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 8: Optimal generators

So far we have focused mainly on the rank of the homology groups. However, the homology generators, that is, the cycles whose classes constitute the elements of the homology groups carry information about the space. Computing just some generating cycles typically can be done by the standard algorithms for computing homology groups such as the persistence algorithms. In practice, however, we may sometimes be interested in generating cycles that have some optimal property. In particular, if the space has a metric associated with it, one may associate a measure with the cycles that can differentiate them in terms of their 'size'. For example, if K is a simplicial complex embedded in $\mathbb{R}^d$, the measure of a 1-cycle can be its length. Then, we can ask to compute a set of 1-cycles whose classes generate $\mathsf{H}_1(K)$ and has minimum total length among all such sets of cycles. Typically, the locality of these cycles capture interesting geometric features of the space $|K|$; see Figure 8.1 for an example. It turns out that, for $p > 1$, computing an optimal homology generators for p-dimensional homology group H_p is NP-hard [8]. However, the problem is polynomial time solvable for $p = 1$ [15]. A greedy algorithm which was originally devised for computing an optimal H_1-generator for surfaces [16] extends to general simplicial complexes as described in Section 8.1.

There is another version of optimality, namely the *localization* of homology classes. In this problem, given a p-cycle c, we want to compute an optimal p-cycle c^* in the same homology class of c, that is, $[c] = [c^*]$. This problem is NP-hard even for $p = 1$ [5]. Interestingly, there are some special cases for which an integer program formulated for the problem can be solved with a linear program [10]. This is the topic of Section 8.2.

The two versions mentioned above do not consider persistence framework. We may ask what are the optimal cycles for persistent homology classes. Toward formulating the problem precisely, we define a persistent cycle for a given bar in the barcode of a filtration. This is a cycle whose class is created at the birth point and becomes a boundary at the death point of the bar. Among all persistent cycles for a given bar, we want to compute an optimal one. The problem in general is NP-hard, but one can devise polynomial time algorithms for some special cases such as filtrations of what we call weak pseudo-manifolds [12, 13]. Section 8.3 describes these algorithms.

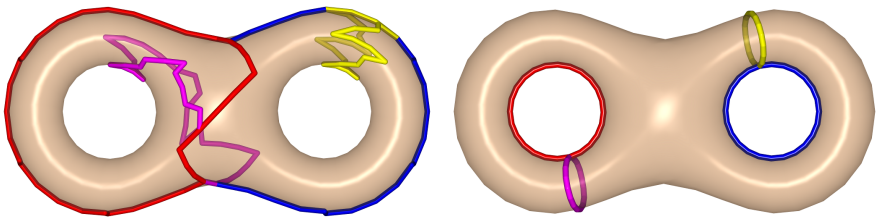

Figure 8.1: Double torus has one-dimensional homology group of rank four meaning that classes of four representative cycles generating H_1; (left) A non-optimal generator, (right) optimal generator.

8.1 Optimal generators/basis

We now formalize the definition for optimal cycles whose classes generate the homology group. Strictly speaking these cycles should not be called generator because it is their classes which generate the group. We take the liberty to call the cycles themselves as the generators.

Definition 1 (Weight of cycles)**.** Let $w : K_p \to \mathbb{R}_{\geq 0}$ be a non-negative weight function defined on the set of p-simplices in a simplical complex K. We extend w to the cycle space Z_p by defining $w(c) = \sum_i \alpha_i w(\sigma_i)$ where $c = \sum_i \alpha_i \sigma_i$ for $\alpha_i \in \mathbb{Z}_2$. For a set of cycles $\mathcal{C} = \{c_1, \ldots, c_g \mid c_i \in \mathsf{Z}_p(K)\}$ define its weight $w(\mathcal{C}) = \sum_{i=1}^{g} w(c_i)$.

Definition 2 (Optimal generator)**.** A set of cycles $\mathcal{C} = \{c_1, c_2, \ldots, c_g \mid c_i \in \mathsf{Z}_p(K)\}$ is a $\mathsf{H}_p(K)$-generator if the classes $\{[c_i] \mid i = 1, \ldots, g\}$ generate $\mathsf{H}_p(K)$. A $\mathsf{H}_p(K)$-generator is optimal if there is no other generator $\mathcal{C}'$ with $w(\mathcal{C}') < w(\mathcal{C})$.

Observe that, an optimal generator may not have minimal number of cycles whose classes generate the homology group because we allow zero weights and hence an optimal generator may contain extra cycles with zero weights. This prompts us to define the following.

Definition 3 (Optimal basis)**.** A $\mathsf{H}_p(K)$-generator $\mathcal{C} = \{c_1, c_2, \ldots, c_g \mid c_i \in \mathsf{Z}_p(K)\}$ is a $\mathsf{H}_p(K)$-cycle basis if $g = \dim \mathsf{H}_p(K)$. The classes of cycles in such a cycle basis constitute a basis for $\mathsf{H}_p(K)$. An optimal $\mathsf{H}_p(K)$-generator that is also a cycle basis is called an optimal $\mathsf{H}_p(K)$-cycle basis or $\mathsf{H}_p(K)$-basis in short.

We observe that optimal $\mathsf{H}_p(K)$-generators with positively weighted cycles are necessarily cycle bases. Notice that to generate $\mathsf{H}_p(K)$, the number of cycles in any $\mathsf{H}_p(K)$-generator has to be at least $\beta_p(K) = \dim \mathsf{H}_p(K)$. On the other hand, an optimal $\mathsf{H}_p(K)$-generator with positively weighted cycles cannot have more than β_p cycles because such a generator must contain a cycle whose class is a linear combination of the classes of other cycles in the generator. Thus, omission of this cycle still generates $\mathsf{H}_p(K)$ while decreasing the weight of the generator. For 1-dimension, similar reasoning can also be applied to conclude that each cycle in a $\mathsf{H}_1(K)$-cycle basis necessarily contains a *simple* cycle which together form a cycle basis (Exercise 1). A 1-cycle is simple if it has a single connected component (seen as a graph) and every vertex has exactly two incident edges.

Fact 1.

(i) An optimal $\mathsf{H}_p(K)$-generator with positively weighted cycles is an optimal $\mathsf{H}_p(K)$-basis.

(ii) Every cycle c_i in a $\mathsf{H}_1(K)$-basis has a simple cycle $c_i' \subseteq c_i$ so that $\{c_i'\}_i$ form a $\mathsf{H}_1(K)$-basis.

We now focus on computing an optimal $\mathsf{H}_p(K)$-basis also known as the optimal homology basis problem or OHBP in short. One may observe that Definition 3 formulates OHBP as a *weighted ℓ^1-optimization* of representatives of bases. This is very general and allows for different types of optimality to be achieved by choosing different weights. For example, assume that the simplicial complex K of dimension p or greater is embedded in $\mathbb{R}^d$, where $d \geq p + 1$. Let the Euclidean p-dimensional volume of p-simplices be their weights. This specializes OHBP to the *Euclidean ℓ^1-optimization* problem. The resulting optimal $\mathsf{H}_p(K)$-basis has the smallest p-dimensional volume amongst all such bases. If the weights are taken to be unit, the resulting optimal solution has the smallest *number* of p-simplices amongst all $\mathsf{H}_p(K)$-bases.

8.1.1 Greedy algorithm for optimal $\mathsf{H}_p(K)$-basis

Consider the following greedy algorithm in which we first sort the input cycles in nondecreasing order of their weights, and then choose a cycle following this order if its class is independent of the classes of the cycles chosen before.

Algorithm 1 GreedyBasis($\mathcal{C}$)

Input:
A set of p-cycles $\mathcal{C}$ in a complex
Output:
A maximal set of cycles from $\mathcal{C}$ whose classes are independent and total weight is minimum

1: Sort the cycles from $\mathcal{C}$ in non-decreasing order of their weights; that is, $\mathcal{C} = \{c_1, \ldots, c_n\}$ implies $w(c_i) \leq w(c_j)$ for $i \leq j$
2: Let $B := \{c_1\}$
3: **for** $i = 2$ **to** n **do**
4: **if** $[c_i]$ is independent wrt B **then**
5: $B := B \cup \{c_i\}$
6: **end if**
7: **end for**
8: if $[c_1]$ is trivial (boundary), output $B \setminus \{c_1\}$ else output B

The greedy algorithm Algorithm 1:GreedyBasis is motivated by Proposition 1. The specific implementation of line 4 (on independence test) will be given in Section 8.1.2.

