The Gudhi Library: Simplicial Complexes and Persistent Homology

Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, Mariette Yvinec

The Gudhi Library: Simplicial Complexes and Persistent Homology

Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, Mariette Yvinec

RESEARCH REPORT
N° 8548
June 2014
Project-Team GUDHI

ISSN 0249-6399   ISRN INRIA/RR--8548--FR+ENG

The Gudhi Library: Simplicial Complexes and Persistent Homology

Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, Mariette Yvinec

Project-Team GUDHI

Research Report n° 8548 — June 2014 — 10 pages

Abstract: We present the main algorithmic and design choices that have been made to represent complexes and compute persistent homology in the Gudhi library. The Gudhi library (Geometric Understanding in Higher Dimensions) is a generic C++ library for computational topology. Its goal is to provide robust, efficient, flexible and easy to use implementations of state-of-the-art algorithms and data structures for computational topology. We present the different components of the software, their interaction and the user interface. We justify the algorithmic and design decisions made in Gudhi and provide benchmarks for the code. The software, which has been developed by the first author, is available at `project.inria.fr/gudhi/software/`.

Key-words: persistent homology, simplicial complex, software library, computational topology, generic programming

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Bibliothèque de Calcul Gudhi : Complexes Simpliciaux et Homologie Persistante

Résumé : Nous présentons le design de la bibliothèque de calcul Gudhi. Gudhi est une bibliothèque générique, programmée en C++, pour la topologie algorithmique.

Mots-clés : homologie persistante, complexes simpliciaux, bibliothèque de calcul, topologie algorithmique, programmation générique

1 Introduction

The principle of algebraic topology is to attach algebraic invariants to topological spaces in order to classify them up to homeomorphism. One can consequently study the property of a discrete algebraic structure (a sequence of homology groups in our case) instead of studying a continuous domain directly, which would be hard to handle algorithmically. Persistent homology [14, 16] may be considered as a "dynamic version" of this principle: given a sequence of topological spaces connected by continuous maps, we study the corresponding sequence of homology groups connected by group homomorphisms, induced by the topological space maps. The whole sequence of groups together with their homomorphisms form an algebraic structure (specifically a module) that we study. Very efficient methods have been developped for computing persistent homology [1, 14] and its dual, persistent cohomology [2, 11, 13]. The generality and stability with regard to noise [9] of persistence have made it a widely used tool in practice.

An application of interest for computational topology is topological data analysis, where one is interested in learning topological invariants of a shape, sampled by a point cloud. A popular approach is to construct, at different scales, an approximation of the shape using complexes built on top of the points, and then compute the persistent homology of these complexes. This approach has been successfully used in various areas of science and engineering, as for example in sensor networks [10], image analysis [6], and data analysis [8], where one typically needs to deal with big data sets in high dimensions and with general metrics. The simplicial complex and persistent homology packages in `Gudhi` provide all software components for this approach.

The challenge is twofold. On the one hand we need to design a generic library in computational topology, in order to adapt to the various configurations of the problem: nature of the complexes (simplicial, cubical, etc) and their representation, nature of the maps between them (inclusions, edge contractions, etc), ordering of the maps (linear, zigzag, etc) and types of algorithm for persistence (homology, cohomology). On the other hand, we need to implement a high-performance library to handle complex practical examples.

We recall in Section 2 the definition of homology and persistent homology constructed from simplicial complexes. In Section 3, we describe the design of the `Gudhi` library. In Section 4, we discuss the implementation choices and the user interface. Specifically, simplicial complexes are implemented with a simplex tree data structure [4]. The simplex tree is an efficient and flexible data structure for representing general (filtered) simplicial complexes. The persistent homology of a filtered simplicial complex is computed by means of the persistent cohomology algorithm [11, 13], implemented with a compressed annotation matrix [2]. The persistent homology package provides the computation of persistence with different coefficient fields, including the implementation of the multi-field persistence algorithm of [3], i.e. the simultaneous computation of persistence with various coefficient fields. Finally, in Section 5 we discuss the future components of the library and their integration in the design.

2 Theoretical Foundation of Persistent Homology

The theory of homology consists in attaching to a topological space a sequence of (homology) groups, capturing global topological features like connected components, holes, cavities, etc. Persistent homology studies the evolution – birth, life and death – of these features when the topological space is changing. Consequently, the theory is essentially composed of three elements: topological spaces, their homology groups and an evolution scheme.

