A peer-reviewed version of this preprint was published in PeerJ on 28 May 2018.

View the peer-reviewed version (peerj.com/articles/cs-153), which is the preferred citable publication unless you specifically need to cite this preprint.

Carvalho JF, Vejdemo-Johansson M, Kragic D, Pokorny FT. 2018. An algorithm for calculating top-dimensional bounding chains. PeerJ Computer Science 4:e153 https://doi.org/10.7717/peerj-cs.153

An algorithm for calculating top-dimensional bounding chains

J. Frederico Carvalho [Corresp., 1], **Mikael Vejdemo-Johansson** [2], **Danica Kragic** [1], **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, New York, United States

Corresponding Author: J. Frederico Carvalho
Email address: jfpbdc@kth.se

We describe the \textsc{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.

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, Stocholm, Sweden.**
[2]**Mathematics Department, CUNY College of Staten Island, NY, USA.**

Corresponding author:
J. Frederico Carvalho

Email address: jfpbdc@kth.se

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.

1 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 has been recognized by Poincaré from the very 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 [1].

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 [2, 3]. 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 [4] by solving the underlying linear system.

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.

1.1 Background

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

1.1.1 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:

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

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:

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

by applying the only possible orientation-reversing permutation to $[p_0, p_2]$ gives $[p_0, p_1] + [p_1, p_2] + [p_2, p_0]$ which 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 $T^d = \{\sigma \mid \dim \sigma \leqslant d\}$ and the d–th* level *as $T^{(d)} = \{\sigma \mid \dim \sigma = d\}$.*

Given two simplices σ, τ we write $\tau \triangleleft \sigma$ 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 $\triangleleft$ 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 $T^{(d)} \neq \emptyset$.

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*

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

- *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.

1.1.2 Algebraic description

We will focus on finite geometrical 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 to S

Definition 3. *Given a simplicial complex* $S = (P,T)$ *the* chain complex *associated to* 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

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

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.

Essentially, an element in k–homology can be represented a chain $p \in C_k(S)$ such that $dp = 0$, these elements are called k–*cycles*, and two cycles p, p' are said to represent the same homology class, or to be *homologous* if there exists some $c \in C_{k+1}(S)$ such that $dc = p - p'$; in this case the cycle $p - p'$ is called a boundary.

1.2 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 $dc = p$; we call c a *bounding chain* of p. To achieve this, we further specialize the problem to

$$\begin{aligned} \text{solve: } & dc = p \\ \text{subject to: } & c_\sigma = v, \end{aligned} \tag{1}$$

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.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

The COEFFICIENT-FLOW algorithm that we present, effectively 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.

1.3 Related Work

In [5] 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 [6]. In [4] 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

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

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] [7]. This complexity quickly becomes prohibitive in case we want to handle very large complexes, such as one might find when dealing with meshes constructed from large pointclouds.

In [8] the authors address the related problem of efficiently computing a cycle p' which is homologous to a given cycle p. Which is an impressive result given that in [9] the authors prove that this cannot be done using $\mathbb{Z}_2$ coefficients.

2 METHODOLOGY

For any simplicial complex S, and any pair of simplices $\sigma, \tau \in S$ such that, $\tau \triangleleft \sigma$, we define the index of τ with respect to σ as $\langle \tau, \partial\sigma \rangle = \langle e_\tau, de_\sigma \rangle$ [10]. Note that the index corresponds to the orientation induced by the boundary of σ on τ and can be computed in $O(d)$ where d is the dimension of σ (i.e. it doesn't require one to actually perform the inner product, see Appendix A for an algorithm and proof).

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 $p_\tau = \sum_{\tau \triangleleft \omega} \langle e_\tau, d(c_\omega e_\omega) \rangle$, recall that by definition $d(c_\omega e_\omega) = \sum_{v \triangleleft \omega} \langle v, \partial\omega \rangle c_\omega e_v$; and so we get $p_\tau = \sum_{\tau \triangleleft \omega} \langle \tau, \partial\omega \rangle c_\omega e_\tau$.

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:

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

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

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

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 *as 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* (07–23) *is executed at most $O(|S^{(n-1)}|)$ times.*

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 iterates while the queue is non-empty, and a triple (σ, τ, v) can only be inserted in line (23) if τ has not been marked 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 only 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 seen the value of c_σ can never be reassigned. This is because the program branches on whether or not σ has been seen, and c_σ can only be assigned on line (14) which is bypassed if σ has been seen before. From this fact we conclude that $c_{\sigma_0} = v$ as it is assigned on line (02) and σ_0 is immediately marked as seen.

