Computing Persistent Homology within Coq/SSReflect*

Jónathan Heras[1], Thierry Coquand[2], Anders Mörtberg[2], and Vincent Siles[2]

[1] School of Computing, University of Dundee, UK
[2] Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Sweden
jonathanheras@computing.dundee.ac.uk,
{coquand,mortberg,siles}@chalmers.se

Abstract. *Persistent homology* is one of the most active branches of *Computational Algebraic Topology* with applications in several contexts such as optical character recognition or analysis of point cloud data. In this paper, we report on the formal development of certified programs to compute *persistent Betti numbers*, an instrumental tool of persistent homology, using the Coq proof assistant together with the SSReflect extension. To this aim it has been necessary to formalize the underlying mathematical theory of these algorithms. This is another example showing that interactive theorem provers have reached a point where they are mature enough to tackle the formalization of nontrivial mathematical theories.

Keywords: Persistent Homology, Computational Algebraic Topology, Formalization of Mathematics, Coq, SSReflect.

1 Introduction

Persistent homology is a branch of Algebraic Topology which appeared simultaneously in three works during the last five years of the 20th century, see [10,18,37]. Since that time it has become one of the central tools in the context of Computational Algebraic Topology and several applications and extensions have been developed.

In a nutshell, persistent homology is a technique which allows one to study the *lifetime* of *topological attributes*; this can be really useful in different contexts such as point cloud data [19], optical character recognition [31], sensor networks [9] and surface reconstruction from noisy samples [11]. The main idea of all of these applications is that relevant features will be *long-lived* in the sense that they persist over a certain parameter range, on contrast with the "noise" which will be *short-lived*.

* Partially supported by Ministerio de Educación y Ciencia, project MTM2009-13842-C02-01, and by the European Union's 7th Framework Programme under grant agreement nr. 243847 (ForMath).

In this work, the main notions about persistent homology have been formalized using the Coq proof assistant [8] together with the SSReflect extension [22]. During such a process we have proved relevant theorems like the *Fundamental Lemma of Persistent Homology* [16, pp. 152]. Moreover, we have implemented certified programs to compute *persistent Betti numbers*, an instrumental tool in the context of persistent homology.

The rest of this paper is organized as follows. The next section is devoted to present the mathematical notions and results which will be formalized using Coq/SSReflect in Section 3. The effective algorithms to compute persistent homology and some experiments performed with such programs are introduced respectively in Section 4 and Section 5. The paper ends with a section of conclusions and further work.

The interested reader can consult the original and complete source code which can be found at [26].

2 Mathematical background

In this section, we briefly provide the necessary mathematical background needed to understand the paper. We mainly focus on definitions, some of them are well known notions of *Algebraic Topology*, see [32], and the rest comes from *persistent homology theory* [17,40,16]. We start by presenting simplicial complexes, a combinatorial object which can be understood as a generalization of graphs to higher dimensions.

2.1 Simplicial complexes

Let V be an *ordered set*, called the *vertex set*. An *(abstract) simplex* over V is any finite subset of V. An *(abstract) n-simplex* over V is a simplex over V whose cardinality is equal to $n+1$. Given a simplex α over V, we call the subsets of α *faces*.

Definition 1 An *(ordered abstract) simplicial complex* over V is a set of simplices $\mathcal{K}$ over V such that it is closed by taking faces (subsets), that is, if $\alpha \in \mathcal{K}$ then all the faces of α are in $\mathcal{K}$ too.

Let $\mathcal{K}$ be a simplicial complex, the set $S_n(\mathcal{K})$ of n-simplices of $\mathcal{K}$ is the set made of the simplices of cardinality $n+1$.

Example 1. Let us consider $V = (0,1,2,3,4,5)$. The small simplicial complex drawn in Figure 1 is mathematically defined as the object:

$$\mathcal{K} = \left\{ \begin{array}{l} (0),(1),(2),(3),(4),(5),(0,1),(0,2),(1,2),(2,3) \\ (3,4),(3,5),(4,5),(0,1,2) \end{array} \right\}.$$

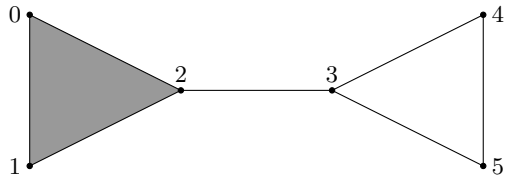


Fig. 1. "Diabolo" Complex

Definition 2 Let $\mathcal{K}$ be a simplicial complex over V. Let n and i be two integers such that $n \geq 1$ and $0 \leq i \leq n$. Then the *face operator* ∂_i^n is the map $\partial_i^n : S_n(\mathcal{K}) \to S_{n-1}(\mathcal{K})$ defined by:

$$\partial_i^n((v_0, \ldots, v_n)) = (v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$$

where the i-th vertex of the simplex is removed, so that a $(n-1)$-simplex is obtained.

2.2 Chain complexes

Now, we introduce a central notion in Algebraic Topology. Notions like rings, modules over a ring and module morphisms (see [30] for details) are assumed to be known.

Definition 3 A *chain complex* C_* is a pair of sequences $(C_n, d_n)_{n\in\mathbb{Z}}$ where for every $n \in \mathbb{Z}$, C_n is a $\mathcal{R}$-module (with $\mathcal{R}$ a ring) and $d_n : C_n \to C_{n-1}$ is a module morphism, called the *differential map*, such that the composition $d_n d_{n+1}$ is null (this is known as *nilpotency* condition).
The module C_n is called the module of *n-chains*. The image $B_n = im\ d_{n+1} \subseteq C_n$ is the (sub)module of *n-boundaries*. The kernel $Z_n = \ker d_n \subseteq C_n$ is the (sub)module of *n-cycles*.

Once we have defined the notions of simplicial complexes and chain complexes, we can define the link between them considering $\mathbb{Z}_2$ as the ground ring. As $\mathbb{Z}_2$ is a field the chain groups are vector spaces.