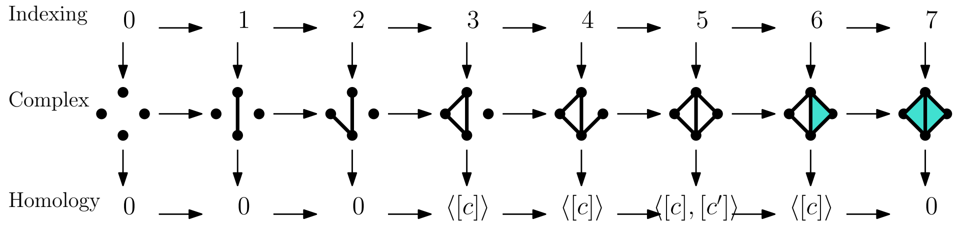

Figure 1: Indexing of eight simplicial complexes and corresponding sequence of homology groups in dimension 1.

2.1 Simplicial Complexes:

In computer science, topological spaces are represented by their discrete counterpart: (cell) complexes. On the following, we focus on simplicial complexes, but our approach applies to all kinds of cell complexes. Let $V = \{1, \cdots, |V|\}$ be a set of *vertices*. A *simplex* σ is a subset of vertices $\sigma \subseteq V$. A *simplicial complex* $\mathbf{K}$ on V is a collection of *simplices* $\{\sigma\}$, $\sigma \subseteq V$, such that $\tau \subseteq \sigma \in \mathbf{K} \Rightarrow \tau \in \mathbf{K}$. The dimension $n = |\sigma| - 1$ of σ is its number of elements minus 1.

A *simplicial map* $f : \mathbf{K} \to \mathbf{K}'$ between simplicial complexes $\mathbf{K}$ and $\mathbf{K}'$, with respective vertex sets V and V', is a map that sends every vertex $v \in V$ to a vertex $f(v) \in V'$, and every simplex $[v_0, \cdots, v_n] \in \mathbf{K}$ to a simplex $[f(v_0), \cdots, f(v_n)] \in \mathbf{K}'$. Note that they may be redundancy in the set $\{f(v_0), \cdots, f(v_n)\}$, in which case the simplex image has lower dimension that its pre-image. In the following, we focus on *inclusions*, which are a particular case of simplicial maps, and discuss the case of general simplicial maps in Section 5.

2.2 Homology:

For a ring $\mathcal{R}$, the group of *n-chains*, denoted $\mathbf{C}_n(\mathbf{K}, \mathcal{R})$, of $\mathbf{K}$ is the group of formal sums of n-simplices with $\mathcal{R}$ coefficients. The *boundary operator* is a linear operator $\partial_n : \mathbf{C}_n(\mathbf{K}, \mathcal{R}) \to \mathbf{C}_{n-1}(\mathbf{K}, \mathcal{R})$ such that $\partial_n \sigma = \partial_n [v_0, \cdots, v_n] = \sum_{i=0}^{n} (-1)^i [v_0, \cdots, \widehat{v_i}, \cdots, v_n]$, where $\widehat{v_i}$ means v_i is omitted from the list. The chain groups form a sequence:

$$\cdots \quad \mathbf{C}_n(\mathbf{K}, \mathcal{R}) \xrightarrow{\partial_n} \mathbf{C}_{n-1}(\mathbf{K}, \mathcal{R}) \xrightarrow{\partial_{n-1}} \cdots \xrightarrow{\partial_2} \mathbf{C}_1(\mathbf{K}, \mathcal{R}) \xrightarrow{\partial_1} \mathbf{C}_0(\mathbf{K}, \mathcal{R})$$

of finitely many groups $\mathbf{C}_n(\mathbf{K}, \mathcal{R})$ and homomorphisms ∂_n, indexed by the dimension $n \geq 0$. The boundary operators satisfy the property $\partial_n \circ \partial_{n+1} = 0$ for every $n > 0$ and we define the homology groups:

$$\mathbf{H}_n(\mathbf{K}, \mathcal{R}) = \ker \partial_n / \mathrm{im}\ \partial_{n+1}$$

We refer to [15] for an introduction to homology theory and to [14] for an introduction to persistent homology.

2.3 Indexing Scheme:

"Changing" a simplicial complex consists in applying a simplicial map. An *indexing scheme* is a directed graph together with a traversal order, such that two consecutive nodes in the graph are connected by an arrow (either forward or backward). The nodes represent simplicial complexes and the directed edges simplicial maps.