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.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

COEFFICIENT-FLOW(p, σ_0, v_0):
param: p — an $n-1$ boundary of the simplicial complex S
param: σ_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)

01: initialize c and mark every n- and $(n-1)$-cell of S as not seen.
02: $c_{\sigma_0} \leftarrow v_0$
03: initialize an empty queue Q
04: mark σ_0 as seen
05: **for each** $\tau \in \partial\sigma_0$:
06: enqueue (σ_0, τ, v) into Q.
07: **while** Q is non-empty:
08: $(\sigma, \tau, v) \leftarrow$ pop first element from Q
09: **if** σ has been marked seen:
10: **if** $v \neq c_\sigma$: the problem has no solution
11: **else**:
12: **if** τ has been marked seen: skip
13: mark τ and σ as seen
14: $c_\sigma \leftarrow v$
15: **for each** $\tau' \in \partial\sigma$:
16: **if** σ is the only coface of τ':
17: mark τ' as seen
18: **if** $p_{\tau'} \neq \langle \partial\sigma, \tau' \rangle v$: the problem has no solution
19: **else**:
20: **if** τ' has not been marked as seen:
21: $\sigma' \leftarrow$ other coface of τ'
22: $v' \leftarrow \langle \partial\sigma', \tau \rangle (p_\tau - \langle \partial\sigma, \tau \rangle v)$
23: enqueue (σ', τ', v') into Q
24: **return** c

Third, we note that for any pair (τ, σ) with $\dim\sigma = n$ and $\tau \triangleleft \sigma$, we have, either σ is the only coface of τ, or τ has a second coface, σ'. In the first case, if $p_\tau \neq c_\sigma \langle \tau, \partial\sigma \rangle$ then an error is detected on line (18). In the second case, assuming that the triple (σ, τ, v) is enqueued before (σ', τ, v') (for some v, v') we then have that $v' = \langle \partial\sigma', \tau \rangle (p_\tau - \langle \partial, \sigma \rangle v)$ as is assigned in line (22) 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 must be true for every pair $\tau \triangleleft \sigma$, then it must be the case that $dc = p$. If this is not the case, then there will be an error in line (10) and the algorithm will abort. □

Note that the connectivity condition can be removed if we instead require a value

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

for one cell in each connected component of the graph $G(S)$ and, 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
01: **for each** $\tau \in S^{n-1}$:
02: **if** τ has a single coface:
03: $\sigma \leftarrow$ the coface of τ
04: $v \leftarrow \langle \partial\sigma, \tau \rangle p_\tau$
05: break loop
06: $c \leftarrow$ COEFFICIENT-FLOW(p, σ, v)
07: **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.*

2.1 Implementation details

We will now discuss some of the choices made in our implementation of the COEFFICIENT-FLOW algorithm[2]. Before we can even tackle 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 [11] provided by the GUDHI library [12] 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, as 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 to p_{i_0} and so on, which in the worst case scenario is in $O(\dim \sigma |S^{(0)}|)$. Thus we need to

[2] available in `https://www.github.com/crvs/coeff-flow`

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

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 [13] 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 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 (09) and (12) 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 (15), 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 (15–23) is executed $\dim \sigma$ times, where $\dim \sigma = d$.
- The loop (15–23) 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 $O(d^2|S^{(d-1)}|)$ □

2.2 Example runs and tests

In Figure 1 we provide an example of the output of the COEFFICIENT-FLOW algorithm for the mesh of the Stanford bunny [14] and the *eulaema meriana* bee model from the Smithsonian 3D model library [15],

For comparison, we performed the same experiments using the Eigen linear algebra library [16] to solve the underlying linear system[3], 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.

In order to draw further conclusions about the run-time, and to properly quantify the improvement over the linear system solution, we performed randomized tests on both implementations. In this scenario we made a mesh on 2–dimensional 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 Figure 2 from

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

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

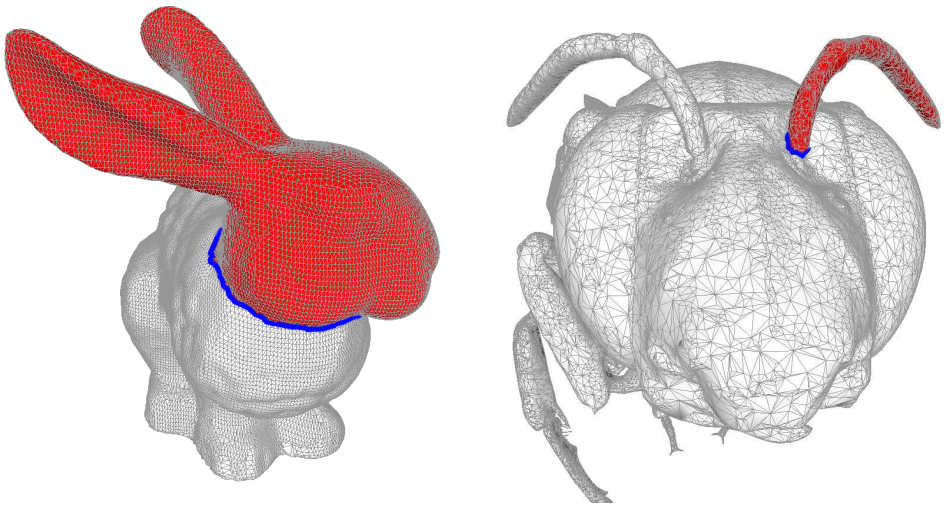

Figure 1. Examples of bounding chains: The edges in blue form cycles in the mesh, and the faces in red form the corresponding bounding chain as computed by the COEFFICIENT-FLOW algorithm.