Proposition 1. *Suppose that $\mathcal{C}$, the input to the algorithm* GreedyBasis*, contains an optimal $\mathsf{H}_p(K)$-basis. Then, the output of* GreedyBasis *is an optimal $\mathsf{H}_p(K)$-basis.*

The above proposition suggests that GreedyBasis can compute an optimal cycle basis if its input set $\mathcal{C}$ contains one. We show next that such an input (i.e., a set of 1-cycles containing an optimal $\mathsf{H}_1(K)$-basis) can be computed for $\mathsf{H}_1(K)$ in $O(n^2 \log n)$ time where the 2-skeleton of K has n simplices.

Specifically, given a simplicial complex K, notice that $\mathsf{H}_1(K)$ is completely determined by the 2-skeleton of K and hence without loss of generality we can assume K to be a 2-complex. Algorithm 2:Generator computes a set $\mathcal{C}$ of 1-cycles from such a complex which includes an optimal basis.

Proposition 2. Generator*(K) computes a $\mathsf{H}_1(K)$-generator $\mathcal{C}$ with their weights in $O(n^2 \log n)$ time for a 2-complex K with n vertices and edges. Furthermore, the set $\mathcal{C}$ contains an optimal basis where $|\mathcal{C}| = O(n^2)$.*

To see that Generator takes time as claimed, observe that each shortest path tree computation takes $O(n \log n)$ time by Dijkstra's algorithm implemented with Fibonacci heap [9]. Summing over $O(n)$ vertices, this gives $O(n^2 \log n)$ time. Each of the $O(n)$ edges in $E \setminus T_v$ for every vertex v gives $O(n)$ cycles in the output accounting for $O(n^2)$ cycles in total giving $|\mathcal{C}| = O(n^2)$. One can save space by representing each such cycle with the the edge $E \setminus T_v$ while keeping T_v for all of them without duplicates. Also, observe that the weight of each cycle $w(c_e)$ can be computed

Algorithm 2 GENERATOR(K)

Input:
A 2-complex K
Output:
A set of 1-cycles containing an optimal $\mathsf{H}_1(K)$-cycle basis

1: Let K^1 be the 1-skeleton of K with vertex set V and edge set E
2: $\mathcal{C} := \{\emptyset\}$
3: **for all** $v \in V$ **do**
4: compute a shortest path tree T_v rooted at v in $K^1 = (V, E)$
5: **for all** $e = (u, w) \in E \setminus T_v$ **do**
6: Compute cycle $c_e = \pi_{u,w} \cup \{e\}$ where $\pi_{u,w}$ is the unique path connecting u and w in T_v
7: $\mathcal{C} := \mathcal{C} \cup \{c_e\}$
8: **end for**
9: **end for**
10: Output $\mathcal{C}$

as a by-product of the Dijkstra's algorithm because it computes the weights of the shortest paths from the root to any of the vertices. Therefore, in $O(n^2 \log n)$ time, GENERATOR can output a $\mathsf{H}_1(K)$-generator with their weights.

8.1.2 Optimal $\mathsf{H}_1(K)$-basis and independence check

To compute an optimal $\mathsf{H}_1(K)$-basis, we first run GENERATOR on K and then feed the output to GREEDYBASIS as presented in Algorithm 3:OPTGEN which outputs an optimal H_1-cycle basis due to Propositions 2.

Algorithm 3 OPTGEN(K)

Input:
A 2-complex K
Output:
An optimal $\mathsf{H}_1(K)$-cycle basis

1: $\mathcal{C}$:= GENERATOR(K)
2: Output $\mathcal{C}^*$:=GREEDYBASIS($\mathcal{C}$)

However, we need to specify how to check the independence of the cycle classes in step 4 of GREEDYBASIS. We do this by using annotations described in previous topic. Recall that $\mathsf{a}(\cdot)$ denotes the annotation of its argument which is a binary vector. Algorithm 4:ANNOTEDGE is a version of the annotation algorithm adapted to edges only.

Assume that each cycle $c_e \in \mathcal{C}$ output by GENERATOR is represented by e and implicitly by the path $\pi_{u,w}$ in T_v. Assume that an annotation of edges has already been computed. This can be done by the algorithm ANNOTEDGE. A straightforward analysis shows that ANNOTEDGE takes $O(n^3)$ time where n is the total number of vertices, edges, and triangles in K. However, for achieving

Algorithm 4 AnnotEdge(K)

Input:
A simplicial 2-complex K
Output:
Annotations for edges in K

Let K^1 be the 1-skeleton of K with edge set E
Compute a spanning tree T of K^1; $m = |E| - |T|$
For every edge $e \in E \cap T$, assign an m-vector $\mathsf{a}(e)$ where $\mathsf{a}(e) = 0$
Index remaining edges in $E \setminus T$ as $e_1, \ldots, e_m$
For every edge e_i, assign $\mathsf{a}(e_i)[j] = 1$ iff $j = i$
for all triangle $t \in K$ **do**
  if $\mathsf{a}(\partial t) \neq 0$ **then**
    pick any non-zero entry b_u in $\mathsf{a}(\partial t)$
    add $\mathsf{a}(\partial t)$ to every edge e s.t. $\mathsf{a}(e)[u] = 1$
    delete uth entry from annotation of every edge
  end if
end for

better time complexity, we can use the earliest basis algorithm described in [4] which runs in time $O(n^\omega)$.

Once the annotations for edges are computed, we need to compute the annotations for the set $\mathcal{C}$ of cycles computed by Generator to check independence among them in GreedyBasis. We first describe how do we compute the annotations for a cycle in $\mathcal{C}$. We compute an *auxiliary* annotation of the vertices in T_v from the annotations of its edges to facilitate computing $\mathsf{a}(c_e)$ for cycles $c_e \in \mathcal{C}$. Traverse the tree T_v top-down and compute the auxiliary annotation $\mathsf{a}(x)$ of a vertex x in T_v as $\mathsf{a}(x) = \mathsf{a}(y) + \mathsf{a}(e_{xy})$ where y is the parent of x and e_{xy} is the edge connecting x and y. The process is initiated by assigning $\mathsf{a}(v)$ for the root v to be the zero vector. It should be immediately clear that all auxiliary annotations of the vertices can be computed in $O(gn)$ time where g, the length of the annotation vectors, equals $\beta_1(K)$. The annotation of each cycle $c_e \in \mathcal{C}$ can be computed as $\mathsf{a}(c_e) = \mathsf{a}(u) + \mathsf{a}(w) + \mathsf{a}(e)$ where $e = (u, w)$. Again, this takes $O(g)$ time per edge e and hence per cycle $c_e \in \mathcal{C}$ giving a time complexity of $O(gn^2)$ in total for the entire set $\mathcal{C}$.

Next, we describe an efficient way of determining the independence of cycles as needed in step 4 of GreedyBasis. Independence of the class $[c_e]$ with respect to all classes already chosen by GreedyBasis is done in a batch mode. One can do it edge by edge incurring more cost. We use a divide-and-conquer strategy instead.

