Computational Topology for Data Analysis: Notes from Book by

Tamal Krishna Dey
Department of Computer Science
Purdue University
West Lafayette, Indiana, USA 46907

Yusu Wang
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, California, USA 92093

Topic 5: Persistence algorithms

For computational purposes, we focus on simplicial filtrations because it is not always easy to compute singular homology of topological spaces. The algorithms that compute the persistence diagram from a given filtration are presented in Section 5.1. First, we introduce it assuming that the input is presented combinatorially with simplices added one at a time in a filtration. The algorithm pairs simplices, one creating and the other destroying an interval. Then, this pairing is translated into matrix operations assuming that the input is a boundary matrix representing the filtration. A more efficient version of the algorithm is obtained by some simple but effective modification.

5.1 Persistence algorithm

First, we describe a combinatorial algorithm originally proposed in [10] and later present a version of it in terms of matrix reductions. For simplicity, assume that the input is a *simplex-wise filtration*

$$\emptyset = K_0 \hookrightarrow K_1 \hookrightarrow K_2 \hookrightarrow \cdots \hookrightarrow K_n = K$$

where $K_j \setminus K_{j-1} = \sigma_j$ is a single simplex for each j.

Fact 1. *When a p-simplex $\sigma_j = K_j \setminus K_{j-1}$ is added, exactly one of the following two possibilities occurs:*

1. *A non-boundary p-cycle c along with its classes $[c] + h$ for any class $h \in \mathsf{H}_p(K_{j-1})$ are born (created). In this case we call σ_j a* positive *simplex (also called a* creator*).*
2. *An existing $(p-1)$-cycle c along with its class $[c]$ dies (destroyed). In this case we call σ_j a* negative *simplex (also called a* destructor*).*

To elaborate the above two changes consider the example depicted in Figure 5.1. When one moves from K_7 to K_8, a non-boundary loop which is a 1-cycle $(e_5 + e_6 + e_7 + e_8)$ is created after adding edge e_8. Strictly speaking, a positive p-simplex σ_j may create more than one p-cycle. Only one of them can be taken as independent and the others become its linear combinations with the existing ones in K_{j-1}. From K_8 to K_9, the introduction of edge e_9 creates two non-boundary loops $(e_5 + e_6 + e_9)$ and $(e_7 + e_8 + e_9)$. But any one of them is the linear combination of the other one with the existing loop $(e_5 + e_6 + e_7 + e_8)$. Notice that there is no canonical way to choose an independent one. However, the creation of a loop is reflected in the increase of the rank of H_1. In other words, in general, the Betti number β_p increases by 1 for a positive simplex. For a negative simplex, we get the opposite effect. In this case β_{p-1} decreases by 1 signifying a death of a cycle. However, unlike positive simplices, the destroyed cycle is determined uniquely up to homology, which is the equivalent class carried by the boundary of σ_j. For example, in Figure 5.1, the loop $(e_7 + e_8 + e_9)$ gets destroyed by triangle t_{10} when we go from K_9 to K_{10}.

Pairing. We already saw that destruction of a class is uniquely paired with a creation of a class through the 'youngest first' rule. This means that each negative simplex is paired uniquely with a positive simplex. The goal of the persistence algorithm is to find out these pairs.

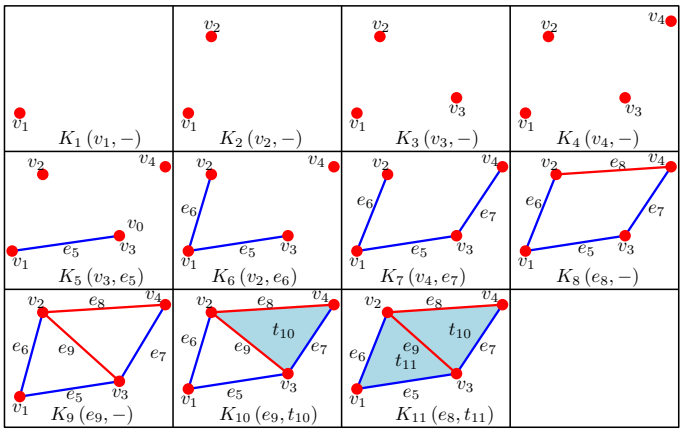

Figure 5.1: Red simplices are positive and blue ones are negative. The simplices are indexed to coincide with their order in the filtration. $(\cdot, \cdot)$ in each subcomplex shows the pairing between the positive and the negative. The second component missing in the parenthesis shows the introducing of a positive simplex.

