An algorithm for calculating top-dimensional bounding chains

J. Frederico Carvalho[1], Mikael Vejdemo-Johansson[2], Danica Kragic[1] and Florian T. Pokorny[1]

[1] CAS/RPL, KTH, Royal Institute of Technology, Stockholm, Sweden
[2] Mathematics Department, City University of New York, College of Staten Island, New York, NY, United States of America

ABSTRACT

We describe the Coefficient-Flow algorithm for calculating the bounding chain of an $(n-1)$-boundary on an n-manifold-like simplicial complex S. We prove its correctness and show that it has a computational time complexity of $O(|S^{(n-1)}|)$ (where $S^{(n-1)}$ is the set of $(n-1)$-faces of S). We estimate the big-O coefficient which depends on the dimension of S and the implementation. We present an implementation, experimentally evaluate the complexity of our algorithm, and compare its performance with that of solving the underlying linear system.

Subjects Algorithms and Analysis of Algorithms, Data Science, Scientific Computing and Simulation
Keywords Homology, Computational algebraic topology

Submitted 4 August 2017
Accepted 10 April 2018
Published 28 May 2018

Corresponding author
J. Frederico Carvalho, jfpbdc@kth.se

Academic editor
Mark Wilson

Additional Information and Declarations can be found on page 11

DOI **10.7717/peerj-cs.153**

INTRODUCTION

Topological spaces are by and large characterized by the cycles in them (i.e., closed paths and their higher dimensional analogues) and the ways in which they can or cannot be deformed into each other. This idea had been recognized by Poincaré from the beginning of the study of topology. Consequently much of the study of topological spaces has been dedicated to understanding cycles, and these are the key features studied by the topological data analysis community (*Edelsbrunner & Harer, 2010*).

One key part of topological data analysis methods is to distinguish between different cycles; more precisely, to characterize different cycles according to their homology class. This can be done efficiently using cohomology (*Pokorny, Hawasly & Ramamoorthy, 2016*; *Edelsbrunner, Letscher & Zomorodian, 2002*). However, such methods only distinguish between non-homologous cycles, and do not quantify the difference between cycles. A possible way to quantify this difference is to solve the problem of finding the chain whose boundary is the union of the two cycles in question, as was proposed in *Chambers & Vejdemo-Johansson (2015)* by solving the underlying linear system, where the authors also take the question of optimality into account (with regards to the size of the resulting chain). This line of research ellaborates on *Dey, Hirani & Krishnamoorthy (2011)* where the authors considered the problem of finding an optimal chain in the same homology class as a given chain. More recently, in *Rodríguez et al. (2017)* the authors have taken a similar approach to ours using a combinatorial method to compute bounding chains of 1-cycles on 3-dimensional simplicial complexes.

In this paper, we explore the geometric properties of simplicial n-manifolds to provide an algorithm that is able to calculate a chain whose boundary is some prescribed $(n-1)$-dimensional cycle, and we show that the proposed algorithm has a complexity which is linear in the number of $(n-1)$-faces of the complex. This is substantially faster than calculating a solution to the linear system as considered in *Chambers & Vejdemo-Johansson (2015)*, for which the complexity would be, at least, quadratic on the number of n-dimensional faces (*Gloub & Van Loan, 1996*).

Background

In what follows we make extensive use of sequences. Therefore, for any $n \in \mathbb{N}$, we abbreviate $x_0, \dots, x_n$ to $x_{0:n}$.

Simplicial complexes

Given a set of points $P \subseteq \mathbb{R}^n$, we define a *k-dimensional simplex*, or k-simplex, on points of P as the ordered set $[p_{0:k}]$, where $p_{0:k} \in P$ are $k+1$ affinely independent points and are called the *vertices* of $[p_{0:k}]$. We represent the simplex $[p_{0:k}]$ by the convex hull of the points $p_{0:k}$, and we say that two simplices are the same if they have the same points and the ordering of the points differs only by an even permutation. If the ordering differs by an odd permutation we say they have opposite *orientations*.

Since a convex hull of a finite set of points is a bounded closed set, it carries the notion of a *boundary* $\partial[p_{0:k}]$ which is defined as:

$$\partial[p_{0:k}] = [p_{1:k}] + \left(\sum_{i=1}^{k-1}(-1)^i [p_{0:i-1}, p_{i+1:k}]\right) + (-1)^k [p_{0:k-1}].$$

The above sum can be interpreted as a "union with orientation", and multiplying by 1 or -1 is the identity or a reversal of orientation, respectively. Note that if $p_{0:k}$ are affinely independent, then the boundary of the convex hull does indeed correspond to the union of the convex hulls of all subsets of $\{p_{0:k}\}$ with k distinct points.