Let $c_{e_1}, c_{e_2}, \ldots, c_{e_k}$ be the sorted order of cycles in $\mathcal{C}$ computed by Generator. We construct a matrix A whose ith column is the vector $\mathsf{a}(c_{e_i})$, and compute the first g columns that are independent called the *earliest basis* of A. Since there are k cycles in $\mathcal{C}$, the matrix A is $g \times k$. We use the following iterative method, based on making blocks, to compute the set J of indices of columns that define the earliest basis. We partition A from left to right into submatrices $A = [A_1|A_2|\cdots]$, where each submatrix A_i contains g columns, with the possible exception of the last submatrix, which contains at most g columns. Initially, we set J to be the empty set. We then iterate over the submatrices A_i by increasing index, that is, as they are ordered from left to right. At each iteration

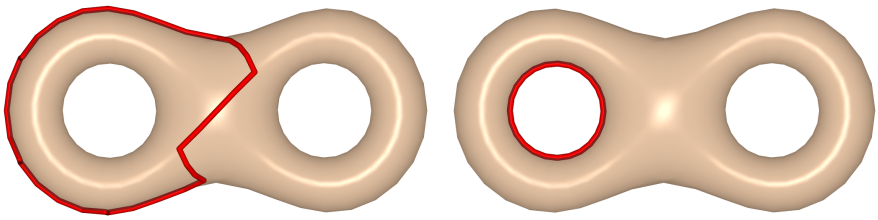

Figure 8.2: (left) A non-trivial cycle in a double torus, (right) optimal cycle in the class of the cycle on left.

we compute the earliest basis for the matrix $[A_J|A_i]$, where A_J is the submatrix whose column indices are in J. We then set J to be the indices from the resulting earliest basis, increase i, and go to the next iteration. At each iteration we need to compute the the earliest basis in a matrix with g rows and at most $|J| + g \leq 2g$ columns. Thus, each iteration takes $O(g^\omega)$ time, and there are at most $O(k/g) = O(n^2/g)$ iterations. Summing over all iterations, this gives a time complexity of $O(n^2 g^{\omega-1})$.

Theorem 3. *Given a simplicial 2-complex K with n simplices, an optimal $\mathsf{H}_1(K)$-basis can be computed in $O(n^\omega + n^2 g^{\omega-1})$ time.*

Proof. A H_1-generator containing an optimal (cycle) basis can be computed in $O(n^2 \log n)$ time due to Proposition 2. One can compute an optimal H_1-basis from $\mathcal{C}$ by GreedyBasis due to Proposition 1. However, instead of using GreedyBasis, we can apply the divide-and-conquer technique outlined above for computing the cycles output by GreedyBasis which takes $O(n^\omega + n^2 g^{\omega-1})$ time. Retaining only the dominating terms, we obtain the claimed complexity for the entire algorithm. □

8.2 Localization

In this section we consider a different optimization problem. Here we are given a p-cycle c in an input complex with non-negative weights on p-simplices and our goal is to compute a cycle c^* that is of optimal (minimal) weight in the homology class $[c]$, see Figure 8.2. We will extend this localization problem from cycles to chains. For this, first we extend the concept of homologous cycles to chains straightforwardly. Two p-chains $c, c' \in \mathsf{C}_p$ are called homologous if and only if they differ by a boundary, that is, $c \in c' + \mathsf{B}_p$. We ask for computing a chain of minimal weight which is homologous to a given chain.

Definition 4. Let $w : K(p) \to \mathbb{R}_{\geq 0}$ be a non-negative weight function defined on the set of p-simplices in a simplicial complex K. We extend w to the chain space C_p by defining $w(c) = \sum_i c_i w(\sigma_i)$ where $c = \sum_i c_i \sigma_i$.

Definition 5. Given a non-negative weight function $w : K(p) \to \mathbb{R}_{\geq 0}$ defined on the set of p-simplices in a simplicial complex K and a p-chain c in $\mathsf{C}_p(K)$, the *optimal homologous chain problem* (OHCP) is to find a chain c^* which has the minimal weight $w(c^*)$ among all chains homologous to c.

If we use $\mathbb{Z}_2$ as the coefficient ring for defining homology classes, the OHCP becomes NP-hard. We are going to show that it becomes polynomial time solvable if (i) the coefficient ring is chosen to be integers $\mathbb{Z}$ and (ii) the complex K is such that $\mathsf{H}_p(K)$ does not have a torsion which may be introduced because of using $\mathbb{Z}$ as the coefficient ring.

We will formulate OHCP as an integer program which requires the chains to be represented as an integer vector. Given a p-chain $x = \sum_{i=0}^{m-1} x_i \sigma_i$ with integer coefficients x_i, we use $\mathbf{x} \in \mathbb{Z}^m$ to denote the vector formed by the coefficients x_i. Thus, $\mathbf{x}$ is the representation of the chain x in the elementary p-chain basis, and we will use $\mathbf{x}$ and x interchangeably.

The main idea is to cast OHCP as an integer program. Unfortunately, integer programs are in general NP-hard and thus cannot be solved in polynomial time unless P=NP. We solve it by a linear program and identify a class of integer programs called *totally unimodular* for which linear programs give exact solution. Then, we interpret total unimodularity in terms of topology. Our approach to solve OHCP can be succinctly stated by following steps.

- write OHCP as an integer program involving 1-norm minimization, subject to linear constraints;
- convert the integer program into an integer *linear* program by converting the 1-norm cost function to a linear one using the standard technique of introducing some extra variables and constraints;
- find the conditions under which the constraint matrix of the integer linear program is totally unimodular; and
- for this class of problems, relax the integer linear program to a linear program by dropping the constraint that the variables be integral. The resulting optimal chain obtained by solving the linear program will be an integer valued chain homologous to the given chain.

8.2.1 Linear program

Now we formally pose OHCP as an optimization problem. After showing existence of solutions we reformulate the optimization problem as an integer linear program and eventually as a linear program.

Assume that the number of p- and $(p+1)$-simplices in K is m and n respectively, and let W be a diagonal $m \times m$ matrix. Let D_p represent the boundary matrix of the boundary operator $\partial_p : \mathsf{C}_p \to \mathsf{C}_{p-1}$ in the elementary chain bases. With these notations, given a p-chain $\mathbf{c}$ represented with an integral vector, the optimal homologous chain problem in dimension p is to solve:

$$\boxed{\min_{\mathbf{x},\mathbf{y}} \|W\,\mathbf{x}\|_1 \quad \text{such that} \quad \mathbf{x} = \mathbf{c} + D_{p+1}\,\mathbf{y}, \text{ and } \mathbf{x} \in \mathbb{Z}^m,\ \mathbf{y} \in \mathbb{Z}^n\,.} \tag{8.1}$$

We assume that W is a diagonal matrix obtained from non-negative *weights* on simplices as follows. Let w be a non-negative real-valued weight function on the oriented p-simplices of K and let W be the corresponding diagonal matrix (the i-th diagonal entry of W is $w(\sigma_i) = w_i$).

The resulting objective function $\|W\,\mathbf{x}\|_1 = \sum_i w_i\,|x_i|$ in (8.1) is not linear in x_i because it uses the absolute value of x_i. However, it is piecewise-linear in these variables. As a result, Eqn (8.1)

can be reformulated as an integer *linear* program in the following way [1, page 18] which splits every variable x_i into two parts x_i^+ and x_i^-:

$$\begin{aligned} \min \ & \sum_i w_i \, (x_i^+ + x_i^-) \\ \text{subject to} \quad & \mathbf{x}^+ - \mathbf{x}^- = \mathbf{c} + D_{p+1} \ \mathbf{y} \\ & \mathbf{x}^+, \ \mathbf{x}^- \geq \mathbf{0} \\ & \mathbf{x}^+, \, \mathbf{x}^- \in \mathbb{Z}^m, \ \mathbf{y} \in \mathbb{Z}^n \, . \end{aligned} \tag{8.2}$$

Comparing the above formulation to the standard form integer linear program in Eqn (8.5), note that the vector $\mathbf{x}$ in Eqn (8.5) corresponds to $[\mathbf{x}^+, \, \mathbf{x}^-, \, \mathbf{y}]^T$ in Eqn (8.2) above. Thus the minimization is over $\mathbf{x}^+, \mathbf{x}^-$ and $\mathbf{y}$, and the coefficients of x_i^+ and x_i^- in the objective function are $|w_i|$, but the coefficients corresponding to y_j are zero. The linear programming relaxation of this formulation just removes the constraints about the variables being integral. The resulting linear program is:

$$\begin{aligned} \min \ & \sum_i w_i \, (x_i^+ + x_i^-) \\ \text{subject to} \quad & \mathbf{x}^+ - \mathbf{x}^- = \mathbf{c} + D_{p+1} \ \mathbf{y} \\ & \mathbf{x}^+, \ \mathbf{x}^- \geq \mathbf{0} \, . \end{aligned} \tag{8.3}$$