Consider the birth and death of the classes by addition of simplices into a filtration. When a p-simplex σ_j is added, we explore if it destroys the class $[c]$ of its boundary $c = \partial\sigma_j$ if it is not a boundary already. The cycle c was created when the youngest $(p-1)$-simplex in it, say σ_i, was added. Note that a simplex is younger if it comes later in the filtration. If σ_i, a positive $(p-1)$-simplex, has already been paired with a p-simplex σ'_j, then a class also created by σ_i got destroyed when σ'_j appeared. We can get the $(p-1)$-cycle representing this destroyed class and add it to $\partial\sigma_j$. The addition provides a cycle that existed before σ_i. We update c to be this new cycle and look for the youngest $(p-1)$-simplex σ_i in c and continue the process till we find one that is unpaired, or the cycle c becomes empty. In the latter case, we discover that $c = \partial\sigma_j$ was a boundary cycle already and thus σ_j creates a new class in $\mathsf{H}_p(K_j)$. In the other case, we discover that σ_j is a negative p-simplex which destroys a class created by σ_i. We pair σ_j with σ_i. Indeed, one can show that the above algorithm produces the persistence pairs whose function values lead to the persistence diagram. We give a proof for a matrix version of the algorithm later (Theorem 2).

Definition 1 (Persistence pairs)**.** Given a simplex-wise filtration $\mathcal{F} : K_0 \hookrightarrow K_1 \hookrightarrow \cdots \hookrightarrow K_n$, for $0 < i < j \leq n$, we say a p-simplex $\sigma_i = K_i \setminus K_{i-1}$ and a $(p+1)$-simplex $\sigma_j = K_j \setminus K_{j-1}$ form a persistence pair (σ_i, σ_j) if and only if $\mu_p^{i,j} > 0$.

The full algorithm is presented in Algorithm 1:PairPersistence, which takes as input a sequence of simplices $\sigma_1, \sigma_2, \cdots, \sigma_n$ ordered according to the filtration of a complex whose persistence diagram is to be computed. It assumes that the complex is represented combinatorially with adjacency structures among its simplices.

Let us again consider the example in Figure 5.1 and see how the algorithm Pair works. From K_7 to K_8, e_8 is added. Its boundary is $c = (v_2 + v_4)$. The vertex v_4 is the youngest positive vertex in c but it is paired with e_7 in K_7. Thus, c is updated to $(v_3 + v_4 + v_4 + v_2) = (v_3 + v_2)$. The vertex

Algorithm 1 PairPersistence($\sigma_1, \sigma_2, \cdots, \sigma_n$)

Input:
An ordered sequence of simplices forming a filtration of a complex
Output:
Determine if a simplex is 'positive' or 'negative' and generate persistent pairs

1: **for** $j = 1$ to n **do**
2: $c = \partial_p \sigma_j$
3: σ_i is the youngest positive $(p-1)$-simplex in c.
4: **while** σ_i is paired and c is not empty **do**
5: Let c' be the cycle destroyed by the simplex paired with σ_i * see step 10 *
6: $c = c' + c$ * this addition may cancel simplices *
7: Update σ_i to be the youngest positive $(p-1)$-simplex in c
8: **end while**
9: **if** c is not empty **then**
10: σ_j is a negative p-simplex; generate pair (σ_i, σ_j); associate c with σ_j as destroyed
11: **else**
12: σ_j is a positive p-simplex * σ_j may get paired later *
13: **end if**
14: **end for**

v_3 becomes the youngest positive one but it is paired with e_5. So, c is updated to $(v_1 + v_2)$. The vertex v_2 becomes the youngest positive one but it is paired with e_6. So, c is updated to be empty. Hence e_8 is a positive edge. Now we examine the addition of the triangle t_{11} from K_{10} to K_{11}. The boundary of t_{11} is $c = (e_5 + e_6 + e_9)$. The youngest positive edge e_9 is paired with t_{10}. Thus, c is updated by adding the cycle destroyed by t_{10} to $(e_5 + e_6 + e_7 + e_8)$. Since e_8 is the youngest positive edge that is not yet paired, t_{11} finds e_8 as its paired positive edge. Observe that, we finally obtain a loop that is destroyed by adding the negative triangle. For example, we obtain the loop $(e_5 + e_6 + e_7 + e_8)$ by adding t_{11}.