Definition 4 Let $\mathcal{K}$ be a simplicial complex over V. Then *the chain complex* $C_*(\mathcal{K})$ *canonically associated with* $\mathcal{K}$ is defined as follows. The chain group $C_n(\mathcal{K})$ is the free $\mathbb{Z}_2$-module generated by the n-simplices of $\mathcal{K}$. In addition, let $(v_0, \ldots, v_n)$ be a n-simplex of $\mathcal{K}$, the differential of this simplex is defined as:

$$d_n := \sum_{i=0}^{n} \partial_i^n.$$

$C_n(\mathcal{K})$ is a free module and the n-simplices form the *standard basis* of it. Therefore, for all n we can represent $d_n : C_n(\mathcal{K}) \to C_{n-1}(\mathcal{K})$ relative to the standard basis of the chain groups as a $\mathbb{Z}_2$ matrix. Such a matrix is called *the n-th incidence matrix* of a simplicial complex.

Let us present an example in order to clarify the notion of chain complex canonically associated with a simplicial complex.

Example 2. Let $\mathcal{K}$ be the simplicial complex defined in Figure 1. The chain complex $C_*(\mathcal{K})$ canonically associated with $\mathcal{K}$ is:

$$\cdots \to 0 \to C_2(\mathcal{K}) \xrightarrow{d_2} C_1(\mathcal{K}) \xrightarrow{d_1} C_0(\mathcal{K}) \to 0 \to \cdots$$

where the 3 associated chain groups are:

- $C_0(\mathcal{K})$, the free $\mathbb{Z}_2$-module on the set of 0-simplices (vertices) $\{(0), (1), (2), (3), (4), (5)\}$.
- $C_1(\mathcal{K})$, the free $\mathbb{Z}_2$-module on the set of 1-simplices (edges) $\{(0,1), (0,2), (1,2), (2,3), (3,4), (3,5), (4,5)\}$.
- $C_2(\mathcal{K})$, the free $\mathbb{Z}_2$-module on the set of 2-simplices (triangles) $\{(0,1,2)\}$.

and the first incidence matrix, d_1, is:

$$\begin{array}{c} \\ (0) \\ (1) \\ (2) \\ (3) \\ (4) \\ (5) \end{array} \begin{array}{c} \begin{array}{ccccccc} (0,1) & (0,2) & (1,2) & (2,3) & (3,4) & (3,5) & (4,5) \end{array} \\ \left(\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array} \right) \end{array}$$

Finally, we introduce one of the most important notions in the context of Computational Algebraic Topology. Given a chain complex $C_* = (C_n, d_n)_{n \in \mathbb{Z}}$, the identities $d_{n-1} \circ d_n = 0$ mean that the inclusion relations $B_n \subseteq Z_n$ hold, that is, every boundary is a cycle (the converse is generally not true). Thus the next definition makes sense.

Definition 5 The *n-th homology group* of C_*, denoted by $H_n(C_*)$, is defined as the quotient $H_n(C_*) = Z_n/B_n$. The elements of $H_n(C_*)$ are called *n-dimensional homology classes* of C_*.

The *n-th Betti number* of C_*, denoted by $\beta_n(C_*)$, is the *rank* of the n-th homology group of C_*.

In an intuitive sense, the n-th Betti number of an object X measures the number of n holes of X; to be more concrete, β_0 measures the number of connected components, and the Betti numbers β_n, with $n > 0$, measure higher dimensional connectedness. For instance, β_0 and β_1 of the simplicial complex of Figure 1 are equal to 1 (1 connected component and 1 hole), and its β_n with $n > 1$ are equal to 0 since it is a 2D object. As another example, we can consider the *sphere* whose β_0, β_1 and β_2 are respectively 1, 0, and 1 (1 connected component, 0 holes and 1 cavity).

The homology groups of a simplicial complex $\mathcal{K}$ are the ones associated with the chain complex $C_*(\mathcal{K})$. Moreover, Betti numbers of a simplicial complex can be easily computed considering the representation of the differential maps as matrices using the formula:

$$\beta_n(C_*(\mathcal{K})) = ns - rank(d_n) - rank(d_{n+1}) \tag{1}$$

where ns is the number of n-simplices.

2.3 Persistent Homology

We end this section by introducing the instrumental notions in persistent homology theory. A more detailed description of this theory can be found in [17,40,16].

Definition 6 Let $\mathcal{K}$ be a simplicial complex, a *subcomplex* of $\mathcal{K}$ is a subset $\mathcal{L} \subseteq \mathcal{K}$ that is also a simplicial complex. A *filtration* of $\mathcal{K}$ is a nested subsequence of complexes:

$$K^0 \subseteq K^1 \subseteq \ldots \subseteq K^m = \mathcal{K}$$

An example of a filtration can be seen in Figure 2 taking the diabolo complex of Figure 1 as $\mathcal{K}$.

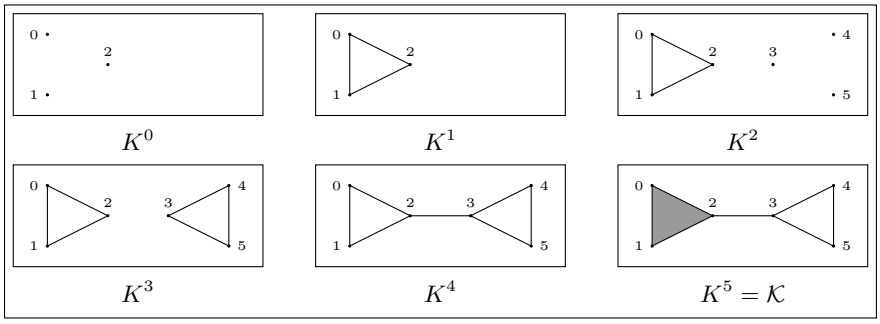
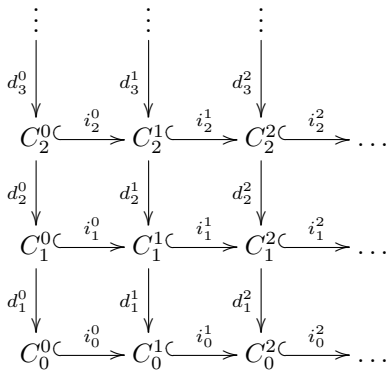