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)**

Table 1. Timing for computation of bounding chains using COEFFICIENT-FLOW, and using Eigen in several meshes. "Bee (Sample)/Bee (Full)" and "Bunny" refer to the meshes in Figure 1 left and right respectively. The mesh "Bee (Full)" is the one obtained from [15], whereas the one labeled "Bee (Sample)" is a sub-sampled version of it.

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 Figure 2 (right), 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[4], whereas COEFFICIENT-FLOW displays linear complexity.

3 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 still leaves open the question of finding bounding chains for boundaries of higher codimension, for which solving a large sparse linear system is still the only

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

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

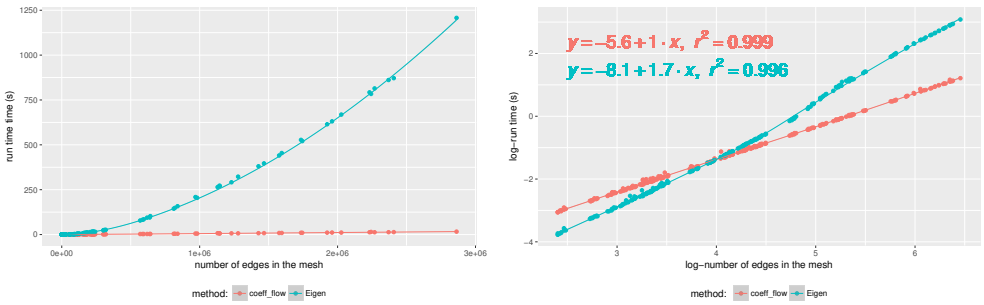

Figure 2. Running times for a calculating the bounding chain of a cycle as a function of the number of edges (left), and Log-log plot of the same data (right).

feasible approach. 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).

Acknowledgements: This work has been supported by the Knut and Alice Wallenberg Foundation and the Swedish Research Council.

REFERENCES

[1] H. Edelsbrunner and J. Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[2] Florian T. Pokorny, Majd Hawasly, and Subramanian Ramamoorthy. Topological trajectory classification with filtrations of simplicial complexes and persistent homology. *International Journal of Robotics Research (IJRR)*, 2016.

[3] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological Persistence and Simplification. *Discrete & Computational Geometry*, 28(4), November 2002.

[4] E. W. Chambers and M. Vejdemo-Johansson. Computing minimum area homologies. In *Computer Graphics Forum*, volume 34. Wiley Online Library, 2015.

[5] E. W. Chambers and Y. Wang. Measuring Similarity Between Curves on 2-manifolds via Homotopy Area. SoCG '13, pages 425–434. ACM, may 2013.

[6] A. Hatcher. *Algebraic topology*. Cambridge University Press, 2 edition, 2002.

[7] A. M. Davie and A. J. Stothers. Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, (2), 2013.

[8] T. K. Dey, A. N. Hirani, and B. Krishnamoorthy. Optimal Homologous Cycles, Total Unimodularity, and Linear Programming. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, STOC '10, New York, NY, USA, 2010. ACM.

[9] Chao Chen and Daniel Freedman. Hardness Results for Homology Localization. *Discrete and Computational Geometry*, 45(3), 2011.

[10] R. Forman. Morse Theory for Cell Complexes. *Advances in Mathematics*, 134(1), March 1998.

[11] J.-D. Boissonnat and C. Maria. The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes. In *Algorithms – ESA 2012*, Lecture Notes in Computer Science. 2012.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017

[12] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. URL `http://gudhi.gforge.inria.fr/doc/latest/`.

[13] Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2), 1988.

[14] The Stanford 3D scanning repository. URL `https://graphics.stanford.edu/data/3Dscanrep/`.

[15] Smithsonian X 3D: Eulaema meriana bee, Smithsonian gardens. URL `https://3d.si.edu`.

[16] Eigen. URL `http://eigen.tuxfamily.org/`.

Appendix

A COMPUTING $\langle \tau, \partial\sigma \rangle$

The following algorithm computes $\langle \tau, \partial\sigma \rangle$ by comparing each vertex of τ and σ at most once.

```
INDEX(σ, τ):
    param: σ – k-simplex represented as a sorted list of indices of points.
    param: τ – (k − 1)-face of σ represented as a sorted list of indices.
for each  i ← 0 . . . dim τ :
  if  τ_i ≠ σ_i:
      orientation ← (−1)^i
      break loop
return orientation
```

By inspecting the main loop we can see that $\textsc{Index}(\sigma, \tau)$ returns $(-1)^i$ where i is the index of the first element where σ and τ differ. Since we assume that τ is indeed a top dimensional face of σ, then if $\sigma = [s_{0:d}]$ by definition τ is of the form $[s_{0:(i-1)}, s_{(i+1):d}]$ 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 first element at which τ and σ differ.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3151v1 | CC BY 4.0 Open Access | rec: 14 Aug 2017, publ: 14 Aug 2017