For example, the boundary of the simplex $[p_0, p_1, p_2]$ is given by:

$$[p_0, p_1] - [p_0, p_2] + [p_1, p_2]$$

Applying the only possible orientation-reversing permutation to $[p_0, p_2]$ gives $[p_0, p_1] + [p_1, p_2] + [p_2, p_0]$. This corresponds to the union of the edges that form the boundary of the triangle $[p_0, p_1, p_2]$ oriented in such a way as to form a *closed path*.

Definition 1. *A set of points $P \subseteq \mathbb{R}^n$ and a set of simplices $T = \{\sigma_{0:N}\}$ defines a* geometric simplicial complex *$S = (P, T)$ if any finite subset of a simplex in T is also in T, and given any two simplices $[p_{0:k}], [q_{0:k'}] \in T$, the intersection of the convex hulls $[p_{0:k}] \cap [q_{0:k'}]$ is the convex hull of $\{p_{0:k}\} \cap \{q_{0:k'}\}$ and is also a simplex in T.*

For any d we define the d-skeleton of T by $T^d = \{\sigma \in T \mid \dim\sigma \leqslant d\}$ and the d-th level *as $T^{(d)} = \{\sigma \in T \mid \dim\sigma = d\}$.*

Given two simplices σ, τ we write $\tau \lhd \sigma$ if $\tau \subset \sigma$ and $\dim\tau = \dim\sigma - 1$ which can be read as "τ is a top-dimensional face of σ". Note that $\lhd$ is not transitive, and therefore it is not a preorder. Its transitive closure, $\tau < \sigma$ however, defines a preorder. We can thus read $\tau < \sigma$ as "τ is contained in the boundary of σ" or simply "τ is a *face* of σ".

We say that a simplicial complex $S = (P, T)$ has *dimension* d if d is the largest integer such that $T^{(d)} \neq \varnothing$.

Definition 2. *A d-dimensional simplicial complex $S = (P, T)$ is called a* d-manifold-like *simplicial complex if*

- *for every $\tau \in T^{d-1}$ there exists some $\sigma \in T^{(d)}$ such that $\sigma > \tau$, and*
- *if $\dim \tau = (d-1)$ then there are at most two $\sigma \in T^{(d)}$ satisfying $\sigma > \tau$.*

Note that a triangulation of a d-manifold is a manifold-like simplicial complex, however the definition also includes other spaces like triangulations of manifolds with boundary and the pinched torus.

Algebraic description

We will focus on finite geometric simplicial complexes $S = (P, T)$ (where $|P|, |T| < \infty$). Since such an S has a finite number of simplices, we can define for each level $0 \leqslant k \leqslant \dim(S)$ an injective function $\iota_k : T^{(k)} \to \mathbb{N}$ such that $\iota_k(T^{(k)}) = \{1, \ldots, |T^{(k)}|\}$; we call ι an *enumeration* of $T^{(k)}$. From this we define the *chain complex* associated with S.

Definition 3. *Given a simplicial complex $S = (P, T)$ the* chain complex *associated with S is defined as the pair $\{(C_k(S), d_k)\}_{k=0}^{+\infty}$ where the $C_k(S)$ are vector spaces defined as $C_k(S) = \mathbb{R}^{|T^{(k)}|}$ and the d_k are linear maps $d_k : C_k(S) \to C_{k-1}(S)$ defined on basis elements as*

$$d_k(e_i) = \sum_{\tau \in \partial(\iota_k^{-1}(i))} o(i, \tau) e_{\iota_{k-1}(\tau)}$$

where $o(i, \tau)$ is the orientation of τ induced by the boundary of $\sigma = \iota_k^{-1}(i)$.

It can be shown that $d_k \circ d_{k+1} = 0$, which allows us to define, for each k, the k-th homology group of S as

$$H_k(S) = \ker(d_k)/\operatorname{im}(d_{k+1}).$$

By a slight abuse of notation, for a simplicial complex $S = (P, T)$ and a k-chain c, we write c_σ for the coefficient corresponding to σ, e_σ for the corresponding basis element and d for the appropriate boundary map whenever these are clear from their contexts.

We call the elements $p \in C_k(S)$ such that $dp = 0$, *k-cycles*. Two k-cycles p, p' are said to be *homologous* if there exists a chain $c \in C_{k+1}(S)$ such that $dc = p - p'$, so that $p - p'$ is called a *k-boundary*. This defines k-homology as the group of k-cycles quotiented by the homology relation.

