ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Patrick Schnider

Introduction to Topological Data Analysis **Scribe Notes 24** **FS23**

Scribe notes by Simon Weber. Please contact me for corrections.
Lecture date: May 26, 2023
Last update: Friday 26th May, 2023, 13:54

Optimal Generators

In a homology class, there are many homologous cycles. Furthermore, there are many different choices of homology classes which form a basis of the homology group. Thus, there are many different choices for cycles as bases of the homology group. How do we find good bases?

We define a weight function $w : K_p \to \mathbb{R}_{\geq 0}$ on the p-simplices, and the weight of a chain is simply the sum, i.e., $w(c) = \sum \alpha_i w(\sigma_i)$ for $c = \sum \alpha_i \sigma_i$. The weight of a set of cycles $\mathcal{C}$ is then the sum of weights of each cycle.

Definition 1. *A set $\mathcal{C}$ of cycles is an* optimal basis *for $H_p(K)$ if it is a basis and there is no other basis $\mathcal{C}'$ with $w(\mathcal{C}') < w(\mathcal{C})$.*

How can we compute such an optimal basis?

In a first step, we are going to compute a set of cycles $\mathcal{C}$ which contains an optimal basis. Then, we sort the cycles by increasing weight, and pick the first cycle to be part of our basis B. Then, we simply iterate through our cycles and add a cycle c_i to our basis if it cannot be written as a linear combination of our current basis. Finally, if c_1 is a boundary, we return the $B \setminus \{c_1\}$, and otherwise we return B.

To do this, we need to be able to compute our beginning set $\mathcal{C}$. Furthermore, we need to be able to check linear independence.

From now on, we will focus on computing a basis for $H_1(K)$. Without loss of generality, we say that K is 2-dimensional, with n triangles, $O(n)$ edges and vertices. To compute $\mathcal{C}$, we begin with $\mathcal{C} = \emptyset$. For all vertices v, we compute the shortest path tree T_v rooted at v. We can do this for example with Dijkstra's algorithm. For every edge e that is not in T_v, we add the unique cycle in $T_v \cup \{e\}$ to $\mathcal{C}$. This can be implemented in $O(n^2 \log n)$, and yields a set of cycles with $|\mathcal{C}| \in O(n^2)$. But, we need to prove that it is indeed a set which contains an optimal basis.

Lemma 2. *$\mathcal{C}$ as computed by the algorithm above contains an optimal basis.*

Proof. Let $\mathcal{C}^*$ be an optimal basis, and towards a contradiction, let c be a cycle contained in $\mathcal{C}^* \setminus \mathcal{C}$. As the weights are non-negative, we can assume that c is simple, i.e., no edge is used multiple times.

Let v be a vertex in c, and let T_v be the corresponding shortest path tree. There must be an edge $e = \{u, w\}$ in c, which is not in T_v, since T_v is a tree. Let $\Pi_{v,u}$ and $\Pi_{v,w}$ be the shortest paths from v to u, w respectively. These paths must be contained in T_v. Let us similarly consider $\Pi'_{v,u}$ and $\Pi'_{v,w}$ to be the (shortest) paths from v to u, w in c. We know that not both $\Pi'_{v,u} = \Pi_{v,u}$ and $\Pi'_{v,w} = \Pi_{v,w}$, so w.l.o.g. assume that $\Pi'_{v,u} \neq \Pi_{v,u}$.

We now define the cycle $c_1 = \{\Pi'_{v,w}, e, \Pi_{v,u}\}$ and $c_2 = \{\Pi_{v,u}, \Pi'_{v,u}\}$. We can now see that as we work in $\mathbb{Z}_2$, $c = c_1 + c_2$. Furthermore, we have $w(c_1) \leq w(c)$, since $\Pi_{v,u}$ is a shortest path (in K), while $\Pi'_{v,u}$ is not necessarily shortest. The same also holds for c_2: $w(c_2) \leq w(c)$ since $\{\Pi'_{v,w}, e\}$ can not be shorter than $\Pi_{v,u}$.

Let us now consider the homology classes of c_1 and c_2. If both $[c_1]$ and $[c_2]$ were dependent on $\mathcal{C}^* \setminus \{c\}$, then so would $[c]$, since $c = c_1 + c_2$. Then, $\mathcal{C}^*$ would not be a basis. Thus, at least one of $[c_1]$ and $[c_2]$ has to be dependent of $\mathcal{C}^* \setminus \{c\}$. Let us consider first that c_1 is independent. Then, we could replace c by c_1 in $\mathcal{C}^*$ and get a basis which is at least as good as $\mathcal{C}^*$. We can repeat the argument for that basis with v', the common ancestor of $\Pi_{v,u}$ and $\Pi'_{v,w}$. If c_2 is independent, we replace c by c_2 in $\mathcal{C}^*$ and repeat the argument with v' the common ancestor of $\Pi_{v,u}, \Pi'_{v,u}$ and e an edge incident to u.

