Computing multiparameter persistent homology through a discrete Morse-based approach

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*

Abstract

Persistent Homology (PH) allows tracking homology features like loops, holes and their higher-dimensional analogs, along with a single-parameter family of nested spaces. Currently, computing descriptors for complex data characterized by multiple functions is becoming a major challenging task in several applications, including physics, chemistry, medicine, geography, etc. *Multiparameter Persistent Homology* (MPH) generalizes persistent homology opening to the exploration and analysis of shapes endowed with multiple filtering functions. Still, computational constraints prevent MPH to be feasible over real-sized data. In this paper, we consider *discrete Morse Theory* [1] as a tool to simplify the computation of MPH on a multiparameter dataset. We propose a new algorithm, well suited for parallel and distributed implementations and we provide the first evaluation of the impact on MPH computations of a preprocessing approach.

Keywords: persistent homology, topological data analysis, Multiparameter persistent homology, Morse reduction, discrete Morse theory

1. Introduction

In recent years, the increasing amount of data has led to the improvement and development of information handling techniques. The basic goal of Topological Data Analysis (TDA) is to retrieve and organize qualitative information about data.

Homology is one of the most relevant invariants studied in TDA but has the drawback of being scarcely descriptive.

Persistent Homology (PH) allows for a multiresolution analysis of homologies by means of a *filtration*. It is used in data analysis to study evolutions of qualitative features of data and it is appreciated for its computability, robustness to noise, and dimension independence. So far, many optimization methods in computing PH have been proposed. Those more tightly related to this paper refer to another relevant tool for TDA,

*Corresponding author
Email addresses: sara.scaramuccia@dibris.unige.it (Sara Scaramuccia), fiurici@clemson.edu (Federico Iuricich), deflo@umiacs.umd.edu (Leila De Floriani), claudia.landi@unimore.it (Claudia Landi)

namely *discrete Morse theory* [1]. In this case, discrete Morse theory provides an important preprocessing tool for homology computations by defining a *discrete gradient field* (also called discrete gradient) over the input datum. This allows reducing the size of the input space to the critical parts, generally few, with respect to the retrieved discrete gradient. The discrete gradient can be also built so that to preserve the filtration structure thus enhancing PH computations. Although other PH optimizations outperform this Morse-based preprocessing, this no longer applies to the generalization of PH called *multiparameter persistent homology* (MPH).

MPH is an extension of the PH theory motivated by the fact that data analysis and comparisons often involve the examination of properties that are naturally described by multiple parameters, for instance in computer vision with respect to photometric properties. Alternatively, for point cloud data, several criteria might be chosen in order to filter the input datum for investigating both the domain itself under several criteria at once and the explanatory power of the different criteria over the same domain.

The entire information provided by MPH is captured by the *persistence module*. Alternatively, the *persistence space* summarizes the MPH information into a collection of PH descriptors.

All available MPH methods suffer from high computational costs and scalability problems. This prevents them to be feasible over real-sized data sets. A Morse-based preprocessing solution, generalized to the multiparameter case, has been proposed in [2]. This can have, in theory, a valuable impact on MPH computations. However, that preprocessing still presents limitations in scalability with real data.

We propose here the first algorithm capable of computing a discrete gradient on real-world data. Our approach is easy-to-use and well suited for parallel and distributed implementations. We consider the applicative domain where the obtained representation can be successfully adopted, namely, for reducing the complexity of computing MPH.

Our contributions is composed of:

- **a new algorithm** for retrieving a discrete gradient which preserves MPH and suitable for real-sized data sets;
- **a comparison** of the scalability of our proposed algorithm with respect to the equivalent method in state of the art;
- **an evaluation** of the advantages obtained by using our proposed preprocessing in persistence module and persistence space computation.

The remainder of this paper is organized as follows. In Section 2, we introduce the notions at the basis of our work. Related work is reviewed in Section 3. The new preprocessing algorithm is described in Section 4 where we also present a detailed analysis of complexity. In Section 5, we present the proof of correctness and a theoretical and experimental comparison, of our approach, with the one presented in [2]. The results of computing MPH with our approach are discussed in Section 6. In Section 7, we draw concluding remarks and we discuss future developments.

2. Background

2.1. Simplicial complexes

A simplicial complex is a discrete topological structure introduced to formalize the input of our algorithm. 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 shortly mean a k-simplex. A *face* τ of σ is the convex hull of any subset of points generating σ. The partial order relation "τ is face of σ" is denoted $\tau \ll \sigma$. If the dimensions of τ and σ differ by one we call τ a *facet* of σ and denote it by $\tau < \sigma$. Dually, σ is a *coface* of τ and a *cofacet* when the two dimensions differ by one (respectively indicated $\sigma \gg \tau$ and $\sigma > \tau$).

Definition 1 (Simplicial complex)**.** A *simplicial complex* S is a finite collection of simplices such that:

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

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 has dimension d (d-complex for short) if the maximum of the dimensions of its simplices is d. The *boundary* of a simplex σ in S is the set of all faces of σ in S. The *coboundary* (or the *star*) of σ in S is the set of cofaces of σ in S.

2.2. Discrete Morse theory

The algorithm proposed in this paper retrieves a combinatorial object called *discrete gradient* over the domain S. The relevance of this output has to be seen within the framework of Forman's discrete Morse Theory [1]. In discrete Morse Theory, a *(discrete) vector* is a pair of simplices (σ, τ) such that $\sigma < \tau$. A *discrete vector field* is any collection of vectors V such that each simplex is 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 $0 \leqslant i \leqslant r-1$, $\sigma_{i+1} < \tau_i$ and $\sigma_i \neq \sigma_{i+1}$. A V-path is said to be *closed* if $\sigma_0 = \sigma_r$ and *trivial* if $r = 0$. We say that a V-path $\{(\sigma_0^k, \tau_0^{k+1}), \ldots, (\sigma_l^k, \tau_l^{k+1})\}$ is *from σ^k to $\bar{\sigma}^k$*, if $\sigma_0^k = \sigma^k$ and $\tau_l^{k+1} > \bar{\sigma}^k$. If $\sigma^k = \bar{\sigma}^k$ the empty set is a valid V-path from σ^k to itself.

Definition 2 (discrete gradient)**.** 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 *critical*. Moreover, instead of critical simplices, we use the term critical *cell* with dimension equal to the simplex dimension.

The set of critical cells $M(V)$ is called a *critical set over* X. From now on, we wright M instead of $M(V)$ when the discrete gradient V is understood. 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 face of τ^{k+1} to σ^k.

2.3. Homology

Intuitively, the homology of a simplicial complex S detects independent k-dimensional cycles of S, i.e., different connected components (0-cycles), tunnels (1-cycles), voids (2-cycles), and so on. More precisely, the incidence relations among simplices in a simplicial complex are combinatorially translated into algebraic terms and, the intuition of a loop is captured by linear combinations of simplices called *chains*. We focus on the case of linear combinations over $\mathbb{F}_2$ the field with the two elements 0 and 1. Incidence relations are translated into algebraic terms by means of a suitable incidence function.

The *simplicial incidence function* $\chi : S \times S \longrightarrow \mathbb{F}_2$ associated with a simplicial complex S is defined by

$$\begin{array}{lll} \chi(\tau,\sigma) = 1 & \leftrightarrow & \tau > \sigma \\ \chi(\tau,\sigma) = 0 & \leftrightarrow & \text{otherwise} \end{array}$$

It is not necessary to have a simplicial complex to construct a chain complex, and thus, to get the homology associated with it. Indeed, in the case of a discrete gradient V over S, we define a *separatrix from a critical cell* τ^{k+1} *to a critical cell* σ^k to be a V-path from any face of τ^{k+1} to σ^k. Then, we can translate incidence relations among critical cells in M into algebraic terms by means of the *critical incidence function* $\mu : M \times M \longrightarrow \mathbb{F}_2$ defined by

$$\begin{array}{lll} \mu(\tau,\sigma) = 1 & \leftrightarrow & \text{the number of different separatrices from } \tau \text{ to } \sigma \text{ is odd} \\ \mu(\tau,\sigma) = 0 & \leftrightarrow & \text{the number of different separatrices from } \tau \text{ to } \sigma \text{ is even} \end{array}$$

We remark that the definition of critical incidence function in this form comes from Theorem 8.10 in [1] in the particular case of $\mathbb{F}_2$ coefficients. Moreover, notice that only cells whose dimensions differ by 1 can have non-null value under the incidence function.

Both a simplicial complex with the simplicial incidence function and a critical set with the critical incidence functions belong to a class of combinatorial structures called Lefschetz complexes. For Lefschetz complexes, we report only the basic notions and refer the reader to [3] for more details about Lefschetz complexes or to [4] for a notation closer to ours, where the Lefschetz complexes are called S-complexes.

Definition 3 (Lefschetz complex)**.** A *Lefschetz complex* (X, κ) *over* $\mathbb{F}_2$ consists of a graded finite set $X = \bigsqcup_{k\in\mathbb{Z}} X_k$ along with an *incidence function* $\kappa : X \times X \longrightarrow \mathbb{F}_2$ satisfying:

i) $\kappa(\tau,\sigma) \neq 0$ implies that the dimensions of τ and σ differ by 1
ii) for every two cells τ^{k+2} and σ^k in X then

$$\sum_{\rho\in X_{k-1}} \kappa(\tau,\rho)\kappa(\rho,\sigma) = 0$$

A simplicial complex X corresponds to a Lefschetz complex (X, κ) with X graded by the simplex dimensions, that is with $X_k = X_k$, and κ equal to the simplicial incidence function χ. Condition i) in Definition 3 is straightforward to check. Condition ii) follows easily by noticing that only simplices ρ satisfying $\tau > \rho > \sigma$ in X lead to non null

summands. Moreover, in X, if existing, there are exactly two such ρ, thus summing up to zero in $\mathbb{F}_2$.

For a Lefschetz complex (X, κ), in analogy with the simplicial case, if $\kappa(\tau, \sigma) \neq 0$, we call τ a *cofacet* of σ (resp. σ a *facet* of τ) and write $\tau > \sigma$ (resp. $\sigma < \tau$). Moreover, each cell $\tau \in X_k$ will be called a k-cell and often shortly denoted by τ^k.

On the other hand, a critical set M corresponds to a Lefschetz complex (X, κ) by defining M_k the set of all k-dimensional cells in M, by setting $X_k = M_k$, and by choosing κ equal to the critical incidence function μ. We call (M, μ) a *critical Lefschetz complex over* (X, κ). We already noticed that Condition i) in Definition 3 is satisfied. The fact that the function μ fulfills condition ii) follows from Theorem 8.10 in [1] by applying the Remark 1 which we postpone.

