The Compressed Annotation Matrix: an Efficient Data Structure for Computing Persistent Cohomology

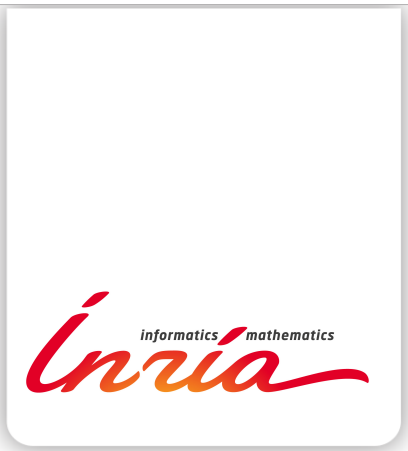
Jean-Daniel Boissonnat, Tamal K. Dey, Clément Maria

The Compressed Annotation Matrix: an Efficient Data Structure for Computing Persistent Cohomology

Jean-Daniel Boissonnat , Tamal K. Dey , Clément Maria

RESEARCH REPORT
N° 8195
April 2013
Project-Team GEOMETRICA

ISSN 0249-6399 ISRN INRIA/RR--8195--FR+ENG



The Compressed Annotation Matrix: an Efficient Data Structure for Computing Persistent Cohomology

Jean-Daniel Boissonnat *, Tamal K. Dey **, Clément Maria *

Project-Team GEOMETRICA

Research Report n° 8195 — April 2013 — 14 pages

Abstract: The persistent homology with coefficients in a field $\mathbb{F}$ coincides with the same for cohomology because of duality. We propose an implementation of a recently introduced algorithm for persistent cohomology that attaches annotation vectors with the simplices. We separate the representation of the simplicial complex from the representation of the cohomology groups, and introduce a new data structure for maintaining the annotation matrix, which is more compact and reduces substancially the amount of matrix operations. In addition, we propose heuristics to simplify further the representation of the cohomology groups and improve both time and space complexities. The paper provides a theoretical analysis, as well as a detailed experimental study of our implementation and comparison with state-of-the-art software for persistent homology and cohomology.

Key-words: persistent cohomology, implementation, data structure, annotation, simplex tree

* INRIA Sophia Antipolis-Méditerranée, {jean-daniel.boissonnat, clement.maria}@inria.fr
** The Ohio State University, tamaldey@cse.ohio-state.edu

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Une Structure de Données Efficace pour le Calcul de Cohomologie Persistente

Résumé : Le calcul d'homologie et de cohomologie persistentes dans un corps $\mathbb{K}$ coincident par dualité. Nous proposons une implémentation de l'algorithme de cohomologie persistente, qui associe un vecteur d'annotation à chaque simplexe du complexe simplicial. Nous séparons la représentation du complexe de celle des groupes de cohomologie, et nous introduisons une nouvelle structure de données pour représenter les annotations. Cette structure est plus compacte et permet de réduire le nombre d'opérations dans la matrice. Nous introduisons également des heuristiques pour réordonner les simplexes, ceci pour simplifier les groupes de cohomologie et améliorer les performances en temps et espace de l'implémentation. Cet article fournit une analyse de complexité ainsi qu'une étude expérimentale détaillée de l'implémentation. Nous comparons notamment notre implémentation avec les librairies disponibles de calculs d'homologie et de cohomologie persistentes.

Mots-clés : cohomologie persistente, implémentation, structure de données, annotation, simplex tree

1 Introduction

Persistent homology [9] is an algebraic method for measuring the topological features of a space induced by the sublevel sets of a function. Its generality and stability with regard to noise have made it a widely used tool for the study of data, where it does not need any knowledge a priori. A common approach is the study of the topological invariants of a nested family of simplicial complexes built on top of the data, seen as a set of points in a geometric space. This approach has been successfully used in various areas of science and engineering, as for example in sensor networks, image analysis, and data analysis where one typically needs to deal with big data sets in high dimensions. Consequently, the demand for designing efficient algorithms and implementation to compute the persistent homology of filtered simplicial complexes has grown.

The first persistence algorithm [10,13] can be implemented by reducing a matrix defined by face incidence relations, through column operations. The running time is $O(m^3)$ where m is the number of simplices of the simplicial complex and, despite good performance in practice, Morozov proved that this bound is tight [12]. Recent optimizations taking advantage of the special structure of the matrix to be reduced have led to significant progress in the theoretical analysis [11,4] as well as in practice [4,1].

A different approach [7,6] interprets the persistent homology groups in terms of their dual, the persistent cohomology groups. The cohomology algorithm has been reported to work better in practice than the standard homology algorithm [6] but this advantage seems to fade away when compared to the recent optimized homology algorithms [1]. An elegant description of the cohomology algorithm, using the notion of annotations, has been introduced in [8] and used to design more general algorithms for maintaining cohomology groups under simplicial maps.

In this work, we propose an implementation of the annotation-based algorithm for computing persistent cohomology. A key feature of our implementation is a distinct separation between the representation of the simplicial complex and the representation of the cohomology groups. In our implementation, the simplicial complex can be represented either by its Hasse diagram or by using the more compact simplex tree [2]. The cohomology groups are stored in a new data structure that represents a Compressed version of the Annotation Matrix. As a consequence, the time and space complexities of our algorithm depend mostly on properties of the cohomology groups we maintain along the computation and only linearly on the size of the simplicial complex.

Moreover, maintaining the simplicial complex and the cohomology groups separately allows us to reorder the simplices while keeping the same persistent cohomology. This significantly reduces the size of the cohomology groups to be maintained, and improves considerably both the time and memory performance as shown by our detailed experimental analysis on a variety of examples. Our method compares favourably with state-of-the-art software for computing persistent homology and cohomology.