5.1.1 Matrix reduction algorithm

There is a version of the algorithm PairPersistence that uses only matrix operations. First notice the following:

- The boundary operator $\partial_p : \mathsf{C}_p \to \mathsf{C}_{p-1}$ can be represented by a boundary matrix D_p where the columns correspond to the p-simplices and rows correspond to $(p-1)$-simplices.
- It represents the transformation of a basis of C_p given by the set of p-simplices to a basis of C_{p-1} given by the set of $(p-1)$-simplices.

$$D_p[i,j] = \begin{cases} 1 & \text{if } \sigma_i \in \partial_p \sigma_j \\ 0 & \text{otherwise.} \end{cases}$$

- One can combine all boundary matrices into a single matrix D that represents all linear maps $\bigoplus_p \partial_p = \bigoplus_p (\mathsf{C}_p \to \mathsf{C}_{p-1})$, that is, transformation of a basis of all chain groups

together to a basis of itself, but with a shift to a one lower dimension.

$$D[i,j] = \begin{cases} 1 & \text{if } \sigma_i \in \partial_* \sigma_j \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2 (Filtered boundary matrix)**.** Let $\mathcal{F} : K_0 = \emptyset \subset K_1 \subset \ldots \subset K_m = K$ be a filtration induced by an ordering of simplices $(\sigma_1, \sigma_2, \ldots, \sigma_m)$ in K. Let D denote the boundary matrix for simplices in K that respects the ordering of the simplices in the filtration, that is, the simplex σ_i in the filtration occupies column and row i in D. We call D the filtered boundary matrix for $\mathcal{F}$.

Given any matrix A, let $\text{row}_A[i]$ and $\text{col}_A[j]$ denote the ith row and jth column of A, respectively. We abuse the notation slightly to let $\text{col}_A[j]$ denote also the chain $\{\sigma_i \mid A[i,j] = 1\}$, which is the collection of simplices corresponding to 1's in the column $\text{col}_A[j]$.

Definition 3 (Reduced matrix)**.** Let $\text{low}_A[j]$ denote the row index of the last 1 in the jth column of A, which we call the *low-row index* of the column j. It is undefined for empty columns (marked with -1 in Algorithm 2). The matrix A is *reduced* (or is in *reduced form*) if $\text{low}_A[j] \neq \text{low}_A[j']$ for any $j \neq j'$; that is, no two columns share the same low-row indices.

Fact 2. *Given a matrix A in reduced form, we have that the set of non-zero columns in A are all linearly independent over $\mathbb{Z}_2$.*

We define a matrix A to be *upper-triangular* if all of its diagonal elements are 1, and there is no entry $A[i,j] = 1$ with $i > j$. We will compute a reduced matrix from a given boundary matrix by left-to-right column additions. A series of such column additions is equivalent to multiplying the boundary matrix on right with an upper triangular matrix.

Now, we state a result saying that if a reduced form is obtained via only left-to-right column additions, then for each column, the low-row index is unique in the sense that it does not depend on how the reduced form is obtained. Using this result we show that persistence pairing of simplices can be obtained from these low-row indices. Given an $n_1 \times n_2$ matrix A, let $A^{[c,d]}_{[a,b]}$, $a \leq b$ and $c \leq d$, denote the sub-matrix formed by rows a to b, and columns from c to d. In cases when $b = n_2$ and $c = 1$, we also write it as $A^d_a := A^{1,d}_{a,n_2}$ for simplicity. Define the quantity $r_A(i,j)$ as follows:

$$r_A(i,j) = \text{rank}\,(A^j_i) - \text{rank}\,(A^j_{i+1}) + \text{rank}\,(A^{j-1}_{i+1}) - \text{rank}\,(A^{j-1}_i).$$

Proposition 1 (Paring Uniqueness [7])**.** *Let $R = DV$, where R is in reduced form and V is upper triangular. Then* $\text{low}_R[j] = i$ *if and only if* $r_D(i,j) = 1$.

