Tracking a Generator by Persistence

Oleksiy Busaryev* Tamal K. Dey† Yusu Wang‡

Abstract

The persistent homology provides a mathematical tool to describe "features" in a principled manner. The persistence algorithm proposed by Edelsbrunner et al. [9] can compute not only the persistent homology for a filtered simplicial complex, but also representative generating cycles for persistent homology groups. However, if there are dynamic changes either in the filtration or in the underlying simplicial complex, the representative generating cycle can change wildly.

In this paper, we consider the problem of tracking generating cycles with temporal coherence. Specifically, our goal is to track a chosen essential generating cycle so that the changes in it are "local". This requires reordering simplices in the filtration. To handle reordering operations, we build upon the matrix framework proposed by Cohen-Steiner et al. [6] to swap two consecutive simplices, so that we can process a reordering directly. We present an application showing how our algorithm can track an essential cycle in a complex constructed out of a point cloud data.

1 Introduction

The notion of prominent feature in a geometric shape plays important role in various scientific studies and engineering applications. The persistent homology introduced by Edelsbrunner et al. [9] provides a mathematical tool along with a combinatorial algorithm that aids capturing these features in a principled manner. Various aspects of persistent homology have been studied including its extension to more generalized spaces and filtrations [2, 5, 8, 12], its stability against noise [1, 4] and maintenance under changes in the filtration [6]. In this paper, we elaborate upon the last aspect, that is, maintaining persistence and a persistent generator when the simplicial complex and its filtration changes.

Let $\mathcal{K}$ be a simplicial complex. A sequence of its simplices in non-decreasing order of their dimensions induces a *filtration* of $\mathcal{K}$. A persistent pairing of simplices can be obtained from a filtration following the persistence algorithm [9]. A simplex is a *creator* if it creates a new homology class and it is a *destroyer* if it kills a homology class. A destroyer pairs with a creator if it kills a class created by the creator. Each destroyer pairs with a unique creator. The unpaired creators get associated with homology classes that survive and represent essential classes in the homology of $\mathcal{K}$. Given a filtration, the persistence algorithm computes the pairing and also the representative cycles of the classes that survive. If the filtration changes, the persistent pairing changes - so do the essential cycles. Similar phenomena happen when $\mathcal{K}$ changes with insertion and deletion of simplices.

*Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210, USA. Email: busaryev@cse.ohio-state.edu

†Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210, USA. Email: tamaldey@cse.ohio-state.edu

‡Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210, USA. Email: yusu@cse.ohio-state.edu

Our main goal is to track a chosen essential generator under insertions and deletions of simplices. This also requires reordering of simplices in the filtration. Such operations have been considered before [6, 1]. Specifically, Cohen-Steiner et al. [6] gave an efficient algorithm to swap two consecutive simplices in a filtration and maintain persistent pairing using a matrix view of the persistence. Insertions and deletions can be implemented using such swaps. We observe that reordering (and thus insertions and deletions) can be implemented more directly avoiding repeated swaps. Although this does not improve the worst-case time analysis, it indeed saves computing time in practice as our empirical results confirm.

The essential cycle that we track is kept associated with the first creator in the filtration that remains unpaired. With each update, we make necessary changes in this cycle. However, if the persistence algorithm is executed, this cycle may change wildly. We show how to maintain all invariants necessary to detect persistent pairs and still make only "local" changes to the cycle being tracked.

2 Background on Persistence

In this section, we introduce necessary background See [10] for more information on algebraic topology, and [11] for more discussion on persistent homology.

Simplicial homology. Let $\mathcal{K}$ be a simplicial complex. Under $\mathbb{Z}_2$ coefficients, a *p-chain* is a subset of p-simplices of $\mathcal{K}$. The set of p-chains, together with modulo 2 addition, forms a free abelian group C_p called the *p-th chain group* of $\mathcal{K}$. The *boundary* $\partial_p(\sigma)$ of a k-simplex σ is the set of its $(p-1)$-faces, and that of a p-chain is the modulo 2 addition of the boundaries of its simplices. The boundary operator defines a homomorphism $\partial_p : \mathsf{C}_p \to \mathsf{C}_{p-1}$.

Now define a *p-cycle* to be a p-chain with empty boundary and a *p-boundary* to be a p-chain in the image of ∂_{p+1}. The spaces of p-cycles and p-boundaries are called the *p-th cycle group* $\mathsf{Z}_p = \ker \partial_p$ and the *p-th boundary group* $\mathsf{B}_p = \operatorname{im} \partial_{p+1}$, respectively, and they are both subgroups of C_p. The *p-th homology group* is the quotient group $\mathsf{H}_p(\mathcal{K}) = \mathsf{Z}_p/\mathsf{B}_p$. Elements of H_p are the *homology classes* $c+\mathsf{B}_p = \{c+b \mid b \in \mathsf{B}_p\}$ for p-cycles c; c is also referred to as a *generating cycle* of the homology class $[c] = c+\mathsf{B}_p$. Two p-cycles c and d are *homologous* if $[c] = [d]$, that is, $c+d \in \mathsf{B}_p$ is the boundary of some $(p+1)$-chain.

The rank of H_p is called the *p-th Betti number* of $\mathcal{K}$, denoted by $\beta_p = \operatorname{rank} \mathsf{H}_p$. A *basis* of H_p is a minimal set of homology classes that generates H_p. A set of p-cycles $C = \{c_1, \ldots, c_l\}$ *generates* H_p, if $\{[c_i]\}$ forms a basis for H_p; $|C| = \beta_p$.

