Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Patrick Schnider

Introduction to Topological Data Analysis Scribe Notes 10 FS23

Scribe notes by Simon Weber. Please contact me for corrections.
Lecture date: March 24, 2023
Last update: Friday 24th March, 2023, 14:21

We say that a p-homology class $[c]$ (a p-hole) is *born at* K_i if $[c] \in H_p(K_i)$ but $[c] \in H_p^{i-1,i}$. Similarly, $[c]$ *dies entering* K_j, if $[c] \neq 0$ in $H_p(K_{j-1})$ but $h_p^{j-1,j}([c]) = 0$.

It is not always obvious which homology class dies. If two homology classes merge, they both do not die, but their sum dies. In some way, the definition depends on the choice of basis. There is a consistent choice of basis which allows us to only look at persistent homology in terms of basis elements, but we do not go into this here.

If we have a simplex-wise filtration, we can sort homology classes by the time where they were born, and when they merge, we just say the "younger one" dies. This can be seen as adapting the considered basis along the way.

Persistence pairings are another way around this issue. We add some final complex K_{n+1} which has trivial homology (i.e., by adding all simplices that are not yet present). Then, we aim to figure out how many holes get born at K_i and die entering K_j. For this, we define

$$\mu_p^{i,j} = (\beta_p^{i,j-1} - \beta_p^{i,j}) - (\beta_p^{i-1,j-1} - \beta_p^{i-1,j}), \text{ for } i < j \leq n+1.$$

Here, the content of the left parenthesis denotes the number of holes born at or before K_i, which die entering K_j. Conversely, the right parenthesis denotes the number of holes born strictly before K_i, and die entering K_j. Thus, subtracting the two, gives the number of holes born exactly at K_i and die entering K_j.

The persistence diagram is a birth-death diagram which contains a point for every pair i, j for which $\mu_p^{i,j} > 0$. If we give each K_i a timestamp a_i, the point is drawn at the coordinates (a_i, a_j). We give each point multiplicity $\mu_p^{i,j}$. On the diagram we add points on the diagonal with infinite multiplicity, for some technical reasons that will only become apparent in a few weeks.

We can also represent the same information by barcodes: For every i, j, we draw $\mu_p^{i,j}$ many intervals $[a_i, a_j]$. This is then called the p-th persistence barcode.

Algorithms

We consider a simple-wise filtration. Consider some j, and let p be the dimension of the simplex added in K_j, i.e., $K_j \setminus K_{j-1} = \sigma_j$ is a p-simplex. There are only two things that can happen when adding σ_j: Either, a new non-boundary p-cycle c (a hole) is born. Then we say that σ_j is a *creator*. It is also possible that adding σ_j, a $(p-1)$-cycle becomes a boundary, thus a hole dies. Then we say that σ_j is a *destructor*. The fact that at exactly one of the two events happens is a consequence of the Euler characteristic, which was discussed in the Exercise Sheet 3.

The persistence pairing algorithm pairs a destructor σ with the youngest still-unpaired creator within the cycle it destroys. To find this youngest unpaired creator, we look at the boundary of σ. We try pairing σ to the youngest element ρ of its boundary. If this element is already paired with some element τ, we replace it by the sum of ρ and the boundary of τ. We now have a new set of candidate creators. We repeat this process until we found an unpaired creator we can pair to, or until we cannot continue (there are no more candidates). If we cannot pair σ to anything, it must be a new creator. Whatever unpaired creators remain at the end of the algorithm are paired to an element ∞.

What is the runtime of this algorithm? Let N be the total number of simplices in the final complex of our filtration. Whenever we add a simplex, and we replace a simplex by the boundary of its paired destructor, we add at most $O(N)$ simplices. We have to do this at most $O(N)$ times. Since we do this for each simplex, we get a runtime of $O(N^3)$. Surprisingly, this runtime is tight.

In practice, we actually use a different algorithm, which actually does the same but in the language of matrices. This is the *Matrix Reduction Algorithm*. Here, the filtration does not necessarily have to be simplex-wise. We write a large matrix, which is $N \times N$. Both rows and columns are labelled by the simplices, ordered by order of insertion. We then insert a 1 at row σ and column τ, if σ is part of the boundary of τ. For each column, we now look at the lowest 1 in the column. If there is a 1 towards the left of that column, add the column containing that other 1 to that column (in $\mathbb{Z}_2$). At the end, empty columns then correspond to creators (births). To find the death of a creator, look at its corresponding row, and find a pivot element in this row (a 1 which is the lowest 1 of its column). If there is no pivot element, this creator never dies, i.e., is unpaired.

Let's again look at the runtime. For each column $(O(N))$, we might have to add $O(N)$ times a column, and each addition takes $O(N)$. So again, we have $O(N^3)$ runtime. But, we can write the algorithm in a way such that it runs in $O(N^\omega)$, where ω is the matrix-multiplication exponent. In practice, it is essentially $O(N)$.