Background: A *simplicial complex* is a pair $\mathcal{K} = (V, S)$ where V is a finite set whose elements are called the *vertices* of $\mathcal{K}$ and S is a set of non-empty subsets of V that is required to satisfy the following two conditions : 1. $p \in V \Rightarrow \{p\} \in S$ and 2. $\sigma \in S, \tau \subseteq \sigma \Rightarrow \tau \in S$. Each element $\sigma \in S$ is called a *simplex* or a *face* of $\mathcal{K}$ and, if $\sigma \in S$ has precisely $s+1$ elements ($s \geq -1$), σ is called an s-simplex and its dimension is s. The dimension of the simplicial complex $\mathcal{K}$ is the largest k such that S contains a k-simplex. We define $\mathcal{K}^p$ to be the set of p-dimensional simplices of $\mathcal{K}$, and note its size $|\mathcal{K}^p|$. Given two simplices τ and σ in $\mathcal{K}$, τ is a subface (resp. coface) of σ if $\tau \subseteq \sigma$ (resp. $\tau \supseteq \sigma$). The *boundary* of a simplex σ, denoted $\partial\sigma$, is the set its subfaces with codimension 1.

A *filtration* [9] of a simplicial complex is an order relation on its simplices which respects inclusion. Consider a simplicial complex $\mathcal{K} = (V, S)$ and a function $\rho : S \to \mathbb{R}$. We require ρ to be monotonic in the sense that, for any two simplices $\tau \subseteq \sigma$ in $\mathcal{K}$, ρ satisfies $\rho(\tau) \leq \rho(\sigma)$. We will call $\rho(\sigma)$ the *filtration value* of the simplex σ. Monotonicity implies that the sublevel sets

$\mathcal{K}(r) = \rho^{-1}(-\infty, r]$ are subcomplexes of $\mathcal{K}$, for every $r \in \mathbb{R}$. Let m be the number of simplices of $\mathcal{K}$, and let $(\rho_i)_{i=1\cdots n}$ be the n different values ρ takes on the simplices of $\mathcal{K}$. Plainly $n \leq m$, and we have the following sequence of $n+1$ subcomplexes:

$$\emptyset = \mathcal{K}_0 \subseteq \cdots \subseteq \mathcal{K}_n = \mathcal{K}, \;\; -\infty = \rho_0 < \cdots < \rho_n, \;\; \mathcal{K}_i = \rho^{-1}(-\infty, \rho_i]$$

Applying a (co)homology functor to this sequence of simplicial complexes turns (combinatorial) complexes into (algebraic) abelian groups and inclusion into group homomorphisms. Roughly speaking, a simplicial complex defines a domain as an arrangement of local bricks and (co)homology catches the global features of this domain, like the connected components, the tunnels, the cavities, etc. The homomorphisms catch the evolution of these global features when inserting the simplices in the order of the filtration. Let $H_p(\mathcal{K})$ and $H^p(\mathcal{K})$ denote respectively the homology and cohomology groups of $\mathcal{K}$ of dimension p with coefficients in a field $\mathbb{F}$. The filtration induces a sequence of homomorphisms in the homology and cohomology groups in opposite directions:

$$0 = H_p(\mathcal{K}_0) \to H_p(\mathcal{K}_1) \to \cdots \to H_p(\mathcal{K}_{n-1}) \to H_p(\mathcal{K}_n) = H_p(\mathcal{K}) \tag{1}$$

$$0 = H^p(\mathcal{K}_0) \leftarrow H^p(\mathcal{K}_1) \leftarrow \cdots \leftarrow H^p(\mathcal{K}_{n-1}) \leftarrow H^p(\mathcal{K}_n) = H^p(\mathcal{K}) \tag{2}$$

We refer to [9] for an introduction to the theory of (co)homology and persistent homology. Computing the persistent homology of such a sequence consists in pairing each simplex that creates a homology feature with the one that destroys it. The usual output is a *persistence diagram*, which is a plot of the points $(\rho(\tau), \rho(\sigma))$ for each persistent pair (τ, σ). It is known that because of duality the two sequences above provide the same persistence diagram [7].

The original persistence algorithm [10] considers the homology sequence in Equation 1 that aligns with the filtration direction. It detects when a new homology class is born and when an existing class dies as we proceed forward through the filtration. Recently, a few algorithms have considered the cohomology sequence in Equation 2 which runs in the opposite direction of the filtration [7,6,8]. The birth of a cohomology class coincides with the death of a homology class and the death of a cohomology class coincides with the birth of a homology class. Therefore, by tracking a cohomology basis along the filtration direction and switching the notions of births and deaths, one can obtain all information about the persistent homology. The algorithm of de Silva et al. [7] computes the persistent cohomology following this principle which is reported to work better in practice than the original persistence algorithm [6]. Recently, Dey et al. [8] recognized that tracking cohomology bases provides a simple and natural extension of the persistence algorithm for filtrations connected with simplicial maps. Their algorithm is based on the notion of annotation and, when restricted to only inclusions, is a re-formulation of the algorithm of de Silva et al. [7]. Here we follow this annotation based algorithm.

2 Persistent Cohomology Algorithm and Annotations

In this section, we recall the annotation-based persistent cohomology algorithm of [8]. It maintains a cohomology basis under simplex insertions, where representative cocycles are maintained by the value they take on the simplices. We rephrase the description of this algorithm with coefficients in an arbitrary field $\mathbb{F}$, and use classic field notations $\langle \mathbb{F}, +, \cdot, -, /, 0, 1 \rangle$.

Definition 1. *Given a simplicial complex $\mathcal{K}$, let $\mathcal{K}^p$ denote the set of p-simplices in $\mathcal{K}$. An annotation for $\mathcal{K}^p$ is an assignement $\mathsf{a}^p : \mathcal{K}^p \to \mathbb{F}^g$ of an $\mathbb{F}$-vector $\mathsf{a}_\sigma = \mathsf{a}^p(\sigma)$ of same length g for each p-simplex $\sigma \in \mathcal{K}^p$. We use a when there is no ambiguity in the dimension. We also have an induced annotation for any p-chain $c = \sum_i f_i \sigma_i$ given by linear extension: $\mathsf{a}_c = \sum_i f_i \cdot \mathsf{a}_{\sigma_i}$.*

Definition 2. *An annotation* $\mathsf{a} : \mathcal{K}^p \to \mathbb{F}^g$ *is* valid *if:*
1. $g = rank\, H_p(\mathcal{K})$*, and*
2. two p*-cycles* z_1 *and* z_2 *have* $\mathsf{a}_{z_1} = \mathsf{a}_{z_2}$ *iff their homology classes* $[z_1]$ *and* $[z_2]$ *are identical.*