Fig. 2. Filtration of the diabolo simplicial complex

Given a filtration of a simplicial complex and the j-th component of the filtration, let us say K^j, we will denote $C_n(K^j), Z_n(K^j)$ and $B_n(K^j)$ by C_n^j, Z_n^j and B_n^j respectively. Therefore, we can represent the chain complexes associated with a filtration using the following diagram.



where i_n^j is the map induced by the inclusion between the n-simplices of K^j and the ones of K^{j+1}. Moreover, for $j < p$ we will use $i_n^{j,p}$ to denote the map induced by the inclusion between the n-simplices of K^j and the ones of K^p. Now, we can introduce the notion of persistent homology groups.

Definition 7 The *p-persistent n-th homology group* of K^j, denoted by $H_n^{j,p}$, is defined as the quotient $H_n^{j,p} = i_n^{j,p}(Z_n^j)/(B_n^p \cap i_n^{j,p}(Z_n^j))$.

The *p-persistent n-th Betti number* of K^j, denoted by $\beta_n^{j,p}$, is defined as the *rank* of $H_n^{j,p}$.

The elements of $H_n^{j,p}$ are the n-dimensional homology classes of K^j which are still alive at K^p. Hence $\beta_n^{j,p}$ measures the number of n-dimensional classes of K^j which are still alive at K^p. For instance, $\beta_0^{2,3}$ of the filtration depicted in Figure 2 is equal to 2, this means that there are two connected components, the triangle $(0, 1, 2)$ and the vertex (3), of K^2 which are still alive at K^3.

In order to shed light on the meaning of persistent homology, we introduce the usual way of visualizing persistence. The lifetime of a n-dimensional homology class can be represented as an interval; namely, a homology class which is born at level K^i of the filtration and dies entering K^j (with $i < j$) is represented as the interval $[i, j)$, and if it is born at level K^i but never dies we use the interval $[i, \infty)$. A *barcode* is defined to be the set of resulting intervals of a filtration.

Example 8 The barcodes in degree 0 and 1 associated with the filtration of Figure 2 are the ones depicted in Figure 3. In the case of the β_0 barcode, the vertex (0) is a connected component which *is born* at level 0 of the filtration and lives forever; on the contrary, for example, the vertex (3) *is born* at level 2 of the filtrations and *dies* entering the level 4 when it is merged with another connected component. In the case of the β_1 barcode (where holes are the 1-homology classes), a hole appears at level 1 of the filtration between the edges $(0, 1), (0, 2), (1, 2)$ and dies entering the last level of the filtration when it is filled with the triangle $(0, 1, 2)$; on the other hand, the hole which appears at level 3 of the filtration never dies.

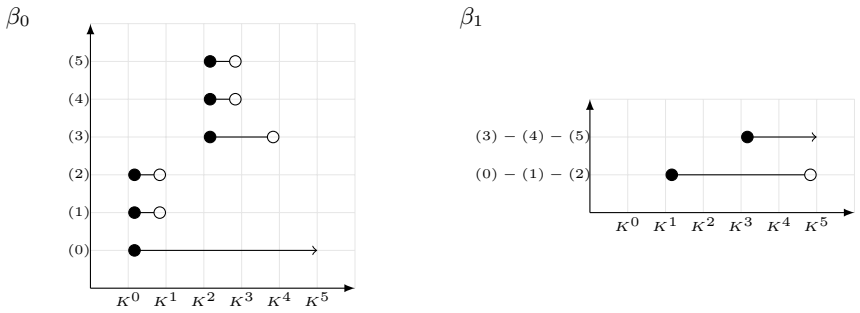


Fig. 3. Barcodes of the diabolo filtration

If we are interested in computing the n-dimensional homology classes which *are born* at K^j and *die* entering K^p, we have the following formula:

$$\mu_n^{j,p} = (\beta_n^{j,p-1} - \beta_n^{j,p}) - (\beta_n^{j-1,p-1} - \beta_n^{j-1,p}). \tag{2}$$

The first difference on the right hand side of the above formula measures the number of n-dimensional classes which are born at or before K^j and die entering K^p, and the second one counts the number of n-dimensional classes which are born at or before K^{j-1} and die entering K^p. Now, we can state the Fundamental Lemma of Persistent Homology.

Theorem 9 (Fundamental Lemma of Persistent Homology) Let $K^0 \subseteq K^1 \subseteq \ldots \subseteq K^m = \mathcal{K}$ be a filtration. For every pair of indices $0 \leq k \leq l \leq m$ and every dimension n, the l-persistent n-th Betti number of K^k is

$$\beta_n^{k,l} = \sum_{0 \leq i \leq k} \sum_{l < j \leq m} \mu_n^{i,j} + \beta_n^{k,m}.$$

The importance of the Fundamental Lemma of Persistent Homology lies in the fact that it says that the barcodes encode all the information about persistent homology groups.

This version of the Fundamental Lemma of Persistent Homology is stated slightly different from the one presented in Edelsbrunner and Harer [16, pp.152]; however both of them are equivalent. There are two differences between the statements: The former is related to the definition of $\mu_n^{j,p}$ which is defined for the case $p \in \mathbb{N}$ and has to be extended for $p = \infty$. In particular for such a case, we have the following formula:

$$\mu_n^{j,\infty} = \beta_n^{j,\infty} - \beta_n^{j-1,\infty}$$

where $\beta_n^{j,\infty}$ is defined as the n-dimensional classes which are born at or before K^j and never die (in other words, dies in the ∞). It is worth noting that $\beta_n^{j,\infty}$ is equal to $\beta_n^{j,m}$ (where m is the last level of the filtration).