Definition 4 (Chain complex)**.** The *chain complex* $\mathcal{C}(X) = (\mathcal{C}_*(X), \partial_*)$ associated with the Lefschetz complex (X, κ) consists of the family $\mathcal{C}_*(X) = \{\mathcal{C}_k(X)\}_{k\in\mathbb{Z}}$ of $\mathbb{F}_2$-vector spaces along with the collection of linear maps $\partial_* = \{\partial_k : \mathcal{C}_k(X) \longrightarrow \mathcal{C}_{k-1}(X)\}_{k\in\mathbb{Z}}$ defined as:

- $\mathcal{C}_k(X)$ is the vector spaces generated by X_k whose elements are called *k-chains*
- ∂_k is called *boundary map* and defined by linear extension from the image of each cell τ^k defined by
$$\partial_k(\tau) = \sum_{\sigma\in X} \kappa(\tau, \sigma)\sigma \tag{1}$$

Notice that, by definition of Lefschetz complex, only $(k-1)$-cells can contribute to the image of the *boundary* $\partial_k(\tau)$ for a k-cell τ. Moreover, condition (ii) in Definition 3 guarantees that $\operatorname{im}\partial_{k+1}$ is included in $\ker\partial_k$, that is

$$\partial_{k+1} \circ \partial_{k+2} = 0 \tag{2}$$

Remark 1. With our assumptions, condition (2) implies Condition ii) in Definition 3. Indeed, by extending the terms in (2) when applied to a specific cell and manipulating the sums, we get that a sum of linearly independent elements gives the null vector. Hence, we deduce that all coefficients are null and coefficients corresponds the left hand side in Condition ii).

By Remark 1, we can look at Lefschetz complexes as a way of dealing with chain complexes in terms of their bases rather than the entire vector spaces. Loops are formalized as k-chains with trivial boundary but such k-chains when bounding $(k+1)$-chains do not detect an actual hole. A *k-cycle* is an element in the kernel of ∂_k. A *k-boundary* is an element in the image of ∂_{k+1}. The *k^{th}-homology of the Lefschetz complex* (X, κ) is defined as the quotient vector space of k-cycles over k-boundaries:

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

The combination of Theorems 7.3 and 8.2 in Forman's [1] proves, in particular, the following

Theorem 5 (Homology invariance)**.** *Given a critical set M over a simplicial complex S, the following isomorphisms hold*

$$\forall k \in \mathbb{Z}, \quad H_k(X) \cong H_k(M)$$

2.4. Multiparameter Persistent Homology

Intuitively, persistent homology studies the homological changes along an increasing sequence of Lefschetz complexes called *filtration*. We start by considering a finite total order $(I, \leqslant)$ and we call *grades* its elements. In this paper, the grade set I is to be thought of as a finite sampling in either $\mathbb{R}$ or $\mathbb{Z}$.

Definition 6 (One-parameter filtration)**.** A filtration $\mathcal{X}$ of a Lefschetz complex (X, κ) is a finite collection of Lefschetz subcomplexes $\mathcal{X}^u = (X^u, \kappa^u)$ in X indexed by $u \in I$ such that: for all grades $u \leqslant v$, $\mathcal{X}^u$ is a Lefschetz subcomplex in $\mathcal{X}^v$

Definition 7 (Compatible discrete gradient)**.** Given a one-parameter filtration $\mathcal{X}$ of a simplicial complex X indexed on I, a discrete gradient V over X is called *compatible with* $\mathcal{X}$ if, for each $(\sigma, \tau) \in V$ and any filtration grade $u \in I$, it holds that

$$\sigma \in \mathcal{X}^u \quad \Rightarrow \quad \tau \in \mathcal{X}^u$$

If a critical Lefschetz complex (M, μ) comes from a discrete gradient compatible with a filtration $\mathcal{X}$, we call (M, μ) a *critical Lefschetz complex compatible with* $\mathcal{X}$.

The latter concepts can be generalized to the multiparameter case. In place of a finite total order $(I, \leqslant)$, we can consider the partial ordered set $(I^n, \preceq)$ with I^n the n-fold Cartesian product for some non-negative integer n and, for any $u = (u_1, \ldots, u_n), v = (v_1, \ldots, v_n) \in I^n$,

$$u \preceq v \qquad \leftrightarrow \qquad \forall i \in \{1, \ldots, n\}, u_i \leqslant v_i.$$

We call $(I^n, \preceq)$ the *grade poset* and, again, *grades* its elements. If neither $u \preceq v$ or $v \preceq u$, we call the two grades u and v *incomparable* and *comparable* otherwise. If $u \preceq v$ and $u \neq v$, we shortly wright $u \precneqq v$.

Definition 8 (Multiparamter filtration)**.** A multiparameter filtration, or simply *filtration* $\mathcal{X}$ of a Lefschetz complex (X, κ) is a finite collection of Lefschetz subcomplexes $\mathcal{X}^u = (X^u, \kappa^u)$ in X indexed by $u \in I^n$ such that: for all grades $u \preceq v$, $\mathcal{X}^u$ is a Lefscehtz subcomplex in $\mathcal{X}^v$

In this paper, we are interested in a specific way of getting a filtration, that is by sublevel sets with respect to a function over X. A *filtering function* is a function $\phi : X \longrightarrow I^n$ such that if σ is a face of τ, then $\phi(\sigma) \preceq \phi(\tau)$. The *filtration induced by* ϕ is the family of Lefschetz subcomplexes $\mathcal{X}(\phi) := \{\mathcal{X}(\phi)^u\}_{u \in I^n}$ in (X, κ) with $\mathcal{X}(\phi)^u = (X^u, \kappa^u)$ defined by

$$X^u := \{\sigma \in X \mid \phi(\sigma) \preceq u\}$$
$$\kappa^u := \kappa \text{ restricted to } X^u \times X^u$$

A subset T is *closed* in X if, for all $\tau \in T$, the condition $\kappa_X(\tau, \sigma) \neq 0$ for some $\sigma \in X$ implies $\sigma \in T$. It is a known fact (Theorems 3.1, 3.2 in [4]) that a closed subset T in X is a Lefschetz subcomplex with incidence function taken by restriction.

Remark 2. For each pair of grades $u \preceq v$, we have that X^u is closed in $\mathcal{X}^v$. Indeed, by definition of filtering function, faces cannot have higher grades than cofaces. This guarantees that $\mathcal{X}(\phi)$ is actually a filtration of (X, κ).

It is worth to remark that a one-parameter filtration $\mathcal{X}$ can always be thought of as a $\mathcal{X}(\phi)$ for some scalar-valued filtering function ϕ. On the contrary, if $n \geqslant 2$, filtrations induced by functions give a subclass of general filtrations. For instance, filtrations of kind $\mathcal{X}(\phi)$ are *one-critical*, which according to [5] , means that the minimal grade $u \in I^n$ a cell in X belong to in $\mathcal{X}(\phi)$ is unique.

Once we have a filtration, we can investigate how homology properties change from one step to another. For any homology degree $k \in \mathbb{Z}$, we can associate each step $\mathcal{X}^u$ with its homology space $H_k(\mathcal{X}^u)$. Moreover, since for all grades $u \preceq v$, the inclusion of corresponding Lefschetz complexes preserves cycles and boundaries, we get induced a linear map $\iota^{u,v} : H_k(\mathcal{X}^u) \longrightarrow H_k(\mathcal{X}^v)$ at homology level, not necessarily injective since cycles can possibly become boundaries by adding cells.

The *persistent* k^{th}*-homology* relative to the the grades $u \preceq v$ is the image of $\iota^{u,v}$ as a subspace in $H_k(\mathcal{X}^v)$, that is the space of all the homology classes of $H_k(\mathcal{X}^u)$ which are persistent in $H_k(\mathcal{X}^v)$. The global information of persistent homology for all possible grades $u \preceq v$ is encoded in the *persistence module*.

Definition 9 (Persistence module)**.** The *persistence* k^{th}*-module* $H_k(\mathcal{X})$ of the filtered complex $\mathcal{X}$ consists of:

- the collection of $\mathbb{F}_2$-vector spaces $H_k(\mathcal{X}^u)$, for each filtration step u
- the collection of all inclusion-induced linear maps $\iota^{u,v} : H_k(\mathcal{X}^u) \longrightarrow H_k(\mathcal{X}^v)$, for each pair of grades in I^n satisfying $u \preceq v$.

In the case of $n = 1$, we talk about *one-parameter persistent homology*, or simply one-parameter persistence.

Now, we are ready to formalize the main property of the discrete gradient retrieved by the algorithm we are proposing in this paper.

Definition 10 (Compatible discrete gradient)**.** Given a filtration $\mathcal{X}$ of a simplicial complex X indexed on I^n, a discrete gradient V over X is called *compatible with* $\mathcal{X}$ if, for each $(\sigma, \tau) \in V$ and any filtration grade $u \in I^n$, it holds that

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

If a critical Lefschetz complex comes from a discrete gradient compatible with a filtration $\mathcal{X}$, we call (M, μ) a *critical Lefschetz complex compatible with* $\mathcal{X}$. We already know from Theorem 5 that the homology of a simplicial complex X is preserved by any critical Lefschetz complex (M, μ) over (X, χ). In fact, the filtration structure can be also preserved. The following result generalizes to the multiparameter case Theorem 4.3 in [6] and equivalently Corollary 2 in [2].

Theorem 11 (Persistence module invariance)**.** *Given a filtration $\mathcal{S}$ of a simplicial complex (S, χ) and a critical Lefschetz complex (X, μ) compatible with $\mathcal{X}$, it holds that*

$$\forall k \in \mathbb{Z}, \quad H_k(\mathcal{X}) \cong H_k(\mathcal{M}) \tag{3}$$

This result guarantees that studying a critical Lefschetz complex compatible with a given filtration is equivalent to study the original filtered complex. As an advantage, in the critical Lefschetz complex, we have generally fewer cells to deal with.

3. Related work

In this section we review the related work on the computation of persistent homology and multi-parameter persistent homology.

3.1. Computing persistent homology

In the one-parameter case, computing the persistence module with coefficients in a field means reducing the *boundary matrix* via the standard algorithm [7]. The latter algorithm has cubic complexity in the worst case. For this reason new approaches have been studied to improve efficiency. We present the resulting optimizations divided in three groups: *integrated*, *annotation-based*, and *preprocessing*.

Integrated optimizations aim at improving the efficiency of the standard approach by either reducing the number of steps reguired to get to a reduced matrix or by progressively removing columns during the computation. These approches exploit the total order defined on simplices of the simplicial complex to improve efficiency. We classify as integrated optimizations the *Twist* algorithm [8] the *Row* algorithm [9], the approach based on sparsity presented in [10], the one based on *Spectral sequences* [11], and the *Chunk* algorithm [12].

Annotation-based techniques [13, 14, 15] take advantage of an efficient data structure, namely the annotation matrix, to efficiently compute the persistent co-homology of a complex.

Preprocessing optimizations aim at reducing the size of the input filtered complex while preserving the output persistence diagram. In [16], homology preserving techniques, such as reductions, coreductions [17, 18, 19] and acyclic subspaces [20], are adapted to the case of persistent homology. Approaches rooted in discrete Morse Theory [1] compute a discrete gradient V compatible with the input filtration. The theoretical results in [6] guarantees that the chain complex constructed from V has the same persistence module of 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 [21] is the first one to introduce a divide-and-conquer approach for computing a Forman gradient on real data. However, it has the main drawback of introducing many spurious critical simplices. Two approaches have been defined in [22, 23] for 2D and 3D images respectively. Focusing on a parallel implementation, they provide a substantial speedup in computing the discrete gradient still creating spurious critical simplices. In [24], a dimension-agnostic algorithm is proposed that processes the lower

star of each vertex independently. 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 sublevel sets, i.e. no spurious critical simplices are created. An efficient implementation of [24], focused on regular grids, is discussed in [25] while, for simplicial complexes, the same algorithm as been extended to triangle [26] and tetrahedral meshes [27]. The first dimension independent implementation for simplicial complexes is presented in [28].