Proposition 1 ([8]). *The following two statements are equivalent:*
1. An annotation $\mathsf{a} : \mathcal{K}^p \to \mathbb{F}^g$ *is valid*
2. The cochains $\{\phi_j\}_{j=1\cdots g}$ *given by* $\phi_j(\sigma) = \mathsf{a}_\sigma[j]$ *for all* $\sigma \in \mathcal{K}^p$ *are cocycles whose cohomology classes* $\{[\phi_j]\}_{j=1\cdots g}$ *constitute a basis of* $H^p(\mathcal{K})$*.*

A valid annotation is thus a way to represent a cohomology basis. The algorithm for computing persistent cohomology consists in maintaining a valid annotation for each dimension when inserting all simplices in the order of the filtration. Since we process the filtration in a direction opposite to the cohomology sequence (as in Equation 2), we discover the death points of cohomology classes earlier than their birth points. To avoid the confusion, we still say that a new cocycle (or its class) is born when we discover it for the first time and an existing cocycle (or its class) dies when we see it no more.

We present the algorithm and refer to [8] for its validity. We insert simplices in the order of the filtration. Consider an elementary inclusion $\mathcal{K}_i \hookrightarrow \mathcal{K}_i \cup \{\sigma\}$, with σ a p-simplex. Assume that to every simplex τ of any dimension in $\mathcal{K}_i$ is attached an annotation vector a_τ from a valid annotation a of $\mathcal{K}_i$. We describe how to obtain a valid annotation for $\mathcal{K}_i \cup \{\sigma\}$ from that of $\mathcal{K}_i$. We compute the annotation $\mathsf{a}_{\partial\sigma}$ for the boundary $\partial\sigma$ in $\mathcal{K}_i$ and take actions as follows:
Case 1: If $\mathsf{a}_{\partial\sigma} = 0$, $g \leftarrow g + 1$ and the annotation vector of any p-simplex $\tau \in \mathcal{K}_i$ is augmented with a 0 entry so that $\mathsf{a}_\tau = [f_1, \cdots, f_g]^T$ becomes $[f_1, \cdots, f_g, 0]^T$. We assign to the new simplex σ the annotation vector $\mathsf{a}_\sigma = [0, \cdots, 0, 1]^T$. According to Proposition 1, this is equivalent to creating a new cohomology class represented by $\phi(\tau) = 0$ for $\tau \neq \sigma$ and $\phi(\sigma) = 1$.
Case 2: If $\mathsf{a}_{\partial\sigma} \neq 0$, we consider the non-zero element c_j of $\mathsf{a}_{\partial\sigma}$ with maximal index j. We now look for annotations of those $(p-1)$-simplices τ that have a non-zero element at index j and process them as follows. If the element of index j of a_τ is $f \neq 0$, we add $-f/c_j \cdot \mathsf{a}_{\partial\sigma}$ to a_τ. Note that, in the annotation matrix whose columns are the annotation vectors, this implements simultaneously a series of elementary row operations, where each row ϕ_i receives $\phi_i \leftarrow \phi_i - (\mathsf{a}_{\partial\sigma}[i]/c_j) \times \phi_j$. As a result, all the elements of index j in all columns are now 0 and hence the entire row j becomes 0. We then remove the row j and set $g \leftarrow g - 1$. σ is assigned $\mathsf{a}_\sigma = 0$. According to Proposition 1, this is equivalent to removing the j^{th} cocycle $\phi_j(\tau) = \mathsf{a}_\tau[j]$.

As with the original persistence algorithm, the pairing of simplices is derived from the creation and destruction of the cohomology basis elements.

3 Data Structures and Implementation

In this section we present our implementation of the annotation-based persistent cohomology algorithm. We separate the representation of the simplicial complex from the representation of the cohomology groups.

3.1 Representation of the Simplicial Complex

We represent the simplicial complex in a data structure $\mathcal{SC}$ equipped with the operation ComputeBoundary(σ) that computes the boundary of a simplex σ. We denote by $\mathcal{C}_\partial^p$ the complexity of this operation where p is the dimension of σ. Additionally, the simplices are ordered according to the filtration.

Two data structures to represent simplicial complexes are of particular interest here. The first one is the *Hasse diagram*, which is the graph whose nodes are in bijection with the simplices (of all dimensions) of the simplicial complex and an edge links two nodes representing two simplices τ and σ iff $\tau \subseteq \sigma$, and the dimensions of τ and σ differ by 1. The second data structure is

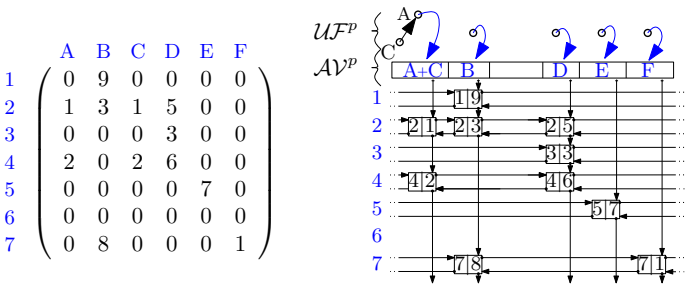


Fig. 1. Compressed annotation matrix of a matrix with integer coefficients.

the *simplex tree* introduced in [2], which is a specific spanning tree of the Hasse diagram. For a simplicial complex $\mathcal{K}$ of dimension k and a simplex $\sigma \in \mathcal{K}$ of dimension p, the Hasse diagram has size $O(k|\mathcal{K}|)$ and allows to compute COMPUTE-BOUNDARY(σ) in time $O(p)$, whereas the simplex tree has size $O(|\mathcal{K}|)$ and allows to compute COMPUTE-BOUNDARY(σ) in time $O(p^2 D_{\mathrm{m}})$, where D_{m} is a small value related to the time needed to traverse the simplex tree.

