TDA TD 4 - programming project

Elliot Thorel, Dieks Scholten

October 2025

In this report we elaborate on our approach and choices for implementing a script which provides the barcode for a given set of simplexes. After this, we concern a number of experiments.

1 Code Architecture

1.1 Matrix Representation

One of the important decisions is to figure out the right matrix representation. We considered three different approaches. In all three we chose to store a matrix as a list of columns, since this suits the algorithm best. In each approach we considered an alternative method of storing a column. We tried: bitvectors, sorted lists, and sets. We have three general operations we have to perform on a column in this algorithm: creating the column, finding the index of the lowest 1, and XOR-ing two columns.

The bitvector approach, in practice, is often a very fast method but it actually is better for a dense matrix, When the size of the matrix gets to the order of 10^5 or 10^6 we see that the other two methods beat the bitvector in speed. The bottleneck of this approach was the creation of the bitvectors theselves. This took in the order of $\mathcal{O}(nd(d+n))$ operations, so a quadratic complexity. In a lower-level language this problem might be less prevalent, but in our tests the other two methods were considerably faster.

The sorted lists approach, in theory, should be the fastest approach. Constructing the matrix takes $\mathcal{O}(nd(d+1) + n\log n)$[1]. Finding the index of the lowest 1 in a column can be done in constant time and XOR-ing two columns can be done in $\mathcal{O}(n)$, by using two pointers. However, the two pointer method required a manual implementation and in practice we see that using a Python standard library implementation is often faster, which gives us our final approach.

The sets approach is in practice the fastest approach. For each column we store a Python Set object which includes only the indices of positive valued entries in a column. Constructing the matrix gives us a small edge over the sorted lists approach: $\mathcal{O}(nd(d+1))$. We do lose a little on finding the lowest 1

[1] Where d is the simplex dimension in the set of simplexes.

which is in $\mathcal{O}(n)$ now and XOR-ing two columns is still in $\mathcal{O}(n)$, but now we can do this with the built-in XOR operator on lists which gives a faster algorithm in practice.

1.2 Reduction

1.2.1 Question 2-3

As previously stated, we are working with matrices represented by a list of columns, which are sets of the non-zero indeces. The total algorithms consists of three parts: reading the simplexes, the reduction of the matrix, and the conversion to a barcode.

The first part is analysed simply by recognizing that it consits of two parts. First, the file with simplexes must be read which is $\mathcal{O}(n)$. Then we need to sort the list of simplexes to ensure a valid bijection between the simplexes and indexes, which is $\mathcal{O}(n \log n)$. The bijection is simply represented as a list object.

The second part of the algorithms consists of creating a matrix and reducing it to echelon form, via Gaussian elimination.

Our procedure for the creation of the matrix is as follows:

Algorithm 1 Matrix Construction

```
1:  vertex_to_index ← {}
2:  for i = 1 to |simplexes| do
3:      vertex_to_index[simplexes[i].vertices] = i
4:  end for
5:  columns ← [ ]
6:  for simplex in simplexes do
7:      col_values ← set()
8:      col_max ← −1
9:      for i = 1 to |simplex.vertices| do
10:         face ← simplex.vertices\simplex.vertices[i]
11:         if face in vertex_to_index then
12:             col_values.add(vertex_to_index[face])
13:             col_max ← max(col_max, vertex_to_index[face])
14:         end if
15:     end for
        Add Column(values = col_values, max = col_max) to columns list.
16: end for
17: return Matrix(columns = columns)
```

This code exists of two loops each of size $\mathcal{O}(n)$. The first loop has a constant-time operation for each iteration. The second loop consists of another loop of size $\mathcal{O}(d)$ where d is the highest dimension among the vertices. In this report we consider this to be bounded, as all example data was bounded in de dimensionality of the vertices. Futhermore, each loop of this inner loop performs a removal of an element in a list, which require shifting the list elements, but since

this list is of size dimension of the simplex. We get that each loop iteration is also considered to bo in constant time.

Putting this together gives that the creation of the matrix is linear in the number of simplexes.

To perform the matrix reduction to echelon form, we proceed as such :

Algorithm 2 Reduction algorithm

Require: $M \in M_n(Z_2)$
1: $zero\_columns \leftarrow [\,]$
2: $lowest\_one \leftarrow \{\}$
3: **for** $i = 1$ **to** n **do**
4: $c \leftarrow i^{th}$ column of M
5: **while** the column has not been reduced **do**
6: **if** $c = 0$ **then**
7: append i to $zero\_columns$
8: mark the column as reduced
9: **else**
10: $low\_1 \leftarrow$ lowest 1 of c
11: **if** $low\_1 \in lowest\_one$ **then**
12: $c \leftarrow c + lowest\_one[low\_1]$
13: **else**
14: $lowest\_one[low\_1] \leftarrow c$
15: mark the column as reduced
16: **end if**
17: **end if**
18: **end while**
19: **end for**
20: **return** M

A few remarks on this algorithm:

- As we work in $M_n(Z/2Z)$, subtracting 2 columns is exactly computing the set difference (XOR operation) between them. This is precisely what gives to the bitvector approach all its interest. Nevertheless, even if the cost of the operation is quite low with this approach, its complexity is still $\mathcal{O}(n)$, (one operation for each bit). For the list and set approach, the complexity is only $\mathcal{O}(k + k')$ where k and k' are the number of 1's in each column.