To cast the program in standard form [1], we can eliminate the free (unrestricted in sign) variables $\mathbf{y}$ by replacing these by $\mathbf{y}^+ - \mathbf{y}^-$ and imposing the non-negativity constraints on the new variables. The resulting linear program has the same objective function, and the equality constraints:

$$\begin{aligned} \min \ & \sum_i w_i \, (x_i^+ + x_i^-) \\ \text{subject to} \quad & \mathbf{x}^+ - \mathbf{x}^- = \mathbf{c} + D_{p+1} \ (\mathbf{y}^+ - \mathbf{y}^-) \\ & \mathbf{x}^+, \ \mathbf{x}^-, \ \mathbf{y}^+, \ \mathbf{y}^- \ \geq \mathbf{0} \, . \end{aligned}$$

We can write the above program as

$$\min \ \mathbf{f}^T \mathbf{z} \quad \text{subject to} \quad A\mathbf{z} = \mathbf{c}, \ \mathbf{z} \geq \mathbf{0} \tag{8.4}$$

where $\mathbf{f} = [\mathbf{w}, 0]^T$, $\mathbf{z} = [\mathbf{x}^+, \mathbf{x}^-, \mathbf{y}^+, \mathbf{y}^-]^T$, and the equality constraint matrix is $A = \begin{bmatrix} I & -I & -B & B \end{bmatrix}$, where $B = D_{p+1}$. This is exactly in the form we want the linear program to be in view of Eqn. (8.5). We now state a result about the total unimodularity of this matrix that allows us to solve the optimization by a linear program.

8.2.2 Total unimodularity

A matrix is called *totally unimodular* if the determinant of each square submatrix is 0, 1, or -1. The significance of total unimodularity in our setting is due to the following Theorem 4 which follows immediately from known results in optimization [18].

Consider an integral vector $\mathbf{b} \in \mathbb{Z}^m$ and a real vector $\mathbf{f} \in \mathbb{R}^n$. Consider the *integer* linear program

$$\min \ \mathbf{f}^T \mathbf{x} \quad \text{subject to} \quad A\mathbf{x} = \mathbf{b}, \ \mathbf{x} \geq \mathbf{0} \text{ and } \mathbf{x} \in \mathbb{Z}^n \, . \tag{8.5}$$

Theorem 4. *Let A be a $m \times n$ totally unimodular matrix. Then the integer linear program* (8.5) *can be solved in time polynomial in the dimensions of A.*

Proposition 5. *If $B = D_{p+1}$ is totally unimodular then so is the matrix $\begin{bmatrix} I & -I & -B & B \end{bmatrix}$.*

As a result of Theorem 4 and Proposition 5, we have the following *algorithmic* result.

Theorem 6. *If the boundary matrix D_{p+1} of a finite simplicial complex of dimension greater than p is totally unimodular, the optimal homologous chain problem* (8.1) *for p-chains can be solved in polynomial time.*

Manifolds. Our results in the next section (Section 8.2.3) are valid for *any* finite simplicial complex. But first we consider a simpler case – simplicial complexes that are triangulations of manifolds. We show that for finite triangulations of compact p-dimensional *orientable* manifolds, the top non-trivial boundary matrix D_p is totally unimodular irrespective of the orientations of its simplices. There are examples of non-orientable manifolds where total unimodularity does not hold (Exercise 3). Further examination of why total unimodularity does not hold in these cases leads to the results in Theorems 9.

Let K be a finite simplicial complex that triangulates a $(p+1)$-dimensional compact orientable manifold M.

Theorem 7. *For a finite simplicial complex triangulating a $(p+1)$-dimensional compact orientable manifold, D_{p+1} is totally unimodular irrespective of the orientations of the simplices.*

As a result of the above theorem and Theorem 6 we have the following result.

Corollary 8. *For a finite simplicial complex triangulating a $(p+1)$-dimensional compact orientable manifold, the optimal homologous chain problem can be solved for p-dimensional chains in polynomial time.*

8.2.3 Relative torsion

Now we consider the more general case of simplicial complexes. We characterize the total unimodularity of boundary matrices for arbitrary simplicial complexes. This characterization leads to a torsion-related condition for the complexes. Since we do not use any conditions about the geometric realization or embedding in $\mathbb{R}^p$ for the complex, the result is also valid for abstract simplicial complexes. As a corollary of the characterization we show that the OHCP can be solved in polynomial time as long as the input complex satisfies the torsion-related condition.

TU and relative torsion

Definition 6. A *pure simplicial complex* of dimension p is a simplicial complex formed by a collection of p-simplices and their faces. Similarly, a *pure subcomplex* is a subcomplex that is a pure simplicial complex.

An example of a pure simplicial complex of dimension p is one that triangulates a p-dimensional manifold. Another example, relevant to our discussion, is a subcomplex formed by a collection of some p-simplices of a simplicial complex and their faces.

Let K be a finite simplicial complex of dimension greater than p. Let $L \subseteq K$ be a pure subcomplex of dimension $p+1$ and $L_0 \subset L$ be a pure subcomplex of dimension p. We use the definition of relative boundary operator in the book used for defining relative homology. Then, the matrix D_{p+1}^{L,L_0} representing the relative boundary operator

$$\partial_{p+1}^{L,L_0} : \mathsf{C}_{p+1}(L, L_0) \to \mathsf{C}_p(L, L_0)\,,$$

is obtained by first *including* the columns of D_{p+1} corresponding to $(p+1)$-simplices in L and then, from the submatrix so obtained, *excluding* the rows corresponding to the p-simplices in L_0 and any zero rows. The zero rows correspond to p-simplices that are not faces of any of the $(p+1)$-simplices of L. Then the following holds.

Theorem 9. *D_{p+1} is totally unimodular if and only if $\mathsf{H}_p(L, L_0)$ is torsion-free, for all pure subcomplexes L_0, L of K of dimensions p and $p+1$ respectively, where $L_0 \subset L$.*

Corollary 10. *For a simplicial complex K of dimension greater than p, there is a polynomial time algorithm for answering the following question: Is $\mathsf{H}_p(L, L_0)$ torsion-free for* all *subcomplexes L_0 and L of dimensions p and $(p+1)$ such that $L_0 \subset L$?*

A special case. In Section 8.2.2 we have seen the special case of compact orientable manifolds. We saw that the top dimensional boundary matrix of a finite triangulation of such a manifold is totally unimodular. Now we show another special case for which the boundary matrix is totally unimodular and hence OHCP is polynomial time solvable. This case occurs when we ask for optimal p-chains in a simplicial complex K which is embedded in $\mathbb{R}^{p+1}$. In particular, OHCP can be solved by linear programming for 2-chains in 3-complexes embedded in $\mathbb{R}^3$. This follows from the following result:

Theorem 11. *Let K be a finite simplicial complex embedded in $\mathbb{R}^{p+1}$. Then, $\mathsf{H}_p(L, L_0)$ is torsion-free for all pure subcomplexes L_0 and L of dimensions p and $p+1$ respectively, such that $L_0 \subset L$.*

Corollary 12. *Given a p-chain c in a weighted finite simplicial complex embedded in $\mathbb{R}^{p+1}$, an optimal chain homologous to c can be computed by a linear program.*

8.3 Persistent cycles

So far, we have considered optimal cycles in a given complex. Now, we consider optimal cycles in the context of a filtration. We know that a filtration of a complex gives rise to persistence of homology classes. An interval module which appears as a bar in the barcode are created by homology classes that get born and die at the endpoints. However, the bar is not associated with the class of a particular cycle because more than one cycle may get born and die at the endpoints. Among all these cycles, we want to identify the cycle that is optimal with respect to a weight assignment as defined earlier. Note that, as indicated earlier, an interval $[b, d-1]$ in the interval decomposition of a persistence module $\mathsf{H}_p(\mathcal{F})$ arising from a simplicial filtration $\mathcal{F}$ corresponds to a closed-open interval $[b, d)$ contributing a point (b, d) in the persistence diagram $\mathrm{Dgm}_p(\mathcal{F})$. We also say that the interval $[b, d)$ belongs to $\mathrm{Dgm}_p(\mathcal{F})$.

Definition 7 (persistent cycle)**.** Given a filtration $\mathcal{F} : \emptyset = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K$, and a finite interval $[b, d) \in \mathrm{Dgm}_p(\mathcal{F})$, we say a cycle c is a *persistent cycle* for $[b, d)$ if c is born at K_b and becomes a boundary in K_d. For an infinite interval $[b, \infty) \in \mathrm{Dgm}_p(\mathcal{F})$, we say a cycle c is a *persistent cycle* for $[b, \infty)$ if c is born at K_b.