3.2 The Compressed Annotation Matrix

For each dimension p, the p^{th} cohomology group can be seen as a valid annotation for the p-simplices of the simplicial complex. Hence, an annotation $\mathsf{a} : \mathcal{K}^p \to \mathbb{F}^g$ can be represented as a $g \times |\mathcal{K}^p|$ matrix with elements in $\mathbb{F}$, where each column is an annotation vector associated to a p-simplex. We describe how to represent this annotation matrix in a efficient way.

Compressing the annotation matrix: In most applications, the annotation matrix is sparse and we store it as illustrated in Figure 1. A column is represented as the singly-linked list of its non-zero elements, where the list contains a pair (i, f) if the i^{th} element of the column is $f \neq 0$. The pairs in the list are ordered according to row index i. All pairs (i, f) with same row index i are linked in a doubly-linked list.

Removing duplicate columns: (see Figure 1) To avoid storing duplicate columns, we use two data structures. The first one, $\mathcal{AV}^p$, stores the annotation vectors and allows fast search, insertion and deletion. $\mathcal{AV}^p$ can be implemented as a red-black tree or a hash table. We denote by $\mathcal{C}^p_{\mathcal{AV}}$ the complexity of an operation in $\mathcal{AV}^p$. The simplices of the same dimension that have the same annotation vector are now stored in a same set and the various (and disjoint) sets are stored in a *union-find* data structure denoted $\mathcal{UF}^p$. $\mathcal{UF}^p$ is encoded as a forest where each tree contains the elements of a set, the root being the "representative" of the set. The trees of $\mathcal{UF}^p$ are in bijection with the different annotation vectors stored in $\mathcal{AV}^p$ and the root of each tree maintains a pointer to the corresponding annotation vector in $\mathcal{AV}^p$. Each node representing a p-simplex σ in the simplicial complex $\mathcal{SC}$ stores a pointer to an element of the tree of $\mathcal{UF}^p$ which is associated to its annotation vector a_σ. Finding the annotation vector of σ consists in getting the element it points to in a tree of $\mathcal{UF}^p$ and then finding the root of the tree which points to a_σ in $\mathcal{AV}^p$. We avail the following operations on $\mathcal{UF}^p$:

- CREATE-SET: creates a new tree containing one element.
- FIND-ROOT: finds the root of a tree, given an element in the tree.
- UNION-SETS: merges two trees.

The number of elements maintained in $\mathcal{UF}^p$ is at most the number of simplices of dimension p, i.e. $|\mathcal{K}^p|$. The operations FIND-ROOT and UNION-SETS on $\mathcal{UF}^p$ can be computed in amortized time $O(\alpha(|\mathcal{K}^p|))$, where $\alpha(n)$ is the very slowly growing inverse Ackermann function (constant

less than 4 in practice), and CREATE-SET is performed in constant time. We will refer to this data structure as the *Compressed Annotation Matrix*.

Operations: The compressed annotation matrix described above supports the following operations. Complexities are expressed for an annotation matrix with g rows and s distinct columns:

• SUM-ANN($\mathsf{a}_1, \mathsf{a}_2$): computes the sum of two annotation vectors a_1 and a_2, and returns the lowest non-zero coefficient if it exists. The column elements are sorted by increasing row index, so the sum is performed in $O(g)$ time.

• SEARCH-ANN/ADD-ANN/REMOVE-ANN (a): searches, adds or removes an annotation vector a from $\mathcal{AV}^p$ in $O(\mathcal{C}^p_{\mathcal{AV}})$ time.

• CREATE-COCYCLE() implements **Case 1** of the algorithm described in section 2. It inserts a new column in $\mathcal{AV}^p$ containing one element $(i_{\text{new}}, 1)$, where i_{new} is the index of the created cocycle. This is performed in time $O(\mathcal{C}^p_{\mathcal{AV}})$. We also create a new disjoint set in $\mathcal{UF}^p$ for the new column. This is done in $O(1)$ time using CREATE-SET.

• KILL-COCYCLE($\mathsf{a}_{\partial\sigma}, c_j, j$) implements **Case 2** of the algorithm. It finds all columns with a non-zero element at index j and, for each such column A, it adds to A the column $-f/c_j \cdot \mathsf{a}_{\partial\sigma}$ if f is the non-zero element at index j in A. To find the columns with a non-zero element at index j, we use the row doubly-linked list at index j. We call SUM-ANN to compute the sums. The overall time needed for all columns is $O(gs)$ in the worst-case. Finally, we remove duplicate columns using operations on $\mathcal{AV}^p$ (in $O(s\,\mathcal{C}^p_{\mathcal{AV}})$ time in the worst-case) and call UNION-SETS on $\mathcal{UF}^p$ if two sets of simplices, which had different annotation vectors before calling KILL-COCYCLE, are assigned the same annotation vector. This is performed in at most $O(s\alpha(|\mathcal{K}^p|))$ time.

3.3 Computing Persistent Cohomology

Given as input a filtered simplicial complex represented in a data structure $\mathcal{SC}$, we compute the persistence diagram of the filtration.

Implementation of the persistent cohomology algorithm: We insert the simplices in the filtration order and update the data structures during the successive insertions. The simplicial complex is stored in a simplicial complex data structure $\mathcal{SC}$ and we maintain, for each dimension p, a compressed annotation matrix, which is empty at the beginning of the computation. Let σ be a p-simplex we insert. We compute $\partial\sigma$ using COMPUTE-BOUNDARY in $\mathcal{SC}$ (in $O(\mathcal{C}^p_\partial)$ time), and find the annotation vectors of the boundary faces using $p+1$ calls to FIND-ROOT in $\mathcal{UF}^{p-1}$ (in $O(p\alpha(|\mathcal{K}^{p-1}|))$ time). We compute $\mathsf{a}_{\partial\sigma}$ by summing the annotation vectors of the faces in the boundary of σ, using $p+1$ calls to SUM-ANN (in $O(pg)$ time). Depending on the value of $\mathsf{a}_{\partial\sigma}$, we call either CREATE-COCYCLE (in $O(\mathcal{C}^p_{\mathcal{AV}})$ time) or KILL-COCYCLE (in $O(s\cdot(g+\mathcal{C}^{p-1}_{\mathcal{AV}}+\alpha(|\mathcal{K}^{p-1}|)))$ time). The algorithm returns the persistence diagram.