The latter difference is a consequence of the former one and is related to the inner sum of the theorem which will be an infinite sum. In particular, the Edelsbrunner and Harer's Fundamental Lemma of Persistent Homology is stated as follows.

Theorem 10 (Fundamental Lemma of Persistent Homology) [16, pp.152] Let $K^0 \subseteq K^1 \subseteq \ldots \subseteq K^m = \mathcal{K}$ be a filtration. For every pair of indices $0 \leq k \leq l \leq m$ and every dimension n, the l-persistent n-th Betti number of K^k is

$$\beta_n^{k,l} = \sum_{0 \leq i \leq k} \sum_{l < j} \mu_n^{i,j}.$$

As we have said previously, both formulations of the theorems are equivalent; however, the one presented in Theorem 9 is more suitable to be formalized since we do not need to handle infinite sums in the Coq/SSReflect theorem prover.

3 An abstract formalization using Coq/SSReflect

Let us now introduce an abstract formalization of the notions presented in the previous section using Coq together with the SSReflect extension.

3.1 Simplicial Complexes and Homology

In previous work, see [28,27], we have formalized the notions presented in subsections 2.1 and 2.2. However, for the sake of clarity of the exposition we include the main definitions and results which have been developed previously.

We begin with the notions related to simplicial complexes. The set of vertices is represented by a finite type `V` (i.e. a type with finitely many elements which in addition has a canonical order associated with it). A simplex is defined as a finite set of vertices. Using this, the definition of a simplicial complex as a set of simplices closed under inclusion is straightforward:

```
Variable V : finType.
Definition simplex := {set V}.
Definition simplicial_complex (c : {set simplex}) :=
  forall x, x \in c -> forall y : simplex, y \subset x -> y \in c.
```

The definition of the n-th incidence matrix of a simplicial complex, which is called `incidence_mx_n`, takes two arguments: a set of simplices `c` and the dimension `n`, and returns a SSReflect matrix. Moreover, we have proved the *nilpotency condition* (see Definition 3) for two consecutive incidence matrices encoded with `incidence_mx_n`.

```
Theorem incidence_matrices_sc_product:
  forall (V:finType) (n:nat) (sc: {set (simplex V)}),
    simplicial_complex sc ->
      (incidence_mx_n sc n) *m (incidence_mx_n sc (n.+1)) = 0.
```

The notion of homology is defined in Coq as follows. Let F be a field, $V1, V2, V3$ vector spaces on F, and $f : V1 \to V2$, $g : V2 \to V3$ linear maps such that $g \circ f = 0$; then, the Homology of f and g is the quotient between the kernel of g and the image of f. This can be defined in Coq in the following way.

```
Variables (F : fieldType) (V1 V2 V3 : vectType F)
          (f : 'Hom(V1,V2)) (g : 'Hom(V2,V3)).
Definition Homology := ((lker g) :\: (limg f)).
Definition Betti := \dim Homology.
```

Finally, this definition of homology can be instantiated for the homology in degree n of a simplicial complex sc using the linear maps associated with the incidence matrices in dimension $n+1$ and n. Given a matrix `M`, the instruction `Vector.Hom(M)` builds the linear map associated with `M`. It is necessary to transpose the incidence matrices to obtain the correct definition of homology groups associated with them.

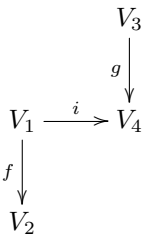
```
Definition Homology_sc_n (sc : {set simplex V}) (n : nat) :=
  Homology (Vector.Hom (incidence_mx_n sc n.+1)^T)
           (Vector.Hom (incidence_mx_n sc n)^T).
```

Analogously, we can define the n-th Betti number of a simplicial complex sc instantiating the `Betti` definition.

```
Definition Betti_sc_n (sc : {set simplex V}) (n : nat) :=
  Betti (Vector.Hom (incidence_mx_n sc n.+1)^T)
        (Vector.Hom (incidence_mx_n sc n)^T).
```

3.2 Persistent homology

In this section, we formalize the results presented in Subsection 2.3. First of all, we define a more generic notion than the one of the persistent homology group $H_n^{j,p}$ associated with a filtration. Such a notion involves the elements presented in the following diagram.



where V_1, V_2, V_3 and V_4 are vector spaces over a field F, $f : V_1 \to V_2$ and $g : V_3 \to V_4$ are linear maps and $i : V_1 \to V_3$ is an injective linear map. Using this we can define the vector space $P_{f,g,i}$, called `PHomology` in Coq, as the quotient $i(\ker(f))/(im(g) \cap i(\ker(f)))$. We use the `vector` library of SSReflect [21] in order to define this notion.

```
Variables (F : fieldType) (V1 V2 V3 V4 : vectType F)
          (f : 'Hom(V1,V2)) (g : 'Hom(V3,V4)) (i : 'Hom(V1,V4)).

Hypothesis (i_inj : injective i).

Definition PHomology :=
  (i @: (lker f)) :\: ((limg g) :&: (i @: (lker f))).
```

As i is an injective linear map and $(im(g) \cap i(\ker(f)))$ is a subspace of $i(\ker(f))$, the dimension of $P_{f,g,i}$ is equal to the dimension of $\ker(f)$ (which in turn is equal to the dimension of V_1 minus the dimension of $im(f)$) minus the dimension of $im(g) \cap i(\ker(f))$. This definition and its correctness are introduced in Coq as follows:

```
Definition PBetti := \dim PHomology.

Lemma PBettiE : PBetti = Vector.dim V1 - \dim (limg f) -
                      \dim ((limg g) :&: (i @: (lker f))).
```

We omit the formal correctness proof for readability, we refer the interested reader to look at the actual formalization [26].