Depending on whether the interval is finite or not, we have two cases captured in the following definitions. Let the cycles be weighted with a weight function $w : K(p) \to \mathbb{R}_{\geq 0}$ defined on the set of p-simplices in a simplicial complex K as before.

Problem 1 (PCYC-FIN$_p$)**.** Given a finite filtration $\mathcal{F}$ and a finite interval $[b, d) \in \mathrm{Dgm}_p(\mathcal{F})$, this problem asks for computing a persitent p-cycle for the bar $[b, d)$.

Problem 2 (PCYC-INF$_p$)**.** Given a finite filtration $\mathcal{F}$ and an infinite interval $[b, \infty) \in \mathrm{Dgm}_p(\mathcal{F})$, this problem asks for computing a persistent p-cycle with the minimal weight for the bar $[b, \infty)$.

When $p \geq 2$, computing minimal persistent p-cycles for both finite and infinite intervals is NP-hard in general. We identify a special but important class of simplicial complexes, which we term as *weak* $(p + 1)$*-pseudomanifolds*, whose minimal persistent p-cycles can be computed in polynomial time. A weak $(p + 1)$-pseudomanifold is a generalization of a $(p + 1)$-manifold and is defined as follows:

Definition 8 (Weak pseudomanifold)**.** A simplicial complex K is a *weak* $(p + 1)$*-pseudomanifold* if each p-simplex is a face of no more than two $(p + 1)$-simplices in K.

Specifically, it turns out that if the given complex is a weak $(p + 1)$-pseudomanifold, the problem of computing minimal persistent p-cycles for finite intervals can be cast into a minimal cut problem (see Section 8.3.1) due to the fact that persistent cycles of such kind are null-homologous in the complex. However, when $p \geq 2$ and intervals are infinite, the computation of the same becomes NP-hard. Nonetheless, for infinite intervals, if we assume that the weak $(p + 1)$-pseudomanifold is embedded in $\mathbb{R}^{p+1}$, then the minimal persistent p-cycle problem reduces to a minimal cut problem (see Section 8.3.3) and hence belongs to P. Note that a simplicial complex that can be embedded in $\mathbb{R}^{p+1}$ is necessarily a weak $(p + 1)$-pseudomanifold. We also note that while there is an algorithm [8] in the non-persistence setting which computes a minimal p-cycle by minimal cuts (the non-persistence algorithm assumes the $(p + 1)$-complex to be embedded in $\mathbb{R}^{p+1}$), the algorithm for finite intervals presented here, to the contrary, does not need the embedding assumption.

Before we present the algorithms for cases where they run in polynomial time, we summarize the complexity results for different cases. In order to make our statements about the hardness results precise, we let WPCYC-FIN$_p$ denote a subproblem[1] of PCYC-FIN$_p$ and let WPCYC-INF$_p$, WEPCYC-INF$_p$ denote two subproblems of PCYC-INF$_p$, with the subproblems requiring additional constraints on the given simplicial complex. Table 8.1 lists the hardness results for all problems of interest, where the column "Restriction on K" specifies the additional constraints subproblems require on the given simplicial complex K. Note that WPCYC-INF$_p$ being NP-hard trivially implies that PCYC-INF$_p$ is NP-hard.

[1]For two problems P_1 and P_2, P_2 is a *subproblem* of P_1 if any instance of P_2 is an instance of P_1 and P_2 asks for computing the same solutions as P_1.

Table 8.1: Hardness results for minimal persistent cycle problems.

Problem	*Restriction on K*	*p*	*Hardness*
PCYC-FIN$_p$	–	≥ 1	NP-hard
WPCYC-FIN$_p$	K a weak $(p+1)$-pseudomanifold	≥ 1	Polynomial
PCYC-INF$_p$	–	$= 1$	Polynomial
WPCYC-INF$_p$	K a weak $(p+1)$-pseudomanifold	≥ 2	NP-hard
WEPCYC-INF$_p$	K a weak $(p+1)$-pseudomanifold in $\mathbb{R}^{p+1}$	≥ 2	Polynomial

The polynomial time algorithms for the cases listed in the table above map the problem of computing optimal persistent cycles into the classic problem of computing minimal cuts in a flow network.

Undirected flow network. An *undirected flow network* (G, s_1, s_2) consists of an undirected graph G with vertex set $V(G)$ and edge set $E(G)$, a capacity function $C : E(G) \to [0, +\infty]$, and two non-empty disjoint subsets s_1 and s_2 of $V(G)$. Vertices in s_1 are referred to as *sources* and vertices in s_2 are referred to as *sinks*. A *cut* (S, T) of (G, s_1, s_2) consists of two disjoint subsets S and T of $V(G)$ such that $S \cup T = V(G)$, $s_1 \subseteq S$, and $s_2 \subseteq T$. We define the set of edges *across* the cut (S, T) as

$$E(S, T) = \{e \in E(G) \,|\, e \text{ connects a vertex in } S \text{ and a vertex in } T\}$$

The *capacity* of a cut (S, T) is defined as $C(S, T) = \sum_{e \in E(S,T)} C(e)$. A *minimal cut* of (G, s_1, s_2) is a cut with the minimal capacity. Note that we allow parallel edges in G (see Figure 8.4) to ease the presentation. These parallel edges can be merged into one edge during computation.

8.3.1 Finite intervals for weak $(p+1)$-pseudomanifolds

In this subsection, we present an algorithm which computes minimal persistent p-cycles for finite intervals given a filtration of a weak $(p+1)$-pseudomanifold when $p \geq 1$. The general approach proceeds as follows: Suppose that the input weak $(p+1)$-pseudomanifold is K which is associated with a simplex-wise filtration $\mathcal{F} : \emptyset = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n$ and the task is to compute the minimal persistent cycle of a finite interval $[b, d) \in \mathrm{Dgm}_p(\mathcal{F})$. Let $\sigma_b^{\mathcal{F}}$ and $\sigma_d^{\mathcal{F}}$ be the creator and destructor pair for the interval $[b, d)$. We first construct an undirected dual graph G for K where vertices of G are dual to $(p+1)$-simplices of K and edges of G are dual to p-simplices of K. One dummy vertex termed as *infinite vertex* which does not correspond to any $(p+1)$-simplices is added to G for graph edges dual to those boundary p-simplices, i.e, the p-simplices that are faces of at most one $(p+1)$-simplex. We then build an undirected flow network on top of G where the source is the vertex dual to $\sigma_d^{\mathcal{F}}$ and the sink is the infinite vertex along with the set of vertices dual to those $(p+1)$-simplices which are added to $\mathcal{F}$ after $\sigma_d^{\mathcal{F}}$. If a p-simplex is $\sigma_b^{\mathcal{F}}$ or added to $\mathcal{F}$ before $\sigma_b^{\mathcal{F}}$, we let the capacity of its dual graph edge be its weight; otherwise, we let the capacity of its dual graph edge be $+\infty$. Finally, we compute a minimal cut of this flow network and return the p-chain dual to the edges across the minimal cut as a minimal persistent cycle of the interval.