Complexity analysis: The complexity for inserting σ of dimension p is:

$$O\left(\mathcal{C}^p_\partial + p(\alpha(|\mathcal{K}^{p-1}|) + g) + \mathcal{C}^p_{\mathcal{AV}} + s(g + \mathcal{C}^{p-1}_{\mathcal{AV}} + \alpha(|\mathcal{K}^{p-1}|))\right)$$

Let k be the dimension of the simplicial complex and m its number of simplices. Let g_{m} and s_{m} be the maximal dimension of a cohomology group and the maximal number of distinct columns in the matrix, respectively, along the computation. The total cost for computing the persistent cohomology and the memory complexity for storing the compressed annotation matrices are respectively:

$$O\left(m \times \left[\mathcal{C}^k_\partial + k(\alpha(m) + g_{\text{m}}) + s_{\text{m}}(g_{\text{m}} + \mathcal{C}_{\mathcal{AV}} + \alpha(m))\right]\right) \text{ and } O\left(m + k g_{\text{m}} s_{\text{m}}\right)$$

with $\mathcal{C}^k_\partial$ the complexity of COMPUTE-BOUNDARY in $\mathcal{SC}$, $\alpha(\cdot)$ the inverse Ackermann function and $\mathcal{C}_{\mathcal{AV}}$ the complexity of an operation in $\mathcal{AV}$.

Although s_{m} is bounded by m and by $2^{g_{\mathrm{m}}}$, it remains close to g_{m} in practice, as illustrated in the experimental section.

4 Filtration Strategies

In this section, we show that we have some freedom in choosing the order in which we insert the simplices. We take advantage of this freedom to improve the performance of the algorithm. In section 4.1, we use the fact that the insertion of a simplex that creates a cocycle can be postponed until one of its cofaces is to be inserted. In section 4.2, we consider the important practical case where the filtration does not provide a strict order on the simplices, i.e. when some simplices have the same filtration value. In both cases, the optimization does not change the persistent cohomology of the simplicial complex.

4.1 Lazy Evaluation

We postpone the insertion of a simplex σ that creates a cocycle until we consider one of its cofaces. Such a delayed insertion of σ does not modify the behaviour of the algorithm since the annotation that is assigned to σ does not affect any subsequent annotation updates until a coface of σ appears. We give in Appendix A.1 a formal proof that the lazy evaluation does not modify the persistent diagram. The lazy evaluation reduces the dimension g of the cohomology groups we maintain, as well as the number s of distinct annotation vectors. This consequently improves the time and space performance of the algorithm.

We implement this lazy evaluation as follows. We mark each simplex whose insertion has been postponed (initially, no simplex is marked). As before, we call Lazy-evaluation, the lazy insertion procedure, on each simplex in the order of the filtration. Let σ be a p-simplex on which we call Lazy-evaluation. If σ is marked, Lazy-evaluation directly inserts it as a creator, without computing the annotation of its boundary, and unmarks σ. If σ is not marked, Lazy-evaluation computes the boundary $\partial\sigma$ of σ and is called recursively on each marked face of $\partial\sigma$. Lazy-evaluation then computes $\mathsf{a}_{\partial\sigma}$ and proceeds as follows. If $\mathsf{a}_{\partial\sigma}$ is non-zero (σ kills a cocycle), we proceed as for the standard insertion of σ and update the annotation. If $\mathsf{a}_{\partial\sigma}$ is null (σ creates a cocycle), we simply mark σ and postpone the insertion of σ. The recursive calls to the marked subfaces guarantee that all subfaces of σ have been inserted prior to the computation of $\mathsf{a}_{\partial\sigma}$. Note that the lazy evaluation does not induce an additional cost since Lazy-evaluation is called at most twice on each simplex and we compute the boundaries only once. In section 5, we give experimental evidence that this lazy evaluation is effective at decreasing the maximal dimension of the cohomology groups we maintain.

4.2 Ordering Iso-simplices

Many simplices, called iso-simplices, may have the same filtration value. This situation is common when the filtration is induced by a geometric scaling parameter. Assume that we want to compute the cohomology groups $H^p(\mathcal{K}_{i+1})$ from $H^p(\mathcal{K}_i)$ where $\mathcal{K}_i \subseteq \mathcal{K}_{i+1}$ and all simplices in $\mathcal{K}_{i+1} \setminus \mathcal{K}_i$ have the same filtration value. Depending on the insertion order of the simplices of $\mathcal{K}_{i+1} \setminus \mathcal{K}_i$, the dimension of the cohomology groups to be maintained along the computation may vary a lot as well as the computing time, potentially leading to a computational bottleneck. We propose a heuristic to order iso-simplices that appears to be efficient in practice (section 5).

Intuitively, we want to avoid the creation of many "holes" of dimension p and want to fill them up as soon as possible with simplices of dimension $p+1$. For example, in Figure 2, we want to avoid inserting all edges first, which will create two holes that will be filled when inserting the triangles. To do so, we look for the maximal faces to be inserted and recursively insert their subfaces. We conduct the recursion so as to minimize the maximum number of holes. In addition,

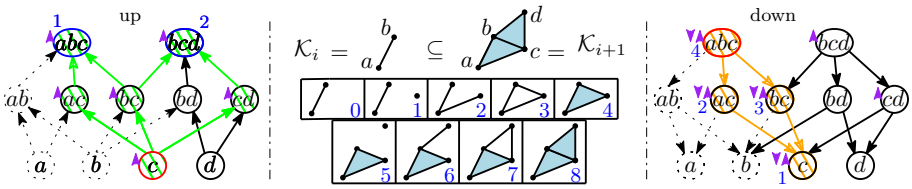


Fig. 2. Inclusion $\mathcal{K}_i \subseteq \mathcal{K}_{i+1}$. Left: upward traversal (in green) from simplex $\{c\}$. The ordering of the maximal cofaces appears in blue. Right: downward traversal (in orange) from simplex $\{abc\}$. The ordering of the subfaces appears in blue.

to avoid the creation of holes due to maximal simplices that are incident, maximal simplices sharing subfaces are inserted next to each other. We can describe the reordering algorithm in terms of a graph traversal. The graph considered is the graph of the Hasse diagram of $\mathcal{K}_{i+1} \setminus \mathcal{K}_i$, defined in section 3.1 (see Figure 2).