Let us now present how we instantiate these definitions for the persistent homology group $H_n^{j,p}$ associated with a filtration. A filtration is defined as a sequence of sets of simplices satisfying both that every element of the sequence is a simplicial complex and that the elements form a nested increasing sequence of sets.

```
Definition filtration (f : seq {set simplex V}) :=
    (forall x, x \in f -> simplicial_complex x) /\
    (forall i j, i <= j -> j < size f ->
              (nth set0 f i) \subset (nth set0 f j)).
```

In order to define the inclusion matrix $i_n^{j,p}$ of a filtration, we first introduce the more generic notion of inclusion matrix of two sequences of simplices. This inclusion matrix will be indexed by two sequences of simplices called `Left` and `Top`.

We can access to the elements of `Left` and `Top` using the function `nth`. A coefficient a_{ij} of the inclusion matrix will be 1 if the `i`-th simplex of `Left` is equal to the `j`-th simplex of `Top` and 0 otherwise.

Therefore, we can define the inclusion matrix of two sequences of simplices as follows.

```
Variables Left Top : seq(simplex).
Variables m n : nat.

Definition inclusionMatrix :=
  \matrix_(i < m, j < n)
    if (nth set0 Left i == nth set0 Top j) then 1 else 0:'F_2.
```

The type annotation `0:'F_2` indicates that the 0 and 1 appearing as coefficients of the matrix are the two elements of `F_2`, that is, $\mathbb{Z}_2$ as a field. The values of `m` and `n` will be latter assigned to define a concrete `inclusionMatrix`.

We now define the inclusion matrix $i_n^{j,p}$ of a filtration `f` by instantiating `Left` and `Top` to the set of n-simplices of the `j` and, respectively, `p` component of `f`. It is worth mentioning that, in SSReflect, finite sets are equipped with a canonical enumeration, and can be transformed to sequences using the `enum` function.

```
Variable f : (seq {set (simplex V)}).
Variables n j p : nat.

Definition n_simplices (c : {set (simplex V)}):=
  [set x \in c | #|x|==n.+1].
Definition n_simplices_k (k : nat) :=
  n_simplices (nth set0 f k) n.
Definition inclusion_mx := inclusionMatrix
  (enum (n_k_simplices j)) (enum (n_k_simplices p)).
```

We have proved that the linear map associated with `inclusion_mx` is injective.

```
Lemma injective_LinearApp_inclusion_mx :
  injective (Vector.Hom (inclusion_mx)).
```

Now we have all of the necessary ingredients to define the persistent homology group $H_n^{j,p}$ and the persistent Betti number $\beta_n^{j,p}$ just instantiating `P_fgi` and `PBetti` with the linear maps associated with the matrices d_n^j, d_{n+1}^p and $i_n^{j,p}$. These matrices are encoded as `(incidence_mx_n (nth set0 f j) n)`, `(incidence_mx_n (nth set0 f p) n.+1)` and `(inclusion_mx f n j p)` respectively.

```
Variable (V:finType) (f: (seq {set (simplex V)})) (n j p:nat).
Hypothesis f_is_filtration : filtration f.
Hypothesis j_is_in_filtration : j < size f.
Hypothesis j_leq_p_is_in_filtration : j <= p < size f.

Definition p_persistent_n_homology_K_j :=
 PHomology (Vector.Hom (incidence_mx_n (nth set0 f j) n)))
           (Vector.Hom (incidence_mx_n (nth set0 f p) n.+1))
           (Vector.Hom (inclusion_mx f n j p)).

Definition p_persistent_n_betti_K_j :=
 PBetti (Vector.Hom (incidence_mx_n (nth set0 f j) n))
        (Vector.Hom (incidence_mx_n (nth set0 f p) n.+1))
        (Vector.Hom (inclusion_mx f n j p)).
```

Finally, we can define $\mu_n^{j,p}$ (see Formula 2) and prove our version of the Fundamental Lemma of Persistent Homology (Theorem 9).

```
Theorem fundamental_lemma_persistent_homology (k l : nat)
  (H : l <= size f) :
  \sum_(0<=j<k.+1) (\sum_(l.+1 <= p < (size f).+1) (mu n j p)) =
  (p_persistent_n_betti_K_j f n k l) -
  (p_persistent_n_betti_K_j f n k (size f)).
```

The `bigop` library of SSReflect, see [5], has played a key role in the proof of the above theorem. This library is devoted to generic indexed big operations, like $\sum_{i=0}^{n} f(i)$ or $\bigcap_{i \in I} f(i)$, and their properties. Again, the interested reader can consult the whole development of the formal proof of the Fundamental Lemma of Persistent Homology in [26].

4 An effective certified implementation

One of the goals of this work was the development of certified programs to compute both Betti and persistent Betti numbers. In the previous section we have provided the definitions of such notions given in terms of linear maps of vector spaces. However, we do not usually work with linear maps when computing Betti and persistent Betti numbers but with the matrices representing those linear maps.

Equation 1 provides the explicit formula to compute Betti numbers from two matrices. So we can use it to define this new notion using SSReflect matrices (where `'M[K]_(m,n)` is a $m \times n$ matrix over `K`) and prove that this new notion is equivalent to the one given by the `Betti` definition.

```
Definition Betti_rank (mxf:'M[K]_(v1,v2)) (mxg:'M[K]_(v2,v3)) :=
  v2 - \rank mxg - \rank mxf.

Lemma Betti_rankE (mxf:'M[K]_(v1,v2))
  (mxg:'M[K]_(v2,v3)), mxf *m mxg = 0 ->
  Betti_rank mxf mxg = Betti (Vector.Hom mxf) (Vector.Hom mxg).
```

Similarly, we can define persistent Betti numbers in terms of matrices and prove the equivalence between such a definition and the one given in `PBetti` definition.

```
Definition PBetti_rank (mxf:'M[K]_(v1,v2)) (mxg:'M[K]_(v3,v4))
  (mxi:'M[K]_(v1,v4)) :=
   (v1 - \rank mxf - (\rank mxg + (v1 - \rank mxf) -
   \rank (col_mx mxg (kermx mxf) *m mxi)))%N.

Lemma PBetti_rankE : forall (mxf:'M[K]_(v1,v2))
 (mxg:'M[K]_(v3,v4)) (mxi:'M[K]_(v1,v4)),
  injective (Vector.Hom mxi) ->
   PBetti_rank mxf mxg mxi =
   PBetti (Vector.Hom mxf) (Vector.Hom mxg) (Vector.Hom mxi).
```

However the use of SSReflect libraries may trigger heavy computations during deduction steps, that would not terminate within a reasonable amount of time. To handle this issue some definitions, like matrices, are locked in a way that do not allow direct computations.