The intuition of the above algorithm is best explained by an example illustrated in Figure 8.3, where $p = 1$. The key to the algorithm is the duality between persistent cycles of the input interval and cuts of the dual flow network having finite capacity. To see this duality, first consider

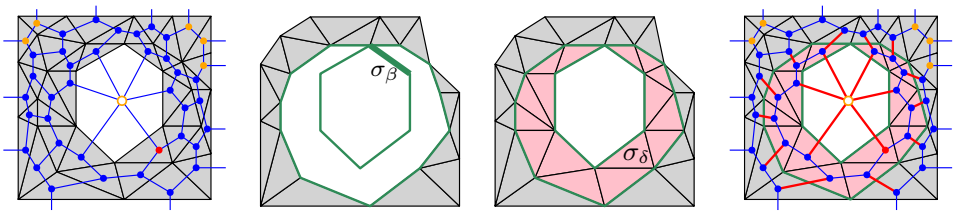

Figure 8.3: An example of the constructions in our algorithm showing the duality between persistent cycles and cuts having finite capacity for $p = 1$. (a) The input weak 2-pseudomanifold K with its dual flow network drawn in blue, where the central hollow vertex denotes the dummy vertex, the red vertex denotes the source, and the orange vertices denote sinks. All graph edges dual to the outer boundary 1-simplices actually connect to the dummy vertex. (b) The partial complex K_b in the input filtration $\mathcal{F}$, where the bold green 1-simplex denotes $\sigma_b^{\mathcal{F}}$ which creates the green 1-cycle. (c) The partial complex K_d in $\mathcal{F}$, where the 2-simplex $\sigma_d^{\mathcal{F}}$ creates the pink 2-chain killing the green 1-cycle. (d) The green persistent 1-cycle of the interval $[b, d)$ is dual to a cut (S, T) having finite capacity, where S contains all the vertices inside the pink 2-chain and T contains all the other vertices. The red graph edges denote those edges across (S, T) and their dual 1-chain is the green persistent 1-cycle.

a persistent p-cycle c of the input interval $[b, d)$. There exists a $(p + 1)$-chain A in K_d created by $\sigma_d^{\mathcal{F}}$ whose boundary equals c, making c killed. We can let S be the set of graph vertices dual to the simplices in A and let T be the set of the remaining graph vertices, then (S, T) is a cut. Furthermore, (S, T) must have finite capacity as the edges across it are exactly dual to the p-simplices in c and the p-simplices in c have indices in $\mathcal{F}$ less than or equal to b. On the other hand, let (S, T) be a cut with finite capacity, then the $(p + 1)$-chain whose simplices are dual to the vertices in S is created by $\sigma_d^{\mathcal{F}}$. Taking the boundary of this $(p + 1)$-chain, we get a p-cycle c. Because p-simplices of c are exactly dual to the edges across (S, T) and each edge across (S, T) has finite capacity, c must reside in K_b. We only need to ensure that c contains $\sigma_b^{\mathcal{F}}$ in order to show that c is a persistent cycle of $[b, d)$. In Section 8.3.2, we argue that c indeed contains $\sigma_b^{\mathcal{F}}$, so c is a persistent cycle.

In the dual graph, an edge is created for each p-simplex. If a p-simplex has two $(p + 1)$-cofaces, we simply let its dual graph edge connect the two vertices dual to its two $(p + 1)$-cofaces; otherwise, its dual graph edge has to connect to the infinite vertex on one end. A problem about this construction is that some weak $(p + 1)$-pseudomanifolds may have p-simplices being face of no $(p + 1)$-simplices and these p-simplices may create self loops around the infinite vertex. To avoid self loops, we simply ignore these p-simplices. The reason why we can ignore these p-simplices is that they cannot be on the boundary of a $(p + 1)$-chain and hence cannot be on a persistent cycle of minimal weight. Algorithmically, we ignore these p-simplices by constructing the dual graph only from what we call the $(p + 1)$-connected component of K containing $\sigma_d^{\mathcal{F}}$.

Definition 9 (q-connected)**.** Let K be a simplicial complex. For $q \geq 1$, two q-simplices σ and σ' of K are *q-connected in K* if there is a sequence of q-simplices of K, $(\sigma_0, \ldots, \sigma_l)$, such that $\sigma_0 = \sigma$, $\sigma_l = \sigma'$, and for all $0 \leq i < l$, σ_i and σ_{i+1} share a $(q - 1)$-face. The property of q-connectedness defines an equivalence relation on q-simplices of K. Each set in the partition

induced by the equivalence relation constitutes a *q-connected component* of K. We say K is *q-connected* if any two q-simplices of K are q-connected in K. See Figure 8.4 for an example of 1-connected components and 2-connected components.

We present the pseudo-code in Algorithm 5:MinPersCycFin and it works as follows: Line 1 and 2 set up a complex $\widetilde{K}$ that the algorithm mainly works on, where $\widetilde{K}$ is taken as the closure of the $(p+1)$-connected component of $\sigma_d^{\mathcal{F}}$. Line 3 constructs the dual graph G from $\widetilde{K}$ and line 4−15 builds the flow network on top of G. Note that we denote the infinite vertex by v_∞. Line 16 computes a minimal cut for the flow network and line 17 returns the p-chain dual to the edges across the minimal cut. In the pseudo-codes, to make presentation of algorithms and some proofs easier, we treat a mathematical function as a programming object. For example, the function θ returned by DualGraphFin in MinPersCycFin denotes the correspondence between the simplices of $\widetilde{K}$ and their dual vertices or edges (see Section 8.3.1 for details). In practice, these constructs can be easily implemented in any programming language.

Algorithm 5 MinPersCycFin$(K, p, \mathcal{F}, [b, d))$

Input:
K: finite p-weighted weak $(p+1)$-pseudomanifold
p: integer ≥ 1
$\mathcal{F}$: filtration $K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n$ of K
$[b, d)$: finite interval of $\mathrm{Dgm}_p(\mathcal{F})$
Output:
minimal persistent p-cycle of $[b, d)$

1: $L^{p+1} \leftarrow (p+1)$-connected component of K containing $\sigma_d^{\mathcal{F}}$ \∗ set up $\widetilde{K}$ ∗\
2: $\widetilde{K} \leftarrow$ closure of the simplicial set L^{p+1}
3: $(G, \theta) \leftarrow$ DualGraphFin$(\widetilde{K}, p)$ \∗ construct dual graph ∗\
4: **for all** $e \in E(G)$ **do**
5: **if** index$(\theta^{-1}(e)) \leq b$ **then**
6: $C(e) \leftarrow w(\theta^{-1}(e))$ \∗assign finite capacity∗\
7: **else**
8: $C(e) \leftarrow +\infty$ \∗assign infinite capacity∗\
9: **end if**
10: **end for**
11: $s_1 \leftarrow \{\theta(\sigma_d^{\mathcal{F}})\}$ \∗set the source∗\
12: $s_2 \leftarrow \{v \in V(G) \,|\, v \neq v_\infty, \text{index}(\theta^{-1}(v)) > d\}$ \∗set the sink∗\
13: **if** $v_\infty \in V(G)$ **then**
14: $s_2 \leftarrow s_2 \cup \{v_\infty\}$
15: **end if**
16: $(S^*, T^*) \leftarrow$ min-cut of (G, s_1, s_2)
17: Output $\theta^{-1}(E(S^*, T^*))$

Complexity. The time complexity of Algorithm 5 depends on the encoding scheme of the input and the data structure used for representing a simplicial complex. For encodings of the input,

we assume K and $\mathcal{F}$ to be represented by a sequence of all the simplices of K ordered by their indices in $\mathcal{F}$, where each simplex is denoted by its set of vertices. We also assume a simple yet reasonable simplicial complex data structure as follows: In each dimension, simplices are mapped to integral identifiers ranging from 0 to the number of simplices in that dimension minus 1; each q-simplex has an array (or linked list) storing all the id's of its $(q+1)$-cofaces; a hash map for each dimension is maintained for the query of the integral id of each simplex in that dimension based on the spanning vertices of the simplex. We further assume p to be constant. By the above assumptions, let n be the size (number of bits) of the encoded input, then there are no more than n elementary $O(1)$ operations in line 1 and 2 so the time complexity of line 1 and 2 is $O(n)$. It is not hard to verify that the flow network construction also takes $O(n)$ time so the time complexity of MinPersCycFin is determined by the minimal cut algorithm. Using the max-flow algorithm by Orlin [20], the time complexity of MinPersCycFin becomes $O(n^2)$.

Dual graph construction. We describe the DualGraphFin subroutine of Algorithm MinPersCycFin, which returns a dual graph G and a θ denoting two bijections which we use to prove the correctness. Given the input $(\widetilde{K}, p)$, DualGraphFin constructs an undirected connected graph G as follows:

- Let each vertex v of $V(G)$ correspond to each $(p+1)$-simplex σ^{p+1} of $\widetilde{K}$. If there is any p-simplex of $\widetilde{K}$ which has less than two $(p+1)$-cofaces in $\widetilde{K}$, we add an infinite vertex v_∞ to $V(G)$. Simultaneously, we define a bijection

$$\theta : \{(p+1)\text{-simplices of } \widetilde{K}\} \to V(G) \setminus \{v_\infty\}$$

  by letting $\theta(\sigma^{p+1}) = v$. Note that in the above range notation of θ, $\{v_\infty\}$ may not be a subset of $V(G)$.