Next, we state that a pairing based on low-row indices indeed provides persistent pairs according to Definition 1 from which the algorithm follows. The proof appears in the book.

Theorem 2. *Let D be the $m \times m$ filtered boundary matrix for a filtration $\mathcal{F}$ (Definition 2). Let $R = DV$, where R is in reduced form and V is upper triangular. Then, the simplices σ_i and σ_j in $\mathcal{F}$ form a persistent pair if and only if* $\text{low}_R[j] = i$.

Algorithm 2 MATPERSISTENCE(D)

Input:
Boundary matrix D of a complex with columns and rows ordered by a given filtration
Output:
Reduced matrix with each column j either being empty or having a unique $\text{low}_D[j]$ entity

1: **for** $j = 1 \rightarrow |\text{col}_D|$ **do**
2: **while** $\exists j' < j$ s.t. $\text{low}_D[j'] == \text{low}_D[j]$ **and** $\text{low}_D[j] \neq -1$ **do**
3: $\text{col}_D[j] := \text{col}_D[j] + \text{col}_D[j']$;
4: **end while**
5: **if** $\text{low}_D[j] \neq -1$ **then**
6: $i := \text{low}_D[j]$; * generate pair (σ_i, σ_j) *\
7: **end if**
8: **end for**

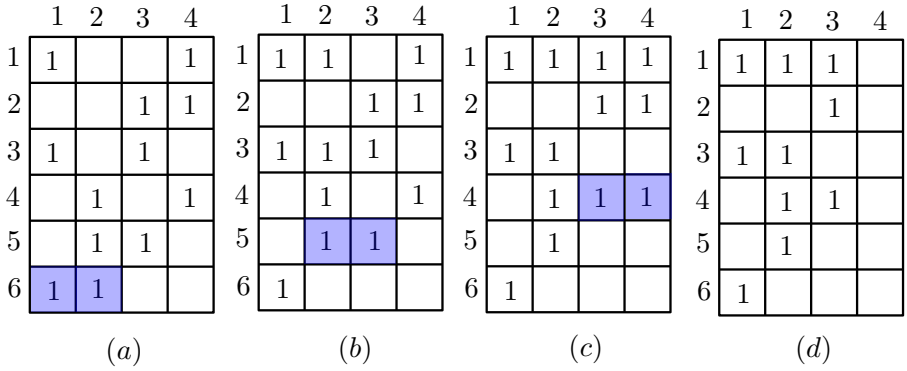

Figure 5.2: Matrix reduction for a 6×4 matrix D: low of columns are shaded to point out the conflicts. (a) $\text{low}_D[1]$ conflicts with $\text{low}_D[2]$ and $\text{col}_D[1]$ is added to $\text{col}_D[2]$, (b) $\text{low}_D[2]$ conflicts with $\text{low}_D[3]$, (c) $\text{low}_D[3]$ conflicts with $\text{low}_D[4]$, (d) the addition of $\text{col}_D[3]$ to $\text{col}_D[4]$ zero out the entire column $\text{col}_D[4]$.

The matrix reduction algorithm. Notice that there are possibly many R and V for a fixed D forming the *reduced-form decomposition*. Theorem 2 implies that the persistent pairing is independent of the particular contents of R and V as long as R is reduced and V is upper triangular. If we reduce a given filtered boundary matrix D to the reduced form R only with left-to-right column additions, indeed then we obtain $R = DV$ as required. With this principle, Algorithm 2:MATPERSISTENCE is designed to compute the persistent pairs of simplices. We process the columns of D from left to right which correspond to the order in which they appear in the filtration. The row indices also follow the same order top down (thus "lower" refers to a larger index, which also means that a simplex is "younger" in the filtration). We assume that $|\text{col}_D|$ denotes the number of columns in D. Suppose we have processed all columns up to $j-1$ and now are going to process the column j. We check if the row $\text{low}_D[j]$ contains any other lowest 1 for any column j' to the left of j, that is $j' < j$. If so, we add $\text{col}_D[j']$ to $\text{col}_D[j]$. This decreases $\text{low}_D[j]$. We

continue this process until either we turn all entries in $\text{col}_D[j]$ to be 0, or settle on $\text{low}_D[j]$ that does not conflict with any other $\text{low}_D[j']$ to its left. In the latter case, σ_j is a negative p-simplex that pairs with the positive $(p-1)$-simplex $\sigma_{\text{low}_D[j]}$. In the algorithm MatPersistence above, we assume that when a column j is zeroed out completely, $\text{low}_D[j]$ returns -1.