Persistent homology. A *filtration* of the simplicial complex $\mathcal{K}$ is a sequence of nested subcomplexes $\mathcal{K}_0 \subset \mathcal{K}_1 \subset \cdots \subset \mathcal{K}_m = \mathcal{K}$. Consider the homomorphism $h_p^{i,j} : \mathsf{H}_p(\mathcal{K}_i) \hookrightarrow \mathsf{H}_p(\mathcal{K}_j)$ induced by the inclusion $\mathcal{K}_i \subset \mathcal{K}_j$. We say that a homology class $h \in \mathsf{H}_p(\mathcal{K}_r)$ was *created* at time $i \leq r$ if it does not have a preimage in $\mathsf{H}_p(\mathcal{K}_{i-1})$ under $h_p^{i-1,r}$, but has a preimage in $\mathsf{H}_p(\mathcal{K}_i)$ under $h_p^{i,r}$. A homology class $h \in \mathsf{H}_p(\mathcal{K}_r)$ is *destroyed* at time $j > r$ if its image in $\mathsf{H}_p(\mathcal{K}_{j-1})$ is non-trivial, but its image in $\mathsf{H}_p(\mathcal{K}_j)$ is trivial. If h is created at i and destroyed at j, then it has *persistence* $j-i$ and produces a *persistence pairing* (i, j).

An ordering $(\sigma_1, \ldots, \sigma_m)$ of simplices in $\mathcal{K}$ induces a filtration Π such that $\mathcal{K}_i = \mathcal{K}_{i-1} \cup \{\sigma_i\}$. We also write $\Pi = (\sigma_1, \sigma_2, \ldots, \sigma_m)$. A valid filtration requires that each simplex has a larger index than any of its faces.

Matrix view. Consider a simplicial complex $\mathcal{K}$ and a filtration $\Pi = (\sigma_1, \sigma_2, \ldots, \sigma_m)$. Let D denote the boundary matrix for $\mathcal{K}$; i.e., $D[i][j] = 1$ if $\sigma_i \in \partial\sigma_j$, and $D[i][j] = 0$ otherwise. Persistent pairing can be computed by the matrix reduction approach [6, 9]. Since we use $\mathbb{Z}_2$ coefficients, matrices we consider have entries only 0 or 1, and we use modulo 2 arithmetic during matrix multiplications.

Given any matrix M, let $\mathrm{row}_M[i]$ and $\mathrm{col}_M[j]$ denote the i-th row and j-th column of M, respectively. We abuse the notation slightly to let $\mathrm{col}_M[j]$ denote also the chain $\{\sigma_i \mid M[i][j] = 1\}$. Let $\mathrm{low}_M[j]$ denote the row index of the last 1 in the j-th column of M, which we call the *low-row index* of the column j. It is undefined for empty columns. The matrix M is *reduced* (or is in *reduced form*) if $\mathrm{low}_M[j] \neq \mathrm{low}_M[j']$ for any $j \neq j'$; that is, no two columns share the same low-row indices. We define a matrix M to be *upper-triangular* if all of its diagonal elements are 1, and there is no entry $M[i][j] = 1$ with $i > j$.

Proposition 2.1 ([6]) *Let $R = DV$, where R is reduced and V is upper-triangular. Then the simplices σ_i and σ_j form a persistent pair if and only if $\mathrm{low}_R[j] = i$.*

Notice that there are many R and V for a fixed D forming the decomposition as described above. Proposition 2.1 implies that the persistent pairing is independent of the contents of R and V. Furthermore, if c_j is the p-chain corresponding to the column $\mathrm{col}_V[j]$, then $\mathrm{col}_R[k]$ corresponds to the $(p-1)$-chain ∂c_j.

Proposition 2.2 *Let c_j and c'_j be the p- and $(p-1)$-chains corresponding to the columns $\mathrm{col}_V[j]$ and $\mathrm{col}_R[j]$ respectively where $R = DV$. Then, $c'_j = \partial_p(c_j)$.*

3 Efficient Updates

In this section, we describe how to perform reordering of simplices. Given a filtration $\Pi = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, we consider basic operations MoveRight(i, j) and MoveLeft(i, j), which move σ_i to the j-th position, with $i < j$ and $i > j$, respectively. Insertion and deletion can be implemented using these operations.

To handle these movements, we use the framework from [6] to perform a *transposition*, which is MoveRight$(i, i+1)$ or MoveLeft$(i, i-1)$. It is equivalent to continuously swapping neighboring simplices till σ_i is moved to position j. However, handling the movement directly enables us to remove certain overhead. Instead of scanning the elements of a certain row $j - i$ times (once per transposition[1]), we only scan it once for the entire operation. We present experimental results at the end of this section showing the practical improvement.

When the complex $\mathcal{K}$ or its filtration changes, we can keep track of the persistence pairs by maintaining the matrices R and V which allow us to apply Propositions 2.1 and 2.2. This means we need to maintain the following invariants:

Invariants. Let R and V be the two matrices after each update [2].

I1. $R = DV$ where D is the boundary matrix for $\mathcal{K}$ with respect to filtration.

I2. R is in reduced form, and V is upper-triangular.

[1] Such a scanning is necessary due to the sparse matrix representation.