Let $\sigma_1 \cdots \sigma_\ell$ be the iso-simplices of $\mathcal{K}_{i+1} \setminus \mathcal{K}_i$, sorted so as to respect the inclusion order. We attach to each simplex two flags, a flag F_{up} and a flag F_{down}, set to 0 originally. When inserting a simplex σ_j, we proceed as follows. We traverse the Hasse diagram upward in a depth-first fashion and list the inclusion-maximal cofaces of σ_j in $\mathcal{K}_{i+1} \setminus \mathcal{K}_i$. The flags F_{up} of all nodes traversed are set to 1 and the maximal cofaces are ordered according to the traversal. From each maximal coface in this order, we then traverse the graph downward and order the subfaces in a depth-first fashion: this last order will be the order of insertion of the simplices. The flags F_{down} of all traversed nodes are set to 1. We stop the upward (resp. downward) traversal when encountering a node whose flag F_{up} (resp. F_{down}) is set to 1. We do not insert already inserted simplices either.

By proceeding as above on all simplices of the sequence $\sigma_1 \cdots \sigma_\ell$, we define a new order which respects the inclusion order between the simplices. Indeed, as the downward traversal starts from a maximal face and is depth first, a face is always inserted after its subfaces. Every edge in the graph is traversed twice, once when going upward and the other when going downward. Indeed, during the upward traversal, at each node N associated to a simplex σ_N, we visit only the edges between N and the nodes associated to the cofaces of σ_N and, during the downward traversal, we visit only the edges between N and the nodes associated to the subfaces of σ_N. If $\mathcal{K}_{i+1} \setminus \mathcal{K}_i$ contains ℓ simplices, the reordering takes in total $O(\ell \times (\mathcal{C}_\partial + \mathcal{C}_{co\partial}))$ time, where $\mathcal{C}_\partial$ (resp. $\mathcal{C}_{co\partial}$) refers to the complexity of computing the codimension 1 subfaces (resp. cofaces) of a simplex in the simplicial complex data structure $\mathcal{SC}$. The reordering of the filtration can either be done as a preprocessing step if the whole filtration is known, or on-the-fly as only the neighboring simplices of a simplex need to be known at a time. It should also be observed that the reordering of iso-simplices is compatible with the lazy evaluation. The reordering of a set of iso-simplices respects the inclusion order of the simplices and the filtration, and therefore does not change the persistence diagram of the filtered simplicial complex. This is a direct consequence of the stability theorem of persistence diagrams [5] (details in Appendix A.2). However, it may change the pairing of simplices.

5 Experiments

In this section we report on the experimental performance of our implementation. Given the filtration of a simplicial complex as input, we measure the time taken by our implementation to compute the persistent cohomology of the filtration, as well as various statistics. We compare the timings with state-of-the-art software for computing persistent homology and cohomology. Specifically, we compare our implementation with the `Dionysus` library (`www.mrzv.org/`

								DioCoH(s.)		PHAT(s.)		CAM(s.)	
Data	Cpx	$\|\mathcal{P}\|$	D	d	$\rho_{\max}$	k	$\|\mathcal{K}\|$	$\mathbb{Z}_2$	$\mathbb{Z}_{11}$	$\mathbb{Z}_2$	$\mathbb{Z}_{11}$	$\mathbb{Z}_2$	$\mathbb{Z}_{11}$
Cy8	Rips	6040	24	2	0.41	16	$21 \cdot 10^6$	420	4822	12.9	–	10.1	10.8
S4	Rips	507	5	4	0.715	5	72×10^6	943	1026	M_∞	–	33.2	33.3
Bro	Wit	500	25	?	0.06	18	3.2×10^6	807	T_∞	2.3	–	1.5	2.8
Kl	Wit	10000	5	2	0.105	5	74×10^6	569	662	14200	–	30.9	31.1
Bud	αSh	49990	3	2	∞	3	1.4×10^6	30.0	30.9	6.1	–	0.85	0.87
Nep	αSh	2×10^6	3	2	∞	3	57×10^6	T_∞	T_∞	T_∞	–	51.1	52.2

Fig. 3. Data, timings and statistics

`software/dionysus/`) which provides implementation for persistent homology [10,13] and persistent cohomology [7] (denoted `DioCoH`) with field coefficients in $\mathbb{Z}_p$, for any prime p. We also compare our implementation with the `PHAT` library (`www.phat.googlecode.com`) which provides an implementation of the optimized algorithm for persistent homology [3,1], with coefficients in $\mathbb{Z}_2$ only. `DioCoH` and `PHAT` have been reported to be the most efficient implementation in practice [6,1]. Our implementation is in `C++`. All timings are measured on a Linux machine with 3.00 GHz processor and 32 GB RAM. They are all averaged over 10 independent runs. The symbols T_∞ means that the computation lasted more than 12 hours, and M_∞ means that the computation ran out of memory.

We use a variety of both real and synthetic datasets: **Cy8** is a set of points in $\mathbb{R}^{24}$, sampled from the space of conformations of the cyclo-octane molecule, which is the union of two intersecting surfaces; **S4** is a set of points sampled from the unit 4-sphere in $\mathbb{R}^5$; **Bro** is a set of 5×5 *high-contrast patches* derived from natural images, interpreted as vectors in $\mathbb{R}^{25}$, from the Brown database; **Kl** is a set of points sampled from the surface of the figure eight Klein Bottle embedded in $\mathbb{R}^5$; **Bud** is a set of points sampled from the surface of the *Happy Buddha* (`http://graphics.stanford.edu/data/3Dscanrep/`) in $\mathbb{R}^3$; and **Nep** is a set of points sampled from the surface of the *Neptune statue* (`http://shapes.aimatshape.net/`). Datasets are listed in Figure 3 with details on the sets of points $\mathcal{P}$, their size $|\mathcal{P}|$, the ambient dimension D, the intrinsic dimension d of the object the sample points belong to (if known), the threshold $\rho_{\max}$, the dimension k of the simplicial complexes and the size $|\mathcal{K}|$ of the simplicial complexes. We use three families of simplicial complexes [9] which are of particular interest in topological data analysis: the Rips complexes (denoted `Rips`), the relaxed witness complexes (denoted `Wit`) and the α-shapes (denoted α`Sh`). Simplicial complexes are constructed up to embedding dimension.