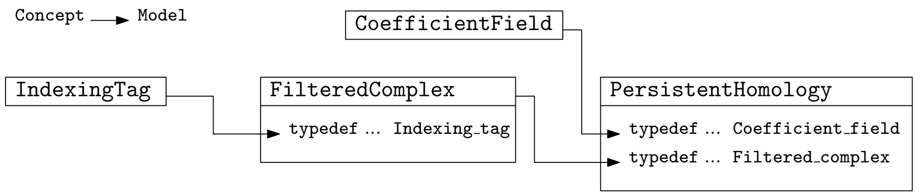

Figure 2: Overview of the design of the library.

From the computational point of view, there are two types of indexing schemes of interest in persistent homology [1]: *linear* ones $\bullet \longrightarrow \bullet \longrightarrow \cdots \longrightarrow \bullet \longrightarrow \bullet$ in persistent homology [16], and *zigzag* ones $\bullet \longrightarrow \bullet \longleftarrow \cdots \longrightarrow \bullet \longleftarrow \bullet$ in zigzag persistent homology [7]. These indexing schemes have a natural left-to-right traversal order, and we describe them with ranges and iterators. We focus in the following on the linear case, and discuss the zigzag case in Section 5.

In the following, we consider the case where the indexing scheme is induced by a filtration. A *filtration* of a simplicial complex is a function $f : \mathbf{K} \to \mathbb{R}$ satisfying $f(\tau) \leq f(\sigma)$ whenever $\tau \subseteq \sigma$. Ordering the simplices by increasing filtration values (breaking ties so as a simplex appears after its subsimplices of same filtration value) provides an indexing scheme.

We refer to Figure 1 for an illustration of the three components of the theory and their connections. The figure pictures the linear indexing of eight simplicial complexes connected by inclusions, and the corresponding sequence of homology groups in dimension 1. Every inclusion induces a group homomorphism at the homology level. Persistent homology studies this sequence of homology groups connected by homomorphisms. Specifically, computing persistent homology consists in computing a primary decomposition of this sequence of homology groups (forming a module); the decomposition is usually represented by means of a *persistence diagram* [14].

2.4 Remark:

The reader may have found a category theory taste to this presentation of persistent homology. In particular, the vertical arrows in Figure 1 represent functors of categories. We refer to [5] for more details on the categorification of persistent homology.

3 Design of the Library

A *concept* is a set of requirements (valid expression, associated types, etc) for a type. If a type satisfies these requirements, it is a *model* of the concept. The general idea under our design is to associate a concept per component presented in Section 2: the three components of the theory (indexing, complex and homology) are illustrated in Figure 1. Given two components related by a vertical arrow in Figure 1, and two models `A` and `B` of their respective associated concepts, we connect `B` with `A` through a template argument `B<A>`.

3.1 IndexingTag Concept:

In order to describe the indexing scheme, we use a tag `IndexingTag` that is either

[1] i.e. from which an interval decomposition of the persistence module exists: Gabriel's theorem [12] in quiver theory classifies these graphs.

```
void compute_persistent_homology( FilteredComplex cpx ) {
    for( Simplex_handle sh : cpx.filtration_simplex_range() ) {
        int dim = cpx.dimension(sh);
        update_cohomology_groups( dim, sh, cpx );
            //inside update_cohomology_groups
            for( Simplex_handle b_sh : cpx.boundary_simplex_range(sh) ) {...}
            //out
} } }
```

Figure 3: Sample code for the computation of persistence, illustrating the use of a model of concept `FilteredComplex`.

`linear_indexing_tag` or `zigzag_indexing_tag`, corresponding to the two indexing schemes of interest mentioned above. The tag is passed as template argument to a model of the concept `FilteredComplex` (described below and representing filtered cell complexes).

3.2 FilteredComplex Concept:

We define the concept `FilteredComplex` that describes the requirement for a type to implement a filtered cell complex. We use the vocabulary of simplicial complexes, but the concept is valid for any type of cell complex. The main requirements are the definition of:

1. type `Indexing_tag`, which is a model of the concept `IndexingTag`, describing the nature of the indexing scheme,
2. type `Simplex_handle` to manipulate simplices,
3. method `int dimension(Simplex_handle)` returning the dimension of a simplex,
4. type and method `Boundary_simplex_range boundary_simplex_range(Simplex_handle)` that returns a range giving access to the codimension 1 subsimplices of the input simplex, as-well-as the coefficients $(-1)^i$ in the definition of the operator ∂. The iterators have value type `Simplex_handle`,
5. type and method `Filtration_simplex_range filtration_simplex_range()` that returns a range giving access to all the simplices of the complex read in the order assigned by the indexing scheme,
6. type and method `Filtration_value filtration(Simplex_handle)` that returns the value of the filtration on the simplex represented by the handle.