[2] We use the decomposition of the form $R = DV$ instead of $RU = D$ from [6].

If I1 and I2 are maintained, we can read the persistent pairs by Proposition 2.1. Moreover, for σ_i that is a creator, $\mathrm{col}_V[i]$ represents a cycle created by σ_i. If σ_i is not paired, then $\mathrm{col}_V[i]$ represents a non-trivial homology class of $\mathcal{K}$.

Implementation of matrix updates involves column additions. To maintain invariant I1, whenever we add two columns in R, we add them in V. To maintain V upper-triangular, we always add a column to a column on its right.

3.1 Moving Right

This operation moves a simplex σ_i to the j-th position with $j > i$. To reflect this change, we should move the i-th column and row of D, R, and V, to the j-th position. Let $\widehat{D}, \widehat{R}$, and $\widehat{V}$ be the resulting matrices. They still satisfy $\widehat{R} = \widehat{D}\widehat{V}$; however, $\widehat{V}$ may not be upper-triangular and $\widehat{R}$ may not be reduced.

Moving a column in V to the right does not destroy its upper-triangular property; however, moving a row down may destroy it. Specifically, $\mathrm{row}_{\widehat{V}}[j]$ may now contain 1's for indices in $[i, j-1]$. Similarly, moving down a row in R may destroy its reduced form. In both cases we can perform a similar procedure which we call RESTORE to restore the desired properties.

Restoring upper-triangular property. Consider the original matrices $R = DV$. Let $I_1 = i$ and let $\{I_2, \ldots, I_s\}$ be the set of indices within $[i+1, j]$ that have entry 1 in $\mathrm{row}_V[i]$. To keep V upper-triangular after we move $\mathrm{row}_V[i]$ to the j-th position, we need to cancel these 1's. To do this, we can add $\mathrm{col}_V[i]$ to each of $\mathrm{col}_V[I_l]$ for $1 < \ell \leq s$. Adding columns in V induces the same additions in R to maintain $R = DV$, which may destroy its reduced form. Fortunately, it turns out that we can maintain R reduced for all columns except possibly $\mathrm{col}_R[i]$.

Let $\rho_l = \mathrm{low}_R[I_l]$. Consider adding column I_1 to I_2 in both V and R to obtain V' and R'. Depending on whether $\rho_1 > \rho_2$ or not, one of them becomes the low-row index of $\mathrm{col}_{R'}[I_2]$ after addition, and the other one becomes a *free low-row index* which can potentially become the low-row index of some other column later. We call the column from R whose low-index now becomes free the *donor column* and its index the *donor index* i_d. Now, to cancel $V'[i][I_3]$, instead of adding $\mathrm{col}_V[i]$ to $\mathrm{col}_{V'}[I_3]$, we add the column $\mathrm{col}_V[i_d]$ in V to $\mathrm{col}_{V'}[I_3]$. This addition can still remove the entry 1 from the i-th element in $\mathrm{col}_{V'}[I_3]$ while not creating any entry 1 in $\mathrm{col}_{V'}[I_3]$ for row indices larger than I_3. This means V' remains upper-triangular. The advantage of adding the donor column is that the matrix R' stays reduced except possibly for the i-th column. We repeat this procedure to cancel each entry 1 in the row $\mathrm{row}_V[i]$ for indices in $\{I_2, \ldots, I_s\}$, maintaining the donor index at each iteration. The entire algorithm is summarized below. It returns new matrices R and V, and also the last donor columns in both R and V which can potentially contribute a free low row index.

Algorithm 1 RESTORE($R, V, \mathbb{I} = \{I_1, \ldots, I_s\}$)

$d\_id \leftarrow I_1$; $d\_low \leftarrow \mathrm{low}_R[d\_id]$; $d\_col_R \leftarrow \mathrm{col}_R[d\_id]$; $d\_colV \leftarrow \mathrm{col}_V[d\_id]$
for $k \leftarrow 2$ to s **do**
  $new\_d\_low \leftarrow \mathrm{low}_R[I_k]$; $new\_d\_colR \leftarrow \mathrm{col}_R[I_k]$; $new\_d\_colV \leftarrow \mathrm{col}_V[I_k]$
  $\mathrm{col}_V[I_k] \leftarrow \mathrm{col}_V[I_k] + d\_colV$; $\mathrm{col}_R[I_k] \leftarrow \mathrm{col}_R[I_k] + d\_colR$
  if $d\_low > new\_d\_low$ **then**
    $d\_id \leftarrow I_k$; $d\_low \leftarrow new\_d\_low$; $d\_colR \leftarrow new\_d\_colR$; $d\_colV \leftarrow new\_d\_colV$
  end if
end for
return (R, V, d_colR, d_colV)