To compute the persistence diagram $\text{Dgm}(\mathcal{F}_f)$ for a filtration $\mathcal{F}_f$, we first run MatPersistence on the boundary matrix D representing $\mathcal{F}_f$. Every computed persistence pair (σ_i, σ_j) gives a finite bar $[f(\sigma_j), f(\sigma_i)]$ or a point with finite coordinates $(f(\sigma_i), f(\sigma_j))$ in $\text{Dgm}(\mathcal{F}_f)$. Every simplex σ_i that remains unpaired provides an infinite bar $[f(\sigma_i), \infty]$ or a point $(f(\sigma_i), \infty)$ at infinity in $\text{Dgm}(\mathcal{F}_f)$. Observe that not every positive p-simplex σ_i (column i is zeroed out) gives a point at infinity in $\text{Dgm}_p(\mathcal{F}_f)$, the only ones that do are the ones that are not paired with a $(p+1)$-simplex whose column is processed afterward. A simple fact about unpaired simplices is:

Fact 3. *The number of unpaired p-simplices in a simplex-wise filtration of a simplicial complex K equals its p-th Betti number $\beta_p(K)$.*

We already mentioned that the input boundary matrix D should respect the filtration order, that is, the row and column indices of D correspond to the indices of the simplices in the input filtration. Observe that we can consider slightly different filtration without changing the persistence pairs. We can arrange all of p-simplices for any $p \geq 0$ together in the filtration without changing their relative orders as follows where σ^i_j denotes the jth i-simplex among all i-simplices in the original filtration.

$$(\sigma^0_1, \sigma^0_2 \ldots, \sigma^0_{n_0}), \ldots, (\sigma^p_1, \sigma^p_2, \ldots, \sigma^p_{n_p}), \ldots, (\sigma^d_1, \sigma^d_2, \ldots, \sigma^d_{n_d}) \tag{5.1}$$

This means columns and rows of p-simplices in D become adjacent though retaining their relative ordering from the original matrix. Observe that, by this rearrangement, all columns that are added to a column j in the original D still remain to the left of j in their newly assigned indices. In other words, processing the rearranged matrix D can be thought of as processing each individual p-boundary matrix $D_p = [\partial_p]$ separately where the column and row indices respect the relative orders of p and $(p-1)$-simplices in the original filtration.

Complexity of MatPersistence. Let the filtration $\mathcal{F}$ based on which the boundary matrix D is constructed insert n simplices. This means that D has at most n rows and columns. Then, the outer **for** loop is executed at most $O(n)$ times. Within this **for** loop, steps 5-7 takes only $O(1)$ time. The complexity is indeed determined by the **while** loop (steps 2-4). We argue that this loop iterates at most $O(n)$ times. This follows from the fact that each column addition in step 3 decreases $\text{low}_D[j]$ by at least one and over the entire algorithm it cannot decrease by more than the length of the column which is $O(n)$. Each column addition in step 3 takes at most $O(n)$ time giving a total time of $O(n^2)$ for the **while** loop. Accounting for the outer **for** loop, we get a complexity of $O(n^3)$ for MatPersistence.

One can implement the above matrix reduction algorithm with a more efficient data structure noting that most of the entries in the input matrix D is empty. A linked list representing the non-zero entries in the columns of D is space-wise more efficient. Edelsbrunner and Harer [9] presents a clever implementation of MatPersistence using such a sparse matrix representation. For every column j, the algorithm executes $O(j-i)$ column additions of $O(j-i)$ length each incurring a cost $O((j-i)^2))$ where $i = 1$ if σ_j is positive and is the index of the simplex σ_i with which it pairs

in case σ_j is negative. Therefore, the total time complexity becomes $O(\sum_{j\in[1,n]}(j-i)^2)$. Here, we assume that the dimension of the complex K is a constant.