- Let each edge e of $E(G)$ correspond to each p-simplex σ^p of $\widetilde{K}$. Note that σ^p has at least one $(p+1)$-coface in $\widetilde{K}$. If σ^p has two $(p+1)$-cofaces σ_0^{p+1} and σ_1^{p+1} in $\widetilde{K}$, then let e connect $\theta(\sigma_0^{p+1})$ and $\theta(\sigma_1^{p+1})$; if σ^p has one $(p+1)$-coface σ_0^{p+1} in $\widetilde{K}$, then let e connect $\theta(\sigma_0^{p+1})$ and v_∞. We define another bijection

$$\theta : \{p\text{-simplices of } \widetilde{K}\} \to E(G)$$

  using the same notation as the bijection for $V(G)$, by letting $\theta(\sigma^p) = e$.

Note that we can take the image of a subset of the domain under a function. Therefore, if (S, T) is a cut for a flow network built on G, then $\theta^{-1}(E(S, T))$ denotes the set of p-simplices dual to the edges across the cut. Also note that since simplicial chains with $\mathbb{Z}_2$ coefficients can be interpreted as sets, $\theta^{-1}(E(S, T))$ is also a p-chain.

8.3.2 Algorithm correctness

In this subsection, we prove the correctness of Algorithm MinPersCycFin. Some of the symbols we use refer to the pseudocode of the algorithm.

Proposition 13. *In Algorithm* MinPersCycFin*, s_2 is not an empty set.*

Proof. For contradiction, suppose that s_2 is an empty set. Then $v_\infty \notin V(G)$ and $\sigma_d^{\mathcal{F}}$ is the $(p+1)$-simplex of $\widetilde{K}$ with the greatest index in $\mathcal{F}$. Since $v_\infty \notin V(G)$, any p-simplex of $\widetilde{K}$ must be face of two $(p+1)$-simplices of $\widetilde{K}$, so the set of $(p+1)$-simplices of $\widetilde{K}$ forms a $(p+1)$-cycle created by $\sigma_d^{\mathcal{F}}$. Then $\sigma_d^{\mathcal{F}}$ must be a positive simplex in $\mathcal{F}$, which is a contradiction. □

The following two propositions specify the duality mentioned at the beginning of this section:

Proposition 14. *For any cut (S,T) of (G, s_1, s_2) with finite capacity, the p-chain $c = \theta^{-1}(E(S,T))$ is a persistent p-cycle of $[b,d)$ and $w(c) = C(S,T)$.*

Proposition 15. *For any persistent p-cycle c of $[b,d)$, there exists a cut (S,T) of (G, s_1, s_2) such that $C(S,T) \leq w(c)$.*

Combining the above results, we conclude:

Theorem 16. *Algorithm* MinPersCycFin *computes a minimal persistent p-cycle for the given interval $[b,d)$.*

8.3.3 Infinite intervals for weak $(p+1)$-pseudomanifolds embedded in $\mathbb{R}^{p+1}$

We already mentioned that computing minimal persistent p-cycles ($p \geq 2$) for infinite intervals is NP-hard even if we restrict to weak $(p+1)$-pseudomanifolds [12].

However, when the complex is embedded in $\mathbb{R}^{p+1}$, the problem becomes polynomially tractable. In this subsection, we present an algorithm for this problem given a weak $(p+1)$-pseudomanifold embedded in $\mathbb{R}^{p+1}$, when $p \geq 1$. For $p = 1$, the problem is polynomial time tractable for arbitrary complexes, see Exercise 4. The algorithm uses a similar duality described in Section 8.3.1. However, a direct use of the approach in Section 8.3.1 does not work. In particular, the dual graph construction is different – previously there is only one dummy vertex corresponding to infinity, now there is one per-void. For example, in Figure 8.4, 1-simplices that do not have any 2-cofaces cannot reside in any 2-connected component of the 2-complex. Hence, no cut in the flow network may correspond to a persistent cycle of the infinite interval created by such a 1-simplex. Furthermore, unlike the finite interval case, we do not have a negative simplex whose dual can act as a source in the flow network.

Let $(K, \mathcal{F}, [b, +\infty))$ be an input to the problem where K is a weak $(p+1)$-pseudomanifold embedded in $\mathbb{R}^{p+1}$, $\mathcal{F} : \emptyset = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n$ is a simplex-wise filtration of K, and $[b, +\infty)$ is an infinite interval of $\mathrm{Dgm}_p(\mathcal{F})$. By the definition of the problem, the task boils down to computing a minimal p-cycle containing $\sigma_b^{\mathcal{F}}$ in K_b. Note that K_b is also a weak $(p+1)$-pseudomanifold embedded in $\mathbb{R}^{p+1}$.

Generically, assume that $\widetilde{K}$ is an arbitrary weak $(p+1)$-pseudomanifold embedded in $\mathbb{R}^{p+1}$ and we want to compute a minimal p-cycle containing a p-simplex $\widetilde{\sigma}$ for $\widetilde{K}$. By the embedding assumption, the connected components of $\mathbb{R}^{p+1} \setminus |\widetilde{K}|$ are well defined and we call them the *voids* of $\widetilde{K}$. The complex $\widetilde{K}$ has a natural (undirected) dual graph structure as illustrated by Figure 8.4 for $p = 1$, where the graph vertices are dual to the $(p+1)$-simplices as well as the voids and the graph edges are dual to the p-simplices. The duality between cycles and cuts is as follows:

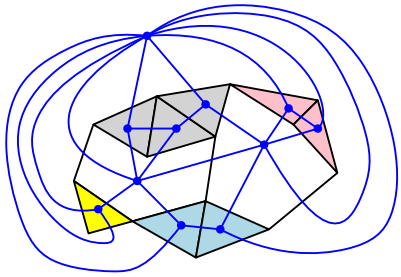

Figure 8.4: (left) A weak 2-pseudomanifold $\widetilde{K}$ embedded in $\mathbb{R}^2$ with three voids. Its dual graph is drawn in blue. The complex has one 1-connected component and four 2-connected components with the 2-simplices in different 2-connected components colored differently.

Since the ambient space $\mathbb{R}^{p+1}$ is contractible (homotopy equivalent to a point), every p-cycle in $\widetilde{K}$ is the boundary of a $(p+1)$-dimensional region obtained by point-wise union of certain $(p+1)$-simplices and/or voids. We can derive a cut[2] of the dual graph by putting all vertices contained in the $(p+1)$-dimensional region into one vertex set and putting the rest into the other vertex set. On the other hand, for every cut of the graph, we can take the point-wise union of all the $(p+1)$-simplices and voids dual to the graph vertices in one set of the cut and derive a $(p+1)$-dimensional region. The boundary of the derived $(p+1)$-dimensional region is then a p-cycle in $\widetilde{K}$. We observe that by making the source and sink dual to the two $(p+1)$-simplices or voids that $\widetilde{\sigma}$ adjoins, we can build a flow network where a minimal cut produces a minimal p-cycle in $\widetilde{K}$ containing $\widetilde{\sigma}$.

The efficiency of the above algorithm is in part determined by the efficiency of the dual graph construction. This step requires identifying the voids that the boundary p-simplices are incident on. A straightforward approach would be to first group the boundary p-simplices into p-cycles by local geometry, and then build the nesting structure of these p-cycles to correctly reconstruct the boundaries of the voids. This approach has a quadratic worst-case complexity. To make the void boundary reconstruction faster, we assume that the simplicial complex being worked on is p-connected so that building the nesting structure is not needed. Our reconstruction then runs in almost linear time. To satisfy the p-connected assumption, we begin our algorithm by taking $\widetilde{K}$ as a p-connected subcomplex of K_b containing $\sigma_b^{\mathcal{F}}$ and continue only with this $\widetilde{K}$. The computed output is still correct because the minimal cycle in $\widetilde{K}$ is again a minimal cycle in K_b. We skip the details of constructing void boundaries which can be done in $O(n \log n)$ time. Also, we skip the proof of correctness of the following theorem. Interested readers can consult [12] for details.