- For a given column c, for each step in the while–loop, we strictly decrease the low_1 variable. Therefore, during the whole algorithm, we execute the while loop at worst one time for every pair of columns.

The two above remarks are enough to show that the algorithm complexity is $\mathcal{O}(n^3)$ in the worst case. As the while–loop does at most $\mathcal{O}(n)$ loops each time doing in the worst case an $\mathcal{O}(n)$ operation. This all repeats n times, which gives the worst–case complexity of $\mathcal{O}(n^3)$.

Finally, the last part of the algorithms — inferring the barcode — is easily implemented in linear time.

Putting all the complexities of the three parts of the algorithm together gives a final worst-case complexity of $\mathcal{O}(n^3)$.

To argue that the complexity is linear or near linear in practice. We have to assume some properties concerned with the matrix reduction step of the algorithm:

1. For each column, we compute a constant number of XOR operations ($\#\text{XOR} = \Theta(1)$).
2. Each XOR operation is fast to compute ($\text{XOR} = \Theta(1)$).

We interpret the "the matrix remains sparse throughout" hypothesis as sufficient to validate both assumptions. Under these assumptions, we achieve the expected complexity of the $\Theta(n)$ complexity for the matrix reduction.

With this, if we combine all parts of this algorithm together, the complexity of the enitre program is dominated by sorting the vertices, which gives us an expected complexity of $\Theta(n \log n)$.

1.2.2 Question 4

A bar $[a, b[$ (eventually with $b = a$ or $b = +\infty$) corresponds to a cycle that is created at the "instant" a and that is killed at the instant b.

To create the barcode, we therefore take each element s of the *zero_column* list.

- The dimension is the dimension of the simplex s.
- The creation time a is the function value of the simplex s.
- The end time b is the function value of the simplex that killed s (or $+\infty$ if the simplex is never killed)

Creating the barcode is therefore of complexity $\mathcal{O}(number\_cycle) = \mathcal{O}(n)$ with the use of what had been previously computed.

2 Experiments

Classical spaces (Q 5-6)

Computing filtration for only one shape requires us to make a choice for the temporal aspect. Indeed, filtration are mostly fit for describing a shape that evolves as it showcases a succession of similar or related shapes.

Therefore, in this section, we will not focus too much on the function value. We can, for example, say that every simplex has the same function value or that each simplex has a function value equal to the dimension. What really matters to topologically describe the shapes are the bars with an infinite lifespan.

d-Ball

We can compute a d-ball by creating the complete simplicial graph of dimension d.

More precisely, we can take $d+1$ vertices numbered from 0 to d and then the filtration is built from $\mathcal{P}([0,d]) \setminus \{\emptyset\}$

Computing the barcode with this filtration gives us the expected-result: a unique infinite bar of dimension 0 that corresponds to the Betti number $(1,0,0,...)$ of any ball.

d-Sphere

We use a similar approach, but we remove the full simplex.

With vertices numbered from 0 to $d+1$,we end up with a filtration built from $\mathcal{P}([|0,d+1|]) \setminus (\emptyset \cup [|0,d+1|])$

Computing the barcode with this filtration also gives us the expected-result: an infinite bar of dimension 0 and one of dimension d that corresponds to the Betti number $(1,0,0,0,...,0,1,0,0,...)$ of the d-sphere.

Moebius Band

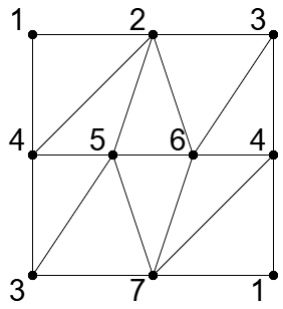

Figure 1: Triangulation of Moebius Band

With this filtration, we end up with the following Betti numbers :

$$b_0 = 1, b_1 = 1, \forall k \geq 2, b_k = 0$$

which matches expectations.

Torus

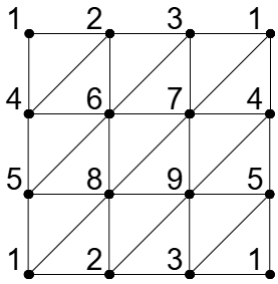

Figure 2: Triangulation Torus

With this filtration, we end up with the following Betti numbers :

$$b_0 = 1, b_1 = 2, b_2 = 1, \forall k \geq 3, b_k = 0$$

which matches expectations.

Klein Bottle

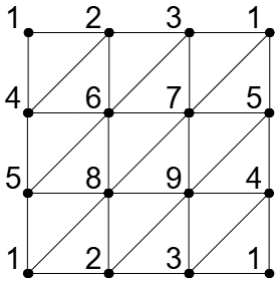

Figure 3: Triangulation Klein Bottle

With this filtration, out code provides 4 infinite-bars :

- 1 of dimension 0
- 2 of dimension 1
- 1 of dimension 2

The Betti numbers for the Klein bottle actually are the following :

$$b_0 = 1,\ b_1 = 1, \quad \forall k \geq 2, b_k = 0$$

This difference occurs because if your field is $Z/2Z$ then the Torus and the Klein Bottle have isomorphic homology groups and thus the same Betti numbers. If we would have considered Z in stead, we would have seen a difference here.