At the end, we get a basis $\mathcal{C}'$ with $w(\mathcal{C}') \leq w(\mathcal{C}^*)$ with $\mathcal{C}' \subseteq \mathcal{C}$. □

So, we have finished the first step of our algorithm. It remains to figure out how to check independence. For this, we introduce *annotations*.

Definition 3. *An* annotation of p-simplices *is a function* $a : K^p \to \mathbb{Z} - 2^g$ *giving each* p*-simplex a binary vector of size* g*. This extends to chains by sums. An annotation must fulfill:*

- $g = \beta_p(K)$
- $a(z_1) = a(z_2)$ *iff* $[z_1] = [z_2]$.

Given an annotation, we can now clearly check linear independence of cycles by simply checking linear independence of a set of vectors, for which we have existing tools such as Gaussian eliminations.

Proposition 4. *In every simplicial complex* K *and for every* $p \geq 0$*, there exists an annotation of* p*-simplices, and can also be computed.*

Proof. (sketch for $p = 1$) We can compute a spanning forest T, and let m be the number of remaining edges. We initialize annotations of length m, and set $a(e) = 0$ for every edge in the spanning forest T. For every remaining edge e_i, we set $a_j(e_i) = 1$ if and only if $j = i$, and 0 otherwise.

For every triangle t, if the annotation of its boundary δt is not 0, we find a non-zero entry b_u in $a(\delta t)$ and add $a(\delta t)$ to every edge with $a_u(e) = 1$, and we delete the u-th entry from all annotations. One can show that this yields a valid annotation, and it can be implemented in $O(n^3)$, and more clever implementations work in $O(n^\omega)$. □

To check independence more efficiently, we add auxiliary annotations also to vertices in a shortest path tree T_v rooted at v. We give v the annotation 0, and for a vertex x that is the child of y, we set $a(x) := a(y) + a(e_{xy})$. For every cycle defined by the non-tree edge $e = uw$, we now have $a(c_e) = a(u) + a(w) + a(e)$. So, we never actually have to compute an explicit representation of a cycle by its edges, we only need to store its weight, the shortest path trees with the auxiliary annotations, and the non-tree edge e. Note that the auxiliary annotations can be computed in $O(gn)$ for the whole tree, thus in $O(gn^2)$ for all trees.

Finally, we have to check independence. Given an $(n \times m)$ matrix M, we can find the lexicographic leftmost set of independent columns in time $O(\max(n, m)^\omega)$. Instead of naively doing this n^2 times (once for every cycle), we group our cycles of $\mathcal{C}$ into groups A_i of size g, and compute the leftmost set for $[B|A_i]$, and thus we get $O(n^2 g^{\omega-1})$ runtime for this step.

To summarize, computing $\mathcal{C}$ takes $O(n^2 \log n)$, sorting the $O(n^2)$ cycles also takes $O(n^2 \log n)$, and for checking linear independence we need $O(n^\omega)$ for the annotations of the edges, $O(gn^2)$ for the auxiliary annotations, and $O(n^2 g^{\omega-1})$ for the block-wise linear independence checking. Overall, we thus get a runtime of $O(n^\omega + n^2 g^{\omega-1})$.

Persistent cycles

In the persistent setting, given a filtration $\mathcal{F}$ and an interval $[b, d]$, can we find an optimal persistent p-cycle c that is born at b and dies at d.

Sadly, this problem is already known to be NP-hard for $d < \infty$ and $p \geq 1$. However, if we assume that K is a weak $(p+1)$-pseudomanifold, i.e., a simplicial complex in which each p-simplex is a face of at most 2 $(p+1)$-simplices, then there exists a polynomial-time algorithm.

If we consider cycles that live until ∞, we can solve the problem in polynomial time for $p = 1$, but it is NP-hard for $p \geq 2$. Here, the assumption of K being a weak $(p+1)$-pseudomanifold does not save us. However, if we further assume that the complex can be embedded in $\mathbb{R}^{p+1}$, then it is again polynomial.

To solve the problem for $d < \infty$ in a weak $(p+1)$-pseudomanifold, we consider undirected flow networks: We have a graph, where every edge has a capacity in $[0, \infty]$, some sources, and some sinks, and we want to find the maximum flow we can send from the sources to the sinks without sending too much flow through any edge. Recall that if we consider a cut which separates the sources from the sinks, the capacity of this cut is an upper bound on the value of the maximum flow. Furthermore, if we consider the minimum

such cut, its capacity is equal to the value of the maximum flow. This can be solved in polynomial time.

We can build a dual graph G, by placing a vertex into every $(p+1)$-simplex and adding an edge whenever they share a p-simplex. We furthermore add a dummy vertex which gets connected to all vertices which only have one neighbor. We are going to make the vertex belonging to the $(p+1)$-simplex which is the destructor of our desired cycle the source. Furthermore, we make the dummy vertex as well as all vertices belonging to $(p+1)$-simplices added after the destructor into sinks. Edges added at or before the birth are getting the capacity equal to their weight, while all other edges get capacity ∞. Then, it turns out that the p-simplices belonging to the edges in a minimum cut separating the sources from the sinks are an optimal persistent cycle.