Theorem 17. *Given an infinite interval $[b, \infty) \in \mathrm{Dgm}_p(\mathcal{F})$ for a filtration $\mathcal{F}$ of a weak $(p+1)$-pseudomanifold K embedded in $\mathbb{R}^{p+1}$, a minimal persistent cycle for $[b, \infty)$ can be computed in $O(n^2)$ time where n is the number of p and $(p+1)$-simplices in K.*

[2]The cut mentioned here is defined on a graph without sources and sinks, so a cut is simply a partition of the graph's vertex set into two sets.

8.4 Notes and Exercises

The algorithm to compute a minimal homology basis based on a greedy strategy was first presented by Erickson and Whitlessey [16] who applied to simplicial 2-manifolds (surfaces). Chen and Friedman [8] showed that the problem is NP-hard for all homology groups of dimensions more than 1. Dey, Sun, and Wang [15] showed that a one dimensional minimal homology basis can be computed in $O(n^4)$ time for a simplicial complex with n simplices. The time complexity got improved to $O(n^{\omega+1})$ by Busaryev et al. [4]. Finally, it was settled to $O(n^3)$ in [14]. Borradaile et al. [2] proposed an algorithm for computing an optimal H_1-basis for graphs embedded on surfaces. For a graph with a total of n vertices and edges, the algorithm runs in $O(g^3 n \log n)$ time where g is the genus plus the number of boundaries in the surface.

The problem of computing a minimal homologous cycle in a given class is NP-hard even in dimension one as shown by Chambers et al. [5]. They proposed an algorithm for 1-cycles on surfaces utilizing the duality between minimal cuts of a surface-embedded graph and optimal homologous cycles of a dual complex. A better algorithm is proposed in [6]. Both algorithms are fixed parameter tractable running in time exponential in the genus of the surface. For general dimension, Borradaile et al. [3] showed that the OHCP problem in dimension p can be $O(\sqrt{\log n})$-approximated and is fixed-parameter tractable for weak $(p+1)$-pseudomanifolds. The only polynomial-time exact algorithm [8] in general dimension for OHCP works for p-cycles in complexes embedded in $\mathbb{R}^{p+1}$, which uses a reduction to minimal (s,t)-cuts. Interestingly, when the coefficient is chosen to be $\mathbb{Z}$ instead of $\mathbb{Z}_2$ for the homology groups, the problem becomes polynomial time solvable if there is no relative torsion as shown in [11]. The material presented in section 8.2 is taken from this paper.

Persistence added an extra layer of complexity to the problem of computing minimal representative cycles. Escolar and Hiraoka [17] and Obayashi [19] formulated the problem as an integer program by adapting a similar formulation for the non-persistent case. Wu et. al [21] adapted the algorithm of Busaryev et al. [4] to present an exponential-time algorithm, as well as an A* heuristics in practice. Dey, Hou, and Mandal [13] showed that the problem of computing minimal persistent cycle is NP-hard even for H_1. In a follow-up paper, the same authors show that the problem becomes polynomial for some special cases such as computing minimal persistent 2-cycles in a 3-complex embedded in 3-dimension [12]. The materials in section 8.3 are taken from this source.

Exercises

1. Show that every cycle in a $\mathsf{H}_1(K)$-basis contains a simple cycle which together form a $\mathsf{H}_1(K)$-basis themselves.

2. Define a minmax basis of $\mathsf{H}_p(K)$ as the set of cycles which generate $\mathsf{H}_p(K)$ and the maximum weight of the cycles is minmized among all such generators. Prove that an optimal H_p-basis defined in Definition 2 is also a minmax basis.

3. Take an example of a triangulation of Möbius strip and show that the integer program formulation of OHCP for it is not totally unimodular.

4. Consider computing a persistent 1-cycle for a bar $[b, d)$ given a filtration of an edge-weighted complex K. Let c be a cycle created by the edge $e = (u, v)$ at the birth time b where c is formed by the edge e and the shortest path between u and v in the 1-skeleton of the complex K_b. If $[c] = 0$ at K_d, prove that c is a minimal persistent cycle for the bar $[b, d]$.

5. Give an example where the above computed cycle using shortest path at the birth time is not a persistent cycle.

6. ([7]) For a vertex v in a complex with non-negative weights on edges, let discrete geodesic ball B_v^r of radius r is defined to be the maximal subcomplex $L \subseteq K$ so that the shortest path from v to every vertex in L is at most r. For a cycle c, let $w(c) = \min\{r \,|\, c \subseteq B_v^r\}$. Give a polynomial time algorithm to compute a shortest H_p-cycle basis for any $p \geq 1$ under this measure.

Bibliography

[1] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, MA., 1997.

[2] Glencora Borradaile, Erin Wolf Chambers, Kyle Fox, and Amir Nayyeri. Minimum cycle and homology bases of surface-embedded graphs. *JoCG*, 8(2):58–79, 2017.

[3] Glencora Borradaile, William Maxwell, and Amir Nayyeri. Minimum bounded chains and minimum homologous chains in embedded simplicial 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 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[4] Oleksiy Busaryev, Sergio Cabello, Chao Chen, Tamal K. Dey, and Yusu Wang. Annotating simplices with a homology basis and its applications. In *Algorithm Theory - SWAT 2012 - 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, pages 189–200, 2012.

[5] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Minimum cuts and shortest homologous cycles. In *SCG '09: Proc. 25th Ann. Sympos. Comput. Geom.*, pages 377–385, 2009.

[6] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. Comput.*, 41(6):1605–1634, 2012.

[7] Chao Chen and Daniel Freedman. Measuring and computing natural generators for homology groups. *Comput. Geometry: Theory & Applications*, 43 (2):169–181, 2010.

[8] Chao Chen and Daniel Freedman. Hardness results for homology localization. *Discrete & Comput. Geometry*, 45 (3):425–448, 2011.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[10] Tamal K. Dey, Anil N. Hirani, and Bala Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. In *STOC '10: Proc. 42nd Ann. Sympos. Theo. Comput.*, pages 221–230, 2010.

[11] Tamal K. Dey, Anil N. Hirani, and Bala Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM J. Comput.*, 40(4):1026–1044, 2011.

[12] Tamal K. Dey, Tao Hou, and Sayan Mandal. Computing minimal persistent cycles: Polynomial and hard cases. *CoRR*, abs/1907.04889, 2019.

[13] Tamal K. Dey, Tao Hou, and Sayan Mandal. Persistent 1-cycles: Definition, computation, and its application. In *Computational Topology in Image Context - 7th International Workshop, CTIC 2019, Málaga, Spain, January 24-25, 2019, Proceedings*, pages 123–136, 2019.

[14] Tamal K. Dey, Tianqi Li, and Yusu Wang. Efficient algorithms for computing a minimal homology basis. In *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, pages 376–398, 2018.

[15] Tamal K. Dey, Jian Sun, and Yusu Wang. Approximating loops in a shortest homology basis from point data. In *SCG '10: Proc. 26th Ann. Sympos. Comput. Geom.*, pages 166–175, 2010.

[16] Jeff Erickson and Kim Whittlesey. Greedy optimal homotopy and homology generators. In *SODA '05: Proc. 16th Ann. ACM-SIAM Sympos. Discrete Algorithms*, pages 1038–1046, 2005.

[17] Emerson G. Escolar and Yasuaki Hiraoka. Optimal cycles for persistent homology via linear programming. *Optimization in the Real World*, 13:79–96, 2016.

[18] Arthur F. Veinott Jr. and George B. Dantzig. Integral extreme points. *SIAM Review*, 10 (3):371–372, 1968.

[19] Ippei Obayashi. Volume-optimal cycle: Tightest representative cycle of a generator in persistent homology. *SIAM J. Appl. Algebra Geom.*, 2(4):508–534, 2018.

[20] James B. Orlin. Max flows in $O(nm)$ time, or better. In *Proc. Forty-Fifth Annual ACM Sympos. Theory Comput.*, pages 765–774, 2013.

[21] Pengxiang Wu, Chao Chen, Yusu Wang, Shaoting Zhang, Changhe Yuan, Zhen Qian, Dimitris N. Metaxas, and Leon Axel. Optimal topological cycles and their application in cardiac trabeculae restoration. In *Information Processing in Medical Imaging - 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings*, pages 80–92, 2017.