Figure 3 illustrates the use of a model of the concept `FilteredComplex`. It sketches the algorithm used for computing persistent homology via the approach of [11, 13].

3.3 PersistentHomology Concept:

The concept `PersistentHomology` describes the requirement for a type to compute the persistent homology of a filtered complex. The requirement are the definition of:

1. a type `Filtered_complex`, which is a model of `FilteredComplex` and provides the type of complex on which persistence is computed,

2. a type `Coefficient_field`, which is a model of `CoefficientField` and provides the coefficient field on which homology is computed.

The requirements of the concept `CoefficientField` are essentially the definition of field operations (addition, multiplication, inversion, etc).

We refer to Figure 2 for a presentation of the concepts and their connections.

4 Implementation

In this section we describe how these concepts are implemented. The code will be available soon at `project.inria.fr/gudhi/software/`.

4.1 Simplicial Complex:

We use a *Simplex Tree* [4] to represent simplicial complexes. The class `Simplex_tree` is a model of `FilteredComplex` and hence furnishes all requirements of the concept. Moreover, it furnishes algorithms to construct efficiently simplicial complexes, and in particular flag complexes [14]. Details on the implementation of the algorithms may be found in [4].

4.2 Persistent Homology:

We use the *Compressed Annotation Matrix* [2] to implement the persistent cohomology algorithm [11, 13] for persistence. This leads to the class `Persistent_cohomology`, which is a model of `PersistentHomology`. The class `Persistent_cohomology` allows the computation of the persistence diagram of a filtered complex, using the method `compute_persistent_homology` (see Figure 3).

The coefficient fields available as models of `CoefficientField` are `Field_Zp` for $\mathbb{Z}_p$ (for any prime p) and `Multi_field` for the multi-field persistence algorithm – computing persistence simultaneously in various coefficient fields – described in [3].

4.3 Example of Use of the Library:

Figure 4 illustrates the user interface for constructing a flag complex [14] from a graph and computing its persistent homology with various coefficient fields.

```
Graph g; ...  //compute the graph
Simplex_tree< linear_indexing_tag > st; //linear ordering
st.insert(g); //insert the graph as 1-skeleton of the complex
st.expand(5); //construct the 5-skeleton of the associated flag complex
Persistent_cohomology< Simplex_tree<linear_indexing_tag>, Multi_field > pcoh;
    //persistence with "multi field coefficients" defined on a simplex tree
pcoh.compute_persistent_homology(st,2,1223); //compute persistent homology of st in all
    fields Zp for p prime between 2 and 1223
```

Figure 4: User interface for the construction of a filtered flag complex with a simplex tree and the computation of its persistent homology.

Data	$\lvert\mathcal{P}\rvert$	D	d	r	$\lvert\mathcal{K}\rvert$	T_{st}	$T^{\text{ph}}_{\mathbb{Z}_2}$	$T^{\text{ph}}_{\mathbb{Z}_{1223}}$	$T^{\text{ph}}_{\mathbb{Z}^2_{1223}}$
Bud	49,990	3	2	0.09	$127 \cdot 10^6$	5.7	161	161	252
Bro	15,000	25	?	0.04	$142 \cdot 10^6$	5.8	252	252	380
Cy8	6,040	24	2	0.8	$193 \cdot 10^6$	8.4	249	249	325
Kl	90,000	5	2	0.25	$114 \cdot 10^6$	8.3	228	227	401
S3	50,000	4	3	0.65	$134 \cdot 10^6$	7.2	176	176	310

Figure 5: Timings in seconds for the various algorithms.

4.4 Experiments:

Figure 5 presents timings T_{st} for the construction of flag complexes with a simplex tree using the algorithm of [4], $T^{\text{ph}}_{\mathbb{Z}_2}$ and $T^{\text{ph}}_{\mathbb{Z}_{1223}}$ for the computation of persistent homology with coefficient is $\mathbb{Z}_2$ and $\mathbb{Z}_{1223}$ respectively, using the implementation of [2], and $T^{\text{ph}}_{\mathbb{Z}^2_{1223}}$ for the simultaneous computation of persistent homology in the 200 coefficient fields $\mathbb{Z}_p$ with p prime, for $2 \leq p \leq 1223$, using the multi-field persistent homology algorithm described in [3]. Experiments have been realized on a Linux machine with 3.00 GHz processor and 32 GB RAM, for Rips complexes [14] built on a variety of data points. Datasets are listed in Figure 5 with the size of points sets $|\mathcal{P}|$, the ambient dimension D and intrinsic dimension d of the sample points ("?" if unknown), the parameter r for the Rips complex and the size of the complex $|\mathcal{K}|$. More details about the implementation, the experimental protocol, the data sets as-well-as additional experiments can be found in [2, 3, 4].