Problem description and contribution

We are interested in the bounding chain problem, that is, given a cycle p, we want to decide whether or not p is a boundary, and in case it is, provide a witness in the form of a chain c such that $\partial c = p$; we call c a *bounding chain* of p. To achieve this, we further specialize the problem to

$$\text{solve: } \partial c = p$$
$$\text{subject to: } c_\sigma = v, \tag{1}$$

where p is a specified $(n-1)$-boundary in an n-manifold-like simplicial complex S, σ is an n-simplex, and v is a pre-specified real number. In general, the equation $\partial c = p$ has more than one solution c, therefore by adding the constraint $c_\sigma = v$ we are able to make this solution unique.

The Coefficient-Flow algorithm that we present solves this restricted form of the bounding chain problem (by providing one such bounding chain if it exists) and has computational time complexity of $O(|S^{(n-1)}|)$. Furthermore, we show how the parameters σ and v can be done away with in cases where the chain is unique, and we discuss how this algorithm can be used to find a minimal bounding chain.

Related work

In *Chambers & Wang (2013)* the authors address the problem of computing the area of a homotopy between two paths on 2-dimensional manifolds, which can be seen as a generalization of the same problem, for 2-dimensional meshes via the Hurewicz map (*Hatcher, 2002*). In *Chambers & Vejdemo-Johansson (2015)* the authors provide a method for calculating the minimum area bounding chain of a 1-cycle on a 2d mesh, that is the solution to the problem

$$\arg\min{}_c \operatorname{area}(c) = p, \qquad \text{where } \partial c = p \tag{2}$$

and p is a 1-chain on a given simplicial complex. This is done by using optimization methods for solving the associated linear system. These methods however have time complexity lower-bounded by matrix multiplication time which is in $\Omega(\min(n,m)^2)$ where n, m are the number of rows and columns of the boundary matrix[1] (*Davie & Stothers, 2013*). This complexity quickly becomes prohibitive when we handle large complexes, such as one might find when dealing with meshes constructed from large pointclouds.

[1]Which corresponds to the number of $(k-1)$- and k-faces of the complex, respectively.

More recently, in *Rodríguez et al. (2017)* the authors proposed a method for computing bounding chains of 1-cycles in 3-dimensional complexes, using a spanning tree of the dual graph of the complex.

In *Dey, Hirani & Krishnamoorthy (2011)* the authors address the related problem of efficiently computing an optimal cycle p' which is homologous to a given cycle p (with $\mathbb{Z}$ coefficients). This is a significant result given that in *Chambers, Erickson & Nayyeri (2009)* the authors proved that this cannot be done efficiently (i.e., in polynomial time) for 1-cycles using $\mathbb{Z}_2$ coefficients, a result that was extended in *Chen & Freedman (2011)* to cycles of any dimension.

METHODOLOGY

For any simplicial complex S, and any pair of simplices $\sigma, \tau \in S$ such that, $\tau \lhd \sigma$, we define the index of τ with respect to σ as $\langle \tau, \partial\sigma \rangle = \langle e_\tau, de_\sigma \rangle$ (*Forman, 1998*). Note that the index corresponds to the orientation induced by the boundary of σ on τ and can be computed in $O(d)$ time by the following algorithm:

```
INDEX($\sigma, \tau$):
    param: $\sigma$ - $k$-simplex represented as a sorted list of indices of points.
    param: $\tau$ - $(k-1)$-face of $\sigma$ represented as a sorted list of indices.
1: for each $i \leftarrow 0 \dots \dim\tau$ :
2:     if $\tau_i \neq \sigma_i$:
3:         orientation $\leftarrow (-1)^i$
4:         break loop
5: return orientation
```

By inspecting the main loop we can see that INDEX(σ,τ) returns $(-1)^i$ where i is the index of the first element at which σ and τ differ. We assume τ is a top dimensional face of σ, so if $\sigma = [s_{0:d}]$, then by definition $\tau = [s_{0:(i-1)}, s_{(i+1):d}]$ for some i, and so the coefficient of τ in the boundary of σ is $(-1)^i$ as per the definition of the boundary operator. This is also the index of the first element at which τ and σ differ, since they are represented as sorted lists of indices.

The following is an intuitive result, that will be the basis for the Coefficient-Flow algorithm that we will present in the sequel.

Proposition 1. *Let S be a manifold-like simplicial complex, let c be an n-chain on S and p its boundary. Then for any pair of n-simplices $\sigma \neq \sigma'$ with $\partial\sigma \cap \partial\sigma' = \{\tau\}$ we have:*

$$c_\sigma = \langle \tau, \partial\sigma \rangle (p_\tau - \langle \tau, \partial\sigma' \rangle c_{\sigma'}).$$

Proof. If we expand the equation $\partial c = p$, we get $p_\tau = \sum_{\tau \lhd \omega} \langle e_\tau, d(c_\omega e_\omega) \rangle$, recall that by definition $d(c_\omega e_\omega) = \sum_{\nu \lhd \omega} \langle \nu, \partial\omega \rangle c_\omega e_\nu$; and so we get $p_\tau = \sum_{\tau \lhd \omega} \langle \tau, \partial\omega \rangle c_\omega e_\tau$.