Restoring reduced form. Let R_1 and V_1 be the matrices returned by RESTORE$(R, V, \mathbb{I})$. If we now move the i-th column and row of matrices R_1, D, V_1 to the j-th position, the new decomposition $\widehat{R} = \widehat{D}\widehat{V}$ guarantees that $\widehat{V}$ is upper-triangular. But $\widehat{R}$ may not be reduced. In particular, (1) the new column $\mathrm{col}_{\widehat{R}}[j]$ may not be reduced since we did not reduce $\mathrm{col}_R[i]$ in RESTORE; and (2) the row $\mathrm{row}_{\widehat{R}}[j]$ corresponding to the simplex σ that we moved may render several columns in $\widehat{R}$ having the same low-row index j. For (1), it turns out that we can simply replace $\mathrm{col}_{\widehat{R}}[j]$ with $\mathrm{col}_R[i_d]$, where i_d is the last donor index after performing the procedure RESTORE above. This will give a unique low-row index to $\mathrm{col}_{\widehat{R}}[j]$ (by the definition of the donor column), and $\mathrm{col}_{\widehat{V}}[j] = \mathrm{col}_V[i_d]$ is a valid column in V. For (2), observe that the indices of those columns having low-row index j in $\widehat{R}$ are $\mathbb{J} = \{J_1, \ldots, J_t\}$ such that for each $l \in [1, t]$, the original low-row index $\mathrm{low}_R[J_l]$ falls inside the range $[i, j]$ and $\mathrm{row}_R[i][J_l]$ (i.e., $R[i][J_l]$) is 1. To resolve these conflicts, we call exactly the same algorithm RESTORE, for matrices R_1, V_1 before moving the i-th columns and rows, but with the set $\mathbb{J}$ substituting the role of $\mathbb{I}$. The entire algorithm for handling a right-movement operation is summarized below.

Algorithm 2 MOVERIGHT(R, V, i, j)

1: compute $\mathbb{I} = \{I_1, \ldots, I_s\}$
2: compute $\mathbb{J} = \{J_1, \ldots, J_t\}$
3: compute $(R_1, V_1, d\_colR, d\_colV) \leftarrow$ RESTORE$(R, V, \mathbb{I})$
4: compute $(R_2, V_2, d\_colR', d\_colV') \leftarrow$ RESTORE$(R_1, V_1, \mathbb{J})$
5: Let $\hat{R}$ and $\hat{V}$ denote matrices after moving the i-th column and row of R_2 and V_2 to the j-th column and row, respectively
6: $\mathrm{col}_{\widehat{R}}[j] \leftarrow d\_colR$; $\mathrm{col}_{\widehat{V}}[j] \leftarrow d\_colV$

Proposition 3.1 *Suppose we move a p-simplex from the i-th position to the j-th position in a filtration of a simplicial complex $\mathcal{K}$ that has n simplices of dimensions $p-1$, p, and $p+1$. Let $R = DV$ where invariants I1 and I2 hold under the given filtration, and $\widehat{R}$ and $\widehat{V}$ be the matrices obtained by* MOVERIGHT(R, V, i, j). *Then, I1 and I2 hold for the new matrices $\widehat{R} = \widehat{D}\widehat{V}$. The algorithm runs in $O(kn)$ time, where $k = (s + t) \leq 2(j - i)$ and s, t are as described above.*

Proof: The main purpose of RESTORE is to maintain the invariants I1 and I2. The number of columns restored in two calls of RESTORE is exactly equal to k. To determine the columns to be restored we need to scan the entries in the i-th row of R corresponding to the $(p+1)$-simplices and the i-th row in V corresponding to the p-simplices. This cannot take more than $O(n)$ time. The complexity of RESTORE is determined by the number of columns it restores multiplied by the time it takes to add two columns. Since we are moving a p-simplex, this column addition can remain restricted to the rows corresponding to the $(p-1)$- and p-simplices in R and p-simplices in V. The claimed time bound follows. ∎

3.2 Other Operations

MOVELEFT(i, j) can be handled by a procedure somewhat dual to MOVERIGHT(i, j) in same time complexity, and we omit the description here. The following property of MOVERIGHT and MOVELEFT is used in justifying our tracking algorithm.

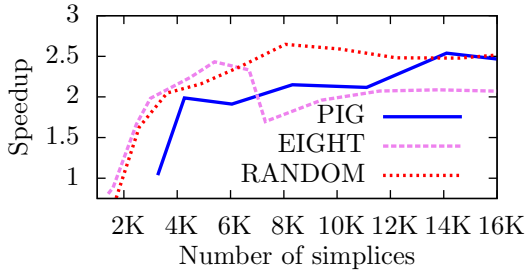
Claim 3.2 *Operation* MOVERIGHT(i, j) *cannot change a destroyer to a creator other than the simplex being moved. Operation* MOVELEFT(i, j) *cannot change a creator to a destroyer other than the simplex being moved. None of the operation changes the status of a simplex whose index is not between i and j.*

We can use MoveLeft and MoveRight to handle insertions and deletions. The routine InsertAt(σ, i) inserts a simplex σ at location i, and can be implemented by first inserting it at the end of the filtration, and then moving it to the i-th location. The routine DeleteAt(i) deletes the i-th simplex from the filtration, and can be achieved by first moving it to the end of the filtration, and then removing it.

3.3 Experiments on Reordering

We use a sparse matrix representation based on unsorted lists to represent matrices R and V. The advantage of using this data structure is that its size is proportional to the number of 1's in each matrix, instead of m^2 which is usually much larger (m is the number of simplices in the input complex). One disadvantage is however, each row is not stored in a linear order, hence searching for an element in it usually requires scanning of this row. The main advantage of our MoveRight(i, j) and MoveLeft(i, j) algorithms is that this cost is needed only once, instead of $j - i$ times when using the transposition algorithm from [6]. Take MoveRight(i, j) as an example. Although we need to move σ_i over $j - i$ columns, we spend linear time only for those columns that will potentially be affected by this movement (which are the columns indexed by $\mathbb{I}$ and $\mathbb{J}$ in the procedure MoveRight), and constant time for other columns. Handling the movement using $j - i$ transpositions, however, will require linear time for every transposition in order to check whether current swapping will affect any column.