The average timings per simplex of the various algorithms are ranging between $4.08 \cdot 10^{-8}$ and $7.28 \cdot 10^{-8}$ seconds per simplex for the construction of the simplex tree, between $1.27 \cdot 10^{-6}$ and $2.00 \cdot 10^{-6}$ seconds per simplex for the computation of persistent homology with coefficient field $\mathbb{Z}_2$ or $\mathbb{Z}_{1223}$, and between $1.68 \cdot 10^{-6}$ and $3.52 \cdot 10^{-6}$ seconds per simplex for the computation of multi-field persistent homology in all fields $\mathbb{Z}_p$ for p prime, $2 \leq p \leq 1223$. Note that most of the time for the computation of persistent homology is spent computing boundaries in the simplex tree.

5 Future Components

The library may be extended in various directions that fit naturally in the design. The first direction is to allow zigzag indexing schemes, by the creation of a tag `zigzag_indexing_tag`. In this case, the method `filtration_simplex_range` must indicate the direction of the arrows.

New implementations and models for `FilteredComplex` may be added. For example, the construction of witness complexes [4] will be added to the class `Simplex_tree`. Additionnaly, new types of complexes (like cubical complexes) and new data structures to represent them may be added to the library: in order to compute their persistent homology, they only need to satisfy the requirements of the concept `FilteredComplex`.

So far, only inclusions have been considered for simplicial maps between simplicial complexes. As explained in [13], any simplicial map may be implemented with a sequence of inclusions and edge contractions. We will consequently add edge contractions as updates in the class `Simplex_tree` and implement the induced updates in the class `Persistent_cohomology` (algorithms exist for edge contractions in a simplex tree [4] and for the corresponding updates at the cohomology level [13]). This way, we will be able to compute persistent homology of simplicial maps. In this case, the range provided by `filtration_simplex_range` must indicate the nature of the map between complexes.

Future works include also the implementation of a class `Field_Q`, model of concept `CoefficientField`, for homology with $\mathbb{Q}$ coefficients. Finally the interface between complexes and persistent homology allows us to implement more persistent homology algorithms.

Acknowledgment

This research has been partially supported by the European Research Council under Advanced Grant 339025 GUDHI (Algorithmic Foundations of Geometric Understanding in Higher Dimensions).

References

[1] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. In *Topological Methods in Data Analysis and Visualization III*, pages 103–117. 2014.

[2] Jean-Daniel Boissonnat, Tamal K. Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. In *ESA*, pages 695–706, 2013.

[3] Jean-Daniel Boissonnat and Clément Maria. Computing Persistent Homology with Various Coefficient Fields in a Single Pass. Rapport de recherche RR-8436, INRIA, December 2013.

[4] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, pages 1–22, 2014.

[5] Peter Bubenik and Jonathan A. Scott. Categorification of persistent homology. *CoRR*, abs/1205.3669, 2012.

[6] Gunnar Carlsson, Tigran Ishkhanov, Vin Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *Int. J. Comput. Vision*, 76:1–12, January 2008.

[7] Gunnar E. Carlsson and Vin de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.

[8] F. Chazal and S. Oudot. Towards persistence-based reconstruction in euclidean spaces. In *Proc. 24th. Annu. Sympos. Comput. Geom.*, pages 231–241, 2008.

[9] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[10] V. de Silva and R. Ghrist. Coverage in sensor network via persistent homology. *Algebraic & Geometric Topology*, 7:339–358, 2007.

[11] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discrete & Computational Geometry*, 45(4):737–759, 2011.

[12] Harm Derksen and Jerzy Weyman. Quiver representations. *Notices of the AMS*, 52(2):200–206, 2005.

[13] Tamal K. Dey, Fengtao Fan, and Yusu Wang. Computing topological persistence for simplicial maps. In *Symposium on Computational Geometry*, page 345, 2014.

[14] Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010.

[15] James R. Munkres. *Elements of algebraic topology*. Addison-Wesley, 1984.

[16] Afra Zomorodian and Gunnar E. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

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