3.2. Computing multiparameter persistent homology

The first difference we encounter when computing multi-parameter persistent homology is that we no longer have any complete descriptor for the persistence module [29]. As a result, either we compute the full persistence module or we compute invariants that deliver only partial information about the multi-paramter persistent homology. The first algorithm for the persistence module retrieval is proposed in [5], 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. As drawbacks, the algorithm introduces an artefact dependency on the chosen basis and implies high computational costs in terms of time: $O(m^4n^3)$, where m is the number of simplices in the complex and n is the number of independent parameters in the multifiltration. Another approach for computing the persistence module is proposed in [30]. The algorithm acts on the multifiltration at chain level rather than at homology level. First, k-cycles and k-boundaries are expressed in terms of the same basis along the multifiltration at the chain level. Then, the Smith Normal Form reduction [31, 32] is applied at each multigrade u in the multifiltration leading to a worst time complexity of $O(m^3\bar{\mu}^n)$, where m is the number of simplices, n the number of parameters in the multifiltration, and $\bar{\mu} := \max_{i=0,\dots,n} \mu_i$, with μ_i the number of multigrades in the multifiltration along the i^{th}-axis. The algorithm has been implemented in the Topcat library [33] and distributed in public domain. A non-complete descriptor for MPH is the *rank invariant*, introduced in [29] for each pair of multigrades $u \preceq v$ as the rank of the corresponding inclusion-induced map, that 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 Topcat persistence module representation. However, the full rank invariant computation requires the iteration of this simple procedure for all possible multigrades satisfying $u \preceq v$ which multiplies the complexity by $\frac{1}{2}\mu^2$, where μ is the cardinality of all multigrades considered in the multifiltration which is typically very large.

The persistence space. The *persistence space* [34] is equivalent to the rank invariant but it practically enhances computational performances by avoiding to precompute the persistence module.

The persistence space can be computed based on the *foliation method* [35]. With such approach the persistence space is constructed incrementally by slicing the space of the input multi-parameter 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 to the persistence space retrieval was limited to the case of 0^{th}-homology [35]. Then, in [36], the foliation method is applied to higher homology degrees. An approximate version of the persistence

space is proposed in [37] for two-parameter filtrations, also called *bifiltrations*: a selection of slices is performed to guarantee a fixed tolerance for the matching distance [36] among persistence spaces. This method finds applications for shape comparison in the PHOG library [38] where authors use the approximate persistence space to deal with photometric attributes.

Limitedly to bifiltrations, a complete representation of the persistence space is computed by the RIVET visualization tool [39] which is available online at `http://rivet.online`. The approach uses the bigraded Betti numbers [40, 41] to locate λ_i multigrades, i.e., where changes in the homology of degree i happen, with $i = 1, 2$. This procedure requires time $O(m^3\lambda)$, where λ is the product of $\lambda_1\lambda_2$ and allows to idenfiy an arrangements of lines such that all filtrations along these lines have the same barcode template. A *barcode template* is constructed in $O(m^3\lambda + (m + \log\lambda)\lambda^2)$. The barcode template encodes the set of bars (i.e., persistence pairs) for every valid filtration in the space of bifiltrations. The actual length for each bar is computed on the fly, upon request, in linear time with respect to m.

Multiparameter optimization methods. Most optimization methods developed for classic persistent homology have not yet found a counterpart in the multiparameter case. So far, the only approach that seem still feasible is simplifying the input filtration into a new one with less cells and less grades. Limited to the study of 0^{th}-homology the algorithm proposed in [42] is the first approach capable of reducing the size of an input complex S without affecting its persistence module.

The approach proposed in [43] can be seen as a Morse-based method generalizing to the multiparameter case the one proposed in [21]. The algorithm computes a discreate gradient field having the same persistence module of the input complex. Like its one-parameter counterpart [21], it suffers from introducing many spurious critical simplices. In a successive paper [2] a new approach is introduced generalizing the idea of [24] of constructing the discrete gradient locally inside the lower star of simplices of S. The resulting discrete gradient is proved to induce a Morse complex with the same persistence module, and then the same persistence space, as the original multifiltration. However, the algorithm requires a global ordering of all the simplices of S and cannot be applied to real-world data. In Section 5, we will further discuss this issue compared with our approach that can be seen as a divide-and-conquer generaliation of [2].

4. Locally computing a discrete gradient over multiparameters

In this section we present our new algorithm for computing a discrete gradient vector field compatible with a multiparameter dataset. For ease of exposition, we describe the method by focusing on simplicial complexes though it is valid for any cell complex satisfying the intersection property such as *cubical complexes*.

In Section 4.1 we provide a high-level description of the algorithm workflow. A detailed description of the auxiliary functions will be provided in Section 4.2, while in Section 4.3 we discuss the algorithm's complexity.

4.1. Outline of the algorithm

The proposed algorithm receives a multiparameter dataset in input and produces a compatible discrete gradient. In what follows, we describe the input multiparameter

dataset 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. Without loss of generality we require the function f to be component-wise injective. In applications, any function can be transformed into a component-wise injective one by means of simulation of simplicity [44]. The obtained output is a pair (V, M) where V is the set of paired simplices of S and M is the set of critical (unpaired) simplices completely defining the discrete gradient.

We recall that a discrete gradient is compatible to f if for each pair of simplices (σ, τ) in V, σ and τ have the same multigrade (see Section 2). Then, the main objective of the algorithm is that of decomposing S according to f so to compute pairings between cells belonging to the same multigrade, possibly in parallel.

The algorithm consists of three main steps: vertex-based decomposition, multigrade grouping, and pairings computation. The main workflow is described in Algorithm 1.

The first objective is that of decomposing S to obtain a first rough subdivision of the simplices (line 2). In this step we only require that simplices belonging to the same multigrade also belong to the same group in the decomposition. This is achieved by the function `ComputeDiscreteGradient` as follows:

- we compute an indexing $I : S_0 \longrightarrow \mathbb{R}$ for the vertices of S. The indexing is extended to the other simplices $\sigma \in S$ by setting $I(\sigma) := \{I(v) \mid v \in S_0 \wedge v \ll \sigma\}$ and is required to be *well-extensible*, i.e., I satisfies, for all simplices $\sigma, \tau \in S$, the following property:

$$f(\sigma) \preceq f(\tau) \qquad \Rightarrow \qquad I(\sigma) \preccurlyeq I(\tau). \tag{4}$$

- We subdivide S into lower stars according to I. We recall the the star of a simplex σ is defined as the set of cofaces of σ. Then, we define the index-based lower star of a simplex σ as the set of cofaces having value of I lower or equal to σ. Formally,

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

The well-extensible indexing, in combination with the index-based lower star provide two fundamental properties for our algorithm:

- each simplex σ belongs to the indexed based lower star of exactly one vertex (Lemma 12 in Section 5)
- if two simplices have the same multigrade, then they belong to the index-based lower star of the same vertex (Lemma 13 in Section 5).

That is, a well-extensible indexing and the index-based lower stars implicitly provide a valid decomposition for the domain S. Thanks to the former properties we can guarantee that by processing the vertices independently we are not missing any valid pairing (see Section 5 for the formal proof).

The next step requires grouping the simplices of $\mathrm{Low}_I(v)$, with $v \in S_0$, having the same multigrade (lines 3-5). The latter is done by explicitly computing the index-based lower star for each vertex v (function `ComputeIndexLowerStar`) and by subdividing the resulting set of simplices such that simplices with equal value of f end up in the same

Algorithm 1 ComputeDiscreteGradient(S, f)

Input: S a simplicial complex
Input: $f : S_0 \longrightarrow \mathbb{R}^n$ component-wise injective function
Output: V list of simplex pairs # discrete gradient compatible with f
Output: M list of simplices # critical cells of V

1: V, M are empty lists
2: $I \leftarrow$ `ComputeIndexing`(S_0, f) # I is a well extensible indexing with respect to f
3: **for all** v in S_0 **do** # independently from the order
4: $\quad \mathrm{Low}_I(v) \leftarrow$ `ComputeIndexLowerStar`(v, I, S)
5: $\quad K(v) \leftarrow$ `SplitIndexLowerStar`$(f, \mathrm{Low}_I(v))$
6: $\quad$ **for all** Lset in K_v **do** # independently from the order
7: $\quad\quad (V_{\mathrm{Lset}}, M_{\mathrm{Lset}}) \leftarrow$ `HomotopyExpansion`(S, Lset)
8: $\quad\quad$ append V_{Lset} to V
9: $\quad\quad$ append M_{Lset} to M
10: **return** (V, M)

set. This is done by the auxiliary function `SplitIndexLowerStar` which organizes the simplices and returns K_v, a list of sets where each set contains simplices with the same multigrade.

In the last step (lines 6-9), each multigrade Lset $\in K_v$ is independently processed by the auxiliary function `HomotopyExpansion` responsible for computing the actual pairings. Paired and critical simplices found in the multigrade set Lset will contribute to the final discrete gradient. Since simplices are subdivided based on their multigrade, each simplex S appears in a exactly one level set and it will be classified, as either paired or critical, only once. This makes the approach embarrassingly parallel.

4.2. Auxiliary functions

This section provides additional information about the auxiliary functions we use Algorithm 1 following the order of appearance.

The first auxiliary function is `ComputeIndexing` which is used for computing a well-extensible indexing on the vertices of S. There are many ways to obtain a well-extensible indexing I, we have chosen to sort all the vertices 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 as, for each pair of simplices σ and τ, $f(\sigma) \preceq f(\tau)$ implies $f_1(\sigma) \preceq f_1(\tau)$. Thus, a vertex $v \in \tau$ exists such that $f_i(v) \geqslant f_i(w)$ for every vertex $w \ll \sigma$. This implies $I(v) \geqslant I(w)$, for every vertex $w \ll \sigma$ and we conclude that $\tilde{I}(\sigma) \leqslant \tilde{I}(\tau)$.

Next, `ComputeIndexLowerStar` is used for computing the index-based lower star of a vertex from the indexing I. The function extracts the set of simplices incident into a vertex v. 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. $I(v_0) > I(v_1) > \cdots > I(v_k)$.