It is worth noting that essentially the matrix reduction algorithm is a version of the classical Gaussian elimination method with a given column order and a specific choice of row pivots. In this respect, persistence of a given filtration can be computed by the PLU factorization of a matrix for which Bunch and Hopcroft [5] gives an $O(M(n))$ time algorithm where $M(n)$ is the time to multiply two $n \times n$ matrices. It is known that $M(n) = O(n^\omega)$ where $\omega \in [2, 2.373)$ is called the exponent for matrix multiplication.

5.1.2 Efficient Implementation

The matrix reduction algorithm considers a column from left to right and reduces it by left-to-right additions. As we have observed, every addition to a column with index j pushes $\text{low}_D[j]$ upward. In the case, that σ_j is a positive simplex, the entire column is zeroed out. In general, positive simplices incur more cost than the negative ones because $\text{low}_D[\cdot]$ needs to be pushed all the way up for zeroing out the entire column. However, they do not participate in any future left-to-right column additions. Therefore, if it is known beforehand that the simplex σ_j will be a positive simplex, then the costly step of zeroing out the column j can be avoided.

Chen and Kerber [6] observed the following simple fact. If we process the input filtration backward in dimension, that is, process the boundary matrices D_p, $p = 1, \ldots, d$ in decreasing order of dimensions, then a persistence pair (σ^{p-1}, σ^p) is detected from D_p before processing the column for σ^{p-1} in D_{p-1}. Fortunately, we already know that σ^{p-1} has to be a positive simplex because it cannot pair with a negative simplex σ^p otherwise. So, we can simply ignore the column of σ^{p-1} while processing D_{p-1}. We call it *clearing* out column $p-1$. In practice, this saves a considerable amount of computation in cases where a lot of positive simplices occur such as in Rips filtrations. Algorithm 3:ClearPersistence implements this idea.

We cannot take advantage of the clearing for the last dimension in the filtration. If d is the highest dimension of the simplices in the input filtration, the matrix D_d has to be processed for all columns because the pairings for the positive d-simplices are not available.

If the number of d-simplices is large compared to simplices of lower dimensions, the incurred cost of processing their columns can still be high. For example, in a Rips filtration restricted up to a certain dimension d, the number of d-simplices becomes usually much larger than the number of, say, 1-simplices. In those cases, the clearing can be more cost-effective if it can be applied forward.

In this respect, the following observation becomes helpful. Let D_p^* denote the anti-transpose of the matrix D_p, defined by the transpose of D_p with the columns and rows being ordered in reverse. This means that if D_p has row and column indices $1, \ldots, m$ and $1, \ldots, n$ respectively, then $D_p^*(i, j) = D_p(n+1-j, m+1-i)$. Call it the twisted matrix of D_p. Figure 5.3 shows the twisted matrix D^* of the matrix D in Figure 5.2 where the rows and columns are marked with the indices of the original matrix. The following proposition guarantees that we can compute the persistence pairs in D_p from the matrix D_p^*.

Proposition 3. *(σ^{p-1}, σ^p) is a persistence pair computed from D_p if and only if (σ^p, σ^{p-1}) is computed as a persistence pair from D_p^*.*

The proof of the above proposition can be found in the book. We skip the details here.