Now since S is a manifold-like simplicial complex and $\tau = \partial\sigma \cap \partial\sigma'$, then σ, σ' are the *only* cofaces of τ, and hence we have:

$$p_\tau = \langle \tau, \partial\sigma \rangle c_\sigma + \langle \tau, \partial\sigma' \rangle c_{\sigma'}$$

which can be reorganized to $c_\sigma = \frac{p_\tau - \langle \tau, \partial\sigma' \rangle c_{\sigma'}}{\langle \tau, \partial\sigma \rangle}$. Finally, since the index $\langle \tau, \partial\sigma \rangle$ is either 1 or -1, we can rewrite this equation as:

$$c_\sigma = \langle \tau, \partial\sigma \rangle (p_\tau - \langle \tau, \partial\sigma' \rangle c_{\sigma'})$$

Next, we present an algorithm to calculate a bounding chain for a $(n-1)$-cycle in an n-manifold-like simplicial complex. The algorithm proceeds by checking every top-dimensional face σ, and calculating the value of the chain on adjacent top-dimensional faces, using Proposition 1.

In order to prove that the Coefficient-Flow algorithm solves problem (1), will use the fact that we can see the algorithm as a traversal of the *dual graph*.

Definition 4. *Given an n-dimensional simplicial complex S, recall that the* dual graph *is a graph $G(S) = (V, E)$ with set of vertices $V = S^{(n)}$ and $(\sigma, \sigma') \in E$ if $\dim(\sigma \cap \sigma') = n-1$.*

Proposition 2. *If S is a manifold-like simplicial complex, where $G(S)$ is connected, and p is an $(n-1)$-boundary, then* COEFFICIENT- FLOW(p, σ, v) *returns a bounding chain c of p satisfying $c_\sigma = v$, if such a boundary exists. Furthermore, the main loop (05–21) is executed at most $O(|S^{(n-1)}|)$ times.*

```
COEFFICIENT-FLOW($p, \sigma_0, v_0$):
     param: $p$ — an $n-1$ boundary of the simplicial complex $S$
     param: $\sigma_0$ — an $n$–simplex from where the calculation will start
     param: $v_0$ — the value to assign the bounding chain at sigma
     return: $c$ — a bounding chain of $p$ satisfying $c_{\sigma_0} = v_0$ (if it exists)
6:  initialize $c$ and mark every $n$- and $(n-1)$-cell of $S$ as not seen.
7:  initialize an empty queue $Q$
8:  let $\tau_0 \in \partial\sigma_0$
9:  enqueue $(\sigma_0, \tau_0, v_0)$ into Q.
10: while $Q$ is non-empty:
11:    $(\sigma, \tau, v) \leftarrow$ pop first element from $Q$
12:    if $\sigma$ has been marked seen:
13:       if $v \neq c_\sigma$: the problem has no solution
14:    else:
15:       if $\tau$ has been marked seen: skip
16:       $c_\sigma \leftarrow v$
17:       mark $\tau$ and $\sigma$ as seen
18:       for each $\tau' \in \partial\sigma$:
19:          if $\sigma$ is the only coface of $\tau'$:
20:             mark $\tau'$ as seen
21:             if $p_{\tau'} \neq \langle \partial\sigma, \tau' \rangle v$: the problem has no solution
22:          else:
23:             if $\tau'$ has not been marked as seen:
24:                $\sigma' \leftarrow$ other coface of $\tau'$
25:                $v' \leftarrow \langle \partial\sigma', \tau \rangle (p_\tau - \langle \partial\sigma, \tau \rangle v)$
26:                enqueue $(\sigma', \tau', v')$ into $Q$
27: return c
```

Proof. We start by proving the bound on the number of executions of the main loop. This is guaranteed by the fact that the loop is executed while the queue is non-empty, and a triple (σ, τ, v) can only be inserted in line (21) if τ has not been marked as seen. Furthermore, since τ has at most two cofaces, say, σ, σ', we can only enqueue τ if we are analyzing σ or σ' and so for each τ, at most two elements are placed in the queue, and hence the main loop gets executed at most $2|S^{(n-1)}|$ times.

To prove correctness of the algorithm, we have to prove that it outputs an error if and only if the problem has no solution, and otherwise it outputs a bounding chain of p with $c_{\sigma_0} = v$.