The computed index-based lower stars are then subdivided in independent sets by `SplitIndexLowerStar` according to multigrades. This function initializes an associative

Algorithm 2 HomotopyExpansion(X, Lset)

Input: X, a simplicial complex, Lset, a list of cells in X forming a level set w.r.t. f
Output: V_{Lset} list of discrete vectors, M_{Lset} list of simplices

1: set $V_{\text{Lset}}, M_{\text{Lset}}$ to be empty lists
2: set `Ord0`,`Ord1` to be empty ordered lists
3: set `declared` to be an array of length $|\,\text{Lset}\,|$ with Boolean values equal to `false`
4: **for all** τ in Lset **do**
5: **if** `num_undeclared_facets`(τ, Lset)$= 0$ **then**
6: insert τ into `Ord0`
7: **else if** `num_undeclared_facets`(τ, Lset)$= 1$ **then**
8: insert τ into `Ord1`
9: **while** `Ord1`$\neq \varnothing$ or `Ord0`$\neq \varnothing$ **do**
10: **while** `Ord1`$\neq \varnothing$ **do**
11: $\tau \leftarrow$ the first element in `Ord1` # τ is removed from `Ord1`
12: **if** `num_undeclared_facets`(τ, Lset)$= 0$ **then**
13: insert τ into `Ord1`
14: **else**
15: $\rho \leftarrow$ `unpaired_facet`(τ, Lset) # ρ is removed from `Ord0`
16: add (ρ, τ) to V_{Lset}
17: `declared`[ρ] , `declared`[τ] $\leftarrow$`true`
18: `add_cofacets`(ρ,Lset,`Ord1`)
19: `add_cofacets`(τ, Lset,`Ord1`)
20: **if** `Ord0`$\neq \varnothing$ **then**
21: $\tau \leftarrow$ the first element in `Ord0` # τ is removed from `Ord0`
22: append τ to M_{Lset}
23: `declared`[τ] $\leftarrow$ `true`
24: `add_cofacets`(τ, Lset,`Ord1`)
25: **return** $(V_{\text{Lset}}, M_{\text{Lset}})$

array mapping from a multigrade (a vector of floats) to the list of simplices sharing the same multigrade. We recall that the filtration values f are assumed to be associated to the vertices of S only. For any other simplex σ the filtration value is computed for each component i as $f_i(\sigma) := max_{v \in \sigma} f_i(v)$.

As a last step, function `HomotopyExpansion` classifies simplices with the same multigrade. We present its pseudocode in Algorithm 2. The execution has no conceptual differences from the one described in [24]. A k-simplex σ and a $(k+1)$-simplex τ are considered *pairable* only when σ is the only unclassified facet of τ. So, 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 lists `Ord0` and `Ord1` are used to keep track of those simplices that have exactly zero unpaired facets or one unpaired facet, respectively. Intuitively, simplices in `Ord0` are candidates to be classified as critical or as tails of arrows in a discrete vector, since they have no face to be paired with, while simplices in `Ord1` are the candidate to be heads or arrows in a discrete vector. The two lists are initialized by cycling on the simplices in the input set (lines 4 to 8 of Algorithm 2). The auxiliary function

`num_undeclared_facets`() is used to count the number of unclassified facets for each simplex. Both lists `Ord0` and `Ord1` are ordered in such a way to have faces taking priority over cofaces. The 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`.

Inside the two nested while loops (lines 9 and 10) is where simplices are classified. If `Ord1` is not empty, we extract the first simplex τ from it and we verify if the number of unclassified facets of τ has not changed (lines 12 and 13). Notice that the number of unpaired facets can only decrease. If this number is now zero (i.e., its facet has been classified), we add τ to `Ord0`. Otherwise we retrieve its unique unclassified facet σ (line 15), we add (σ,τ) to the set of pairs V_{Lset}, and we update the array `declared` accordingly. After classifying σ and τ, all their cofacets are visited and added to either `Ord1` or `Ord0`, if they have the necessary number of unclassified facets (lines 18 and 19).

When no pairable simplex is available (i.e., `Ord1` is empty) the first cell in `Ord0` is extracted and declared critical (lines 21 to 23). All its cofacets are processed and added to `Ord1` if it is the case. The algorithm stops when both lists are empty. In Proposition 4 in [24], authors show that we exit the outer while loop when all cells have been classified.

4.3. Complexity

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

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

- for each simplex $\sigma \in S$, we assume $\text{Star}(\sigma)$ to be computed and stored off-line. If computed on the fly, $\text{Star}(\sigma)$ would require $O(|\,\text{Star}(\sigma)|)$ [45].
- the ordered lists `Ord0`, `Ord1` are implemented as self-balancing binary search trees. Inserting, or removing an element from the tree has a logarithmic cost in the list's size.
- For each k-simplex $\sigma \in S$, $f(\sigma)$ can be retrieved in $O(k+1)$ by retrieving the filtration values of the vertices of σ. We will overestimate this by always 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 5.4).

4.3.1. Analysis of the auxiliary functions

Here, we present the time and storage costs of the auxiliary functions introduced in Section 4.2.

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

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

Once a lower star is extracted, the level sets are created by means of `SplitIndexLowerStar`. This requires retrieving the filtration value $f(\sigma)$ for each simplex σ in the index-based lower star $\mathrm{Low}_I(v)$ of a vertex v. Searching for the set of cells with a specific multigrade takes at most $O(\log|\,\mathrm{Low}_I(v)\,|)$. Then, the overall cost of `SplitIndexLowerStar` is $C_{LS} = O(|\,\mathrm{Low}_I(v)\,| \cdot (d + \log|\,\mathrm{Low}_I(v)\,|))$.

For the last step, `HomotopyExpansion` classifies the cells in each level set. Preparing the two lists requires $O(|\,\mathrm{Lset}\,|\cdot|\log(|\,\mathrm{Lset}\,|)$ as for each simplex σ in $|\,\mathrm{Lset}\,|$, `num_undeclared_facets` requires visiting its facets which number is limited from above by a constant factor. Inserting each simplex in the list takes $O(\log|\,\mathrm{Lset}\,|)$.

Within the two while loops, each simplex enters 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 $O(|\,\mathrm{Lset}\,|)$ as the number of cofacets is not limited by any constant number,
- inserting the simplex in a list takes $O(\log|\,\mathrm{Lset}\,|)$.

Overall the contribution of `HomotopyExpansion` is $C_{HE} = |\,\mathrm{Lset}\,|(|\,\mathrm{Lset}\,| + 2\log(|\,\mathrm{Lset}\,|))$

4.3.2. Analysis of `ComputeDiscreteGradient` algorithm

By analizing the worst case complexity of the single auxiliary function we obtain a worst case complexity of

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

For the internal summation we can notice that in the worst case $|\,\mathrm{Lset}\,|$ is as big as the entire index-based lower star. Thus, we can overestimate

$$\sum_{\mathrm{Lset}\subseteq\mathrm{Low}_I(v)} C_{HE} = O(|\,\mathrm{Low}_I(v)\,|(|\,\mathrm{Low}_I(v)\,| + 2\log(|\,\mathrm{Low}_I(v)\,|)))$$

.

We recall that each k-simplex appears in the star of its $k+1$ vertices. If we overestimate the dimension of each simplex k with the dimension of the complex d, we can rewrite $\sum_{v\in S}|\,\mathrm{Star}(v)\,|$ as $|S|(d+1)$.

In a similar fashion, every simplex appears in exactly one index-based lower star. Thus, we can rewrite $\sum_{v\in S}(|\,\mathrm{Low}_I(v)\,|(|\,\mathrm{Low}_I(v)\,| + 2\log(|\,\mathrm{Low}_I(v)\,|)))$ as $|S|(|\,\mathrm{Low}_I(v)\,| + 2\log(|\,\mathrm{Low}_I(v)\,|)$ and C_{LS} as $|S|(d + \log(|\,\mathrm{Low}_I(v)\,|)$.

Moreover, we notice that in the worst case $\mathrm{Low}_I(v)$ is as big as Star. Based on this observation we can rewrite the overall complexity as

$$O(|S_0|\log|S_0| + |S|(d + \log|\,\mathrm{Star}\,| + |\,\mathrm{Star}\,|))$$

We should also mention that in applications we are often interested in filtrations defined on low dimensional complexes (i.e., with d=2,3). In such cases the number of simplices in each star becomes negligible leading us to a worst case complexity of $O(|S_0| \log |S_0| + |S|)$.

5. Proof of correctness and comparisons

In this section, we provide a formal proof of correctness for algorithm `ComputeDiscreteGradient`. We formalize the correctness statement as follows:

> "The vector field returned by algorithm `ComputeDiscreteGradient`, with input the multifiltration (X, f), is a discrete gradient field V and the corresponding Morse complex M is compatible with (X, f)."

The discrete gradient retrieved by algorithm `ComputeDiscreteGradient` is generated by the outputs of the auxiliary function `HomotopyExpansion` wich is run over a single level set Lset in the index-based lower star $\mathrm{Low}_I(v)$ of some vertex $v \in S_0$. The strategy to prove correctness consists in showing equivalence to the algorithm `Matching` introduced in [2], and fully proved to be correct in [46]. In order to do so, in Section 5.1 we first review how algorithm `Matching` acts. In Section 5.2, we prove the equivalence of `ComputeDiscreteGradient` and `Matching`.

5.1. Globally computing a discrete gradient for multiparameters

In this section, we recall the procedure applied by algorithm `Matching` [46, 2] to retrieve the same object as our proposed algorithm `ComputeDiscreteGradient` introduced in Section 4.

Input assumptions. As for the case of `ComputeDiscreteGradient`, the `Matching` algorithm acts on a simplicial complex S and a function $f : S_0 \longrightarrow \mathbb{R}^n$ required to be component-wise injective on vertexes and extended to higher dimensional simplices by function f as defined in Section 4. 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 has to be compatible both to the coface partial order $\ll$ among simplices and to the value ordering under f. Explicitly, J has to satisfy, the following property for every $\sigma \neq \tau \in S$,

$$\sigma \ll \tau \text{ or } f(\sigma) \lneq f(\tau) \quad \Rightarrow \quad J(\sigma) < J(\tau).$$

Description of `Matching`. The algorithm processes all simplices in S in a for-cycle. Simplices have to be processed according to increasing values of the indexing J. This implies that, as opposed to the local algorithm of Section 4, `Matching` cannot be broken into a parallel or distributed approach.

An auxiliary vector `classified` of length $|S|$ with Boolean entries is initialized with all entries set to `false`. For each simplex σ, the algorithm `Matching` checks whether σ is classified. An already classified simplex is not processed. A non-classified simplex σ is passed to an auxiliary function extracting the lower star of σ with respect to f

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

To do so, the auxiliary function visits $\mathrm{Star}(\sigma)$ to select each simplex τ satisfying condition $f(\tau) \preceq f(\sigma)$. Afterwards, an auxiliary function equivalent to the one in the algorithm `HomotopyExpansion` (pseudocode reported in Algorithm 2) is run with input $(\mathrm{Low}_f(\sigma), J)$. We recall that algorithm `ComputeDiscreteGradient`, instead, calls `HomotopyExpansion` with input $(\mathrm{Lset}, i(\leqslant_{\mathrm{Lex}}))$, where Lset is a level set inside an index-based lower star $\mathrm{Low}_I(v)$ for some vertex v, and $i(\leqslant_{\mathrm{Lex}})$ is the lexicographic order imposed on simplices by I by ordering the vertexes in decreasing order of values of I. Algorithm `HomotopyExpansion` returns a pair of lists $(V_{\mathrm{Low}_f(\sigma)}, M_{\mathrm{Low}_f(\sigma)})$ and all entries in the auxiliary vector `classified` corresponding to the simplices in the two lists are set to `true`. The global output is given by the independent contributions of all pairs $(V_{\mathrm{Low}_f(\sigma)}, M_{\mathrm{Low}_f(\sigma)})$. We denote by P the set of all simplices $\sigma \in S$ such that $\mathrm{Low}_f(\sigma)$ is processed by `Matching`, also called primary simplices.