To overcome this pitfall, we use the methodology presented in [15] whose key idea is the one of *refinements*. Roughly speaking, the correctness of mathematical algorithms are proved using all the high-level theory available in the SSReflect libraries and then the algorithms are refined to an implementation on simpler data structures that will be the ones running on the machine. In our particular case of matrices we use lists of lists as the low level data type for representing them.

The methodology presented in [15] has been implemented as a new library, built on top of SSReflect libraries, which is called CoqEAL [14]. This library includes the refinements of almost all the algorithms involved in the computation of Betti and persistent Betti numbers. To be more concrete, the only algorithm which has been necessary to refine is the one in charge of computing the row kernel of a matrix.

The `kermx` function is already available in the SSReflect library and implements the row kernel of a matrix. This algorithm has been refined into an efficient version, called `ker`, which works with abstract matrices. The equivalence between both algorithms has been proved in the following lemma.

```
Lemma eqmx_ker m n (M : 'M[K]_(m,n)) : ker M :=: kermx M.
```

Linear subspaces are represented in SSReflect by means of matrices which means that the same linear subspace may have multiple representations. The notation `:=:` checks whether the two matrices represent the same subspace. The reason that we cannot prove `ker M = kermx M` is that it is simply not true, the reason for this is that the `kermx` algorithm of SSReflect is implemented by picking an arbitrary nonzero pivot element using the choice mechanism of SSReflect [22]. However for the concrete implementation we cannot just pick an arbitrary nonzero pivot but we need to take one in an efficient manner and hence the output matrix may not be exactly equal to the one of `kermx` but only a matrix representing the same subspace.

After defining the `ker` algorithm, we have translated it to the low level data type, list of lists, as the function `ker_seqmx`. In the rest of the paper, the functions whose name ends with `_seqmx` are refined versions of matrix functions. Finally we have ensured that `ker_seqmx` perform the same operation as its high-level counterpart `ker` by proving:

```
Lemma ker_seqmxE : forall m n (M : 'M[K]_(m,n)),
  seqmx_of_mx _ (ker M) = ker_seqmx m n (seqmx_of_mx _ M).
```

This lemma says that computing first with `ker` and then converting to lists of lists (using `seqmx_of_mx`) give the same output as first converting and then computing using `ker_seqmx`.

Now, we have all the necessary programs to define an executable version of both Betti and persistent Betti numbers and prove the equivalence with their high-level versions.

```
Definition ex_Betti_rank (mxf mxg:seqmatrix K) :=
  v2 - (rank_elim_seqmx v2 v3 mxg) - (rank_elim_seqmx v1 v2 mxf).

Definition ex_PBetti_rank (mxf mxg mxi : seqmatrix K) :=
  let rf := rank_elim_seqmx v1 v2 mxf in
  let rg := rank_elim_seqmx v3 v4 mxg in
  v1 - rf - (rg + (v1 - rf) -
  rank_elim_seqmx (v3 + size (ker_seqmx v1 v2 mxf)) v4
      (col_seqmx mxg (mulseqmx (ker_seqmx v1 v2 mxf) mxi))).

Lemma ex_Betti_rankE:
  forall (mxf: 'M[K]_(v1,v2)) (mxg : 'M[K]_(v2,v3)),
   ex_Betti_rank (seqmx_of_mx _ mxf) (seqmx_of_mx _ mxg) =
      Betti_rank mxf mxg.

Lemma ex_PBetti_rank_PBetti_rank_E : forall (mxf: 'M[K]_(v1,v2))
  (mxg : 'M[K]_(v3,v4)) (mxi : 'M[K]_(v1,v4)),
  ex_PBetti_rank (seqmx_of_mx _ mxf) (seqmx_of_mx _ mxg)
                 (seqmx_of_mx _ mxi) = PBetti_rank mxf mxg mxi.
```

It is worth noting that the executable functions on matrices, represented as lists of lists, usually need the size of the matrices, as can be seen for instance in the `rank_elim_seqmx` function. Moreover, the function `seqmx_of_mx` is the one in charge of transforming abstract matrices into lists of lists (which are encoded as the type `seqmatrix`).

Following the same pattern, we have defined executable simplicial complexes and their connection with the computation of Betti and persistent Betti numbers. In particular, the `ex_Betti_sc` function takes as argument a simplicial complex `c` and a natural number `n` and computes the `n`-th Betti number of `c` and the `ex_p_persistent_n_Betti_K_j` function takes as argument a filtration `f` and three natural numbers `p,n,j` and computes the `p` persistent `n`-th Betti number of the `j` level of the filtration `f`. Some examples of the usage of these functions are introduced in the following section.

5 Experimental results

In this section we try to clarify how Betti and persistent Betti numbers can be computed within Coq. Let us start with the computation of Betti numbers of the simplicial complex of Figure 1.

Simplicial complexes are built in Coq providing their *facets*. A *facet* of a simplicial complex $\mathcal{K}$ is a maximal simplex with respect to the subset order $\subseteq$ among the simplices of $\mathcal{K}$. To construct the simplicial complex associated with a

sequence of facets, $\mathcal{F}$, we generate all the faces of the simplices of $\mathcal{F}$; subsequently, if we perform the set union of all the faces we obtain the simplicial complex associated with $\mathcal{F}$. This procedure has been implemented, and its correctness has been proved, using Coq in [28]. In the case of the diabolo complex of Figure 1 its facets are: $\{(2,3),(3,4),(3,5),(4,5),(0,1,2)\}$.