First, we note that if a face σ is marked as seen, the value of c_σ can never be reassigned. This is because the program branches on whether or not σ has been marked as seen, and c_σ can only be assigned on line (11) which is bypassed if σ has been previously marked as seen. From this fact we conclude that $c_{\sigma_0} = v_0$ as it is assigned on line (11) and σ_0 is marked as seen in the first iteration of the main loop.

Second, note that there is an edge between two n-faces in the dual graph if and only if they share an $(n-1)$-face. This implies that as we execute the algorithm, analyze a new n-face σ and successfully add the other cofaces of elements of the boundary of σ, we add the vertices neighboring σ in the dual graph. Since the dual graph is connected all of the nodes in the graph are eventually added, and hence all of the n-faces are analyzed.

Third, we note that for any pair (τ,σ) with $\dim\sigma = n$ and $\tau \lhd \sigma$, either σ is the only coface of τ, or τ has another coface, σ'. In the first case, if $p_\tau \neq c_\sigma\langle\tau,\partial\sigma\rangle$ an error is detected on line (16). In the second case, assuming that the triple (σ,τ,v) is enqueued before (σ',τ,v') we have $v' = \langle\partial\sigma',\tau\rangle(p_\tau - \langle\partial,\sigma\rangle v)$ as is assigned in line (20) then

$$
\begin{aligned}
(dc)_\tau &= \langle\partial\sigma,\tau\rangle v + \langle\partial\sigma',\tau\rangle v' \\
&= \langle\partial\sigma,\tau\rangle v + \langle\partial\sigma',\tau\rangle(\langle\partial\sigma',\tau\rangle(p_\tau - \langle\partial,\sigma\rangle v)) \\
&= \langle\partial\sigma,\tau\rangle v + p_\tau - \langle\partial\sigma,\tau\rangle v = p_\tau .
\end{aligned}
$$

Finally, since upon the successful return of the algorithm, this equation must be satisfied by every pair $\tau \lhd \sigma$, it must be the case that $dc = p$. If this is not the case, then there will be an error in line (08) and the algorithm will abort. □

Note that the connectivity condition can be removed if we instead require a value for one cell in each connected component of the graph $G(S)$, and throw an error in case there is an $(n-1)$-simplex τ with no cofaces, such that $p_\tau \neq 0$. Furthermore, the algorithm can be easily parallelized using a thread pool that iteratively processes elements from the queue.

Finally, in the case where it is known that S has an $(n-1)$-face τ with a single coface, we do not need to specify σ or v in COEFFICIENT-FLOW, and instead use the fact that we know the relationship between the coefficient p_τ and that of its coface in a bounding chain c of p, i.e., $p_\tau = \langle\partial\sigma,\tau\rangle c_\sigma$. This proves Corollary 1.

```
BOUNDING-CHAIN($p$):
    param: $p$ — an $n-1$ boundary of the simplicial complex $S$
    return: $c$ — a bounding chain of $p$
28: for each $\tau \in S^{n-1}$:
29:   if $\tau$ has a single coface:
30:     $\sigma \leftarrow$ the coface of $\tau$
31:     $v \leftarrow \langle\partial\sigma,\tau\rangle p_\tau$
32:     break loop
33: $c \leftarrow$ COEFFICIENT-FLOW($p,\sigma,v$)
34: return $c$
```

Corollary 1. *If S is a connected n-manifold-like simplicial complex with a connected dual graph, and has an $(n-1)$-face with a single coface, then given an $(n-1)$-cycle p on S,* BOUNDING-CHAIN(p) *returns a bounding chain of p if one exists.*

Implementation details

We will now discuss some of the choices made in our implementation of the Coefficient-Flow algorithm (https://www.github.com/crvs/coeff-flow). Before we can address the problem of considering chains on a simplicial complex we first need to have a model of a simplicial complex. For this, we decided to use a simplex tree model (*Boissonnat & Maria, 2014*) provided by the GUDHI library (*The GUDHI Project, 2015*) as it provides a compact representation of a simplicial complex (the number of nodes in the tree is in bijection with the number of simplices) which allows us to quickly get the enumeration of a given simplex σ. Indeed, the complexity of calculating $\iota_k(\sigma)$ is in $O(\dim\sigma)$.

It is important to compute the enumeration quickly because in our implementation of Coefficient-Flow we use arrays of booleans to keep track of which faces of the simplicial complex have been seen before, as well as numerical arrays to store the coefficients of the cycle and its bounding chain, which need to be consulted at every iteration of the loop.