Correctness for the algorithm ***`Matching`****.* It follows from Proposition 5 in [24] implying that, over each $\mathrm{Low}_f(\sigma)$, the output is a valid (local) discrete gradient. The fact that the union of all the independent discrete gradients returned by the auxiliary function `HomotopyExpansion` forms a discrete gradient is given by Theorem 3.8 in [46]. Finally, the compatibility with the input multifiltration (X, f) is guaranteed by Theorem 3.7 in [46]. In particular, Proposition 3.6 in [46] directly implies that lower stars $\mathrm{Low}_f(\sigma)$ for $\sigma \in P$ form a partition of X.

Summing up. Both algorithms `ComputeDiscreteGradient` and `Matching` build their output discrete gradient by running `HomotopyExpansion` on a partition of the input complex X:

- `Matching` finds the discrete gradient independently over each lower star $\mathrm{Low}_f(\sigma)$ with $\sigma \in P$,
- `ComputeDiscreteGradient` finds the discrete gradient independently over each level set $\mathrm{Lset} \in K_v$ with $v \in X_0$.

In the next section, we show the two algorithms to be equivalent by proving that the two partitions are the same.

5.2. Proof of equivalence

Under the notation of Section 5.1, in this section, we prove the equivalence between algorithms `ComputeDiscreteGradient` and `Matching`. The proof of all statements of this section is postponed to the appendix. First, we show that the two algorithms apply the auxiliary function `HomotopyExpansion` to the same partition of the input simplicial complex S. Then, we show the desired equivalence. In order for the partition into level sets Lset belonging to $\mathrm{Low}_I(v)$ for some vertex v to be coherent with the partition into $\mathrm{Low}_f(\sigma)$'s, it is crucial that the indexing I is well-extensible as defined in (4). Then, the following statement holds:

Lemma 12. Let I be a well-extensible indexing with respect to f. Then, for every $\sigma \in S$ there is exactly one vertex $v \in S_0$ such that $\mathrm{Low}_f(\sigma) \subseteq \mathrm{Low}_I(v)$ and $v \in \sigma$.

The lemma above states that each filtration-based lower star $\mathrm{Low}_f(\sigma)$ is contained in exactly one index-based lower star $\mathrm{Low}_I(v)$, with v a vertex of σ. The following lemma states that each level set $\mathrm{Lset} \subseteq L_I(v)$ coincides with the maximal of the lower stars $\mathrm{Low}_f(\sigma)$ contained therein.

Lemma 13. Let S be a simplicial complex, $f : S_0 \longrightarrow \mathbb{R}^n$ a component-wise injective function and $I : S_0 \longrightarrow \mathbb{R}$ a well-extensible indexing map with respect to f. Fix $\mathrm{Lset} \in K_v$ for some $v \in S_0$. Then, there exists a unique simplex $\sigma \in S$ such that $\mathrm{Low}_f(\sigma) = \mathrm{Lset}$ and, moreover, $\sigma \in \mathrm{Lset}$.

Lemma 12 and Lemma 13 are used to prove that both algorithms apply `HomotopyExpansion` to the same portions of the domain.

Lemma 14. Let $I : S \longrightarrow \mathbb{R}$ be a well-extensible indexing with respect to f and $J : S \longrightarrow \mathbb{R}$ a suitable input indexing for `Matching`. Let P be the set $\{\sigma \in S \mid$ `Matching` runs `HomotopyExpansion` over $\mathrm{Low}_f(\sigma)\}$. Then, for any level set $\mathrm{Lset} \in K_v$, there exists a $\sigma \in S$ such that

$$\mathrm{Lset} = \mathrm{Low}_f(\sigma) \Leftrightarrow \sigma \in P$$

Hence, the equivalence of the two approaches can be shown by focusing on the reduction of one input into the other.

Proposition 15. For every input (X, f) for `ComputeDiscreteGradient`, there exists a suitable input indexing $J : X \longrightarrow \mathbb{R}$ for `Matching` such that the output of `Matching`(X, f, J) equals that of `ComputeDiscreteGradient`(X, f).

As a corollary, we get the correctness of `ComputeDiscreteGradient`.

Corollary 16. Algorithm `ComputeDiscreteGradient` with input (X, f) returns a discrete gradient V compatible with the multifiltration induced by (S, f).

The following proposition completes the equivalence between the algorithms `ComputeDiscreteGradient` and `Matching`. In particular, it states that all possible gradients retrieved by the global algorithm `Matching` can be obtained by the local strategy of `ComputeDiscreteGradient`.

Proposition 17. Let (X, f, J) be a suitable input for `Matching`. Let $f : X_0 \longrightarrow \mathbb{R}^n$ be the restriction of f to the vertexes. Then, the output of `Matching`(X, f, J) equals that of `ComputeDiscreteGradient`(X, f), provided that, for each level set Lset under f, the function `HomotopyExpansion` is given (X, Lset, J) as input.

This last statement provides not simply correctness, but it also states that `ComputeDiscreteGradient` is as general as `Matching`. This means that the introduction of the well-extensible indexing over the vertexes which is needed in algorithm `ComputeDiscreteGradient` can be performed for every possible input.

5.3. Comparison of Asymptotical Complexity

In this section, we compare the computational complexity of algorithm `ComputeDiscreteGradient` , provided in Section 4.3, with that of algorithm `Matching` as discussed in [2]. As reviewed in Section 5, `Matching` and `ComputeDiscreteGradient` apply `HomotopyExpansion` to exactly the same level sets. In [2], authors express this cost as $O(|S| \cdot |\operatorname{Star}| \cdot \log |\operatorname{Star}|)$. The actual difference in complexity between the two algorithms is relative to the different number of stars which need to be visited in order to apply `HomotopyExpansion`. Indeed, `Matching` computes a lower star, for each level set of f. Instead, in `ComputeDiscreteGradient`, computes a lower star for each vertex in the input complex. This means that, in our case, the exact number of cells visited by `ComputeIndexLowerStar` is $|S|$. Instead, the number of cells visited by `Matching` depends on the number of level sets $|P|$ in the input dataset, satisfying

$$|X_0| \leqslant |P| \leqslant |X|.$$

In the case where $|P| = |X_0|$, the two algorithms visit the same number of cells. In the case where $|P| = |X|$, it means that each cell belongs to a different level set. Since each j-cell has $\binom{j+1}{k+1}$ different k-dimensional faces, each j-cell belongs to exactly $\binom{j+1}{k+1}$ cell stars. In that case, the amount of visited cells is given by

$$\sum_{\sigma \in X} |\operatorname{Star}(\sigma)| = \sum_{j=0}^{d} |X_j| \sum_{k=0}^{j+1} \binom{j+1}{k+1}. \tag{5}$$

In Section 5.4.1 we provide experimental results showing that, even if both algorithms works linearly in the number of cells, our approach guarantees an improved scalability.

5.4. Reducing a multiparameter filtration in practice

In this section we will compare experimentally our local preprocessing approach and the global matching algorithm introduced in [2]. Each *original dataset* is formed by a pair (S, f), where S is a simplicial complex and $f : S \longrightarrow \mathbb{R}^n$ 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 is a bifiltration that assign to each vertex its x and y coordinates (i.e., for $v = (x, y, z)$, $f(v) = (x, y)$). In order to guarantee a fair comparison, both algorithms have been implemented by using the FG_Multi library [47] which provides an efficient encoding for the triangle mesh as well as for the computed discrete gradient.

Representing a simplicial complex. A triangle mesh S is a simplicial 2-complex formed by vertices, edges, and triangles. The FG_Multi library [47] implements an incidence-based data structure for compactly encoding the relations among these simplices. Vertices and triangles are the only simplices which are explicitly encoded for a total of $|S_0| + |S_2|$ entities. Each vertex encodes the list of triangles incident while each triangle encodes a reference to its three vertices. Notice that, for each triangle σ referencing a vertex v, we also have that v references σ. Then, if each triangle references three vertices the triangle-vertex relation costs $3|S_0|$ while the vertex-triangle relation doubles this cost leading to a total of $7|X_2| + |X_0|$. The filtering function f is stored for each vertex by encoding a vector of floating point values, one value for each parameter.

Discrete gradient representation. The discrete gradient V is here encoded by adopting the representation described in [26]. The latter focuses on encoding all the gradient pairs locally to each triangle. The encoding uses the following rationale. Since each triangle σ can be paired with at most three edges and each edge can be paired with two vertices, locally for each triangle we have 9 possible pairs. If we consider also the possible pairs between an edge 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. that 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 1 byte per triangle and, thus, encoding the gradient only requires $|X_2|$ bytes. Notice that the latter approach has been generalized to tetrahedral meshes [27] and d-dimensional simplicial complexes [48].

5.4.1. Experimental results

The dataset used in this comparison is originated by three triangle meshes. For each mesh we obtain two refined versions of the latter by recursively applying the Catmull-Clark algorithm [49] to it. The nine triangle meshes composing the final dataset are described 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*.

Dataset	Parameters	Cells		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
Sphere	2	2.9M	0.28M	10.2
		11.7M	0.11M	10.2
		47.1M	0.46M	10.1
Gorilla	2	3.8M	0.4M	9.5
		15.2M	1.6M	9.4
		60.9M	6.4M	9.4

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*).

The experiments have been performed on a dual Intel Xeon E5-2630 v4 CPU at 2.20 Ghz with 64GB of RAM.

Timings are shown in Figure 1. The local approach takes between 0.89 seconds and 4.8 minutes to finish depending on the dataset and it is generally 7 times faster than our implementation of the global approach. Time performances show the practical efficiency of the local approach compared to the global one. As seen in 4.3, the expected asymptotical complexity over a triangle mesh S (i.e., $d = 2$) is $O(|S| \log |\operatorname{Star}|)$ for both algorithms. For the datasets considered in these tests the number of simplices withing each star is negligible which makes the algorithm linear in the number of simplices.

In Figure 2, we show the trends as the number of simplices increases. This confirms our argument that the number of stars to be retrieved and visited has a direct consequence on the algorithm complexity.

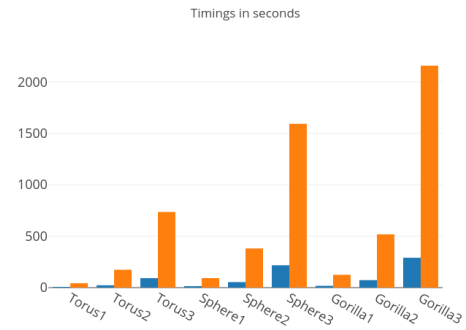


Figure 1: Timings required by `ComputeDiscreteGradient` (in blue) and `Matching` (in orange).

