (2012) 965–974.
[61] R. van de Weygaert, G. Vegter, H. Edelsbrunner, B. Jones, P. Pranav, C. Park, W.A. Hellwing, B. Eldering, N. Kruithof, P. Bos, J. Hidding, J. Feldbrugge, E. ten Have, M. van Engelen, M. Caroli, M. Teillaud, Alpha, Betti and the megaparsec universe: on the topology of the cosmic web, in: M.L. Gavrilova, C.K. Tan, M.A. Mostafavi (Eds.), Transactions on Computational Science XIV, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 60–101.
[62] K. Weiss, F. Iuricich, R. Fellegara, L. De Floriani, A primal/dual representation for discrete Morse complexes on tetrahedral meshes, Comput. Graph. Forum 32 (3) (2013) 361–370.
[63] P. Wu, C. Chen, Y. Wang, S. Zhang, C. Yuan, Z. Qian, D. Metaxas, L. Axel, Optimal topological cycles and their application in cardiac trabeculae restoration, in: M. Niethammer, M. Styner, S. Aylward, H. Zhu, I. Oguz, P.-T. Yap, D. Shen (Eds.), Information Processing in Medical Imaging, vol. 10265, Springer International Publishing, Cham, 2017, pp. 80–92.