Model	CEM	Our	Speedup
Random	27.2s	11.5s	2.19
Eight	30.3s	15.0s	1.85
Pig	33.0s	15.1s	2.02



We tested our algorithm on three point cloud models, including a randomly generated set of 1000 points. For each model we produced 12 simplicial complexes of variable sizes; for each complex, 10 scenarios of 10000 random moves were generated. In the table above we show the running time for each model as well as the speedup relative to the algorithm from [6] denoted with acronym CEM. The speedup is computed as $\frac{t_{CEM}}{t_{our}}$. Running times and speedup values were separately averaged over 120 test cases.

4 Tracking a Generator

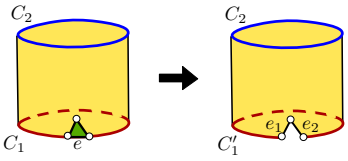
We consider the problem of tracking an essential generator in a simplicial complex under insertions and deletions of simplices. Precisely, if we are tracking a p-cycle G in a simplicial complex $\mathcal{K}$, we require that $[G]$ is not zero in $\mathsf{H}_p(\mathcal{K})$. This means we should be able to detect if G remains essential or $[G]$ exists.

The class $[G]$ may become non-essential by insertion of a $(p+1)$-simplex. Essentiality of G can be detected using matrices R and V. Detecting whether the class $[G]$ disappears is more subtle. Of course, this happens only with deletions of simplices. If a simplex which is not in G is deleted, $[G]$ is not affected. Now consider the case when a simplex $\alpha \in G$ is being deleted. We cannot delete α if it has co-faces; so, we assume that α has no co-faces at the time of its deletion. But in that case $[G]$ is always destroyed. Therefore, any deletion of a simplex from G would destroy its class. To

overcome this technical difficulty, we define existence of a class under deletion differently: instead of deleting a simplex, we delete a simplex and all its co-faces and check whether $[G]$ survives or not.

In particular, let $\widehat{\mathcal{K}}$ be the complex obtained by deleting α and its co-faces in $\mathcal{K}$ where $\alpha \in G$. Consider the map $i_p : \mathsf{H}_p(\widehat{\mathcal{K}}) \hookrightarrow \mathsf{H}_p(\mathcal{K})$ induced by the inclusion $\widehat{\mathcal{K}} \subset \mathcal{K}$. The homology class $[G]$ in $\mathsf{H}_p(\mathcal{K})$ may have a pre-image in $\mathsf{H}_p(\widehat{\mathcal{K}})$ under i_p, in which case we say that the class $[G]$ *exists* in $\mathsf{H}_p(\widehat{\mathcal{K}})$. $[G]$ does not exist in $\mathsf{H}_p(\widehat{\mathcal{K}})$ if and only if α did not have any $(p+1)$-dimensional co-face in $\mathcal{K}$. In this case, $[G]$ is destroyed and the tracking of G ends.

A motivating application. In many applications deploying scanning technology, the shape is often represented by a discrete sample. Inference of geometry and topology from such data is a topic of active research. For topological inferences, often a data structure called *Rips complex* is built on top of this discrete data. Essential generators in the Rips complex approximate the essential generators in the sampled space [7]. Therefore, if we track such a generator, say under motion of the vertices, we have an idea of the deformation of the true generator in the underlying space. For a set of points $P = \{p_1, p_2, \ldots, p_n\}$, the Rips complex $\mathcal{R}^\varepsilon(P)$ is defined as the collection of all simplices whose vertices have pairwise distances at most ε. Obviously, the topological change in $\mathcal{R}^\varepsilon(P)$ occurs only at *critical events* when the distance between two vertices p_i, p_j changes from larger to smaller than ε or vice versa. The former corresponds to an *insertion* event that inserts the edge $e = p_i p_j$, as well as all the higher-dimensional simplices it induces. The latter corresponds to a *deletion* event that deletes the edge $e = p_i p_j$ together with all its co-faces. This is why we consider the tracking of a cycle with respect to insertion and deletion of a simplex together with its co-faces.



Suppose e has k co-faces either before its deletion or after its insertion. Then using the reordering operations described above, we can maintain a set of generating cycles of any dimension p in $O(km^2)$ time, where $m = |\mathcal{K}|$. However, during the updates, generating cycles can change wildly. See the right figure for an example. The initial generating cycle for a cylinder is C_1. However, after we remove the edge e and its incident triangle t, the creator of the new cycle C_1' may change to say e_1, which may come later than all edges in the cycle C_2. In this case, the generating cycle will suddenly change to C_2.

We wish to track a given input generating cycle G so that there is good temporal coherence in the cycle we produce (the meaning of this is made more precise later). We also keep the generator G *simple*. A p-cycle C is simple if one cannot decompose it into two independent p-cycles C_1 and C_2 with $C = C_1 + C_2$. Simplicity of cycles is often required in graphics and visualization applications.