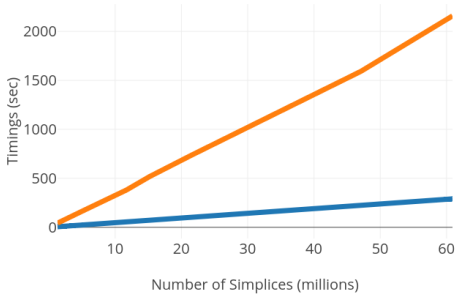


Figure 2: Trend in the timings for `ComputeDiscreteGradient` (in blue) and `Matching` (orange).

In the local case we need to process each vertex star only while the global approach requires processing a star for each multigrade.

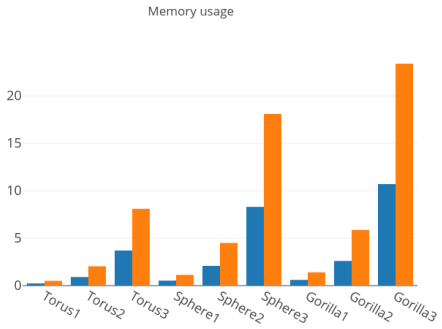


Figure 3: Maximum peaks of memory (in gigabytes) required by `ComputeDiscreteGradient` (in blue) and `Matching` (in orange).

Other than time efficiency, our divide-and-conquer strategy also requires a limited use of memory. The memory consumption is shown in Figure 3 where we are reporting the maximum peak of memory used by the two algorithms. The local approach uses up to 10 gigabytes of memory versus more than 20 of the global approach. The storage cost difference between the two implementations grows linearly in the number of simplices in the datasets. As seen in Section 5.3, the advantage is due to the different memory consumption at runtime. In particular, the global approach stores a global queue over all the simplices in the dataset and needs to track all classified cells. Both these steps are performed locally, over each level set, by the local approach.

6. Computing multiparameter persistence homology on reduced datasets

In this section, we evaluate the impact of our preprocessing method for the computation of the persistence module (Section 6.1) and of the persistence space (Section 6.2). Before presenting our results, we describe how to extract, from a discrete gradient V, the Morse complex that will be used as input of persistence computations.

Computing the Morse complex. We recall that a discrete gradient V implicitly represents a Morse complex M having the cells in one to one correspondence with the critical simplices of V. The incidence relations among the cells of M are described by the gradient paths originating and having destination in a pair of criticial simplices. Intuitively, two cells of M are incident to each other if there is a gradient path that connects the corresponding critical simplices in V.

To compute such relations we process all the critical cells of V. For each critical cell σ of dimension k, a breadth-first traversal is performed as follows. From σ we extract its incident $(k-1)$-simplices. For each $(k-1)$-simplex we extract its paired 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 register the two cells σ and τ as incident to each other.

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

6.1. Computing the persistence module

In this section, we evaluate the impact of the reduction method to the computation of the persistence module. The persistence module will be computed by means of the open-source library Topcat [33], which is currently the only available library for this task. Due to its strong limitations in terms of time and memory costs, we used a simplified dataset for our experiments. We use six triangle meshes of limited size, three representing a torus and the other three representing a sphere. The bifiltration used for each triangulation is defined by the x and y coordinates of the vertexes. Table 2 presents a description of the dataset. Number of entities (column *Cells*) and number of multigrades (column *Grades*) are reported for each simplicial complex (column *Original*) and each Morse complex (column *Reduced*).

	Original				Reduced				Reduction Time
			Persistence Module				Persistence Module		
Dataset	Cells	Grades	Time	Memory	Cells	Grades	Time	Memory	
Sphere	38	8x8	0.3	0.24	4	5x5	0.18	0.1	0.0264
	242	42x42	4.4	0.86	20	10x10	0.28	0.2	0.0244
	2882	482x482	-	-	278	92x89	24.3	1.5	0.0473
Torus	96	16x16	0.5	0.1	8	9x9	0.25	0.2	0.0255
	4608	768x768	-	-	128	65x66	7.96	2.4	0.0643
	7200	1200x1200	-	-	156	70x80	12.05	3.0	0.0815

Table 2: Timings (in seconds) and storage costs (in gigabytes) for the persistence module retrieval over the original (columns *Original*) and the corresponding reduced (columns *Reduced*) datasets. Columns *Cells* and *Grades* reports the number of cell in the dataset and the number of grades along each parameter, respectively. Missing entries indicate where the Topcat library run out of memory. Column *Reduction Time* explicit the timings (in seconds) for obtaining the reduced cell complex.

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 to make it read multifiltrations defined over general cell complexes.

We compute the persistence module of each dataset, both the original and the reduced ones, 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 Gigabyte) required for computing the persistence module. Where no result is reported, the Topcat library runs out of memory. We notice that timings are always in favor of the Morse complex. 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 algorithm as it is mainly affected by the number of cells and the number of multigrades. The use of the reduction approach reduces this problem by shrinking the number of boths. In our experiment all successful executions have used a limited amount of memory, significantly below the machine limit of 64GB. This suggests a dramatic increase of memory usage in the ones where the computations have failed. For instance, the failure of the test over, for instance, the Sphere dataset with 2882 cells and 482x482 multifiltration multigrades suggests that computing the persistence module on a reduced dataset 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 not that of overcoming the limitation of Topcat. Column *Reduction Time* reports the partial timings required for computing the reduced cell complex. These include the timings contribution of running `ComputeDiscreteGradient` together with the retrieval of the boundary matrix through the algorithm [48]. We can notice that the measured reduction timings, ranging from 0.0244 to 0.0815 seconds are negligible with respect to the time required for computing the persistence module.

6.2. Computing the persistence space

In this section, we evaluate the impact of the reduction method on the computation of the persistence space. We recall that the persistence space can be computed via the *foliation method* introduced in [35]. The foliation method consists in restricting

each multifiltration to several linearization, i.e., single-parameter filtrations, called *slices*. On each slice, any computational technique from single-parameter persistence can be applied to obtain a persistence diagram. The collection of persistence diagrams, gives an approximation of the persistence space. The number of slices to consider varies based on the application. As a rule of thumb, the more slices we consider, the more accurate is the approximation of the persistence space obtained.

The foliation method. The foliation method can be seen as a two-step approach for the computation of the persistence space. For sake of simplicity we present a description for the foliation method specific for a bifiltration ϕ.

The first step consists of uniformely selecting ω^2 lines of non-negative slope in the Euclidean plane. First we compute the extremal values for the bifiltration ϕ. For each component $i = 1, 2$, we compute parameters $C_i := \max_{x \in S} \phi_i(x)$ and $c_i := \min_{x \in S} \phi_i(x)$. Each line l that we will extract is determined by two parameters: λ, i.e. the slope coefficient, and b, i.e. the base point. For creating ω^2 we uniformely select ω values for both λ and b. Values of λ range from 0 to $\frac{\pi}{2}$. Value of b are computed as follow. For each value of λ, we select the bisector of the II and IV quadrant with slope λ. The projections of points (c_1, C_2) and (C_1, c_2) over the bisector will limit the interval on which sampling the values of b. All possible values for λ and b are combined to represent the ω^2 possible lines. Each line $l = (m, b)$ is the line of unit vector with $m = (\cos(\lambda), \sin(\lambda))$ and passing through b.

For each line extracted $l = (m, b)$ we create a new 1-dimensional filtration over the simplices of S. Each simplex σ obtains the filtration value Φ^l according to l as: $\Phi^l(\sigma) := \min_{i=1,2} m_i \cdot \max_{i=1,2} \frac{\phi_i(\sigma) - b_i}{m_i}$. The obtained filtration is used to compute classic persitent homology. The resulting persistence pairs within each persistence diagram will form the approximated persistence space.

6.2.1. Computing the persistence space of the Morse complex.

In this subsection we present results for evaluating the impact of our reduction approach when computing the persistence space. The foliation method requires the choice of two parameters: the number of slices and the method used for computing classic persistent homology. In the following we will present results providing insights on both, either by varying the number of slices (between 2 and 100) or by varying the method for computing persistent homology.

Datasets considered are from the Princeton Shape Benchmark [50]. Table 3 describes the dataset and the corresponding results obtained when computing the persistence space by using 100 slices and by using the standard algorithm implemented in PHAT. For each dataset 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 cells (column *Cells*) and the average number of persistence pairs found per slice (column *Pairs*).

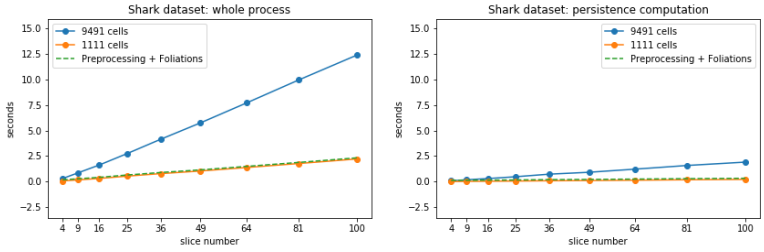
Timings are reported separately for the computation of the Morse complex (column *Reduction*), for the extraction of slices (column *Line Extraction*) and for the actual computationa of the persistence space (column *Foliations Time*). The latter is formerly subdivided into three partial timings accounting for the construction of the boundary matrix (column *Building Pers. input*), computation of persistent homology (column *Computing*

Persistence), reindexing of the persistence pairs according to the multifiltration (column *Reindexing Pers. output*). Column *Foliations Total* shows the sum of the partial timings.

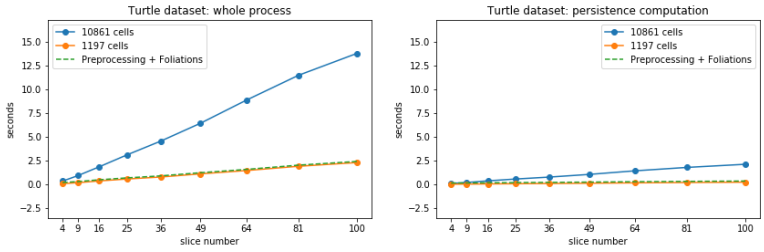
						Foliations Time			
Dataset	Cells	Pairs		Reduction Time	Line Extraction	Building Pers. input	Computing Persistence	Reindexing Pers. output	Foliations Total
Shark	9491	4744	(81.4)	0.11	0.86	9.04	1.91	1.45	12.42
	1111	554				1.15	0.21	0.84	2.22
Turtle	10861	5426	(8.8)	0.12	0.63	10.21	2.11	1.53	13.87
	1197	594				1.22	0.22	0.84	2.29
Gun	27826	13873	(10.2)	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	(79.5)	1.14	0.85	118.14	26.56	10.33	155.91
	10955	5286				11.09	2.26	1.65	15.01

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 reduced dataset considered over the same 100 slices. Column *Cells* reports the number of cells in the multifiltration. Column *Pairs* reports the average number of persistence pairs found per slice. In parantheses, the number of pairs with positive persistence (equal for original and reduced datasets). Reported timings are subdivided into phases a) (column *Reduction Time*), b) (column *Line Extraction*), and c) (column *Foliations Time*). The latter subdivided into step 1) (column *Building Pers. input*), step 2) (column *Computing Persistence*), and 3) (column *Reindexing Pers. output*).