However, finding the cofaces of the simplicial complex is not as easy in a simplex tree, since, if $\sigma = [p_{i_0:i_k}]$, this would require to search every child of the root node of the tree that has an index smaller than i_0, followed by every child of the node associated with p_{i_0} and so on, which in the worst case scenario is in $O(\dim\sigma |S^{(0)}|)$. Thus we need to adopt a different method. For this, we keep a Hasse diagram of the face relation which comprises a directed graph whose vertices are nodes in the simplex tree and has edges from each face to its codimension-1 cofaces (see for instance *Di Battista & Tamassia (1988)*) for more details of this data-structure). This allows us to find the codimension-1 cofaces of a simplex of S in $O(1)$ with a space overhead in $O(|S|)$.

With these elements in place, we can analyze the complexity of our implementation of the Coefficient-Flow algorithm in full:

Lemma 1. *The* Coefficient-Flow *algorithm using a simplex tree and a Hasse diagram has computational (time) complexity* $O(d^2|S^{(d-1)}|)$ *where* $d = \dim S$.

Proof. In Proposition 2 we saw that the Coefficient-Flow algorithm executes the main loop at most $O(|S^{(d)}|)$ times, so we only need to measure the complexity of the main loop, in order to obtain its time complexity. This can be done by checking which are the costly steps:

- In lines (07) and (10) checking whether a face has been marked as seen requires first computing $\iota_k(\tau)$ which, as we stated above, has time complexity $O(k)$, with $k \leq d$.
- In line (13), computing the faces of a simplex σ requires $O(\dim\sigma^2)$ steps, and yields a list of size $\dim\sigma$, hence the inner loop (13–21) is executed $\dim\sigma$ times, where $\dim\sigma = d$, for each element placed in the queue.
- The loop (13–21) requires once again, computing $\iota(\tau')$, and $\langle\partial\sigma,\tau\rangle$, each of these operations, as we explained before, carries a time complexity $O(d)$.
- All other operations have complexity $O(1)$.

Composing these elements, the total complexity of one iteration of the main loop is $O(\max\{d, d^2, d\cdot d\}) = O(d^2)$, which yields a final complexity for the proposed implementation, of $O(d^2|S^{(d-1)}|)$. □

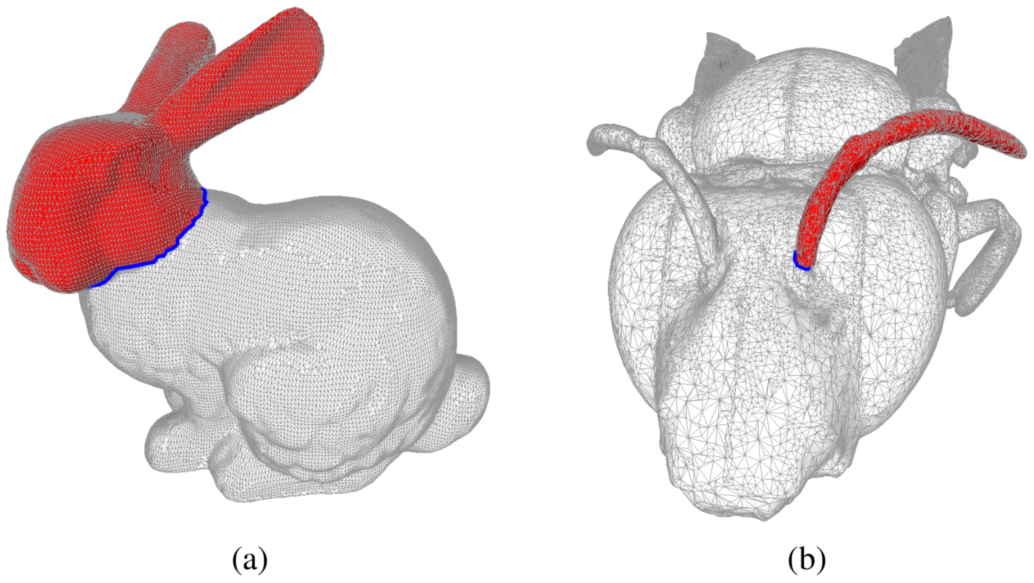

Figure 1 Examples of bounding chains: the edges in blue form cycles in the mesh, and the faces in red form the corresponding bounding chain as computed by the COEFFICIENT-FLOW algorithm. In (A) we depict the mesh of the Stanford Bunny *The Stanford 3D scanning repository (1994)* and in (B) we show the mesh of a Bee *Smithsonian (2013)*. In both these meshes we depict examples of bounding chains, where the edges in blue form cycles, and the faces in red form the corresponding bounding chain as computed by the COEFFICIENT-FLOW algorithm. In both cases the depicted bounding chains correspond to the optimal bounding chains (w.r.t. the number of faces), these can by obtained by choosing σ_0 and v_0 so as to yield the desired chain. In this case, since the two complexes are topological spheres and the cycles are simple cycles (meaning they are connected and do not self-intersect), there are only two possible bounding chains that do not include all the faces of the complex, which can be obtained by running the algorithm three times, choosing σ_0 arbitrarily, and setting v_0 to be $0, n$ or $-n$ where $n = \max_{\tau \in S^{(1)}} |p_\tau|$. In the case of non-simple cycles, more alternatives would exist.