The procedure to compute Betti numbers of the diabolo complex is as follows. First, we define the list of facets:

```
Definition diabolo_facets : seq (seq 'I_6) :=
  [::[::2;3];[::3;4];[::3;5];[::4;5];[::0;1;2]].
```

and, subsequently, we compute Betti numbers (of dimension 0 and 1) using the instruction `ex_Betti_sc` which takes as arguments the facets of the simplicial complex and the dimension.

```
Eval vm_compute in ex_Betti_sc diabolo_facets 0.
Eval vm_compute in ex_Betti_sc diabolo_facets 1.
```

obtaining in both cases 1 (this means that the diabolo complex has a connected component and a hole) in just milliseconds. The tactic `Eval vm_compute` evaluates the goal using the optimized call-by-value evaluation bytecode based virtual machine of Coq.

The procedure to compute persistent Betti numbers is quite similar. First of all, we define the filtration providing the facets of each one of the simplicial complexes of the filtration. In the case of the diabolo filtration of Figure 2, the representation in Coq of the filtration is the following one:

```
Definition diabolo_filtration : seq (seq (seq 'I_6)) :=
 [::[::[::0];[::1];[::2]];
    [::[::0;1];[::1;2];[::0;2]];
    [::[::0;1];[::1;2];[::0;2];[::3];[::4];[::5]];
    [::[::0;1];[::1;2];[::0;2];[::3;4];[::4;5];[::3;5]];
    [::[::0;1];[::1;2];[::0;2];[::3;4];[::4;5];[::3;5];[::2;3]];
    [::[::0;1;2];[::3;4];[::4;5];[::3;5];[::2;3]]].
```

Using this, we can compute persistent Betti numbers by calling the function `ex_p_persistent_n_betti_K_j`. Therefore, we can combine the functions to compute Betti and persistent Betti numbers in order to obtain information about the filtration. For instance, if we want to know how many connected components which live at the level 0 of the filtration (this is computed by `(nth nil diabolo_filtration 0)`) are still alive at level 4, we use the following instructions:

```
Eval vm_compute in ex_Betti_sc (nth nil diabolo_filtration 0) 0.
Eval vm_compute in
  ex_p_persistent_n_Betti_K_j diabolo_filtration 0 0 4.
```

obtaining as result 3 and 1 respectively. This means that there are three connected components at level 0 of the filtration but just one of them is still alive at level 4.

As a benchmark to test the efficiency of Coq programs, we have considered several random simplicial complexes and filtrations generated from a fixed number of triangles. The results can be seen in Table 1 where we show for each number of triangles the times (in seconds) to compute both Betti and persistent Betti numbers.

Triangles	Betti	Persistent Betti
10	0.024 s	0.036 s
50	2 s	3 s
100	18 s	25 s
200	146 s	190 s
500	1856 s	2731 s

Table 1. Execution time, in seconds, for different simplicial complexes

Of course, our Coq programs take much more time to compute Betti and persistent Betti numbers than special purpose software packages such as Chomp [1] and the GAP homology package [13] for Betti numbers or *JavaPlex* [39] and Dionysus [34] for persistent Betti numbers. However, it is worth remarking that Coq is an Interative Theorem Prover and in this kind of systems, unlike Computer Algebra Systems or special purpose packages, efficient computational capabilities have not been the main goal up to now.

Nevertheless, things are changing and there is an on-going effort in the implementation of efficient mathematical algorithms running inside Coq. In this line, we can highlight the works on machine integers and arrays [4], efficient real numbers [23] or an approach which consists in internally compiling Coq terms to the functional programming language OCaml [33].

The importance of performing computations inside Coq is twofold. First of all, we are using algorithms whose correctness has been proved; therefore, we can be sure that the results are correct. Moreover, computations inside Coq can be used in proofs by reflection, which is something that cannot be done with computations in Computer Algebra Systems.

6 Conclusions and further work

In this paper we have presented a set of formally verified programs which allows us to *effectively* compute persistent Betti numbers within Coq. To carry out this task, it has been necessary a formalization of the basic notions related to persistent homology. Moreover, we have formalized relevant theorems like the Fundamental Lemma of Persistent Homology. This illustrates that Interactive Theorem Provers are mature enough to tackle the formalization of theories in non-trivial mathematical domains. This fact can be seen also in the proof of the Four Color Theorem [20]; in the Flyspeck project [25], devoted to the formal

proof of the Kepler conjecture [24]; or in the classification of finite groups [2]. The infrastructures developed in those projects allow one to formalize results like the ones presented throughout this paper in reasonable time.

One of our main concerns for the future is associated with the formalization of efficient mathematical algorithms. This is a necessary effort which has to be carried out before undertaking other of our goals: the application of our programs to biomedical problems.

Homological techniques have been successfully applied in the biomedical context, for instance to classify brain blood vessel data [36], to analyze 2D colonoscope images [35] and to measure the synaptic density of neurons [27]. In this environment it is necessary to have both efficient and reliable software systems; therefore the use of formally verified efficient algorithms seems desirable.

It is also appealing to use this work as a basis for further developments. We can tackle the formalization of different extensions of persistent homology; for instance, *multidimensional persistence* [7] or *ZigZag persistence* [6]. In addition, we would like to consider the approach presented in [38], where the usual computations of persistent homology over $\mathbb{Z}_2$ are generalized to the case of $\mathbb{Z}$.

In summary, we are working towards an efficient formal library of Computational Algebraic Topology. In this line of work we can mention the formalizations of Effective Homology [3,12] and Discrete Morse Theory [29], however more work is still necessary to reach our goal.