4.1 Tracking Invariant

The basic operation of the dynamic updates is the following: given a simplicial complex $\mathcal{K}$, we can either insert a p-simplex α and all of its faces if that are not already in $\mathcal{K}$, or remove α together with all its co-faces. We track a p-cycle G as follows: we maintain a decomposition $R = DV$ for the current simplicial complex, and the tracked cycle we return is always the chain $G = \mathrm{col}_V[\theta]$, where θ is the first p-dimensional creator in the filtration. We call this cycle the *first p-cycle*. We guarantee that G is essential and simple throughout the updates, and its change after insertions or deletions is "local" in some sense. If G becomes non-essential, its tracking ends. The following observation provides some intuition behind the choice of using the first p-cycle as the essential cycle we are tracking.

Proposition 4.1 *Let $G = \mathrm{col}_V[\theta]$ be the cycle created by the first p-dimensional creator θ in Π. G must be simple, and $[G]$ is essential if and only if θ is unpaired.*

Proof: If G is not simple, the youngest simplex in one of its subcycles not containing θ is a creator, contradicting that θ is the first p-dimensional creator.

If θ is unpaired, then obviously $[G]$ is essential. To see the opposite direction, assume that $[G]$ is essential, but θ is paired with a $(p+1)$-simplex τ. Then, there is a p-cycle G' which is the chain stored in $\mathrm{col}_R[\tau]$, such that θ is its youngest simplex. The cycle G' cannot be homologous to G, as $[G']$ is trivial in $\mathcal{K}$ while $[G]$ is essential. Hence $G' \neq G$ and $G' + G$ is a p-cycle containing simplices all older than θ. In this case the youngest simplex of $C' + C$ is necessarily a creator, but θ cannot be the first p-creator, reaching a contradiction. ■

4.2 Insertions and Deletions

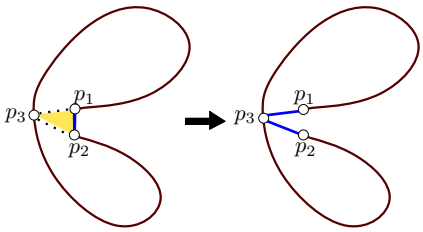
A simplex α is always inserted at the end of the filtration. This adds one rightmost column and one bottom row to the matrix D. The new rightmost column can be reduced in $O(m^2)$ time, and no other columns/rows in R are affected. Hence θ is still the first p-creator, and $\mathrm{col}_V[\theta]$ (i.e. G) remains the same. However, if the dimension of α is $p+1$, α may be paired with θ. In this case, $[G]$ is trivial in the new complex, and the tracking of G ends.

Claim 4.2 *We can update the reduced matrix decomposition after inserting a simplex α and its k co-faces in $O(km^2)$ time. Either $[G]$ is trivial and the tracking terminates, or we still return G as the new generator.*

Tracking G after deletion is much more involved. We delete simplices in decreasing order of dimensions. After deletion of each simplex, we update the matrices R and V using DeleteAt operation. The status of the p-cycle G and θ can potentially be affected only when deleting p-simplices. However, whether $[G]$ is destroyed or not depends also on the $(p+1)$-simplices deleted. Hence from now on, we discuss only the basic case of deleting a p-simplex α and its $(p+1)$-dimensional co-faces. For deleting simplices of dimension other than p and $p+1$, there is no additional operations required other than the standard DeleteAt operations. Now let $\mathcal{K}$ and $\widehat{\mathcal{K}}$ denote the simplicial complex before and after deleting the p-simplex α and its $(p+1)$-dimensional co-faces, respectively.

If $\alpha \notin G$, then the deletion of it changes neither the status nor the composition of G and θ (this is not true in general, but holds for the first p-creator θ). We simply delete α and its co-faces, and update R and V for $\widehat{\mathcal{K}}$ appropriately.

If α is contained in the cycle G, and α did not have any $(p+1)$-dimensional co-faces in $\mathcal{K}$, we observe that $[G]$ is destroyed and the tracking of G ends.



$[G]$ exists after deletion. The remaining case is that there exists at least one $(p+1)$-simplex τ incident to α in $\mathcal{K}$, and thus $[G]$ exists in $\mathsf{H}_p(\widehat{\mathcal{K}})$. Consider the cycle $G' = G + \partial\tau$ which is homologous to G in $\mathcal{K}$. Observe that the cycle G' exists in $\widehat{\mathcal{K}}$, $[G']$ is necessarily non-trivial in $\mathsf{H}_p(\widehat{\mathcal{K}})$ as the pre-image of $[G]$ under the map i_p, and the change between G and G' is very local (differs by at most p simplices). Our goal is thus to return G' as the new generator $\widehat{G}$ as long as it is simple. If, however, G' is not simple anymore, then we aim to identify a simple sub-cycle $\widehat{G}$ of G' such that $[\widehat{G}]$ is still non-trivial. Intuitively, this generating cycle is still close to the previous one. Unfortunately, we cannot expect always that both $\widehat{G}$ is simple and $[\widehat{G}] = [G'](= [G])$, as such a $\widehat{G}$ may not exist. See the figure for an example

where we are to delete the edge p_1p_2 and its only co-face is the triangle $p_1p_2p_3$. After the deletion, the only representative cycle of the class $[G]$ has two sub-cycles and is thus not simple. We now describe how these goals are achieved.