Full-size DOI: 10.7717/peerjcs.153/fig-1

Example runs and tests

In Fig. 1 we provide an example of the output of the COEFFICIENT-FLOW algorithm for the mesh of the Stanford bunny (*The Stanford 3D scanning repository, 1994*) and the *eulaema meriana* bee model from the Smithsonian 3D model library (*Smithsonian, 2013*).

For comparison, we performed the same experiments using the Eigen linear algebra library (*Eigen, 2017*) to solve the underlying linear system,[2] and summarized the results in Table 1. This allowed us to see that even though both approaches remain feasible with relatively large meshes, solving the linear system consistently underperforms using the COEFFICIENT-FLOW algorithm.

Even though COEFFICIENT-FLOW is expected to outperform a linear system solver (an exact solution to a linear system has $\Omega(n^2)$ time complexity), we wanted to test it against an approximate sparse system solver. Such solvers (e.g., conjugate gradient descent (*Gloub & Van Loan, 1996*)) rely on iterative matrix products, which in the case of boundary matrices of dimension d can be performed in $O((d+1)n)$ where n is the number of d-dimensional simplices, placing the complexity of the method in $\Omega((d+1)n)$. In order to observe the difference in complexity class we performed randomized tests on both implementations.

[2]We use the least squares conjugate gradient descent method to solve the system.

Table 1 **Timing for computation of bounding chains using Coefficient-Flow, and using Eigen in several meshes.** "Bunny" and "Bee (Sample)/Bee (Full)" refer to the meshes in Figs. 1A and 1B, respectively. The mesh "Bee (Full)" is the one obtained from *Smithsonian (2013)*, whereas the one labeled "Bee (Sample)" is a sub-sampled version of it.

Mesh	Faces	Edges	Vertices	Eigen	Coefficient-Flow
Bunny	69 663	104 496	34 834	2.073 (s)	**0.48911 (s)**
Bee (Sample)	499 932	749 898	249 926	116.553 (s)	**3.04668 (s)**
Bee (Full)	999 864	1 499 796	499 892	595.023 (s)	**7.15014 (s)**

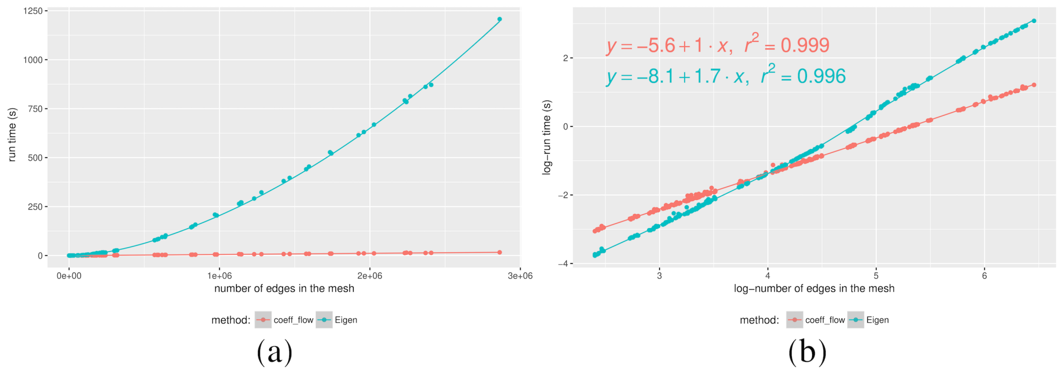

Figure 2 **Running times for a calculating the bounding chain of a cycle as a function of the number of edges (A), and Log–log plot of the same data (B).**

Full-size DOI: 10.7717/peerjcs.153/fig-2

In this scenario we made a mesh on a unit square from a random sample. By varying the number of points sampled from the square, we effectively varied the resolution of the mesh. Finally, at each resolution level we snapped a cycle onto the mesh, and computed its bounding chain using both Coefficient-Flow and by solving the sparse linear system as before. We plotted the timings in Fig. 2 from where we can experimentally observe the difference in the complexity class between our algorithm and the solution to the sparse linear system.

Furthermore by analyzing the Log-Log plot in Fig. 2B, we can confirm our complexity estimates by analyzing the slope of the lines where the samples are distributed, i.e., solving the sparse linear system is approximately $O(n^{1.7})$ complexity,[3] whereas Coefficient-Flow displays linear complexity.