Time Performance: As `Dionysus` and `PHAT` encode explicitely the boundaries of the simplices, we use a Hasse diagram for implementing $\mathcal{SC}$ and having the same time complexity for accessing the boundaries of simplices. As illustrated in Figure 3 our implementation (noted `CAM`) outperforms `DioCoH` and `PHAT` on all examples. The persistent cohomology algorithm of `Dionysus` is always several times slower than our implementation. Moreover, `DioCoH` is very sensitive to non-orientability (as in the case of **Cy8** and **Bro**) where changing the field (here from $\mathbb{Z}_2$ to $\mathbb{Z}_{11}$) changes the structure of the cohomology groups and leads to more complex calculations. On the other hand, our implementation `CAM` shows almost identical performance for $\mathbb{Z}_2$ and $\mathbb{Z}_{11}$ coefficients on all examples.

The persistent homology algorithm of `PHAT` shows good performance in the case of small simplicial complexes ($\leq$ 21M simplices): even if `CAM` is always faster, the timings are sometimes close. However, `PHAT` provides only computation of persistent homology with $\mathbb{Z}_2$ coefficients, where the computation is sped up by the fact that the field operations $+, \cdot, -, /$ become straighforward, and the sum of two sparse vectors can be computed as a symmetric difference. Moreover, `PHAT` did not scale to bigger simplicial complexes ($>$ 50M simplices), where the computational cost

Nep	$\lvert M\rvert$	$\#\mathbb{F}op.$
Comp.	126057	84×10^6
¬Comp.	574426	3860×10^6

Nep	G_{m}	S_{m}
Strat.	74107	115676
¬Strat.	383684	409084

$\mathbb{Z}_{11}$			$\mathbb{Z}_{393}$		
M_{DS}	$\mathsf{a}_{\partial\sigma}$	M_{op}	M_{DS}	$\mathsf{a}_{\partial\sigma}$	M_{op}
72%	18%	10%	58%	15%	27%

Fig. 4. Statistics on the effect of the optimizations.

increases dramatically in terms of time and memory (for example `PHAT` runs out of RAM after 9H of computation for **S4**). On all examples, `CAM` did not take more than half a minute to compute persistence.

Statistics and Optimization: Figure 4 presents the effects on the computation of both compression of the matrix (from section 3) and filtration strategies (from section 4) on the example **Nep**, which is of particular interest because no other method can compute its persistent homology: interestingly, we have noticed that the dimensions of the cohomology groups increase up to several thousands, for a simplicial complex with a simple topology. The left table presents the number of non-zero elements of the compressed annotation matrix ($|M|$) and the number of field operations in $\mathbb{F}$ ($\#\mathbb{F}op.$) with and without the compression of the matrix (removal of duplicate columns). We note a reduction factor of 4.5 for the size of the matrix, and we proceed to 46 times less field operations with the compression. Considering **Nep** is 57 million simplices, this makes less than 1.5 field operations per simplex in average, for our implementation. The middle table presents the effect of the filtration strategies (section 4) on the dimensions of the cohomology groups and the number of distinct annotation vectors, which are two key values in the complexity analysis. Here G_{m} stands for the sum of the dimensions of the cohomology groups and S_{m} stands for the sum of distinct annotation vectors (for all dimension of simplices), both maximized over the computation. The filtration strategies reduce the dimension of the cohomology groups by a factor 5. Note also that S_{m} remains close to G_{m}. Finally, the right table presents the repartition of computational time between maintaining the compressed annotation matrix (under removal of duplicate vectors, etc, noted M_{DS}), computing the annotation vector $\mathsf{a}_{\partial\sigma}$ and modifying the values of the elements in the compressed annotation matrix (noted M_{op}), when computing persistent cohomology with $\mathbb{Z}_{11}$ and $\mathbb{Z}_{393}$ coefficients. The computational complexity of field operations $\langle \mathbb{F}, +, \cdot, -, /, 0, 1\rangle$, and in particular the inversion $/$, varies depending of the field we use. As the inversion is needed only when modifying the matrix and M_{op} represents a small part of the computation, our implementation is quite insensitive to the field we use. Specifically, when the time spent for modifying the matrix elements is multiplied by a factor of 3.3 when using $\mathbb{Z}_{393}$ instead of $\mathbb{Z}_{11}$, the total cost for computing persistent cohomology only increases by 23%.

On all our experiments, the size of the compressed annotation matrix is negligible compare to the size of the simplicial complex. Consequently, combined with the simplex tree data structure [2] for representing the simplicial complex, we have been able to compute the persistent cohomology of simplicial complexes of several hundred million simplices in high dimension.

Acknowledgements

This research is partially supported by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL Computational Geometry Learning). This research is also partially supported by NSF (National Science Foundation, USA) grants CCF-1048983 and CCF-1116258

References

1. Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. 2013.
2. Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. In *ESA*, pages 731–742, 2012.
3. Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, 2011.
4. Chao Chen and Michael Kerber. An output-sensitive algorithm for persistent homology. *Comput. Geom.*, 46(4):435–447, 2013.
5. David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.
6. V. de Silva, D. Morozov, and M. Vejdemo-Johansson. Dualities in persistent (co)homology. *Inverse Problems*, 27:124003, 2011.
7. V. de Silva, D. Morozov, and M. Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discrete Comput. Geom.*, 45:737–759, 2011.
8. Tamal K. Dey, Fengtao Fan, and Yusu Wang. Computing topological persistence for simplicial maps. *CoRR*, abs/1208.5018, 2012.
9. Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010.
10. Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.
11. Nikola Milosavljevic, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Symposium on Computational Geometry*, pages 216–225, 2011.
12. Dmitriy Morozov. Persistence algorithm takes cubic time in worst case. *BioGeometry News, Dept. Comput. Sci., Duke Univ*, 2005.
13. Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