First, delete all co-faces of α, including τ, using the algorithm from Sec. 3. Now we handle the deletion of α. Let $\sigma_0 = \alpha, \ldots, \sigma_r$ be the set of p-faces of τ that are contained in G, and $\beta_1, \ldots, \beta_u$ the remaining set of p-faces of τ. We first perform the following: (1) move $\beta_1, \ldots, \beta_u$ immediately after θ (to the right of θ); (2) delete $\sigma_0 (= \alpha)$ and move $\sigma_1, \ldots, \sigma_r$ to the right of β_u. This can be completed in $O((r+u)m^2) = O(pm^2)$ time using operations from Sec. 3. Let $R_1 = \widehat{D}V_1$ be the resulting decomposition of the new filtration Π_1. Obviously, β_u is necessarily a creator, as it creates the cycle G' $(= G + \partial\tau)$ that cannot exist before β_u is introduced. However, note that G' is not necessarily the cycle stored at $\mathrm{col}_{V_1}[\beta_u]$ at this point. Specifically, β_u may not be the first p-dimensional creator. One or more β_is for $1 \leq i < u$ may become creators (by Claim 3.2, all other simplices older than θ necessarily remain as destroyers). Suppose β_j is the first creator in Π_1 and consider the cycle $\mathrm{col}_{V_1}[\beta_j]$ that it creates.

case i. If there is a simplex $\gamma \in \mathrm{col}_{V_1}[\beta_j]$ that is not in G', then we move γ to the right of β_u. This renders γ a creator (as it creates $\mathrm{col}_{V_1}[\beta_j]$) and β_j a destroyer, as otherwise, β_j could not have been the first p-creator in Π_1.

case ii. Assume that all simplices in $\mathrm{col}_{V_1}[\beta_j]$ are contained in G'. This means that G' is no longer simple (as it contains another cycle $\mathrm{col}_{V_1}[\beta_j]$ in it) and hence it is necessary to break G'.

(ii.a) If β_j is unpaired, then our algorithm terminates with β_j being the new first p-creator and $\widehat{G} = \mathrm{col}_{V_1}[\beta_j]$ which is necessarily a subset of G'.

(ii.b) If β_j is paired, then we modify $G' \leftarrow G' + \mathrm{col}_{V_1}[\beta_j]$. From now on we target to track the new G', and note this new G' is a sub-cycle of the previous one. We next move β_j to the right of β_u.

In the two cases above, either our algorithm returns with a new first p-creator β_j with $j \in [1, u)$ and the update for deletion is over (case ii.a), or the first p-creator is getting closer to β_u (cases i and ii.b). If it is the latter, we repeat the above procedure till either the algorithm returns in case ii.a or β_u becomes the first p-creator, in which case, the update terminates with $\widehat{G} = \mathrm{col}_{V_1}[\beta_u]$. This procedure can be called at most $u \leq p$ number of time, leading to an $O(pm^2)$ update time for deleting a p-simplex σ.

Claim 4.3 *The above algorithm deletes a p-simplex α and all its k co-faces in $O((p+k)m^2)$ time. It detects correctly whether $[G]$ is destroyed after the deletion or not, and terminates the tracking if and only if $[G]$ is destroyed. If $[G]$ remains, then the algorithm returns a simple essential cycle $\widehat{G}$ such that either (i) $[\widehat{G}] = [G]$ and $\widehat{G}$ differs from G by at most p simplices, or (ii) $\widehat{G}$ is a subset of $G + \partial\tau$ for some $(p+1)$-coface τ of α.*

Proof: Set $\bar{G} = G + \partial\tau$ for some $(p+1)$-coface τ of α as picked by the algorithm. Note that if $\bar{G}$ is simple, then we will never reach case (ii). Hence the algorithm will terminate with β_u being the first creator, with all simplices in $\bar{G}$ necessarily to the left of β_u. In this case, the cycle contained in $\mathrm{col}_{\widehat{V}}[\beta_u]$ is exactly $\bar{G}$.

If $\bar{G}$ is not simple, then case (ii) will be called at least once. Otherwise, the algorithm would have to terminate with β_u being the first p-creator and all simplices in $\bar{G}$ to the left of β_u. This is not possible as the last simplex in one of the sub-cycles of $\bar{G}$ has to be a creator. We now argue that if $\bar{G}$ is not simple, then the cycle $\widehat{G}$ returned by the algorithm is simple, essential, and a subset

of $\bar{G}$. That $\widehat{G}$ is simple is guaranteed by that it is the cycle created by the first p-creator in current filtration. To see that it is essential and is a subset of $\bar{G}$, observe that (ii.b) guarantees that the new G' is always an essential subset of the previous G', thus an essential subset of $\bar{G}$ transitively. Similarly, (ii.a) returns an essential subset of current G', thus of $\bar{G}$. ■