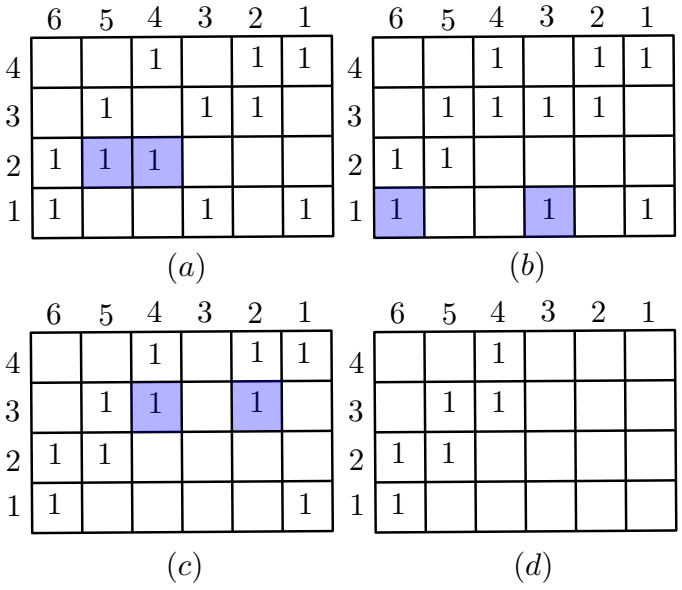

Figure 5.3: Matrix reduction with the twisted matrix D^* of the matrix D in Figure 5.2 which is first transposed and then got its rows and columns reversed in order; the conflicts in $\text{low}_D[\cdot]$ are resolved to obtain the intermediate matrices shown (a) through (d); The last transformation from (c) to (d) assumes to complete all conflict resolutions from columns 3 through 1. Observe that every column-row pair correspond to row-column pair in the original matrix. Also, all columns that are zeroed out here correspond to all rows in the original that did not get paired with any column meaning that they are either negative simplex, or positive simplex not paired with any.

To apply clearing we process D^*_{p+1} after D^*_p by calling ClearPersistence$(D^*_1, D^*_2, \cdots, D^*_d)$ because if we get a pair (σ^{p+1}, σ^p) while processing D^*_p, we already know that σ^{p+1} is a negative simplex and its column in D^*_{p+1} cannot contain a defined low entry. This means that the column of σ^{p+1} in D^*_{p+1} can be zeroed out and hence can be ignored. Now, the only boundary matrix that needs to be processed without any clearing is D^*_1. So, depending on whether D_d or D_1 is large, one can choose to process the filtration in increasing or decreasing dimensions respectively.

5.2 Notes and Exercises

The concept of topological persistence came to the fore in early 2000 with the paper by Edelsbrunner, Letscher, and Zomorodian [10] though the concept was proposed in a rudimentary form (for 0-dimensional homology) in other papers by Frosini [11] and Robins [12]. The persistence algorithm as described in this chapter was presented in [10] which has become the cornerstone of topological data analysis. The original algorithm was described without any matrix reduction which first appeared in [7]. Since then various versions of the algorithm has been presented. The persistence for filtrations of simplicial 1-complexes (graphs) with n simplices can be computed in $O(n\alpha(n))$ time (see the book). Persistence for filtrations of simplicial 2-manifolds also can be

Algorithm 3 ClearPersistence($D_1, D_2, \ldots, D_d$)

Input:
Boundary matrices ordered by dimension of the boundary operators with columns ordered by filtration

Output:
Reduced matrices with each column for negative simplex having a unique low entry

```
1: MatPersistence(D_d);
2: for i = (d − 1) → 1 do
3:   for j = 1 → |col_{D_i}| do
4:     if σ_j is not paired while processing D_{i+1} then
5:       \* column j is not processed if σ_j is already paired*\
6:       while ∃j' < j s.t. low_D[j] ≠ −1 and low_{D_i}[j'] == low_{D_i}[j] do
7:         col_{D_i}[j] := col_{D_i}[j] + col_{D_i}[j'];
8:       end while
9:       if low_D[j] ≠ −1 then
10:        k := low_{D_i}[j] \* generate pair (σ_k, σ_j) *\
11:      end if
12:    end if
13:  end for
14: end for
```

computed in $O(n\alpha(n))$ time algorithm by essentially reducing the problem to computing persistence on a dual graph. In general, for any constant $d \geq 1$, the persistence pairs between d- and $(d-1)$-simplices of a simplicial d-manifold can be computed in $O(n\alpha(n))$ time by considering the dual graph. If the manifold has boundary, then one has to consider a 'dummy' vertex that connects to every dual vertex of a d-simplex adjoining a *boundary* $(d-1)$-simplex.