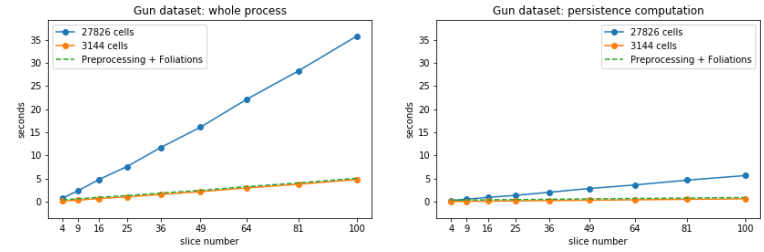
We notice that, by reducing the number of cells of approximately one order, we get a one-order reduction on all timings. Looking at column *Line Extraction* we notice that the extraction of the lines has little to no differences across the triangle meshes. This happens because in this case we are always 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 this values. Both this phase and the following one (i.e., the actual computation of persitent homology) are affected by the number of input cells, indeed the results for the reduced dataset reflect the one-order reduction in the number of cells. Results shown in column *Reducing Pers. Output* depend on the number of persistence pairs found. The difference in the results obtained with the original triangle mesh and the corresponding Morse complex suggests that our reduction step let us consider fewer spurious persistence pairs. Column *Foliation Total* indicates timings for computing the persistence space as a whole. The total timings required by a reduced dataset 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).



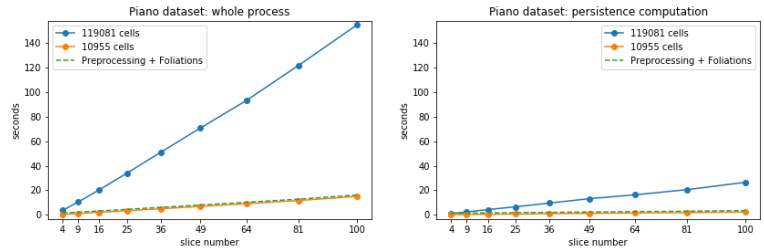
(a)



(b)



(c)



(d)

Figure 4: Time performances plotted with respect to a number of slices varying form 4 to 100 over the same dataset. Datasets considered are triangle meshes: (a) Shark, (b) Turtle, (c) Gun, and (d) Piano. In all the figures, on the left, performances are indicated in blue for the original dataset and in orange for the corresponding reduced dataset. On the right, we show the same plotting with respect to step 2) in the foliation phase only.

Varying the number of slices. In Figures 4, we compare the time performances achieved by the foliation method using a number of slices ranging from 4 to 100. The one-parameter persistence over each slice 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, the green dotted line presents 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 overlapp indicating that the preprocessing step used for computing the Morse complex is almost negligible with respect to the computation of the persistence space.

We also notice the linear dependency of the process from the number of slices. For reduced datasets (orange line), the slope coefficient is smaller than for the original datasets (blue line). This is more evident for global timings suggesting that a preprocessing reduction is preferable independently from the number of considered slices. Notice that, limitedly to the computation of persistent homology when using 4 slices, we get the blue line just below the green dashed line. This is the only exception where the preprocessing step could be avoided.

Our tests confirm that the time complexity in the foliation method primarily depends on the number of slices considered. Our reduction approach impacts on 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.

Varying the persistent homology computation algorithm. In Figure 5, we report the results obtained by using the five algorithms implemented in PHAT for computing persistent homology on the original (a) and reduced datasets (b) over 100 slices. Also here we can notice that computing persistent homology on the reduced datasets takes an order of magnitude less than on the original triangle mesh.

In our test, performances of the standard algorithm are comparable to the other appraoches implementing optimizations. This was not expected according to [12]. This can be explained, in part, by the low dimension of the chosen meshes and their limited size, but we should also notice that in the foliation method we have to run the same algorithm multiple times. For this reason, the number of slices may have a more profound impact on the overall timing than the optimization implemented on the single slice. On top of that, our results already suggest that a multiparameter reduction strategy is preferable since it can be computed only once and used for all the slices.

7. Concluding remarks

In Section 4, we have proposed a new preprocessing algorithm for MPH suitable for applications to real-sized data sets. We have highlighted the local character of our approach as opposed to the global character of the equivalent existing approach in [2]. Our complexity analysis makes it clear that the two preprocessing algorithm might have the same worst-case time complexity depending on the input. In fact, we have discussed how the presence of multiparameter in place of one parameter affects the average case rather than the worst-case time.

Concerning the issue of quantifying the advantage of our proposed MPH preprocessing to computing the persistence module, our local MPH preprocessing increases of up to about 50 times the size of the input complex that can be treated, and up to about 250 times the size of the filtration that can be treated. In all considered datasets (rather small), the reduction allows to complete the pipeline. Some non-reduced datasets have failed for running out of memory. These failures for rather small datasets suggest that, at the moment, our preprocessing is not enough to make the persistence module computation feasible over real-size data. In particular, we detected memory costs as a bottleneck for current persistence module computational methods. Optimizations of current algorithms require better handling of memory usage in terms of size of the multifiltration and number of cells in the input complex.

Concerning the issue of quantifying the advantage of our proposed MPH preprocessing to computing the persistence space, our local MPH preprocessing shows its advantages in all considered datasets. The foliation method allows to retrieve the persistence space by multiple iterations of PH computations. One goal was that of evaluating the trade-off between the number of iterations and the advantages of the MPH preprocessing. We found that, in all considered datasets, the reduced datasets outperforms the corresponding original dataset, regardless of the number of iterations applied. Instead, when limited to the PH computation timings, only the case of 4 slices shows advantages for non-processed datasets. This is coherent with other comparisons made for PH efficiency such as [12, 51] (non-processed datasets should be preferable for few iterations). Moreover, we have found that our MPH preprocessing is preferable over all considered PH optimized algorithm to be iterated. Finally, we notice that, in our test, performances of the standard algorithm are comparable to the considered optimizations. This was not expected according to [12]. Our choice of triangle meshes datasets, that is with low geometric dimensions and cell stars limited in size, may explain that results.

7.1. Future Work

The results discussed in this paper suggest future works in multiple directions. From a computational point of view, the results obtained motivate the need for studying and developing finer implementations of the available techniques, especially in the case of the persistence module retrieval.

Additionally, we think that the idea of a discrete gradient compatible with a multifiltration deserves further insights from the theoretical point of view. Currently, we are working on defining a notion of optimal reduction for a multifiltration. The optimality should extend the property satisfied by the algorithm [24] equivalent to our proposed one in the case of a single parameter filtration.

Moreover, comparisons between reductions based on critical cells of a discrete gradient and other bifiltration reductions, such as the one implemented in RIVET [39], should be studied both theoretically and computationally. We are working on this problem by trying to relate critical cells in a multifiltration to the notion of *multigraded Betti numbers*.

Finally, the study of critical cells of a multifiltration may be addressed from the topology-based visualization perspective. The critical cells of a multifiltration might be interpreted as a fully discrete counterpart to other approaches to visualize mutual behavior of multiple scalar fields, such as Pareto sets [52] or Jacobi sets [53]. At the moment, we are working on defining an incidence structure among critical cells arranged

into a compact graph to be compared to other similar structures available for piecewise linear functions such as the Reachability graph [52].

8. Acknowledgements

This work has been partially supported by the US National Science Foundation [grant number IIS-1116747]. The authors wish to thank Michael Kerber for interesting discussion on the results.

References

References

[1] R. Forman, Morse Theory for Cell Complexes, Advances in Mathematics 134 (1998) 90–145. doi:10.1006/aima.1997.1650.

[2] M. Allili, T. Kaczynski, C. Landi, F. Masoni, Algorithmic Construction of Acyclic Partial Matchings for Multidimensional Persistence, in: Kropatsch W., Artner N., Janusch I. (Eds.), Discrete Geometry for Computer Imagery. DGCI 2017. Lecture Notes in Computer Science, vol 10502, Springer, Cham, 2017, pp. 375–387. doi:10.1007/978-3-319-66272-5_30.

[3] S. Lefschetz, Algebraic Topology, Vol. 27 of Colloquium Publications, American Mathematical Society, 1942.

[4] M. Mrozek, B. Batko, Coreduction homology algorithm, Discrete and Computational Geometry 41 (1) (2009) 96–118. doi:10.1007/s00454-008-9073-y.

[5] 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 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 730–739. doi:10.1007/978-3-642-10631-6_74.

[6] K. Mischaikow, V. Nanda, Morse Theory for Filtrations and Efficient Computation of Persistent Homology, Discrete & Computational Geometry 50 (2) (2013) 330–353. doi:10.1007/s00454-013-9529-6.

[7] H. Edelsbrunner, D. Letscher, A. Zomorodian, Topological persistence and simplification, Discrete and Computational Geometry 28 (4) (2002) 511–533. doi:10.1007/s00454-002-2885-2.

[8] C. Chen, M. Kerber, Persistent homology computation with a twist, in: 27th European Workshop on Computational Geometry, Vol. 45, 2011, pp. 28–31. doi:10.1.1.224.6560.

[9] V. de Silva, D. Morozov, M. Vejdemo-Johansson, Dualities in persistent (co)homology, Inverse Problems 124003 (12) (2011) 16. doi:10.1088/0266-5611/27/12/124003.

[10] N. Milosavljević, D. Morozov, P. Skraba, Zigzag Persistent Homology in Matrix Multiplication Time, in: Proc. 27th Ann. Symp. Comput. Geom., SoCG '11, ACM, New York, NY, USA, 2011, pp. 216–225. doi:10.1145/1998196.1998229.

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

[12] U. Bauer, M. Kerber, J. Reininghaus, Clear and Compress: Computing Persistent Homology in Chunks, in: arXiv preprint arXiv:1303.0477, Springer International Publishing, 2013, pp. 1–12. doi:10.1007/978-3-319-04099-8__7.

[13] 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. doi:10.1145/2582112.2582165.

[14] J. D. Boissonnat, T. K. Dey, C. Maria, The Compressed Annotation Matrix: en Efficient Data Structure for Computing Persistent Cohomology, in: Algorithms - ESA 2013, Vol. 8125 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 695–706. doi:10.1007/978-3-642-40450-4_59.

[15] 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 of Lecture Notes in Computer Science, Springer, 2012, pp. 189–200. doi:10.1007/978-3-642-31155-0_17.

[16] P. Dłotko, H. Wagner, Simplification of complexes for persistent homology computations, Homology, Homotopy and Applications 16 (1) (2014) 49–63. doi:10.4310/HHA.2014.v16.n1.a3.
[17] M. Mrozek, B. Batko, Coreduction Homology Algorithm, Discrete & Computational Geometry 41 (1) (2009) 96–118. doi:10.1007/s00454-008-9073-y.
[18] M. Mrozek, T. Wanner, Coreduction homology algorithm for inclusions and persistent homology, Computers & Mathematics with Applications 60 (10) (2010) 2812–2833. doi:10.1016/j.camwa.2010.09.036.
[19] P. Dłotko, T. Kaczynski, M. Mrozek, T. Wanner, Coreduction Homology Algorithm for Regular CW-Complexes, Discrete and Computational Geometry 46 (2) (2011) 361–388. doi:10.1007/s00454-010-9303-y.
[20] M. Mrozek, P. Pilarczyk, N. Zelazna, Homology algorithm based on acyclic subspace, Computers and Mathematics with Applications 55 (11) (2008) 2395–2412. doi:10.1016/j.camwa.2007.08.044.
[21] H. King, K. Knudson, N. Mramor, Generating Discrete Morse Functions from Point Data, Experimental Mathematics 14 (4) (2005) 435–444. doi:10.1080/10586458.2005.10128941.
[22] N. Shivashankar, S. Maadasamy, V. Natarajan, Parallel computation of 2d morse-smale complexes, IEEE Trans. Vis. Comput. Graph. 18 (10) (2012) 1757–1770. doi:10.1109/TVCG.2011.284.
[23] N. Shivashankar, V. Natarajan, Parallel computation of 3d morse-smale complexes, Comput. Graph. Forum 31 (3) (2012) 965–974. doi:10.1111/j.1467-8659.2012.03089.x.
[24] V. Robins, P. J. Wood, A. P. Sheppard, Theory and algorithms for constructing discrete Morse complexes from grayscale digital images, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (8) (2011) 1646–1658. doi:10.1109/TPAMI.2011.95.
[25] D. Günther, J. Reininghaus, H. Wagner, I. Hotz, Efficient computation of 3d morse–smale complexes and persistent homology using discrete morse theory, The Visual Computer 28 (10) (2012) 959–969. doi:10.1007/s00371-012-0726-8.
[26] R. Fellegara, F. luricich, L. De Floriani, K. Weiss, Efficient computation and simplification of discrete morse decompositions on triangulated terrains, in: Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '14, ACM, 2014, pp. 223–232. doi:10.1145/2666310.2666412.
[27] K. Weiss, F. Iuricich, R. Fellegara, L. De Floriani, A primal/dual representation for discrete Morse complexes on tetrahedral meshes, Computer Graphics Forum 32 (3pt3) (2013) 361–370. doi:10.1111/cgf.12123.
[28] U. Fugacci, F. Iuricich, L. De Floriani, Efficient computation of simplicial homology through acyclic matching, in: 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, IEEE, 2014, pp. 587–593. doi:10.1109/SYNASC.2014.84.
[29] G. Carlsson, A. Zomorodian, The theory of multidimensional persistence, in: SoCG '07 Proceedings of the twenty-third annual symposium on Computational geometry, Vol. 392, ACM New York, Gyeongju, South-Korea, 2007, pp. 184–193. doi:10.1145/1247069.1247105.
[30] O. Gäfvert, Algorithms for Multidimensional Persistence, Master thesis, KTH Royal Institute of Technology.
[31] J. R. Munkres, Elements of Algebraic Topology, Perseus Books, 1984.
[32] M. K. Agoston, Computer Graphics and Geometric Modeling: Mathematics, Springer Verlag London Ltd., 2005.
[33] Oliver Gäfvert, TopCat: a Java library for computing invariants on multidimensional persistence modules (2016) [cited 2017-03-30].
URL https://github.com/olivergafvert/topcat
[34] 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, Vol. 7749 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 180–191. doi:10.1007/978-3-642-37067-0_16.
[35] S. Biasotti, A. Cerri, P. Frosini, D. Giorgi, C. Landi, Multidimensional size functions for shape comparison, Journal of Mathematical Imaging and Vision 32 (2) (2008) 161–179. doi:10.1007/s10851-008-0096-z.
[36] F. Cagliari, B. Di Fabio, M. Ferri, One-dimensional reduction of multidimensional persistent homology, Proceedings of the American Mathematical Society 138 (08) (2010) 3003–3003. doi:10.1090/S0002-9939-10-10312-8.
[37] S. Biasotti, A. Cerri, P. Frosini, D. Giorgi, A new algorithm for computing the 2-dimensional matching distance between size functions, Pattern Recognition Letters 32 (14) (2011) 1735–1746. doi:10.1016/j.patrec.2011.07.014.
[38] S. Biasotti, A. Cerri, D. Giorgi, M. Spagnuolo, PHOG: Photometric and geometric functions for

textured shape retrieval, Computer Graphics Forum 32 (5) (2013) 13–22. doi:10.1111/cgf.12168.
[39] M. Lesnick, M. Wright, Interactive Visualization of 2-D Persistence Modules, ArXiv preprint (2015) 1–75arXiv:1512.00180.
[40] K. P. Knudson, A refinement of multi-dimensional persistence, Homology, Homotopy and Applications 10 (1) (2008) 259–281. doi:10.4310/HHA.2008.v10.n1.a11.
[41] D. Eisenbud, The Geometry of Syzygies: A second course in Commutative Algebra and Algebraic Geometry, Springer, New York, NY, 2005. doi:10.1007/b137572.
[42] A. Cerri, P. Frosini, C. Landi, A global reduction method for multidimensional size graphs, Electronic Notes in Discrete Mathematics 26 (2006) 21–28. doi:10.1016/j.endm.2006.08.004.
[43] M. Allili, T. Kaczynski, C. Landi, Reducing complexes in multidimensional persistent homology theory, Journal of Symbolic Computation 78 (C) (2017) 61–75. doi:10.1016/j.jsc.2015.11.020.
[44] H. Edelsbrunner, E. P. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, ACM Transactions on Graphics 9 (1) (1990) 66–104. doi:10.1145/77635.77639.
[45] D. Canino, L. D. Floriani, K. Weiss, IA*: An adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions, Computers & Graphics 35 (3) (2011) 747 – 753, shape Modeling International (SMI) Conference 2011. doi:https://doi.org/10.1016/j.cag.2011.03.009.
[46] M. Allili, T. Kaczynski, C. Landi, F. Masoni, A New Matching Algorithm for Multidimensional Persistence (Nov 2015). arXiv:1511.05427.
[47] Federico Iuricich, MDG: a C++ library for computing discrete gradients on multivariate data (2018) [cited 2018-09-30].
URL https://github.com/IuricichF/fg_multi
[48] U. Fugacci, F. Iuricich, L. De Floriani, Computing discrete Morse complexes from simplicial complexes, Graphical ModelsManuscript submitted for publication.
[49] E. Catmull, J. Clark, Recursively generated b-spline surfaces on arbitrary topological meshes, Computer-Aided Design 10 (6) (1978) 350 – 355. doi:https://doi.org/10.1016/0010-4485(78)90110-0.
[50] P. Shilane, P. Min, M. Kazhdan, T. Funkhouser, The Princeton Shape Benchmark, in: Shape Modeling Applications, 2004. Proceedings, Genova, Italy, 2004, pp. 167–178. doi:10.1109/SMI.2004.1314504.
[51] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, H. A. Harrington, A roadmap for the computation of persistent homology, EPJ Data Science 6 (1) (2017) 17. doi:10.1140/epjds/s13688-017-0109-5.
[52] L. Huettenberger, C. Heine, C. Garth, Decomposition and simplification of multivariate data using Pareto sets, IEEE Transactions on Visualization and Computer Graphics 20 (12) (2014) 2684–2693. doi:10.1109/TVCG.2014.2346447.
[53] H. Edelsbrunner, J. L. Harer, Jacobi sets, in: Foundations of Computational Mathematics: Minneapolis, 2002, Vol. 312 of London Mathematical Society Lecture Note Series, Cambridge University Press, 2004, pp. 37–57. doi:10.1017/CBO9781139106962.003.

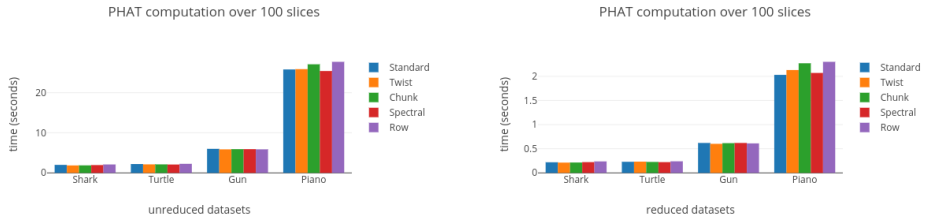
Appendix

In this appendix, we report the proofs of results we omitted in Section 5.2.

Proof of Lemma 1. Let σ be a simplex in S. It is easy to see that the $\mathrm{Low}_I(v)$'s form a partition of S. Hence, there exists a unique vertex $v \ll \sigma$ such that σ belongs to $\mathrm{Low}_I(v)$. Let τ be a simplex in $\mathrm{Low}_{\tilde{f}}(\sigma)$. By definition of lower star, $\tau \gg \sigma$ and $\tilde{f}(\tau) \preceq \tilde{f}(\sigma)$. The former condition implies that $\tilde{I}(\tau) \geqslant \tilde{I}(\sigma)$ and that $v \in \tau$. The latter condition together with the assumption on I being well-extensible give $\tilde{I}(\tau) \leqslant \tilde{I}(\sigma)$. Hence, $\tilde{I}(\tau) = \tilde{I}(\sigma) = I(v)$, which concludes the proof.

Proof of Lemma 2. Let v and Lset be as in the lemma statement. In order to prove uniqueness, suppose there are two simplices $\sigma, \sigma' \in S$ such that $\mathrm{Low}_f(\sigma) = \mathrm{Lset} = \mathrm{Low}_f(\sigma')$. Notice that, $\sigma, \sigma' \in \mathrm{Lset}$, since any simplex belongs to its own lower star.

Moreover, condition $\mathrm{Low}_f(\sigma') = \mathrm{Low}_f(\sigma)$ implies that $\sigma' \in \mathrm{Low}_f(\sigma)$ and $\sigma \in \mathrm{Low}_f(\sigma')$ at the same time. By definition of lower star, we get in particular $\sigma' \ll \sigma$ and $\sigma \ll \sigma'$, that is $\sigma' = \sigma$. In order to prove existence, we define σ to be the intersection of all simplices belonging to Lset. We know that $v \in \tau$ for any $\tau \in$ Lset and that when two simplices intersect they do it in a single shared face, so σ is a non-empty simplex. Notice that σ belongs to $\mathrm{Low}_I(v)$ and, since f is component-wise injective, for any $i = 1, \ldots, n$ $\tau \in$ Lset, it holds that $\tilde{f}_i(\tau) = f_i(w)$ with $w \in \tau$. This implies, $w \in \sigma$ and, thus, $\sigma \in$ Lset. We claim that $\mathrm{Low}_f(\sigma) =$ Lset. Indeed, Lset is trivially part of $\mathrm{Low}_f(\sigma)$ since any $\tau \in$ Lset has the same value under $\tilde{f}$ and $\tau \gg \sigma$. Conversely,, let τ be a simplex in $\mathrm{Low}_f(\sigma)$. Since I is well-extensible, the Lemma 12 ensures that $\tau \in \mathrm{Low}_I(v)$. Notice that, for a general $\tau \in \mathrm{Low}_f(\sigma)$, it holds that $\tilde{f}(\tau) = \tilde{f}(\sigma)$. Being $\sigma \in$ Lset, we get $\tau \in$ Lset, which proves our claim and concludes the proof.



(a) (b)

Figure 5: Timings for 100 slice computations via the five algorithms for persistence implemented in the PHAT library. Original datasets (a) are compared to reduced datasets (b).