Projective Plane

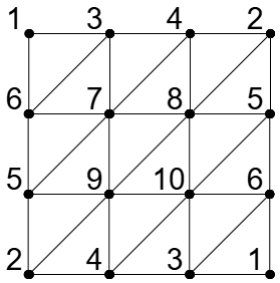

Figure 4: Triangulation Projective Plane

Similarly to what happened with the Klein bottle, the algorithm gives the following results:

- 1 infinite bar of dimension 0
- 1 infinite bar of dimension 1
- 1 infinite bar of dimension 2

Once again, we attribute the fact that it does not match the expected Betti number of

$$b_0 = 1, \forall k \geq 1, b_k = 0$$

to the use of the $Z/2Z$ field.

Q7

We recorded the following average timings over four runs (in seconds):

filtration	μ	σ
B	4.017	0.0683
C	8.484	0.0973
A	83.860	0.9009
D	311.244	7.0012

Table 1: Computation timing data

When we plot the results, we get the following:

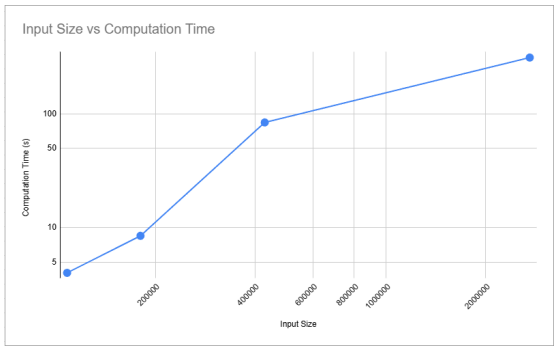

Figure 5: Log-log plot of computation time vs input size

If we consider this plot we see that it does not reflect a linear complexity. We however also concluded to have a near linear expected complexity of $\Theta(n \log n)$, which might better reflect the actual complexity.

When we plot a trendline in the loglog-plot we get a slope of about 1.36. This would suggest that the expected computation time would be proportional to $n^{1.36}$. This also support the claim of an expected complexity between $\Theta(n)$ and $\Theta(n^2)$.

To investigate further we provide the following plots:

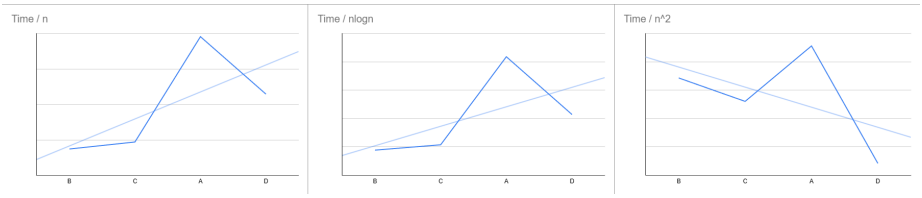

Figure 6: Computation time divided by a function of n

The labels are ordered from smallest n to largest n. We do not include the scale of the vertical axis as it provides no information, it is a linear plot which is sufficient.

If we the time complexity $T(n)$ would be in $\Theta(f(n))$ we would get that $T(n)/f(n) = \Theta(1)$ for large n. In these plots we see that assuming a linear complexity is too optimistic as it shows a clear upward trend, similarly assuming a quadratic complexity seems to be too pessimistic as it shows a clear downward trend. Even though, the near linear complexity still shows an upward trend, we believe that these plots show that an expected complexity of about $\Theta(n \log n)$ seems to be a suitable complexity.

Q8

1. **Filtration A:**

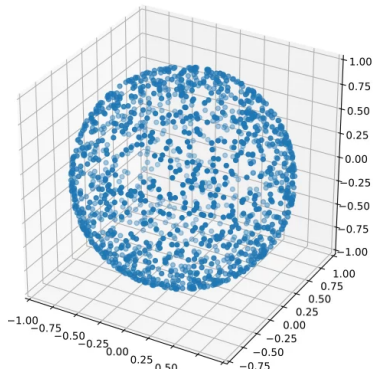

Figure 7: Points scattered near surface of sphere. [2]

This filtration appears fairly typical. We first see the cloud of distinct points. They are quickly connected to form 1 connected component. During this merge, a lot of 1-cycles are also created but they are quickly filled (killed), meaning the original distribution probably did not have any real 1-dimensional-hole.

The most interesting feature is the long bar indicating the presence of a 2-dimensional-hole for a relatively important amount of time. This hole is too persistant to be "accidental". On the opposite, the presence of a few other 2 dimensional-holes indicates that the original distribution is likely not a 2D-shape.

Our interpretation is that the barcode is the one of a compact distribution with a "hole" having a 0 or a very low probability, like a spherical-shell.

2. **Filtration B:**

[2] