Advances in Pure Mathematics, 2016, 6, 113-137
Published Online February 2016 in SciRes. http://www.scirp.org/journal/apm
http://dx.doi.org/10.4236/apm.2016.63010

Scientific Research Publishing

A Global Reduction Based Algorithm for Computing Homology of Chain Complexes

Madjid Allili, David Corriveau

Department of Mathematics, Bishop's University, Sherbrooke, Canada
Email: mallili@ubishops.ca

Received 22 September 2015; accepted 21 February 2016; published 26 February 2016

Abstract

In this paper, we propose a new algorithm to compute the homology of a finitely generated chain complex. Our method is based on grouping several reductions into structures that can be encoded as directed acyclic graphs. The organized reduction pairs lead to sequences of projection maps that reduce the number of generators while preserving the homology groups of the original chain complex. This sequencing of reduction pairs allows updating the boundary information in a single step for a whole set of reductions, which shows impressive gains in computational performance compared to existing methods. In addition, our method gives the homology generators for a small additional cost.

Keywords

Homology Algorithm, Chain Complex, Homology Generators

1. Introduction

Homology has been used recently in a wide variety of applications in domains such as dynamical systems, and image processing and recognition. In dynamics, typical problems are translated into problems in topology where invariants such as the Conley index are computed using homology algorithms. In digital image analysis, topological invariants are useful in shape description, indexation, and classification. Among shape descriptors based on homology theory, there are the Morse shape descriptor [1] [2], the Morse Connection Graph [3], and the persistence barcodes for shape [4]. The necessity of improved algorithms appears evident as new applications of the homology computation arise in research for very large data sets. Although several algorithms and software packages have been developed for this purpose, there is still a lot of room for improvement as processing very large data sets is often very time- and memory-consuming. The classical approach to compute homology of a chain complex with integer coefficients reduces to the calculation of the Smith Normal Form (SNF) of the

How to cite this paper: Allili, M. and Corriveau, D. (2016) A Global Reduction Based Algorithm for Computing Homology of Chain Complexes. *Advances in Pure Mathematics*, **6**, 113-137. http://dx.doi.org/10.4236/apm.2016.63010

boundary matrices which are in general sparse [5] [6].

Unfortunately, this approach has a very poor running-time and its direct implementation yields exponential bounds. Polynomial time algorithms for computing the SNF were given by Kannan & Bachem [7] and later improved by Chou & Collins [8] and Illiopoulos [9]. The best currently known Smith Normal Form algorithms have super cubical complexity [10]. Another generic approach is the method of reduction proposed in [5] [11]. In this method the original complex is simplified through a sequence of reductions of cells that preserve the homology groups at each step. The idea is to replace the chain complex (or even the object under study) by a smaller one with the same homology. A reduction consists of cancelling a cell B through an element of its boundary a (reduction pair (a,B)). The two cells are suppressed from the structure of the complex and the boundary homomorphisms are updated. Direct implementations of the reduction method use unordered lists to encode the boundary homomorphisms and have cubical complexity.

Several algorithms based on the idea of reduction and that improve the time complexity for particular types of data sets have been designed. For cubical sets embedded in $\mathbb{R}^3$, Mrozek *et al*. proposed a method [12] based on the computation of acyclic subspaces by using lookup tables. Experimentally, this method performed in linear time. Another method running in quasi linear time for finite simplicial complexes embedded in $\mathbb{R}^3$ was proposed by Delfinado and Edelsbrunner in [13]. For problems dealing with higher dimensions, Mrozek and Batko developed an algorithm [14] based on the concept of coreduction which showed interesting performance results on cubical sets.

Despite the existence of efficient homology algorithms for cubical sets and lower dimension spaces, there remain many problems where the data is higher dimensional and not cubical. Generally, for these problems it is often difficult to adapt the data because the cost is too high or too complex. In this case, one has to use one of the classical methods: SNF or reduction. However, the classical methods spend most of their time in updating the boundary homomorphisms after each reduction step. This requires repetitive manipulations of the data structures that store the chain complex which incurs a high running time to accomplish the reduction.

An idea is to identify admissible reduction pairs and organize them in a structure that allows efficiently updating the boundary information in a single step for a whole set of reductions. This reduces to the minimum manipulations of the data structures that store the boundary information. Our method works as follows. Starting at an arbitrary cell A with a face a in its boundary, we identify successive adjacent admissible pairs of reduction and build the longest possible sequence originating at A. Several sequences can originate at the same cell, and each sequence can be seen as a path in a directed acyclic graph, called a *reduction DAG*, where nodes are reduction pairs and oriented edges denote an adjacency relation between the reduction pairs. Given a reduction DAG across a chain complex, we achieve a global simplification of the complex by performing all the reductions in the DAG at once. We will establish direct algebraic formulas that allow updating the boundary of remaining cells. The net advantage of our approach is that the boundaries are not explicitly updated after each reduction step. Instead, this information is always preserved implicitly within the data structures and processed globally for each reduction DAG allowing reducing considerably the computational time. The communication [15] contains a summary of the method developed systematically here and the preliminary experimentations that are now developed in Section 5.

2. Chain Complexes and Homology

Chain complexes are an algebraic tool for defining and computing homology. Although they are usually defined from geometrical structures such as simplicial or cellular complexes, they can also be considered in an abstract manner as pure algebraic entities. A finitely generated free *chain complex* $(\mathcal{C},\partial)$ with coefficients in a ring $\mathcal{R}$ is a sequence of finitely generated free abelian groups $\{C_q\}_{q\in\mathbb{Z}}$ together with a sequence of homomorphisms, called boundary maps or operators, $\{\partial_q : C_q \to C_{q-1}\}_{q\in\mathbb{Z}}$ satisfying $\partial_{q-1}\circ\partial_q = 0$ for all $q\in\mathbb{Z}$. Typically, $C_q = 0$ for $q<0$. For each p, the elements of C_p are called p-chains and the kernel of $\partial_p : C_p \to C_{p-1}$ is called the group of p-cycles and denoted $Z_p = \{c\in C_p \,|\, \partial_p c = 0\}$. The image of ∂_{p+1}, called the group of boundaries, is denoted by $B_p = \{c\in C_p \,|\, \exists b\in C_{p+1} \text{ such that } \partial_{p+1} b = c\}$. B_p is a subgroup of Z_p because of the property $\partial_{p-1}\circ\partial_p = 0$. The quotient groups $H_p := Z_p / B_p$ are the homology groups of the chain complex $\mathcal{C}$.

Computation of Homology by Reduction of Chain Complexes

The reduction of a chain complex is a procedure that consists of removing successively pairs of generators from the bases of its chain groups while preserving the homology of the original complex. This method is originally motivated by a simple geometric idea known as the collapsing. At the algebraic level, each removal of a pair of generators that form a reduction pair is equivalent to a collection of projection maps $\{\pi_d : C_d \to C_d\}_d$ that send each generator of the removed pair into 0. Moreover, it projects the other cells into the remaining generators taking into account the modifications of their boundaries caused by the removal of the pair. Let $C'_d = \pi_d(C_d)$ for each *d*. It is shown in [5] that C' is a chain complex and $H_*(C') \cong H_*(C)$, that is, the homologies of C' and *C* coincide. To calculate the homology of the original chain complex *C*, the idea is to define a sequence of projections associated to the removal of pairs in all dimensions and then compute the homology of the resulting chain complex that is the image of the successive projections. A sequence of projections is complete if no other projection can be added to the sequence. Such a sequence always exits when the chain complex is finitely generated. Let (C, ∂) be the original chain complex and (C^f, ∂^f) is the final chain complex obtained after a complete sequence of projections. We already know that the two complexes have the same homology, moreover it is easily observed that if $\partial^f = 0$ then $H_*(C) \cong H_*(C^f) \cong C^f$, that is, the Betti number β_d is given by the number of *d*-cells remaining in the complex C^f.

Formally, a reduction pair and its associated collection of projection maps are defined as follows.

Definition 1. *Let* $\mathcal{C} = \{C_d, \partial_d\}$ *be an abstract chain complex. A pair of generators* (a, B) *such that* $a \in C_{m-1}$, $B \in C_m$ *and* $\langle \partial B, a \rangle = \pm 1$ *is called a reduction pair. It induces a collection of group homomorphisms*

$$\pi_d c := \begin{cases} c - \dfrac{\langle c, a \rangle}{\langle \partial B, a \rangle} \partial B & \text{if } d = m-1, \\ c - \dfrac{\langle \partial c, a \rangle}{\langle \partial B, a \rangle} B & \text{if } d = m, \\ c & \text{otherwise,} \end{cases}$$

where $c \in C_d$ and $\langle \cdot, \cdot \rangle$ is the canonical bilinear form on chains.

A way to understand the global reduction of a complex is to reformulate it as an algorithm and follow an example of its execution. The following implementation is basic and simply aims at understanding the process. For instance, if the chain complex has torsion or if the dimensions are reduced in arbitrary order, then we should refer to the more general version of the *reduction* method described in [5] [11]. Also, we use the data structures as defined in section 4.0.

Algorithm 2. *Reduction* (***Complex*** *K*) *reduces K into* K' *using the one-step reduction formula. Consecutive calls of this function will simplify K until it remains unchanged. Finally, if all the boundaries are null, then the homology generators are given by the remaining cells. If not, then we continue the computation of homology by using a classical method such as Smith Normal Form.*

1) (*Find a reduction pair*) (a, B) is a reduction pair if *a* is a face of *B* with an incidence coefficient of ± 1. If no reduction pair is found, return K unmodified. Otherwise continue. It is assumed that reduction pairs are chosen by decreasing order of dimension. First the highest dimension is reduced, then the lower dimensions.

2) (*Projection of K*) Assign $\partial B \leftarrow B$. **boundary**, $cob\, a \leftarrow a$. **coboundary** and $\lambda \leftarrow \langle \partial B, a \rangle$, the coefficient of *a* in ∂B.

(a) (*Update cofaces of a*) For all cofaces *C* of *a*, add $-\frac{\gamma}{\lambda}\partial B$ to *C*. **boundary**, where γ is $\langle \partial C, a \rangle$, the coefficient of *a* in *C*. **boundary**.

(b) (*Update faces of B*) For all faces *c* of *B*, add $-\frac{\gamma}{\lambda} cob\, a$ to *c*. **coboundary**, where γ is $\langle \partial B, c \rangle$, the coefficient of *c* in *B*. **boundary**.

(c) (*Project a and B*) Remove cells *a* and *B* from *K*.

3) (*Return the result*) Return the simplified *K*.

An example of using the *reduction* algorithm is illustrated in **Figure 1**. With the prescribed orientation of the

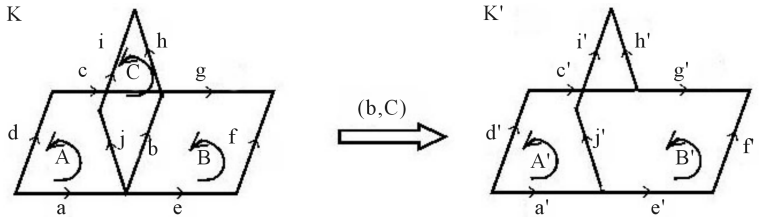


Figure 1. Result of the projection of a complex after one reduction.

given complex, the boundaries of the cells in K are

$$\partial A = a + b - c - d$$

$$\partial B = -b + e + f - g$$

$$\partial C = b + h - i - j.$$

The pair (b, C) is a reduction pair since the incidence number $\langle \partial C, b \rangle = 1$. It follows that

$$A' = \pi_2 A = A - \frac{\langle \partial A, b \rangle}{\langle \partial C, b \rangle} C = A - \frac{1}{1} C = A - C$$

$$B' = \pi_2 B = B - \frac{\langle \partial B, b \rangle}{\langle \partial C, b \rangle} C = B - \frac{-1}{1} C = B + C$$

$$C' = \pi_2 C = C - \frac{\langle \partial C, b \rangle}{\langle \partial C, b \rangle} C = C - \frac{1}{1} C = 0.$$

Clearly, π_0 is the identity map since there is no 0-cell removed from the complex and $\pi_1 \alpha = \alpha$ for all α except for $\pi_1 b = 0$. The altered boundaries in the resulting complex K' are

$$\partial A' = \partial (A - C) = a - c - d - h + i + j$$

$$\partial B' = \partial (B + C) = e + f - g + h - i - j.$$

To calculate the homology of the complex we continue the simplifications until $\partial = 0$ for all dimensions.

3. Computation of Homology by Grouping Reductions

The *reduction* algorithm chooses reduction pairs in arbitrary order and spends most of its time updating the boundaries at each reduction step. Instead of performing the reductions in arbitrary order, we organize them into reduction sequences that make up *reduction directed acyclic graphs* (RDAGs). We will show that this is very advantageous algorithmically.

3.1. Forming RDAGs

Starting from an arbitrary cell, we form the maximal possible directed acyclic graph (DAG) whose nodes are reduction pairs and a directed edge between two pairs (a, B) and (c, D) means that (a, B) is adjacent to (c, D) (see definition 4). A directed path in the DAG corresponds to a reduction sequence. That type of reduction sequence affects only adjacent cells to the path and leaves other cells unchanged. Information about each reduction is carried out by the reduced cells. After performing the whole set of reductions, we obtain the reduced complex by extracting the boundary information from the data structures associated to the cells visited by the reduction DAGs.

We introduce the main concepts through an example. The example details how the projections of cells are calculated given a reduction sequence. This will show what information is exactly needed to be kept about the original complex in order to be able to rebuild the reduced complex.

Example 3. *Consider the complex given in* **Figure 2**. *The boundaries of the* 2*-cell are*

$$\partial A = -a + b - h$$

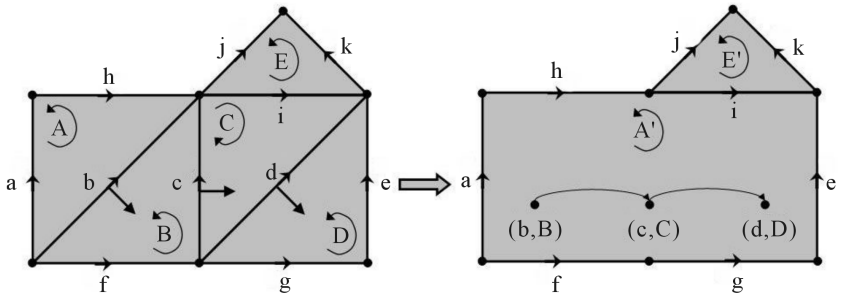


Figure 2. Example of collapsing by using RDAGs.

$$\partial B = -b + c + f$$

$$\partial C = c - d + i$$

$$\partial D = -d + e + g$$

$$\partial E = i - j + k.$$

The depicted sequence of reduction pairs originating at cell A is $(b,B),(c,C)$ and (d,D). The projection maps at the level of the 2-cell give the following

$$(b,B):\quad \pi_2^b \alpha = \alpha - \frac{\langle \partial\alpha, b\rangle}{\langle \partial B, b\rangle} B$$

$$(c,C):\quad \pi_2^c \alpha = \alpha - \frac{\langle \partial\alpha, c\rangle}{\langle \partial C, c\rangle} C$$

$$(d,D):\quad \pi_2^d \alpha = \alpha - \frac{\langle \partial\alpha, d\rangle}{\langle \partial D, d\rangle} D$$

It follows that the projection map associated to the whole sequence is $\pi_2 = \pi_2^d \circ \pi_2^c \circ \pi_2^b$. The 2-cell A is the only cell adjacent to the sequence while the 2-cell E is not adjacent. Their respective projections with respect to the sequence are

$$\begin{aligned}
\pi_2 A = A' &= \pi_2^d \circ \pi_2^c \left(A - \frac{\langle \partial A, b\rangle}{\langle \partial B, b\rangle} B \right) = \pi_2^d \circ \pi_2^c \left(A - \lambda_b B \right) \\
&= \pi_2^d \left((A - \lambda_b B) - \frac{\langle \partial (A - \lambda_b B), c\rangle}{\langle \partial C, c\rangle} C \right) \\
&= \pi_2^d \left((A - \lambda_b B) - \frac{\langle \partial A - \lambda_b \partial B, c\rangle}{\langle \partial C, c\rangle} C \right) \\
&= \pi_2^d \left((A - \lambda_b B) - \frac{-\lambda_b \langle \partial B, c\rangle}{\langle \partial C, c\rangle} C \right) \\
&= \pi_2^d \left(A - \lambda_b B + \lambda_b \lambda_c C \right) \\
&= A - \lambda_b B + \lambda_b \lambda_c C - \frac{\langle \partial (A - \lambda_b B + \lambda_b \lambda_c C), d\rangle D}{\langle D, d\rangle} \\
&= (A - \lambda_b B + \lambda_b \lambda_c C) - \frac{\lambda_b \lambda_c \langle \partial C, d\rangle}{\langle D, d\rangle} D \\
&= A - \lambda_b B + \lambda_b \lambda_c C - \lambda_b \lambda_c \lambda_d D.
\end{aligned}$$

$$\pi_2 E = E' = \pi_2^d \circ \pi_2^c \left(E - \frac{\langle \partial E, b \rangle}{\langle \partial B, b \rangle} B \right)$$
$$= \pi_2^d \circ \pi_2^c \left(E - 0 \cdot B \right) = \cdots = E.$$

Each coefficient in the projection is called the coefficient of contribution of the corresponding cell to the projection of *A*. For instance, $-\lambda_b \lambda_c \lambda_d$ is the coefficient of contribution of *D* in the projection of *A*. We can compute the coefficients λ_b, λ_c and λ_d and express concretely the projection of *A*:

$$\lambda_b = \frac{\langle \partial A, b \rangle}{\langle \partial B, b \rangle} = \frac{1}{-1} = -1$$

$$\lambda_c = \frac{\langle \partial B, c \rangle}{\langle \partial C, c \rangle} = \frac{1}{1} = 1$$

$$\lambda_d = \frac{\langle \partial C, d \rangle}{\langle \partial D, d \rangle} = \frac{-1}{-1} = 1$$

$$A' = A + B - C + D$$

Once a reduction DAG is built, we are interested in building the reduced complex. The cells of the new complex are the projected cells of the original complex. The projection can be used effectively to update or build the boundary operator for each projected cell. In the previous example, the boundary of A' can be reconstructed from the formula of its projection

$$\partial A' = \partial A + \partial B - \partial C + \partial D = f + g + e - a - i - h.$$

The incidence number of a given cell, say *g*, with respect to the boundary of the projection of *A* is computed by adding all the contribution coefficients of the cofaces of *g* present in the projection of *A*, each multiplied by the incidence number of *g* with respect to that coface. A particular coface of *g* can be present several times in the projection of A if it belongs to different sequences that originate at A.

For instance $\langle \partial A', g \rangle$ is given by

$$\langle \partial A', g \rangle = \langle \partial \left(A - \lambda_b B + \lambda_b \lambda_c C - \lambda_b \lambda_c \lambda_d D \right), g \rangle = -\lambda_b \lambda_c \lambda_d \langle \partial D, g \rangle$$

since *D* is the only coface of *g* in the original complex and it is present in the projection of *A*. Note that the value of $\langle \partial D, g \rangle$ is known from the original complex.

Thus to calculate the reduced complex, it is enough to keep track of the coefficients of contribution of every reduced cell in any of the projected cells for which it contributes. In general, after a sequence of reductions is achieved, a projected cell adjacent to reduction DAGs will look as depicted in **Figure 3**.

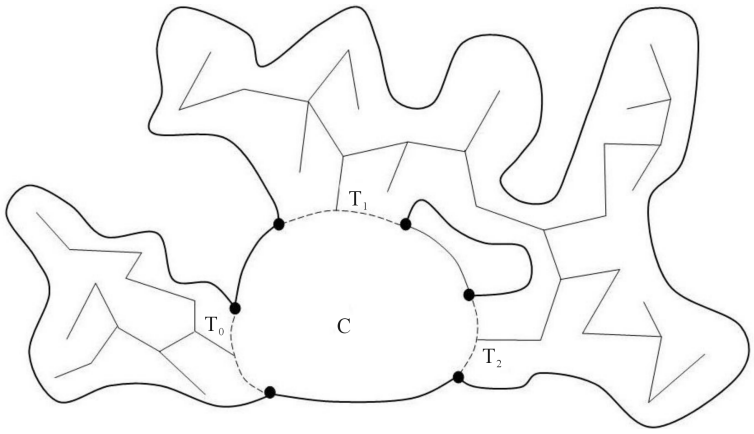


Figure 3. A cell extended by reduction DAGs.

3.2. Projection Formulas for Grouped Reductions

We define how to organize the reductions and the complex using reduction DAGs.

Definition 4. *A cell* A_1 *is adjacent to the pair* (a_2, A_2) *if the cells* A_1 *and* A_2 *are of the same dimension m and* a_2 *is a cell with dimension* $m-1$ *in the boundaries of both* A_1 *and* A_2. *A pair* (a_1, A_1) *is adjacent to a pair* (a_2, A_2) *if* A_1 *is adjacent to* (a_2, A_2).

Definition 5. *A reduction DAG is a directed acyclic graph whose nodes are reduction pairs and a directed edge between two pairs* (a_1, A_1) *and* (a_2, A_2) *means that* (a_1, A_1) *is adjacent to* (a_2, A_2).

Definition 6. *A path P from* (a_1, A_1) *to* (a_n, A_n) *in a reduction DAG G is a reduction sequence* $(a_1, A_1), \cdots, (a_n, A_n)$ *whose elements are nodes in G and* (a_i, A_i) *is adjacent to* (a_{i+1}, A_{i+1}), *for* $i = 1, \cdots, n-1$.

A cell *A* is said to be *adjacent* to a reduction sequence if it is adjacent to some pair of the sequence.

Definition 7. *A path from* (a_1, A_1) *to* (a_n, A_n) *in a reduction DAG G is said to originate at* A_0 *if* A_0 *is adjacent to* (a_1, A_1).

Definition 8. *A cell not appearing in any reduction DAG is called a projected cell. The cells that appear in a reduction DAG are called reduced cells.*

All individual paths originating at a given cell contribute to the projection of the cell. In the following theorem, we give a formula to calculate the projection of a cell by considering the contribution of a single path and ignoring the input from other paths and sub-paths. This formula is called "path projection".

Theorem 9. *Let* $P : (a_1, A_1), \cdots, (a_n, A_n)$ *be a path originating at* A_0. *The path projection of* A_0 *by the path P is given by*

$$\pi_P(A_0) = A_0 + \sum_{i=1}^{n} \Gamma_i A_i,$$

where $\Gamma_i = (-1)^i \prod_{j=1}^{i} \lambda_j$, and $\lambda_j = \dfrac{\langle \partial A_{j-1}, a_j \rangle}{\langle \partial A_j, a_j \rangle}$ for $1 \le i, j \le n$. Γ_i is called the *coefficient of contribution* of A_i in the path projection of A_0.

Proof: We proceed by induction on the length of the path. For $n = 0$ or $n = 1$, the path projection formula is easily checked as shown in Example 3. Suppose that the path projection of A_0 by the path of length *n*, $P_n : (a_1, A_1), \cdots, (a_n, A_n)$ is

$$\pi_{P_n}(A_0) = A_0 + \sum_{i=1}^{n} \Gamma_i A_i.$$

Adding a $(n+1)^{\text{th}}$ pair of reduction (a_{n+1}, A_{n+1}) to the path P_n so that a_{n+1} is a face of A_n leads to the path of length $(n+1)$, $P_{n+1} : (a_1, A_1), \cdots, (a_{n+1}, A_{n+1})$. The projection of A_0 by P_{n+1} is equivalent to the projection of $\pi_{P_n}(A_0)$ by the path $P_1 : (a_{n+1}, A_{n+1})$ consisting of a single reduction pair, that is:

$$\begin{aligned}
\pi_{P_{n+1}}(A_0) &= \pi_{P_1}\left(\pi_{P_n}(A_0)\right) \\
&= \pi_{P_1}\left(A_0 + \sum_{i=1}^{n} \Gamma_i A_i\right) \\
&= \pi_{P_n}(A_0) - \frac{\langle \partial \pi_{P_n}(A_0), a_{n+1} \rangle}{\langle \partial A_{n+1}, a_{n+1} \rangle} A_{n+1} \\
&= \pi_{P_n}(A_0) - \Gamma_n \frac{\langle \partial A_n, a_{n+1} \rangle}{\langle \partial A_{n_1}, a_{n+1} \rangle} A_{n+1}.
\end{aligned}$$

The last equality is due to the fact that none of the cells $A_1, A_2, \cdots, A_{n-1}$ contains a_{n+1} as a face. Thus,

$$\pi_{P_{n+1}}(A_0) = \pi_{P_n}(A_0) - \Gamma_n \lambda_{n+1} A_{n+1} = \pi_{P_n}(A_0) + (-1)^{n+1} \left(\prod_{j=1}^{n+1} \lambda_j \right) A_{n+1}.$$

Finally:

$$\pi_{P_{n+1}}(A_0) = A_0 + \sum_{i=1}^{n+1} \Gamma_i A_i,$$

where $\Gamma_i = (-1)^i \prod_{j=1}^{i} \lambda_j$ and $\lambda_j = \dfrac{\langle \partial A_{j-1}, a_j \rangle}{\langle \partial A_j, a_j \rangle}$, which proves the induction hypothesis.

Now, we can write

$$\pi_{P_n}(A_0) = A_0 + \underbrace{\sum_{i=1}^{n} \Gamma_i A_i}_{\Psi_{P_n}}.$$

We denote by Ψ_{P_n} the *projection chain* of the cell A_0 by the path P_n. □

Note that when the path P_n is extended by one reduction pair (a_{n+1}, A_{n+1}), then the formula for the projection of A_0 by P_{n+1} is given by:

$$\pi_{P_{n+1}}(A_0) = \pi_{P_n}(A_0) + \Gamma_n\left((-1)\lambda_{n+1} A_{n+1}\right).$$

More generally, if $P:(a_1, A_1),\cdots,(a_n, A_n)$ is extended by a path $Q:(b_1, B_1),\cdots,(b_m, B_m)$, then the formula generalizes to

$$\pi_{PQ}(A_0) = \pi_P(A_0) + \Gamma_n\left(\sum_{j=1}^{m} \alpha_j B_j\right) \tag{1}$$

where $\alpha_j = (-1)^j \prod_{k=1}^{j} \mu_k$ and $\mu_k = \dfrac{\langle \partial B_{k-1}, b_k \rangle}{\langle \partial B_k, b_k \rangle}$. Here $B_0 = A_n$.

Corollary 1. *Let $P_1, P_2, \cdots, P_k$ be disjoint non overlapping paths originating at A_0. The total projection of A_0 by $P_1, \cdots, P_k$ denoted by*

$$\pi_{P_1, P_2, \cdots, P_k}(A_0) = A_0 + \sum_{i=1}^{k} \Psi_{P_i}$$

where Ψ_{P_i} is the *projection chain* of the cell A_0 by the path P_i.

Proof: It is important to mention that *k* cannot exceed the number of boundary faces of A_0 which are of dimension $dim(A_0) - 1$. Since the *k* paths are disjoint and non overlapping, each path has to start from a different face of A_0. Without loss of generality, it is sufficient to prove the corollary for the case where $k = 2$ and with paths consisting of single reduction pairs as shown in **Figure 4**. If we apply first the reduction by (a_1, A_1), then

$$\pi_{P_1}(A_0) = A_0 - \lambda_1 A_1, \quad \text{where } \lambda_1 = \left(\frac{\langle \partial A_0, a_1 \rangle}{\langle \partial A_1, a_1 \rangle}\right)$$

Applying the second reduction results in:

$$\pi_{P_2}\left(\pi_{P_1}(A_0)\right) = \pi_{P_1}(A_0) - \frac{\langle \partial(A_0 - \lambda_1 A_1), a_2 \rangle}{\langle \partial A_2, a_2 \rangle} A_2$$

Since P_1 and P_2 are disjoint non overlapping, it follows that a_2 is not a face of A_1, thus

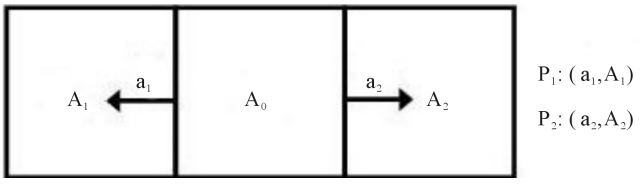


Figure 4. Reduction by two disjoint non overlapping paths.

$$\pi_{P_1P_2}(A_0) = A_0 - \lambda_1 A_1 - \lambda_2 A_2, \quad \left(\lambda_2 = \frac{\langle \partial A_0, a_2 \rangle}{\langle \partial A_2, a_2 \rangle}\right)$$
$$= A_0 + \Psi_{P_1}(A_0) + \Psi_{P_2}(A_0)$$

Corollary 2. *Let T be a reduction tree originating at* A_0, *then the projection of* A_0 *by T is equal to the sum of* A_0 *and the projection chain of* A_0 *by T.*

Proof:

Typically, the trees originating at A_0 can occur as pure paths, in which case the associated projections are given previously. Otherwise, we can find paths that share a common ancestral branch that originates at A_0. This is seen as a bifurcation as shown in **Figure 5**. In this case, the ancestral branch is a path $B:(b_1, B_1),\cdots,(b_{n_B}, B_{n_B})$ which is extended by two paths $C:(c_1, C_1),\cdots,(c_{n_C}, C_{n_C})$ and $D:(d_1, D_1),\cdots,(d_{n_D}, D_{n_D})$. The combination of the results in Theorem 9 and Corollary 1 can be used to show that the total projection of A_0 by $T=(BC),(BD)$ is given by

$$\pi_T(A_0) = A_0 + \sum_{i=1}^{n_B} \Gamma_i B_i + \Gamma_{n_B}\left(\sum_{j=1}^{n_C} \alpha_j C_j + \sum_{k=1}^{n_D} \beta_k D_k\right)$$

The term $\sum_{i=1}^{n_B} \Gamma_i B_i + \Gamma_{n_B}\left(\sum_{j=1}^{n_C} \alpha_j C_j + \sum_{k=1}^{n_D} \beta_k D_k\right) = \pi_T(A_0) - A_0 = \Psi_T(A_0)$ is called the *projection chain of the tree*.

In general, the tree can have many bifurcations at a given node and the bifurcations can occur at different nodes for which cases, the formula given above can be easily generalized. □

One may wonder what happens when two paths are merging into a single path as illustrated in **Figure 6**. For this purpose, let's consider the simple situation where two distinct reduction paths originating at A_0 overlap at a certain node, which is they are extended both by the same path.

In that situation, there are two independent pure paths *CB* and *DB* and the total projection of A_0 is given as follows

$$A_0 + \sum_{i=1}^{n_C} \alpha_i C_i + \Gamma_{n_C} \sum_{j=1}^{n_B} \lambda_j B_j + \sum_{k=1}^{n_D} \beta_k D_k + \Gamma_{n_D} \sum_{j=1}^{n_B} \lambda_j B_j$$
$$= A_0 + \sum_{i=1}^{n_C} \alpha_i C_i + \sum_{k=1}^{n_D} \beta_k D_k + \left(\Gamma_{n_C} + \Gamma_{n_D}\right) \sum_{j=1}^{n_B} \lambda_j B_j.$$

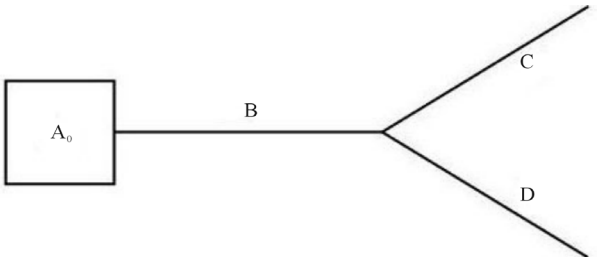


Figure 5. A path *B* bifurcates into two paths *C* and *D*.

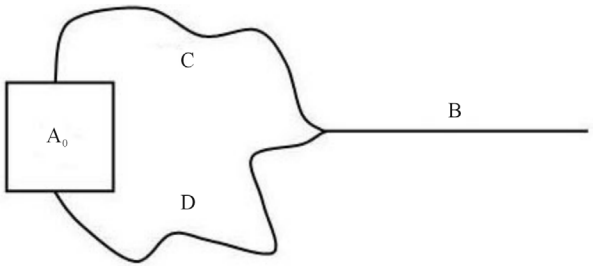


Figure 6. Two paths *C* and *D* merge into a single path *B*.

We see from the formula that it comes down to considering each path (which is also a tree) including the shared sibling branch as pure paths contributing to the projection of A_0. This comes down to considering each tree independently (see **Figure 7**).

Theorem 10. *Let A_0 be a cell adjacent to a RDAG in a chain complex C. The projection of A_0 by the RDAG is equal to A_0 to which we add the projection chains of A_0 by all the trees in the RDAG that originate at A_0, that is*

$$\pi_{RDAG}(A_0) = A_0 + \sum_{T \in \mathcal{T}_{A_0}} \Psi_T(A_0)$$

where $\mathcal{T}_{A_0}$ is the collection of all the reduction trees in the RDAG originating at A_0.

Proof: Since we deal with finite complexes the number of all possible trees starting at a given cell (here A_0) has to be finite. Thus, the results proved for paths in corollaries 1 and 2 and the subsequent remark regarding merging paths can be extended for the case of trees (splitting and merging) to find the formula for the projection of A_0. □

We review the projection formulas of paths and trees with the example of **Figure 8**. There are two trees T_B and T_C originating at A_0. The projection of A_0 with respect to the tree T_B can be written as

$$\lambda_1 B_1 + \lambda_2 B_2 + \lambda_3 D_1 + \lambda_4 D_2 + \lambda_4 \pi_1,$$

where $\pi_1 = \alpha_1 E_1 + \alpha_2 E_2 + \beta_1 F_1 + \beta_2 F_2$. The projection chain of A_0 with respect to the tree T_C is given by

$$\varphi_1 C_1 + \varphi_2 C_2 + \varphi_3 D_1 + \varphi_4 D_2 + \varphi_4 \pi_2,$$

with $\pi_2 = \mu_1 E_1 + \mu_2 E_2 + \sigma_1 F_1 + \sigma_2 F_2$. It follows that the total projection of A_0 is given as the sum of both plus A_0.

3.3. Why Acyclic Graphs

Adjacency alone does not guarantee that cycles are not created in a reduction DAG. To ensure that every reduction sequence is associated with a well defined projection, we enforce acyclicity in the formed directed graphs. More generally, assuming that a reduction sequence $(a_1, A_1), \cdots, (a_n, A_n)$ is such that (a_0, A_0) is adjacent to (a_1, A_1) and (a_n, A_n) is adjacent to (a_0, A_0) (see **Figure 9**), we consider extending the sequence to $(a_1, A_1), \cdots, (a_{n+1}, A_{n+1})$ by setting $(a_{n+1}, A_{n+1}) = (a_0, A_0)$. However, the pair (a_0, A_0) is not necessarily admissible since after performing the sequence of reductions $(a_1, A_1), \cdots, (a_n, A_n)$, the cell A_0 has been modified into A_0'

$$A_0' = A_0 + \sum_{i=1}^{n} (-1)^i \left(\prod_{j=1}^{i} \lambda_j \right) A_i$$

and $a_0 = a_{n+1}$ occurs in the boundary of A_0' as a face of both A_0 and A_n. Its incidence number is therefore calculated as follows

$$\langle \partial A_0', a_0 \rangle = \langle \partial A_0, a_0 \rangle + (-1)^n \left(\prod_{j=1}^{n} \lambda_j \right) \langle \partial A_n, a_0 \rangle$$

which is not necessarily invertible. In **Figure 9**, $\langle \partial A_0', a_0 \rangle = (-1) + (-1)^3 \times (1 \times 1 \times (-1)) \times 1 = 0$ is not invertible which means that (a_0, A_0) is not an admissible reduction pair. In our implementation, we avoid testing if

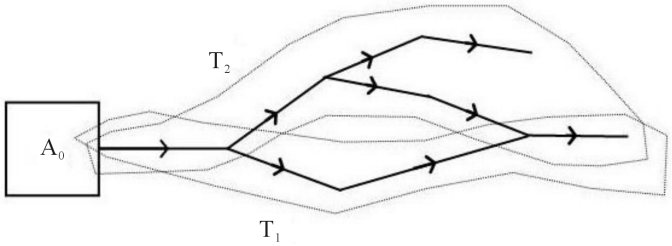


Figure 7. Decomposition of a reduction DAG into two independent trees T_1 and T_2.

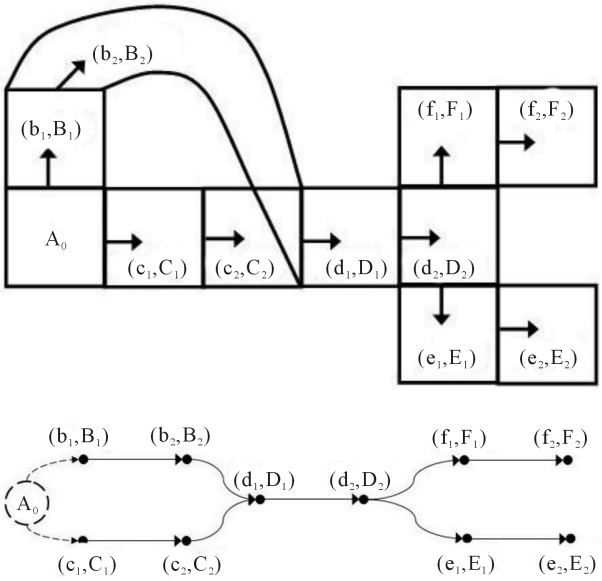


Figure 8. Example of reduction paths and the associated reduction DAG.

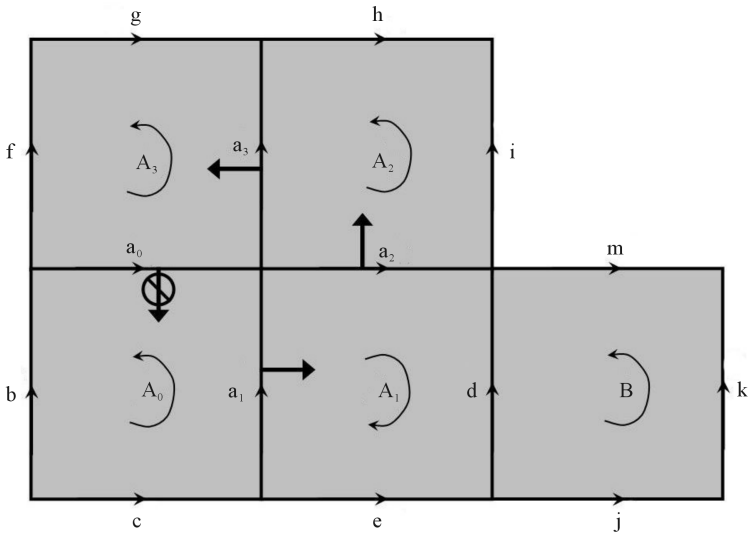


Figure 9. Cycles need to be avoided as reduction pairs are added in the tree.

reduction pairs are admissible or not by avoiding cycles as we build the reduction DAGs. This results in a more efficient algorithm.

3.4. Using Projection Formulas

We recapitulate the formulas with a second example, see **Figure 10**. The first reduction pair is (a, A). Because the cell E is adjacent to the pair (a, A), it is marked as a *projected* cell. In a list associated with the *reduced* cell A, we save the pair $(-\lambda_a, E)$ where $\lambda_a = \dfrac{\langle \partial E, a \rangle}{\langle \partial A, a \rangle} = -1$. The reduction is continued from the faces of A. The second reduction pair is (b, B). We carry the information previously saved on the visited cofaces of b, so we add $(\lambda_b \lambda_a, E)$ $\left(\lambda_b = \dfrac{\langle \partial A, b \rangle}{\langle \partial B, b \rangle} = 1\right)$ to the list associated with the cell B. Continuing from the faces of B, the

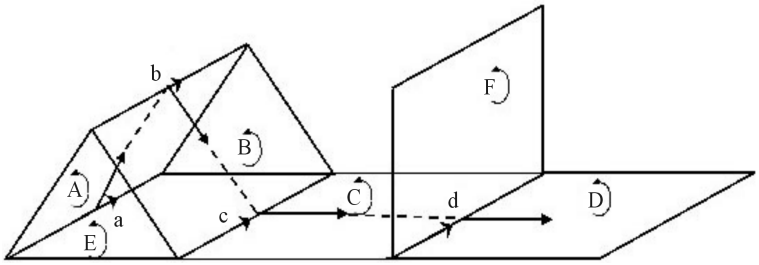


Figure 10. A more complicated reduction sequence.

third reduction pair to be added is (c,C). The cell c is adjacent to both the *reduced* cell B and the *projected* cell E. From this point, we continue a path $((a,A),(b,B),(c,C),\cdots)$ and begin a sub-path $((c,C),\cdots)$. The information saved with C considers the input of both paths. So we save $(-(\lambda_{cE}+\lambda_{cB}\lambda_b\lambda_a),E)$ where $\lambda_{cE}=\frac{\langle \partial E,c\rangle}{\langle \partial C,c\rangle}=-1$ and $\lambda_{cB}=\frac{\langle \partial B,c\rangle}{\langle \partial C,c\rangle}=-1$. But because the coefficients of contribution sum up to 0, it means that C does not contribute to the projection of E. Thus, nothing is saved with C. The last reduction pair (d,D) is added to the sequence. Because, d is adjacent to the unvisited cell F, F is marked as a *projected* cell. The *reduced* cell C is also adjacent to d but no information was saved with C, so the only information to save with D is $(-\lambda_d,F)$ where $\lambda_d=\frac{\langle \partial F,d\rangle}{\langle \partial D,d\rangle}=-1$.

3.5. Building the Simplified Complex

Using reduction DAGs to compute the homology of a chain complex is a recursive process. At each recursion level, the algorithm simplifies the complex by constructing reduction DAGs on the complex and saving the associated projections into appropriate data structures. This is performed simultaneously for each dimension. This process eventually stops when it is impossible to add another reduction pair to any reduction DAG. In that case, the algorithm will build the associated simplified complex and continue the reduction process on the simplified complex.

The simplified complex is rebuilt from the projected cells only (reduced cells are not considered). The boundaries of the cells are updated using global projection formulas that allow to calculate the incidence numbers between cells of contiguous dimensions. Note that the reduced cells are not completely removed from the structures since they may be needed to recover homology generators expressed in terms of cells of the original complex as we explain in subsection 3.8. Contrary to the classical case where the boundary updating is done at each reduction step and may concern cells that can be reduced at a later step, a major benefit of this new approach is that the boundary updating is done only among the projected cells which often constitute a small fraction of the number of cells in the original complex.

Continuing with the example used in **Figure 10**, we proceed to build its simplified complex. At this point from the projection formulas we have the projection of E, which is $E'=E+A+B$ and the projection of F which is $F'=F+D$. To build the simplified complex, we create two 2-dimensional cells, E' and F'. We also create the projected lower dimensional cells (which is not illustrated in the example). Then, knowing that $\partial F'=\partial F+\partial D$, we add the projected 1-cells in the boundary of D to the boundary of F' using the correct coefficients. For example, if we suppose that d_1 is a projected cell of dimension one that was in the boundary of D in the original complex. From the structures, we know that D appears in the projection of F with an incidence coefficient of 1 ($F'=F+D$). So, we add d_1 in the boundary of F' with a coefficient of 1. This is done for all projected cells in all dimensions.

3.6. Performance Benefits of Using Reduction DAGs

We use the complex illustrated in **Figure 9** as a case study. Our objective is to point out how grouping reductions differs from classical methods and leads to substantial performance benefits. Performance is measured by the number of lists updates and compared to the classical *reduction* algorithm (see 2). Usually, the boundary and

coboundary of cells are maintained by lists data structures. Assuming ordered lists, the cost for merging two lists L_1 and L_2 is bounded by $O(|L_1|+|L_2|)$, where L refers to the number of elements in L. Using unordered lists is bounded by $O(|L_1|*|L_2|)$.

1) Classical *Reduction* Method

(a) *Reduction* (a_1, A_1). According to step 2.a) of algorithm 2, we update cofaces of a_1. That is, for each 2-cell β in the coboundary of a_1, we add the set of cells in the boundary of A_1 to the boundary of β using the right coefficient. Therefore, this operation costs an order of $\partial A_1 + \partial A_0 = 4+4=8$ list updates.

In step 2.b), we update the coboundaries of faces of A_1. For each 1-cell α in the boundary of A_1, we add the set of cells in the coboundary of a_1 to the coboundary of α using the right coefficient. This time, the cost is in the order of $(|cobd|+|cobe|+|coba_2|+3\cdot|coba_1|)=11$. Taken all together, it costs an order of $8+11=19$ lists updates for the first reduction.

(b) *Reduction* (a_2, A_2). Now $\partial A_0 = -b+c+d+e-a_2-a_0$, thus updating cofaces of a_2 is in the order of $(|\partial A_0|+|\partial A_2|)=(6+4)=10$.

For step 2.b), updating faces of A_2 is in the order of $(|cobi|+|cobh|+|coba_3|+3\cdot|coba_2|)=10$. Taken together, the cost for the second reduction is in the order of $10+10=20$ lists updates.

(c) *Reduction* (a_3, A_3). Now $\partial A_0 = -b+c+d+e+i-h-a_3-a_0$, and updating cofaces of a_3 costs an order of $\partial A_0 + \partial A_3 = 8+4=12$ lists updates.

For step 2.b), updating faces of A_3 amounts to a cost in the order of $(|cob\,f|+|cobg|+|coba_0|+3\cdot|coba_3|)=10$. The total cost for the third reduction is in the order of $12+10=22$.

Taken all together, the three reductions cost an order of $19+20+22=97$ lists updates. Generally, as the reductions go on, the costs increase because the boundary and coboundary lists tend to grow in size.

2) *Reduction DAGs* Method

(a) *Reduction* (a_1, A_1). The pair (λ_{a_1}, A_0), where $\lambda_{a_1} = -\dfrac{\langle \partial A_0, a_1 \rangle}{\langle \partial A_1, a_1 \rangle}$ is saved in a list maintained by A_1. The cost is one list update.

(b) *Reduction* (a_2, A_2). The pair $(\lambda_{a_2}\lambda_{a_1}, A_0)$, where $\lambda_{a_2} = -\dfrac{\langle \partial A_1, a_2 \rangle}{\langle \partial A_2, a_2 \rangle}$ is saved in a list maintained by A_2. The cost is one list update.

(c) *Reduction* (a_3, A_3). The pair $(\lambda_{a_3}\lambda_{a_2}\lambda_{a_1}, A_0)$ where $\lambda_{a_3} = -\dfrac{\langle \partial A_2, a_3 \rangle}{\langle \partial A_3, a_3 \rangle}$, is saved in a list maintained by a_3. The cost is one list update.

(d) *Reconstruction of the simplified complex*. For each non reduced 1-cell a, we scan through its cofaces of the original complex and link a with the list of projected cells previously saved by using the right coefficients. The total cost is in the order of the number of remaining 1-cells plus the size of the lists of adjacent projected cells. Note that usually, the simplification process continues on the lower dimensions. So at the end there will remain only a fraction of the lower dimensional cells (here 1-cells). The *Reduction DAGs* method is more efficient as long as the cost for reconstructing the simplified complex is low which our experimental results corroborate.

3.7. Minimizing Lists Updates

Different reductions lead to different computational costs (measured by the number of lists updates). We attempt to achieve a good overall performance by choosing reduction pairs that should minimize the number of updates. To select among many candidate reduction pairs, we use a heuristic that estimates their respective costs. Given a reduction pair (a, B), this cost is approximated by the number of non reduced cofaces of a (other than B) plus the size of the projection lists of each reduced coface of a.

We illustrate this on an example in **Figure 11**. Suppose that the 2-cell A is already reduced and has (λ_1, A_1) and (λ_2, A_2) in its list of projected cells. We consider the cost of performing the reduction (c, C). The heuristic approximates this cost to three lists updates, because there is one non reduced coface of c (that is B), and the projection list of the reduced coface A has two elements (λ_1, A_1) and (λ_2, A_2). Indeed, performing the reduction (c, C) amounts to inserting the pairs $(\lambda_3\lambda_1, A_1), (\lambda_3\lambda_2, A_2)$ and (λ_4, B) into the projection list of C. We give more details about the heuristic in section 4.

3.8. Calculating Generators

An interesting aspect of using the reduction DAGs method to compute homology is that the data structures allow to extract the homology generators at a low additional cost. We explain how to proceed. Let us consider the complex of a plane quotiented by its boundary as illustrated in **Figure 12**. This complex is homeomorphic to a 2-sphere and the 2-cell A represents the 2-generator $A' = A + \lambda_1 B + \lambda_2 C + \lambda_3 D$. The generator associated to a projected cell corresponds to the projection of the cell. As we illustrated in **Figure 12**, after each reduction, the projection coefficients λ_i are saved into the projection lists maintained in each reduced cell. Generally, once the complex has been simplified to the level where all boundaries are trivial, one has to examine all projected (non-reduced) cells and find their corresponding coefficients in the projection lists of the reduced cells. Thus, to get the 2-generator associated to the 2-cell A, one has to scan through each reduced 2-cell and extract the projection coefficients associated to A.

However, computing homology by reduction DAGs is a recursive process (see **Figure 13**) and this needs to be considered when calculating the generators. Projections correspond to generators but they can be expressed with cells of the simplified complex at any level of the recursion. However, in order for the generators to carry geometric meaning, it makes sense only to express the generators with cells from the original complex. Due to the recursive simplifications, a projected cell at a previous level of the recursion may become a reduced cell at a later level of the recursion. We illustrate this in **Figure 14**. In this example, there are two levels of recursion.

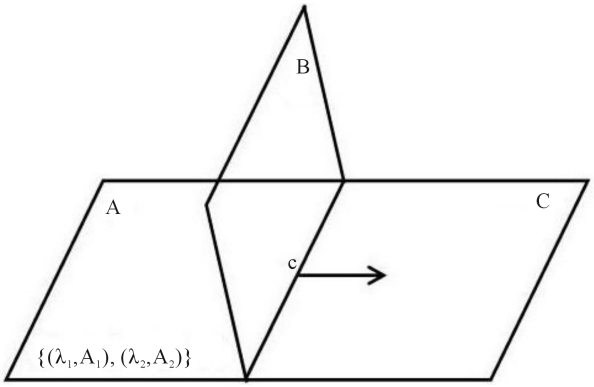


Figure 11. Cost of performing a reduction in terms of list updates.

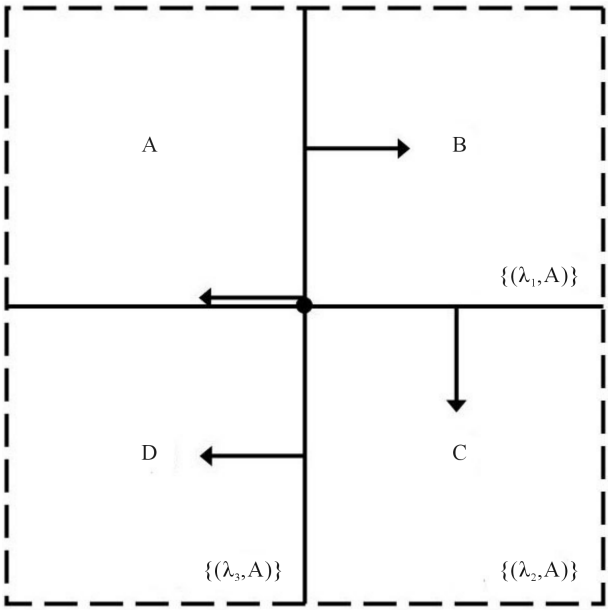


Figure 12. A Complex homeomorphic to a 2-sphere.

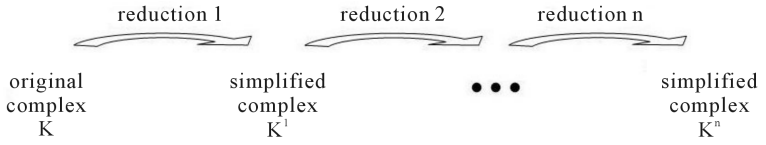


Figure 13. Computing homology by reduction DAGs is a recursive process.

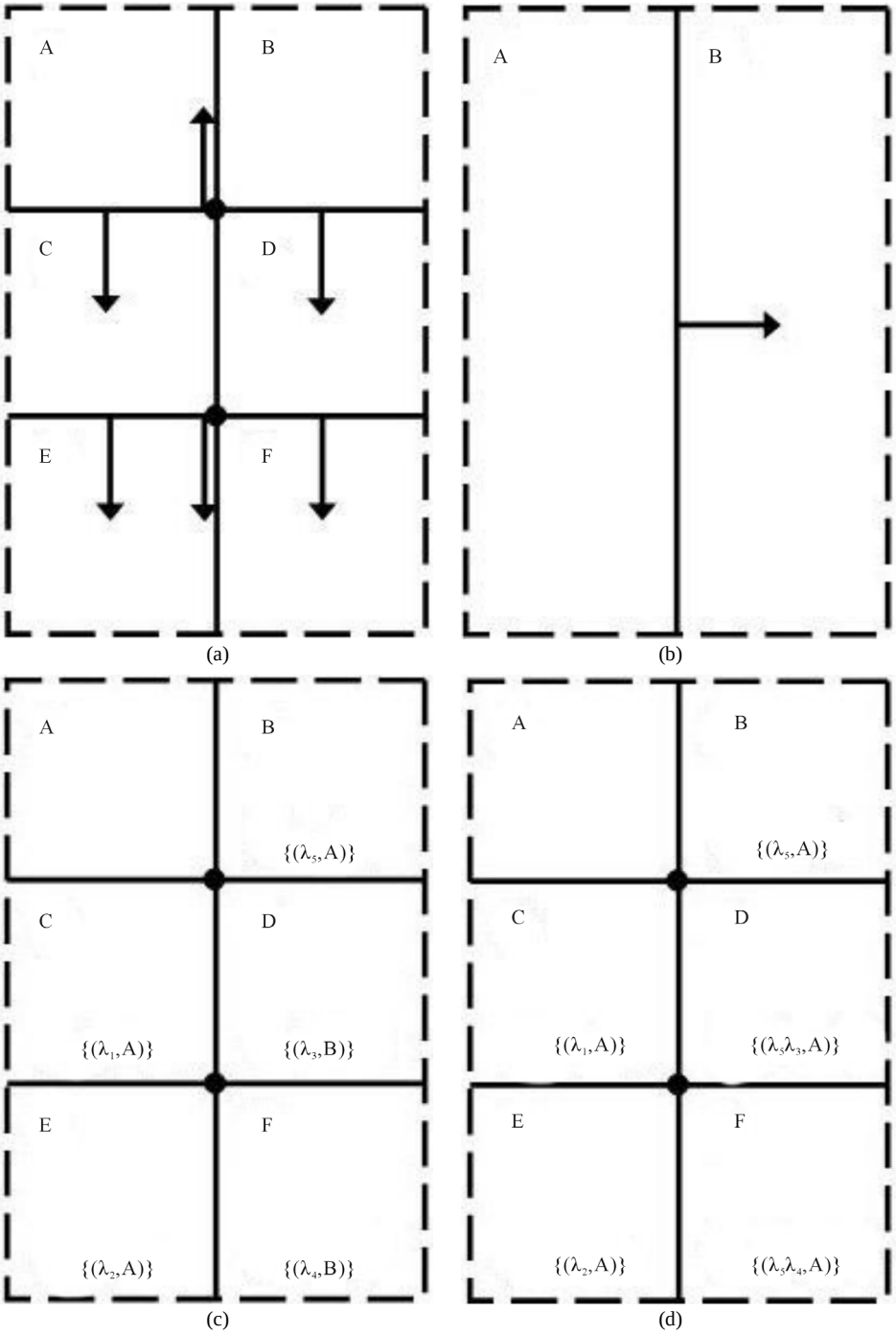


Figure 14. (a) Reduction 1 of the initial complex. (b) Reduction 2 of the simplified complex. (c) Initial complex after returning from the second reduction. (d) Initial complex after returning from the first reduction.

The final simplified complex is obtained after reducing the vertical edge and the 2-cell B shown in **Figure 14(b)**. At this step, the cell A represents a 2-generator that is expressed as $A'' = A' + \lambda_5 B'$ (considering that after the first reduction we rename the original cells A and B as A' and B'). Note that in fact, the algorithm always works on the original cells and never copy nor create any new cell during the whole computation. Returning from the last recursion, we know that $A' = A + \lambda_1 C + \lambda_2 E$ and $B' = B + \lambda_3 D + \lambda_4 F$. The cell A represents a projected cell at both recursion levels (14(b) and 14(c)) and requires no special consideration. On the other hand, the cell B is a projected cell at the first level of recursion but becomes a reduced cell at the second recursion level. Consequently, returning from the second recursion level, we scan through each 2-cell and whenever B is encountered in one of the projection lists, it is replaced by $\lambda_5 A$. This is shown in **Figure 14(d)**. Finally, the generator is expressed by $A'' = A + \lambda_5 B + \lambda_1 C + \lambda_5 \lambda_3 D + \lambda_2 E + \lambda_5 \lambda_4 F$. In **Figure 15** we show different 1-generators that we extracted from 3D models.

3.9. What If Boundaries Are Not Trivial?

In previous subsections, we stressed that all boundaries have to be trivial in order to obtain homology and its generators at the end of the reduction process. But in practice, when all reductions are done we can be in a situation

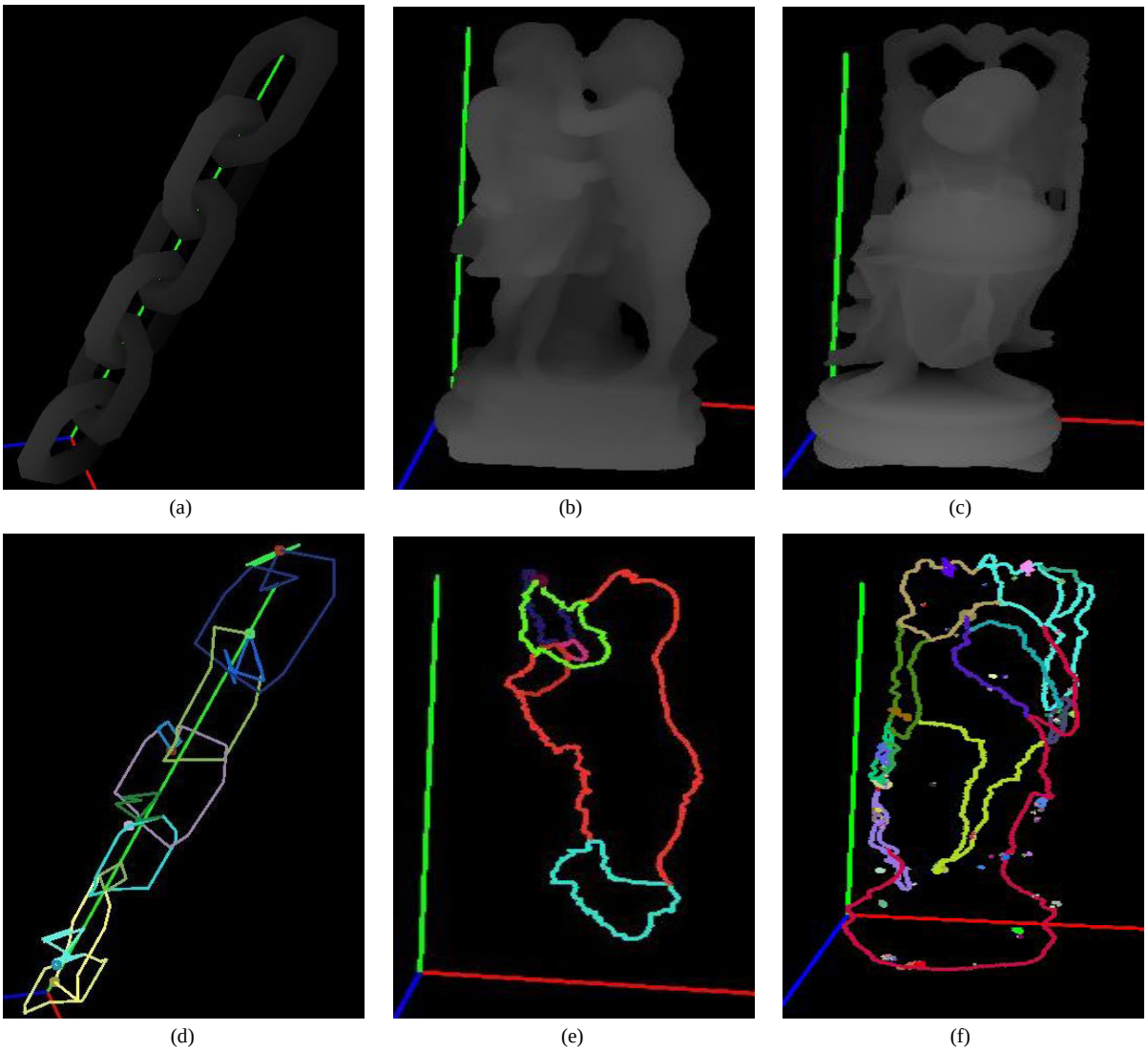


Figure 15. (a)-(c) The 3D models of a chain, two kissing children and a Buddha. (d)-(f) The calculated holes (one dimensional generators).

where some of the boundaries remain non trivial. For example, this may be due to the presence of torsion.

In Definition 1, the pair (a, B) is defined as a reduction pair if $\langle \partial B, a \rangle = \pm 1$. But this condition is in fact too restrictive. In order for π_d to be a group homomorphism, $\lambda = \dfrac{\langle \partial c, a \rangle}{\langle \partial B, a \rangle}$ has to be an integer. This is satisfied when $\langle \partial B, a \rangle$ divides $\langle \partial c, a \rangle$ for every $c \in C_d$. The easiest and most efficient way to guarantee this is to choose B and a such that $\langle \partial B, a \rangle = \pm 1$. Thus, for performance considerations, we designed the reduction DAGs algorithm to consider only reduction pairs with ± 1 coefficients. But the drawback is that once all reductions are done, it is not guaranteed that all boundaries will be trivial. There may remain reduction pairs such that $\langle \partial B, a \rangle \neq \pm 1$ but $\langle \partial B, a \rangle$ divides $\langle \partial c, a \rangle$. In that case, one can modify the present algorithm to test for divisibility and continue the computations with the modified version. Another possibility is to have recourse to a classical algorithm such as the Smith Normal Form. Note that at this level most of the cells generating the complex have already been reduced to a small number.

4. Data Structures and Algorithms

The first data structure represents a chain complex.

ChainComplex:

E: two dimensional array of cells organized by their respective dimension, that is E[*d*][*i*] denotes the *i*-th *d*-cell.

In our implementation, the pointer to a cell is used to identify the cell. These identifiers are saved in **E**. The main benefit to using pointers is that when we build the simplified complex, no new cell is created. Only the pointers of the projected cells are copied into the simplified complex.

Cell:

boundary/coboundary: list of faces/cofaces.

state: a flag taking one of the values in {NORMAL, REDUCED, PROJECTED, VISITED}.

projCells: list to save the PROJECTED cells and their associated coefficients that project onto the given cell when it is REDUCED.

nbUpdates: approximates the number of updates to a projCells list when the given cell is reduced by one of its cofaces.

Initially, all cells are set to the NORMAL state. REDUCED is assigned to the cells in a reduction sequence and PROJECTED is assigned to the cells adjacent to a reduction sequence. The VISITED flag is assigned to the *d*-cells that don't have any $(d+1)$ coface that can form an admissible reduction pair. A reduction pair (a, B) is admissible if $\langle \partial B, a \rangle = \pm 1$ and adding the pair to the reduction DAG does not create a cycle. The flag helps to avoid testing more than once if the given *d*-cells have an admissible reduction pair.

The projCells list is used by the REDUCED cells to keep track of the coefficients of contribution of every PROJECTED cell for which it contributes. For example, if we use the sequence of reduction pairs illustrated in **Figure 2**, then *B*.projCells and *C*.projCells will respectively contain (λ_b, A) and $(\lambda_b \lambda_c, A)$ where $\lambda_b = -\dfrac{\langle \partial A, b \rangle}{\langle \partial B, b \rangle}$ and $\lambda_c = -\dfrac{\langle \partial B, c \rangle}{\langle \partial C, c \rangle}$.

To build a sequence of reduction pairs, nbUpdates is used to select reduction pairs that should minimize the amount of updates to the projCells list. The number of updates is approximated by the number of non reduced cofaces plus the number of entries in the projCells lists of REDUCED cofaces.

We now give the principal steps of the algorithm.

- *HOM_RDAG*(ChainComplex *K*) returns the Betti numbers of a chain complex *K* by using reduction DAGs.

1) (*Initialization*) For all cells in *K*, set its state to NORMAL, empty projCells, set nbUpdates to the number of cofaces.

2) (*Reduce cells*) Proceed by decreasing order of dimension. Order the *d*-cells by increasing value of nbUpdates. Following this order, start a reduction DAG (call *BuildRDAG*()) from each cell whose state is NORMAL.

For dimension 0, change remaining NORMAL 0-cells to PROJECTED.

3) (*Build the simplified complex*) Call *BuildSimplifiedComplex*(*K*). This method extracts the boundary information from the data structures to build a simplified complex K'.

Continue to recursively simplify K' by calling *HOM_RDAG*(K') until there are no reduction pair left.

Test if all boundaries are trivial. If not, then continue the reduction process with another method such as SNF.

4) (*Return the Betti numbers*) Assign the number of PROJECTED *d*-cells to β_d.

- *BuildRDAG*(Cell *c*) builds a reduction DAG from cell *c*.

1) (*Initialization*) Save *c*.nbUpdates into *nbUpdatesLimit* for later use.

2) (*Find a reduction pair*) Find a coface *B* of *c* such that *B* is NORMAL or VISITED. If a coface *B* is found, then call *PairCells*(*c*, *B*), otherwise set *c* to VISITED and exit *BuildRDAG*().

3) (*Expand the reduction DAG*) Expand the reduction DAG (repeat step 2) from all NORMAL faces of *B* that have nbUpdates ≤ *nbUpdatesLimit*. Proceed in a breadth first search approach.

- *PairCells*(Cell *c*, Cell *B*) adds the reduction pair (c, B) to the sequence and updates the data structures accordingly.

1) (*Update projCells*) For all cofaces *C* of *c* different than *B*, if *C* is REDUCED, then add λ_c**C*.projCells to *B*.projCells. Otherwise, set *C* to PROJECTED and add λ_c**C* to *B*.projCells, where $\lambda_c = -\frac{\langle \partial C, c \rangle}{\langle \partial B, c \rangle}$.

2) (*Update the nbUpdates variable*) For all faces *a* of *B* different than *c*, add |*B*.projCells|-1 to *c*.nbUpdates. For all faces of *c*, remove one to nbUpdates.

- *BuildSimplifiedComplex*(Complex *K*) extracts the boundary and coboundary information from the data structures to build a simplified complex K'.

Note: In the preceding methods, the coboundary list of a cell is never modified. Thus, when referring to cofaces, it is about the cofaces in the complex *K* at a given recursion level before any reduction is performed.

1) (*Update boundary*) Proceed by increasing order of dimension. Let *c* be a PROJECTED *d*-cell of *K*. For all cofaces *C* of *c*, iterate through *C*. projCells. Let *B* be a $(d+1)$-cell in *C*. projCells and λ_B its associated coefficient. Add $\lambda_B \langle \partial C, c \rangle c$ to *B*.boundary and update *c*.coboundary accordingly.

Finally, remove all REDUCED cells remaining in the boundary and coboundary of PROJECTED cells. Copy the pointers of all PROJECTED cells into *K*'.*E*.

5. Experimental Results

Because of its heuristic and recursive design, we had major difficulties to analyze the time complexity of our algorithm. Instead, we approached the problem by evaluating its performance on a wide range of experimentations. We compared its performance with the reduction algorithm 2 on different data series: *d*-balls, *d*-spheres, tori, Bing's houses, 3D medical scans and randomly generated *d*-complexes.

The series of *d*-balls and *d*-spheres were created by juxtaposing d dimensional cubes along each dimension. A 3-ball of size 10 is the cube obtained by placing side by side 10 × 10 × 10 3-cubes and making the corresponding identifications of lower dimensional cells. A *d*-sphere is a *d* -ball quotiented by its boundary. We also constructed the tori and Bing's houses from 3-cubes, although they are both 2 dimensional complexes. This construction does not affect the homology. We created two data series from 3D medical scans: brain and heart. Each of them is created from 3D images by selecting the pixels whose values are higher than a threshold. Lastly, we randomly generated *d*-complexes by choosing $(d+1)$ random numbers referring to the number of generators. We created a *d*-cell for each *d*-generator and randomly linked the *d*-cells to the $(d-1)$-cells. By subdivision, we obtained complexes of various data sizes.

In the experimentation, we use the following terminology. A data series refers to one of *d*-balls, *d*-spheres, tori, Bing's houses, 3D medical scans or randomly generated *d*-complexes. Each data series is constituted of many datasets. For example, d-balls contain the 2-ball, 3-ball, ..., 6-ball. Each dataset contains many files of complexes of various data sizes. For example, there are 15 files in the 2-ball data set for which the data size vary from 2000 up to 10,000,000. The size of a complex is measured as the number of cells plus the number of links (entries in boundary lists). A test is an execution of the algorithm on a file of a specified data set.

Initially, we observed that a test could be misleading, especially when dealing with cubical data sets. The reason is because we create the complexes in an orderly fashion. Thus, during its execution, the algorithm is most probably benefiting from a favorable choice of reduction pairs due to the order inherent in the data. Consequently, to avoid misreporting on the algorithm's performance, we took care to randomize the boundary and

coboundary lists of all cells in the complex before each test.

We performed 30 tests for each file and computed the average, maximum and standard deviation of the time used for calculating the homology, the generators and sorting. We measured the sorting time because it has a time complexity of $\Theta(n \cdot \log n)$ and we wanted to verify that sorting would not monopolize the computation time. Also, we saved statistics on the number of recursions to see its influence on the global performance. Here we grouped and summarized the results per dataset.

We begin with our most interesting results (see **Table 1**) which compare the performance of the reduction DAGs algorithm with the classical reduction method on different datasets. We can observe a significant improvement between the two methods. Knowing that the time complexity of the reduction method is quadratic (using ordered lists), those results suggest a subquadratic time complexity for the reduction DAGs algorithm which is what we measured in **Table 2**.

Table 1. Comparison of the reduction DAGs algorithm versus the classical reduction method.

Dataset	Data Size	Times Faster
3-Ball	115,905	195.4
4-Ball	27,841	147.3
2-Sphere	517,145	65.0
3-Sphere	134,777	182.5
Torus	418,608	36.0
Heart	545,492	50.5
3-Complex	265,322	410.0
5-Complex	431,186	458.8
8-Complex	330,285	185.9

Table 2. Approximated performance with respect to dataset, data size and dimension.

		Equation		Time (msec) Vs Data Size			
Dataset	***N***	β	α	10^4	10^5	10^6	10^7
2-Ball	15	9.05E−8	1.11	2.49	32.11	413.66	5329.04
3-Ball	11	4.63E−8	1.15	1.84	26.04	367.77	5194.95
4-Ball	11	4.26E−8	1.14	1.55	21.35	294.72	4068.27
5-Ball	9	5.66E−8	1.12	1.71	22.53	297.04	3915.76
6-Ball	5	3.56E−8	1.15	1.42	20.02	282.78	3994.39
2-Sphere	17	5.77E−8	1.18	3.03	45.83	693.71	10,499.67
3-Sphere	16	2.79E−8	1.22	2.12	35.12	582.91	9673.96
4-Sphere	10	3.38E−8	1.17	1.62	23.93	353.93	5235.00
5-Sphere	9	5.52E−8	1.14	2.00	27.67	381.89	5271.56
6-Sphere	5	3.30E−8	1.17	1.58	23.36	345.55	5111.09
7-Sphere	3	1.31E−8	1.24	1.19	20.76	360.80	6270.05
Torus	5	1.48E−7	1.07	2.82	33.13	389.28	4573.64
Bing's House	5	2.18E−7	1.04	3.15	34.55	378.84	4153.90
Heart	6	6.84E−8	1.13	2.26	30.55	412.15	5559.76
Brain	6	3.13E−8	1.20	1.97	31.30	496.07	7862.20
2-Complex	9	6.78E−8	1.24	6.18	107.46	1867.37	32,451.12
3-Complex	9	7.89E−9	1.36	2.17	49.78	1140.45	26,126.25
4-Complex	9	2.82E−9	1.45	1.78	50.15	1413.35	39,833.56
5-Complex	9	4.95E−9	1.38	1.64	39.32	943.20	22,625.87
6-Complex	9	3.04E−9	1.41	1.33	34.11	876.75	22,535.83
7-Complex	9	2.42E−8	1.23	2.01	34.18	580.52	9858.60
8-Complex	9	2.26E−9	1.43	1.19	31.92	859.23	23126.42

In **Table 2** we report the approximative time that our algorithm uses to calculate homology on various datasets. The second column (*N*) gives the number of files within the dataset. The third and fourth columns show the parameters α and β that best-fit the equation " $\text{time} \approx \beta \cdot \text{data\_size}^{\alpha}$ ", where time is expressed in seconds. We obtained the best-fits from the times measured on the individual files of each dataset. In the last four columns, we present the times in milliseconds that we get from the best-fit equation for different data sizes. Those times approximate the real times that we measured in real experiments. For example, on 30 tests, we measured an average time of 4992.24 ms on a torus of size 10944304. The best-fit equation approximates a time of 4573.64 ms for a torus of size 10000000.