A Correctness of the Filtration Strategies

A.1 Correctness of the Lazy Evaluation

We prove that the persistence diagram we compute with the lazy algorithm is the same as the persistence diagram we compute with the standard persistence algorithm. We first prove the lemma:

Lemma 1. *Given a filtration $F = F_1 \cdot \sigma \cdot \tau \cdot F_2$, the pairing of simplices remains unchanged in the filtration $\widetilde{F} = F_1 \cdot \tau \cdot \sigma \cdot F_2$ if σ is a creator in F and $\sigma \not\subseteq \tau$.*

Proof. Let $\mathcal{B}_1^p$ and $\mathcal{B}_2^p$ be the sets of cocycles, maintained with the annotations in the algorithm, representing the classes forming a basis of the cohomology group of dimension p after considering, respectively, the prefix F_1 and the prefix $F_1 \cdot \sigma \cdot \tau$ of the filtration ordering F. Define similarly $\widetilde{\mathcal{B}}_1^p$ and $\widetilde{\mathcal{B}}_2^p$ for, respectively, the prefix F_1 and the prefix $F_1 \cdot \tau \cdot \sigma$ of the filtration ordering $\widetilde{F}$. We first prove that σ and τ are paired the same way in F and $\widetilde{F}$.

By hypothesis, σ is a creator in F thus $\mathsf{a}_{\partial\sigma} = 0$ after processing the prefix F_1 of the filtration F. We prove that $\mathsf{a}_{\partial\sigma} = 0$ after processing $F_1 \cdot \tau$ in $\widetilde{F}$. Let $\partial\sigma = \{s_1, \cdots, s_{|\sigma|}\}$ be the codimension 1 subfaces of σ. If τ is a creator or the dimension of τ is different from the one of σ, the insertion of τ does not modify the annotation vectors of the s_js and $\mathsf{a}_{\partial\sigma}$ is not changed. If τ is a killer of same dimension as σ, and kills the cocycle of index i, the insertion of τ will modify the value of the annotation vector of a simplex s_j from the boundary of σ such that: $\mathsf{a}_{s_j} \leftarrow \mathsf{a}_{s_j} - \mathsf{a}_{s_j}[i]/c_i \cdot \mathsf{a}_{\partial\tau}$, with $c_i = \mathsf{a}_{\partial\tau}[i]$. If we compute $\mathsf{a}_{\partial\sigma}$ after the insertion of τ we get:

$$\begin{aligned}\mathsf{a}_{\partial\sigma} &= \sum_{j=1\cdots|\sigma|} (-1)^j \left[\mathsf{a}_{s_j} - \frac{\mathsf{a}_{s_j}[i]}{c_i}\mathsf{a}_{\partial\tau}\right] \\ &= \left[\sum_{j=1\cdots|\sigma|} (-1)^j \mathsf{a}_{s_j}\right] - \frac{1}{c_i}\left[\sum_{j=1\cdots|\sigma|} (-1)^j \mathsf{a}_{s_j}[i]\right] \mathsf{a}_\tau = 0\end{aligned}$$

because a_σ is 0 before the insertion of τ. Consequently, σ is also a creator in the filtration $\widetilde{F}$. In both filtrations σ creates a cocycle ϕ^σ with time of appearence ρ_σ.
As $\sigma \not\subseteq \tau$ and σ is a creator, the annotation $\mathsf{a}_{\partial\tau}$ of the boundary of τ remains unchanged whether σ is inserted before or after τ. Consequently, τ is paired the same way in F or $\widetilde{F}$, and the cocycle it creates/kills is born/dies at the same time ρ_τ.

We finally prove that, for any p, $\mathcal{B}_1^p = \widetilde{\mathcal{B}}_1^p$ and $\mathcal{B}_2^p = \widetilde{\mathcal{B}}_2^p$. The first equality $\mathcal{B}_1^p = \widetilde{\mathcal{B}}_1^p$ is straighforward. The second equality $\mathcal{B}_2^p = \widetilde{\mathcal{B}}_2^p$ is deduced from the fact that σ and τ admits the same pairing in both filtrations, and give the same times for births and deaths of cocycles. Thus, the pairing of the simplices in F_1 and F_2 is the same in F and $\widetilde{F}$.

We deduce:

Proposition 2. *The lazy algorithm computes the same persistence diagram as the standard algorithm for computing persistent homology.*

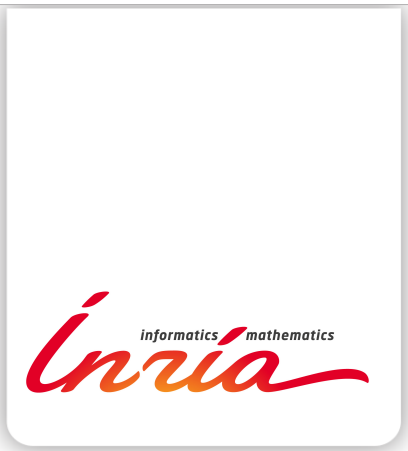
Proof. The shuffle of the filtration induced by the lazy evaluation can be decomposed into successive elementary transpositions of the form $F = F_1 \cdot \sigma \cdot \tau \cdot F_2 \to \widetilde{F} = F_1 \cdot \tau \cdot \sigma \cdot F_2$, with σ creator in F and $\sigma \not\subseteq \tau$. As such a transposition does not modify the pairing nor the time of birth/death of cocycle classes (lemma 1), the output persistence diagram is the same.

A.2 Correctness of the Iso-simplices Ordering

The fact that the reordering within a set of simplices with a same filtration value does not change the persistence diagram of the filtered simplicial complex, while changing the pairing of simplices, is a direct consequence of the stability theorem of persistence diagrams:

Theorem 1. *[5] Let $\mathbb{X}$ be a triangulable space with continuous tame functions $f, g : \mathbb{X} \to \mathbb{R}$. Then the persistence diagrams satisfy $d_B(D(f), D(g)) \leq \|f - g\|_\infty$, where d_B is the bottleneck distance, $D(h)$ the persistence diagram of function h and $\|\cdot\|_\infty$ the L_∞ distance.*

As our reordering still respects the inclusion order of simplices it defines a valid filtration on the simplices. As the filtration values are unchanged, the persistence diagram remains the same.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399