References

1. Chomp: Computational homology project. Software available at `http://chomp.rutgers.edu/software/`.
2. Mathematical components team homepage. `http://www.msr-inria.inria.fr/Projects/math-components`.
3. J. Aransay, C. Ballarin, and J. Rubio. A mechanized proof of the Basic Perturbation Lemma. *Journal of Automated Reasoning*, 40(4):271–292, 2008.
4. M. Armand, B. Grégoire, A. Spiwack, and L. Théry. Extending Coq with Imperative Features and Its Application to SAT Verification. In *Proceedings Interactive Theorem Proving 2010 (ITP'2010)*, volume 6172 of *Lecture Notes in Computer Science*, pages 83–98, 2010.
5. Y. Bertot, G. Gonthier, S. O. Biha, and I. Pasca. Canonical Big Operators. In *Proceedings 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLS'08)*, volume 5170 of *Lecture Notes in Computer Science*, pages 86–101, 2008.
6. G. Carlsson and V. de Silva. Zigzag persistence. *CoRR*, abs/0812.0197, 2008.
7. G. Carlsson and A. Zomorodian. The theory of multidimensional persistence. In *Proceedings of the 23rd annual symposium on Computational geometry (SCG '07)*, pages 184–193. ACM, 2007.
8. Coq development team. The Coq Proof Assistant, version 8.4. Technical report, 2012.
9. V. de Silva and R. Ghrist. Homological sensor networks. *Notices of the American Mathematical Society*, 54(1):10–17, 2007.

10. C. J. A. Delfinado and H. Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3 sphere. *Computer Aided Geometry Design*, 12:771–784, 1995.
11. T. K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *Computational Geometry: Theory and Applications*, 35(1):124–141, 2006.
12. C. Domínguez and J. Rubio. Effective Homology of Bicomplexes, formalized in Coq. *Theoretical Computer Science*, 412:962–970, 2011.
13. J. Dumas, F. Heckenbach, B. D. Saunders, and V. Welker. Gap homology package. Software available at `http://www.linalg.org/gap.html`, 2002.
14. M. Dénès, A. Mörtberg, and V. Siles. CoqEAL, the Coq Effective Algebra Library, 2012. `http://www-sop.inria.fr/members/Maxime.Denes/coqeal`.
15. M. Dénès, A. Mörtberg, and V. Siles. A Refinement Based Approach to Computational Algebra in Coq. In *Proceedings Interactive Theorem Proving 2012 (ITP'2012)*, volume 7406 of *Lectures Notes in Computer Science*, pages 83–98, 2012.
16. H. Edelsbrunner and J. L. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.
17. H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Computional Geometry*, 28:511–533, 2002.
18. P. Frosini and C. Landi. Syze theory as a topological tool for computer vision. *Pattern Recognition and Image Analysis*, 9:596–603, 1999.
19. R. Ghrist. Barcodes: the persistent topology of data. *Bulletin American Mathematical Society*, 45:61–75, 2008.
20. G. Gonthier. *Formal proof - The Four-Color Theorem*, volume 55. Notices of the American Mathematical Society, 2008.
21. G. Gonthier. Point-free, set-free concrete linear algebra. In *Proceedings Interactive Theorem Proving 2011 (ITP'2011)*, volume 6898 of *Lecture Notes in Computer Science*, pages 103–118, 2011.
22. G. Gonthier and A. Mahboubi. An introduction to small scale reflection in Coq. *Journal of Formal Reasoning*, 3(2):95–152, 2010.
23. B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *International Conference on Functional Programming 2002*, pages 235–246. ACM Press, 2002.
24. T. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162:1065–1185, 2005.
25. T. Hales. The flyspeck project fact sheet. Project description available at `http://code.google.com/p/flyspeck/`, 2005.
26. J. Heras, T. Coquand, A. Mörtberg, and V. Siles. Formalization of homology and persistent homology, 2012. `http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/ProofExamples#wp3ex6`.
27. J. Heras, M. Dénès, G. Mata, A. Mörtberg, M. Poza, and V. Siles. Towards a certified computation of homology groups for digital images. In *Proceedings 4th International Workshoph on Computational Topology in Image Context (CTIC'2012)*, volume 7309 of *Lecture Notes in Computer Science*, pages 49–57, 2012.
28. J. Heras, M. Poza, M. Dénès, and L. Rideau. Incidence simplicial matrices formalized in Coq/SSReflect. In *Proceedings 18th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning (Calculemus'2011)*, volume 6824 of *Lecture Notes in Computer Science*, pages 30–44, 2011.
29. J. Heras, M. Poza, and J. Rubio. Verifying an algorithm computing Discrete Vector Fields for digital imaging. In *Proceedings 19th Symposium on the Integration of*

Symbolic Computation and Mechanised Reasoning (Calculemus'2012), volume 7362 of *Lecture Notes in Computer Science*, pages 215–229, 2012.
30. N. Jacobson. *Basic Algebra II*. W. H. Freeman and Company, 2nd edition, 1989.
31. G. Kedenburg. Persistent cubical homology in pattern recognition. Diplomarbeit. Universität Hamburg, 2010.
32. S. MacLane. *Homology*. Springer, 1963.
33. Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. Full Reduction at Full Throttle. In *Proceedings Certified Programs and Proofs*, volume 7086 of *Lecture Notes in Computer Science*, pages 362–377, 2011.
34. D. Morozov. Dionysus. Software available at `http://www.mrzv.org/software/dionysus/`, 2012.
35. M. Mrozek et al. Homological methods for extraction and analysis of linear features in multidimensional images. *Pattern Recognition*, 45(1):285–298, 2012.
36. M. Niethammer et al. Analysis of blood vessel topology by cubical homology. *Image Rochester NY*, 2(2):969–972, 2002.
37. V. Robins. Towards computing homology from finite approximations. *Topology proceedings*, 24:503–532, 1999.
38. A. Romero, J. Heras, J. Rubio, and F. Sergeraert. Defining and computing persistent Z-homology in the general case. Technical report, 2012.
39. A. Tausz, M. Vejdemo-Johansson, and H. Adams. Javaplex: A research software package for persistent (co)homology. Software available at `http://code.google.com/javaplex`, 2011.
40. A. Zomorodian and G. Carlsson. Computing Persistent Homology. *Discrete and Computional Geometry*, 33:249–274, 2005.