From **Table 2**, we observe that $\alpha \approx 1.16$ and $\beta \approx 5E-8$ for *d*-balls and *d*-spheres without regard of the dimension. Except for few values, α is relatively constant while β gradually decreases as the dimension increases. We explain this by the fact that the ratio of exterior face reductions versus interior face reductions increases with the dimension. An exterior face reduction denotes a reduction pair (a, B) for which *a* has only the coface *B* in its coboundary list. On the other hand, we use the term interior face reduction when *a* is shared by more than one coface. Algorithmically speaking, an exterior face reduction is more advantageous because it does not cost any update to the projection lists, while an interior face reduction costs at least an update.

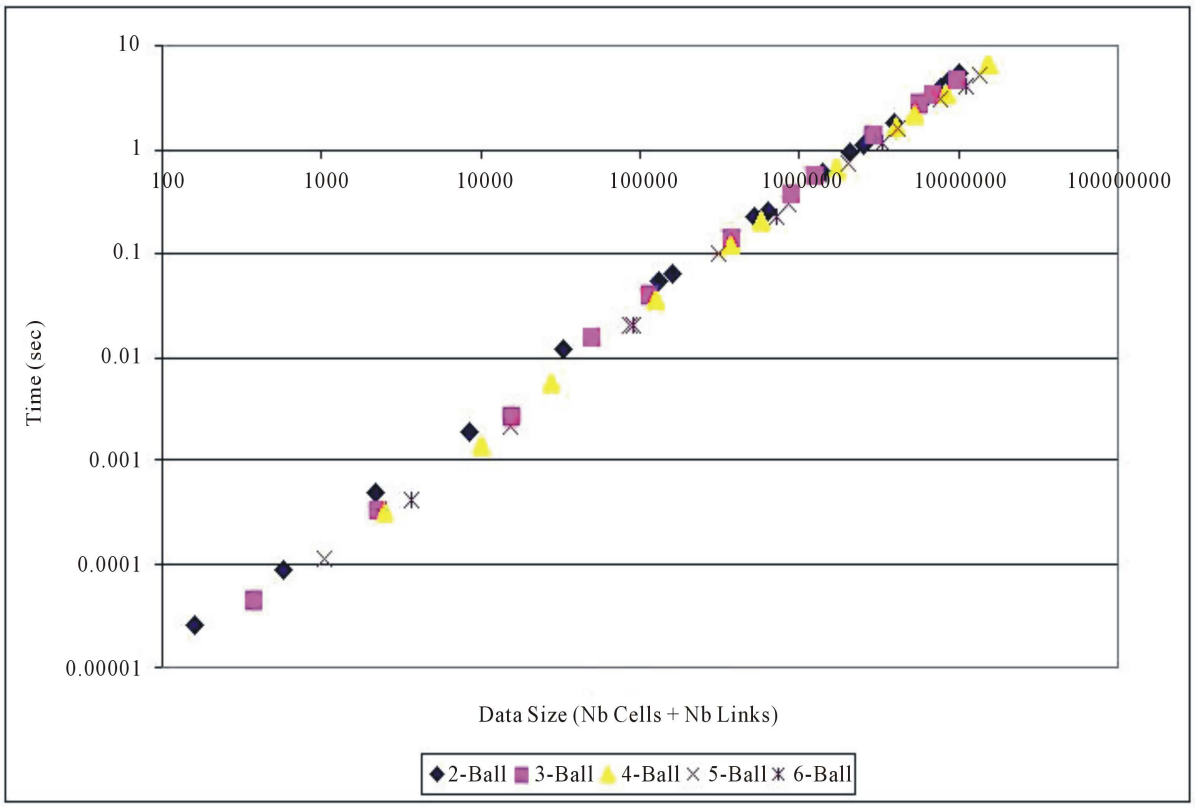
We observe that the approximation times and the parameters α and β for the heart and brain datasets, remain in the same order as those of *d*-balls and *d*-spheres despite the high numbers of generators as shown in **Table 3**. For example, the file heart96 had 30 connected components, 107 holes and 19 voids. For these two datasets, the number of generators was not a relevant factor for the performance of the algorithm. We think it is because those two datasets contain a high ratio of exterior face reductions, which cost near to nothing. If we compare with the random d-complexes then the topology is an important factor. Indeed, as the data size increased we observed bigger differences in the time performance.

In **Figure 16**, we also reproduced the algorithm's performance on the files of each dataset. We used a logarithmic scale on both axes to better distinguish the trends. In **Figure 16(d)**, each file in a *d*-complex has a distinct topology. It explains why we see only a weak trend in opposition to the other datasets where the trends are strong.

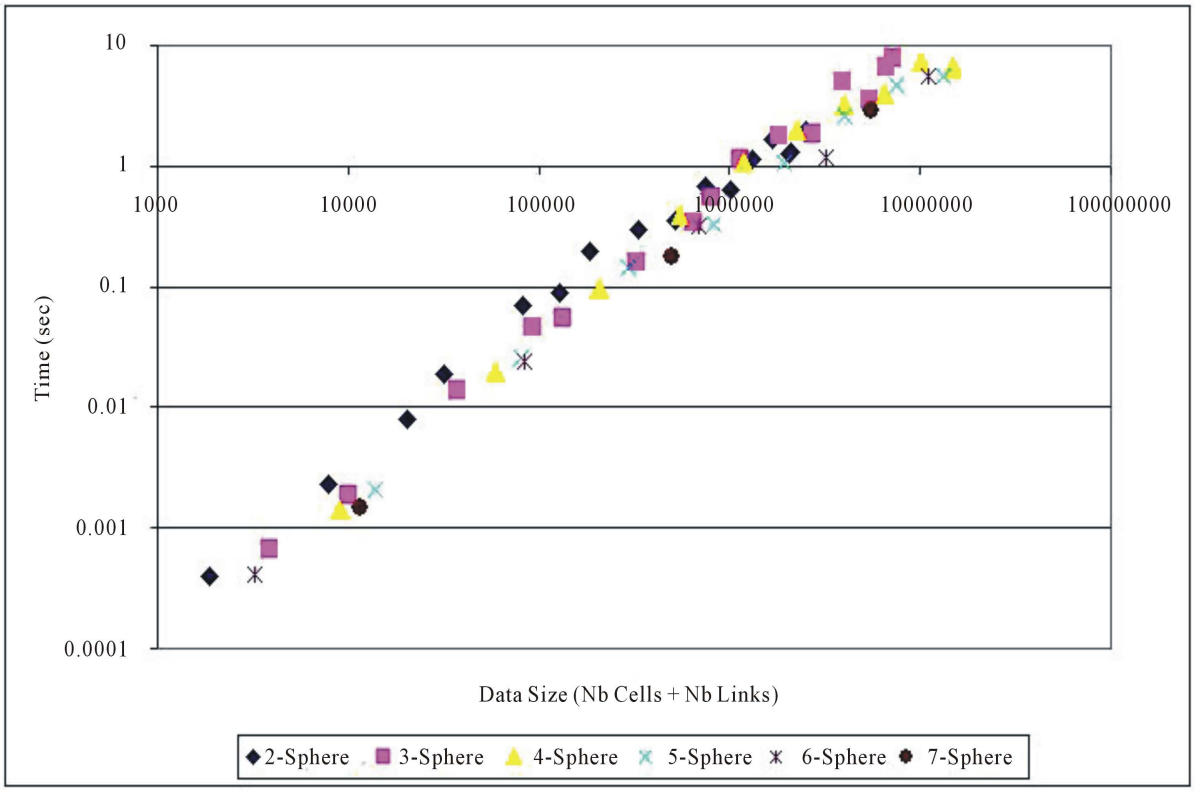
Another interesting aspect of the algorithm to consider is the number of recursions which is expressed as the number of reduction calls in **Table 4**. Interestingly, one reduction call was enough to calculate the homology of all *d*-balls, *d*-spheres and Bing's houses. It means that the algorithm did not reconstruct an intermediate simplified complex to obtain the homology. For other datasets, the algorithm had to reconstruct simplified complexes to carry out the computation. As an example, a value of *n* means that, on average, the algorithm constructed $(n-1)$ simplified complexes. The Max column shows the average of the maximum number of calls and the last column refers to the average standard deviation.

Table 3. Number of generators of the files in the heart and brain datasets.

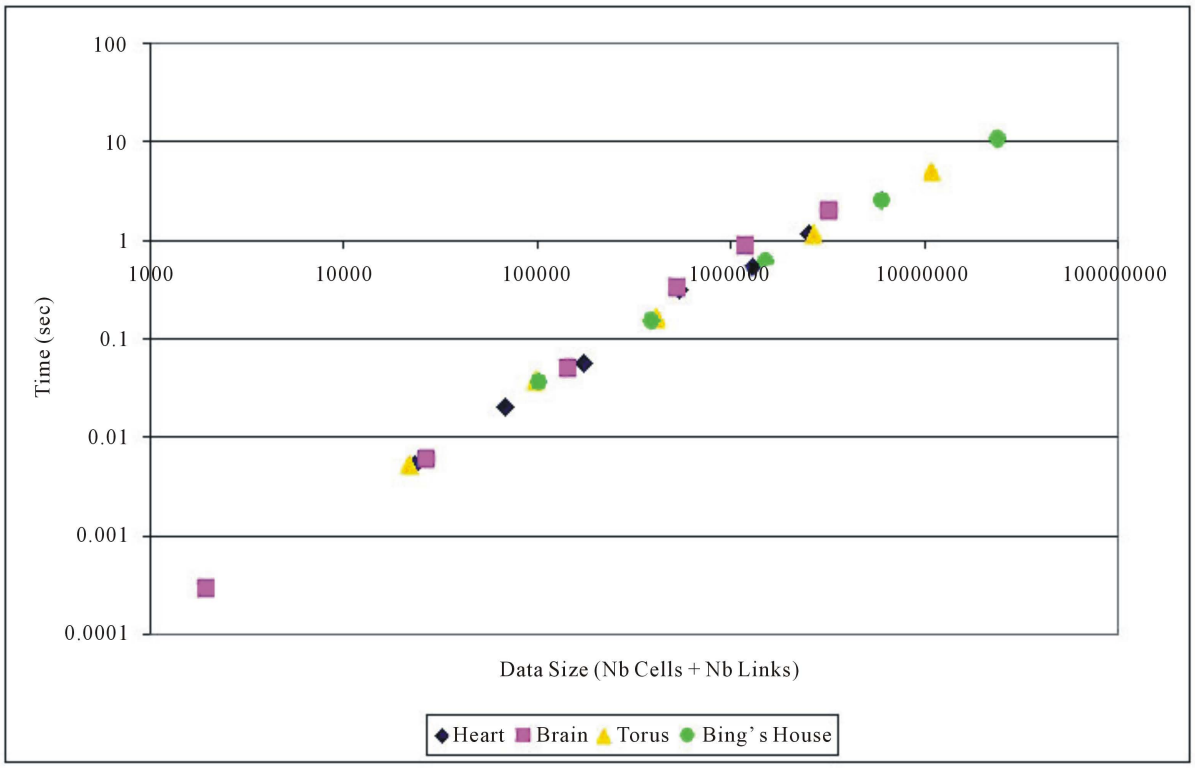
File Name	Number of Generators
heart192	(11, 9, 0)
heart160	(4, 15, 2)
heart128	(39, 12, 6)
heart96	(30, 107, 19)
heart64	(38, 228, 104)
heart32	(111, 220, 208)
brain192	(11, 0, 0)
brain160	(109, 1, 0)
brain128	(230, 25, 0)
brain96	(255, 153, 4)
brain64	(134, 407, 21)
brain32	(41, 524, 252)