[3]Since boundary matrices are naturally sparse, and we are computing an approximate solution, the complexity can be improved beyond the aforementioned $\Omega(n^2)$.

CONCLUSION AND FUTURE WORK

While the problem of finding a bounding chain for a given cycle in a simplicial complex remains a challenging one for large complexes, we showed that this problem can be solved efficiently for codimension-1 boundaries. We implemented and tested our algorithm and have provided complexity bounds for its run-time.

However, this leaves open the question of finding bounding chains for boundaries of higher codimension, for which solving a large sparse linear system is still, to the best of our knowledge, the only feasible approach, save for codimension 2 cycles in dimension

3 (*Rodríguez et al., 2017*). In the future we would like to generalize our algorithm to be able to work with cobounding cochains (i.e., in cohomology), as well as considering the optimality question (i.e., finding the minimal bounding chain w.r.t. some cost function).

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work has been supported by the Knut and Alice Wallenberg Foundation and the Swedish Research Council. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
Knut and Alice Wallenberg Foundation.
Swedish Research Council.

Competing Interests

The authors declare there are no competing interests.

Author Contributions

- J. Frederico Carvalho conceived and designed the experiments, performed the experiments, analyzed the data, prepared figures and/or tables, performed the computation work, authored or reviewed drafts of the paper, approved the final draft.
- Mikael Vejdemo-Johansson, Danica Kragic and Florian T. Pokorny authored or reviewed drafts of the paper.

Data Availability

The following information was supplied regarding data availability:

Github: https://www.github.com/crvs/coeff-flow.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.153#supplemental-information.

REFERENCES

Boissonnat J-D, Maria C. 2014. The simplex tree: an efficient data structure for general simplicial complexes. *Algorithmica* **70(3)**:406–427.

Chambers EW, Erickson J, Nayyeri A. 2009. Minimum cuts and shortest homologous cycles. In: *Symposium on Computational geometry* DOI 10.1145/1542362.1542426.

Chambers EW, Vejdemo-Johansson M. 2015. Computing minimum area homologies. In: *Computer graphics forum, vol. 34*. Wiley Online Library, 13–21 DOI 10.1111/cgf.12514.

Chambers EW, Wang Y. 2013. Measuring similarity between curves on 2-manifolds via homotopy area. SoCG '13. In: *Proceeding of the twenty-ninth annual symposium on computational geometry*. New York: ACM, 425–434 DOI 10.1145/2462356.2462375.

Chen C, Freedman D. 2011. Hardness results for homology localization. *Discrete and Computational Geometry* **45**(**3**):425–448 DOI 10.1007/s00454-010-9322-8.

Davie AM, Stothers AJ. 2013. Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics* **143**(**2**):351–369 DOI 10.1017/S0308210511001648.

Dey TK, Hirani AN, Krishnamoorthy B. 2011. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM Journal on Computing* **40**(**4**):1026–1044 DOI 10.1137/100800245.

Di Battista G, Tamassia R. 1988. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science* **61**(**2**) DOI 10.1016/0304-3975(88)90123-5.

Edelsbrunner H, Harer J. 2010. *Computational topology: an introduction*. Providence: American Mathematical Soc.

Edelsbrunner H, Letscher D, Zomorodian A. 2002. Topological persistence and simplification. *Discrete & Computational Geometry* **28**(**4**):511–533.

Eigen. 2017. Version 3.3.2. *Available at http://eigen.tuxfamily.org/*.

Forman R. 1998. Morse theory for cell complexes. *Advances in Mathematics* **134**(**1**):90–145 DOI 10.1006/aima.1997.1650.

Gloub GH, Van Loan CF. 1996. Matrix computations. 3rd edition. London: Johns Hopkins University Press.

Hatcher A. 2002. Algebraic topology. 2nd edition. Cambridge: Cambridge University Press.

Pokorny FT, Hawasly M, Ramamoorthy S. 2016. Topological trajectory classification with filtrations of simplicial complexes and persistent homology. *International Journal of Robotics Research* **35**(**1–3**):204–223 DOI 10.1177/0278364915586713.

Rodríguez AA, Bertolazzi E, Ghiloni R, Specogna R. 2017. Efficient construction of 2-chains with a prescribed boundary. *Journal of Numerical Analysis* **55**(**3**):1159–1187 DOI 10.1137/15M1025955.

Smithsonian X. 2013. 3D: Eulaema meriana bee, Smithsonian gardens. *Available at https://3d.si.edu*.

The GUDHI Project. 2015. *GUDHI user and reference manual*. GUDHI Editorial Board *Available at http://gudhi.gforge.inria.fr/doc/latest/*.

The Stanford 3D scanning repository. 1994. *Available at https://graphics.stanford.edu/data/3Dscanrep/*.