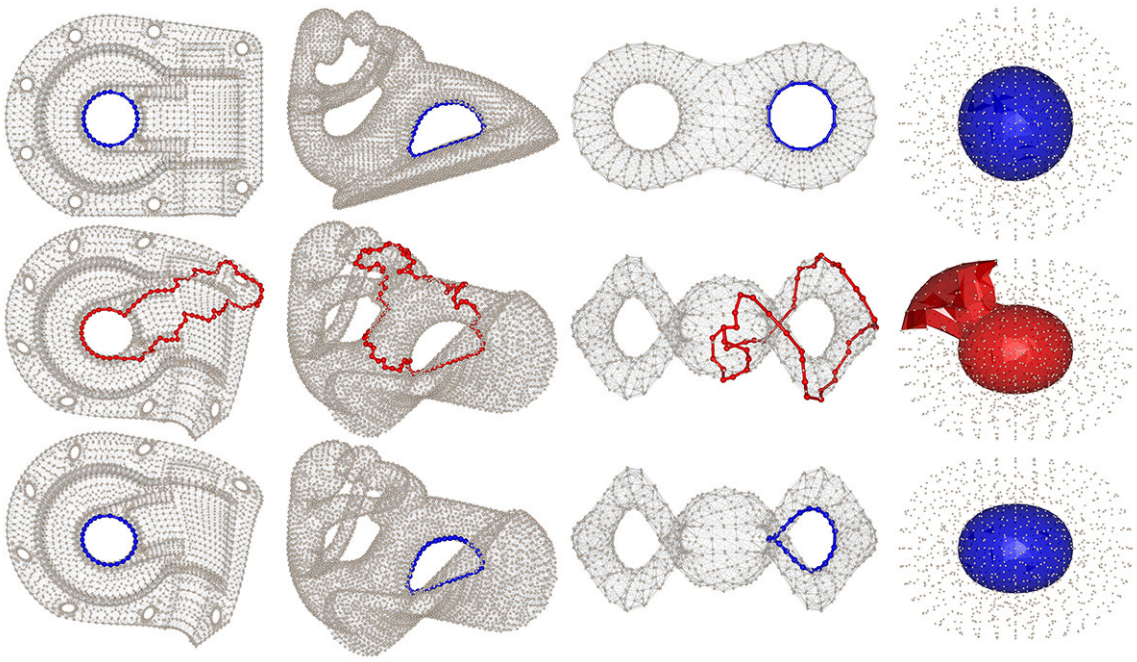


Figure 1: Tracking a generator in a Rips complex as the points move: the original generator chosen to be tracked (top row), the generator after several updates if tracking is not executed (middle row), and if tracking is enabled (bottom row). The rightmost column shows tracking a 2-cycle.

5 Conclusions

Tracking generating cycles of a time-varying simplicial complex is an important problem, having many practical applications. However, theoretical understanding of this topic is limited. In this paper, we made an initial step towards this problem by providing certain "locality" guarantee over a specific type of updates.

Clearly, many open questions remain. For example, our updates are limited to inserting a simplex with all its faces, or deleting a simplex together with all its co-faces. It will be interesting to find out how one can guarantee certain "locality" for more general updates. Our algorithm can potentially be extended to allow an update as a collection of deletions (or insertions). I.e., let $\mathcal{K}$ and $\widehat{\mathcal{K}}$ be the simplicial complex before and after deleting a collection of simplices X, and $G \subset \mathcal{K}$ be the p-cycle being tracked. We wish to return a localized version of G in $\widehat{\mathcal{K}}$ if the homology class $[G] \in \mathsf{H}_p(\mathcal{K})$ has a pre-image in $\mathsf{H}_p(\widehat{\mathcal{K}})$ under the homomorphism induced by the inclusion map $\widehat{\mathcal{K}} \hookrightarrow \mathcal{K}$. Eventually, the problem boils down to finding an appropriate ordering of simplices in X.

It is interesting to find out if the restriction for a cycle being tracked to be simple can be removed, i.e., whether we can develop an algorithm that always returns a cycle G' homologous and close to G as long as such a cycle exists.

Acknowledgments

Tamal Dey acknowledges the support of NSF grants CCF-0830467 and CCF-0915996. Yusu Wang acknowledges the support of NSF grants CCF-0747082 and DBI-0750891.

References

[1] G. Carlsson and V. de Silva and D. Morozov. Zigzag persistent homology and real-valued functions. *Proc. 25th Annu. Sympos. Comput. Geom.* (2009), 247–256.

[2] F. Chazal and D. Cohen-Steiner and M. Glisse and L. J. Guibas and S. Y. Oudot. Proximity of persistence modules and their diagrams. *Proc. 25th Annu. Sympos. Comput. Geom.* (2009), 237–246.

[3] F. Chazal and S. Oudot. Towards persistence-based reconstruction in Euclidean spaces. *Proc. 24th Ann. Sympos. Comput. Geom.* (2008), 232–241.

[4] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discr. & Comput. Geom.* **37** (2007), 103-120.

[5] D. Cohen-Steiner and H. Edelsbrunner and J. Harer. Extending persistence using Poincaré and Lefschetz duality. *Found. Comput. Math.* **9(1)** (2009), 79–103.

[6] D. Cohen-Steiner, H. Edelsbrunner, and D. Morozov. Vines and vineyards by updating persistence in linear time. *Proc. 22nd Annu. Sympos. Comput. Geom.* (2006), 119–134.

[7] T. K. Dey, J. Sun, and Y. Wang. Approximating loops in a shortest homology basis from point data. *26th Annu. Sympos. Comput. Geom.* (2010), to appear.

[8] T. K. Dey and R. Wenger. Stability of critical points with interval persistence. *Discr. & Comput. Geom.* **38** (2007), 479–512.

[9] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discr. & Comput. Geom.* **28** (2002), 511–533.

[10] A. Hatcher. *Algebraic Topology*. Cambridge U. Press, New York, 2002.

[11] H. Edelsbrunner and J. Harer. Persistent homology – a survey. *Surveys on Discrete and Computational Geometry: Twenty Years Later* (2008), 257-282.

[12] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discr. & Comput. Geom.* **33** (2005), 249–274.