10
1
100
1000
10000
100000
1000000
10000000
100000000
0.1
0.01
0.001
0.0001
0.00001
Time (sec)
Data Size (Nb Cells + Nb Links)
2-Ball
3-Ball
4-Ball
5-Ball
6-Ball

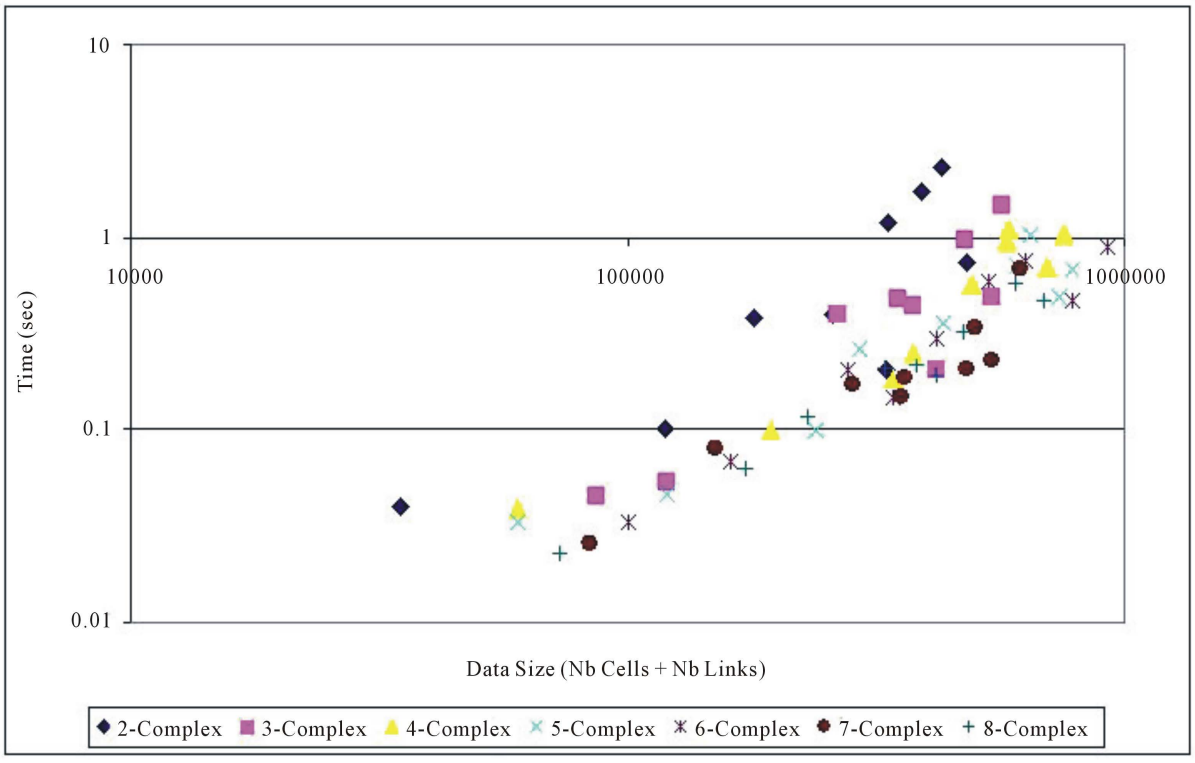
(a)


10
1
1000
10000
100000
1000000
10000000
100000000
0.1
0.01
0.001
0.0001
Time (sec)
Data Size (Nb Cells + Nb Links)
2-Sphere
3-Sphere
4-Sphere
5-Sphere
6-Sphere
7-Sphere

(b)



(c)



(d)

Figure 16. Performance of the algorithm on various data series: (a) *d*-balls, (b) *d*-spheres, (c) heart, brain, Bing's house and torus, (d) random *d*-complexes.

Finally, in **Table 5** we show how much time the algorithm spent on sorting and calculating the generators. Regarding sorting, the issue is that the heuristic uses sorting in attempt to choose reduction pairs that should minimize the number of lists updates. Consequently, we wanted to verify that sorting did not monopolize the computation time. Otherwise, the heuristic would hinder more rather than help the computation. So we measured the average time spent on sorting relative to the total time of a computation and expressed the results in percentage terms. Also, we give the average maximum and average standard deviation. We observe that, on average, the algorithm spent around 5% - 10% of the time on sorting and the extrema are almost always less than 15%. In addition, the algorithm performed much slower when we disabled the heuristic.

Table 4. Number of reduction calls per dataset.

		Reduction Calls		
Dataset	**N**	**Avg**	**Max**	**SD**
Torus	5	1.71	2.20	0.46
Heart	6	3.62	4.33	0.33
Brain	6	3.18	3.33	0.20
2-Complex	9	2.76	3.33	0.34
3-Complex	9	2.68	3	0.27
4-Complex	9	2.94	3.56	0.35
5-Complex	9	2.51	2.78	0.19
6-Complex	9	2.55	2.89	0.21
7-Complex	9	2.89	3.22	0.24
8-Complex	9	2.61	3	0.28
All others	N/A	1	1	0

Table 5. Cost for computing generators and sorting.

		Generators (%)			Sorting (%)		
Dataset	**N**	**Avg**	**Max**	**SD**	**Avg**	**Max**	**SD**
2-Ball	15	105.60	119.85	4.91	10.80	14.17	1.69
3-Ball	11	105.71	118.87	4.86	10.23	13.23	1.55
4-Ball	11	103.99	120.58	5.24	10.81	14.00	1.52
5-Ball	9	104.14	118.88	5.52	10.63	13.21	1.24
6-Ball	5	104.73	113.95	4.84	9.56	12.11	1.14
2-Sphere	17	118.18	140.30	10.89	5.89	7.97	0.82
3-Sphere	16	111.95	129.49	8.85	6.56	8.25	0.62
4-Sphere	10	104.76	126.44	8.68	6.26	8.80	0.90
5-Sphere	9	105.19	122.00	7.76	7.50	10.19	1.00
6-Sphere	5	105.06	120.92	6.81	7.61	11.00	1.31
7-Sphere	3	104.33	118.07	5.09	8.69	10.40	0.87
Torus	5	111.65	130.93	8.39	9.56	13.78	1.65
Bing's House	5	105.51	119.63	5.55	11.68	14.30	1.18
Heart	6	106.11	124.46	7.38	10.36	15.42	2.01
Brain	6	108.76	129.84	10.56	9.55	13.18	1.50
2-Complex	9	115.04	137.84	10.12	3.68	4.67	0.44
3-Complex	9	110.52	131.13	10.33	5.57	7.57	0.77
4-Complex	9	103.44	128.07	10.76	3.56	4.56	0.38
5-Complex	9	101.95	124.02	9.61	5.47	7.89	0.95
6-Complex	9	105.17	127.27	10.63	4.26	5.55	0.43
7-Complex	9	106.66	129.96	11.14	7.11	9.83	0.96
8-Complex	9	104.16	131.26	13.27	4.74	8.50	0.85

Regarding the computation of generators, our algorithm allows to obtain them for an additional cost of about 10% or less on average. From **Table 5** we see that most average times fluctuate around 5% and the maximums fall into the 15% - 30% range. Moreover, the algorithm performed as strongly on datasets with a large amount of generators than on those with a small amount.

6. Conclusions

We developed in this paper a novel and efficient algorithm for the computation of homology groups and homology generators that works at the level of chain complexes, and hence allows handling a variety of geometric complexes. The main idea is based on organizing sequences of cell reductions into a structure of directed acyclic graphs, which makes it possible to derive global projection formulas and perform large collections of reductions at once. In this method, the boundary operators of the complex are not explicitly updated after each reduction step. Instead, this information is always preserved implicitly within the data structures and processed globally for each reduction graph allowing reducing considerably the computational time.

Our experimentations show that this algorithm performs significantly faster than the classical reduction method. In addition, for all the datasets that we tested, which cover a wide range of types of data, their global performance indicated a subquadratic time complexity. Moreover, it allows calculating the homology generators at a small additional time cost. Interestingly enough, in all the datasets we dealt with, the algorithm needed only to construct few intermediate complexes to obtain the homology of the original complex. Our feeling is that if one encounters a specially designed complex in which the number of recursions is substantially high then the algorithm may underperform. However, it is not yet clear how to construct such a complex.

References

[1] Allili, M. and Corriveau, D. (2007) Topological Analysis of Shapes using Morse Theory. *Computer Vision and Image Understanding*, **105**, 188-199. http://dx.doi.org/10.1016/j.cviu.2006.10.004

[2] Allili, M., Corriveau, D. and Ziou, D. (2004) Morse Homology Descriptor for Shape Characterization. *Proceedings of the* 17*th International Conference on Pattern Recognition*, Vol. 4, 27-30. http://dx.doi.org/10.1109/icpr.2004.1333697

[3] Allili, M., Corriveau, D., Derivière, S., Kaczynski, T. and Trahan, A. (2007) Discrete Dynamical System Framework for Construction of Connections between Critical Regions in Lattice Height Data. *Journal of Mathematical Imaging and Vision*, **28**, 99-111. http://dx.doi.org/10.1007/s10851-007-0010-0

[4] Collins, A., Zomorodian, A., Carlsson, G. and Guibas, L. (2004) A Barcode Shape Descriptor for Curve Point Cloud Data. *Computers and Graphics*, **28**, 881-894. http://dx.doi.org/10.1016/j.cag.2004.08.015

[5] Kaczynski, T., Mischaikow, K. and Mrozek, M. (2004) Computational Homology. Applied Mathematical Sciences Series 157, Springer-Verlag, New York.

[6] Munkres, J.R. (1984) Elements of Algebraic Topology. Addison-Wesley.

[7] Kannan, R. and Bachem, A. (1979) Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix. *SIAM Journal on Computing*, **8**, 499-507. http://dx.doi.org/10.1137/0208040

[8] Chou, T.W. and Collins, G.E. (1982) Algorithms for the Solutions of Systems of Linear Diophantine Equations. *SIAM Journal on Computing*, **11**, 687-708. http://dx.doi.org/10.1137/0211057

[9] Iliopoulos, C.S. (1989) Worst-Case Complexity Bounds on Algorithms for Computing the Canonical Structure of Finite Abelian Groups and the Hermite and Smith Normal Forms of an Integer Matrix. *SIAM Journal on Computing*, **18**, 658-669. http://dx.doi.org/10.1137/0218045

[10] Storjohann, A. (1996) Near Optimal Algorithms for Computing Smith Normal Forms of Integer Matrices. *Proceedings of* 1996 *International Symposium on Symbolic and Algebraic Computation*, *ISSAC*'96, Zurich, 24-26 July 1996, 267-274. http://dx.doi.org/10.1145/236869.237084

[11] Kaczynski, T., Mrozek, M. and Slusarek, M. (1998) Homology Computation by Reduction of Chain Complexes. *Computers and Mathematics with Applications*, **35**, 59-70.

[12] Mrozek, M., Pilarczyk, P. and Zelazna, N. (2008) Homology Algorithm Based on Acyclic Subspace. *Computers and Mathematics with Applications*, **55**, 2395-2412. http://dx.doi.org/10.1016/j.camwa.2007.08.044

[13] Delfinado, C.J.A. and Edelsbrunner, H. (1995) An Incremental Algorithm for Betti Numbers of Simplicial Complexes on the 3-Sphere. *Computer Aided Geometric Design*, **12**, 771-784. http://dx.doi.org/10.1016/0167-8396(95)00016-Y

[14] Mrozek, M. and Batko, B. (2009) Coreduction Homology Algorithm. *Discrete and Computational Geometry*, **41**, 96-

118. http://dx.doi.org/10.1007/s00454-008-9073-y

[15] Corriveau, D. and Allili, M. (2009) Computing Homology: A Global Reduction Approach. In: Brlek, S., Reutenauer, C. and Provençal, X., Eds., *Discrete Geometry for Computer Imagery*: *Proceedings of* 15*th IAPR International Conference*, *DGCI* 2009, Springer, Berlin Heidelberg, 313-324. http://dx.doi.org/10.1007/978-3-642-04397-0_27