For efficient implementation, clearing and compression strategies as described in Section 5.1.2 were presented by Chen and Kerber [6]. We have given a proof (in the book) based on matrix reduction that the same persistent pairs can be computed by considering the anti-transpose of the boundary matrix. This is termed as the *cohomology algorithm* first introduced in [8]. The name is justified by the fact that considering cohomology groups and the resulting persistence module that reverses the arrows, we obtain the same barcode. The anti-transpose of the boundary matrix indeed represents the coboundary matrix filtered reversely. These tricks are further used by Bauer for processing Rips filtration efficiently in the Ripser software [2]. Boissonnat et al. [3, 4] have suggested a technique to reduce the size of a given filtration using strong collapse of Barmak and Minian [1]. The collapse on the complex can be efficiently achieved only through simple manipulations of the boundary matrix.

Exercises

1. Let K be a p-complex with every $(p-1)$-simplex incident to exactly two p-simplices. Let M be a boundary matrix of the boundary operator ∂_p for K. We run a different version of

the persistence algorithm on M. We scan its columns from left to right as before, but we add the current column to its right to resolve conflict, i.e., for each $i = 1, \cdots, n$ in this order if there exists $j > i$ so that $\text{low}_M[i] = \text{low}_M[j]$, then add $\text{col}_M[i]$ to $\text{col}_M[j]$. Show that:

   (a) There can be at most one such j,

   (b) At termination, every column of M is either empty or has a unique low entry,

   (c) The algorithm outputs in $O(n^2)$ time the same $\text{low}_M[i]$ as the original persistence algorithm returns on M,

2. For a given matrix with binary entries, a valid column operation is one that adds a column to its right ($\mathbb{Z}_2$-addition). Similarly, define a valid row operation is the one that adds a row to another one above it. Show that there exists a set of valid column and row operations that leave every row and column either empty or with a single non-zero entry.

3. Let $\mathcal{F}$ be a filtration where every p-simplex appear only after all $(p-1)$-simplices like in Eqn. (5.1). Let $\mathcal{F}'$ be a modified filtration of $\mathcal{F}$ as follows. For every $p \geq 0$, all p-simplices in $\mathcal{F}$ are ordered in non-decreasing order of their persistence values in $\mathcal{F}'$ assuming that unpaired p-simplices have persistence value ∞. Show that the persistence pairing remains the same for $\mathcal{F}$ and $\mathcal{F}'$.

4. For a PL-function $f : |K| \to \mathbb{R}$, we know how to produce a simplex-wise filtration $\mathcal{F}$ so that the barcode for f can be read from the barcode of $\mathcal{F}$. Design an algorithm to do the reverse, that is, given a filtration $\mathcal{F}$ on a complex K, produce a filtration $\mathcal{G}$ of a simplicial complex K' so that $\mathcal{G}$ is indeed a simplex-wise filtration of a PL function $g : |K'| \to \mathbb{R}$ where bars for $\mathcal{F}$ can be obtained from those for $\mathcal{G}$. (Hint: use barycentric subdivision of K).

Bibliography

[1] Jonathan Ariel Barmak and Elias Gabriel Minian. Strong homotopy types, nerves and collapses. *Discret. Comput. Geom.*, 47(2):301–328, 2012.

[2] Ulrich Bauer. Ripser: efficient computation of vietoris-rips persistence barcodes. *CoRR*, abs/1908.02518, 2019.

[3] Jean-Daniel Boissonnat and Siddharth Pritam. Edge collapse and persistence of flag complexes. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*, volume 164 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[4] Jean-Daniel Boissonnat, Siddharth Pritam, and Divyansh Pareek. Strong collapse for persistence. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 67:1–67:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[5] James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[6] Chao Chen and Michael Kerber. An output-sensitive algorithm for persistent homology. *Comput. Geom.*, 46(4):435–447, 2013.

[7] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. In *Proc. 22nd Annu. Sympos. Comput. Geom.*, pages 119–126, 2006.

[8] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in persistent (co)homology. *Inverse Problems*, 27:124003, 2011.

[9] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. Applied Mathematics. American Mathematical Society, 2010.

[10] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28:511–533, 2002.

[11] Patrizio Frosini. A distance for similarity classes of submanifolds of a euclidean space. *Bulletin of the Australian Mathematical Society*, 42(3):407–415, 1990.

[12] Vanessa Robins. Towards computing homology from finite approximations. *Topology Proceedings*, 24(1):503–532, 1999.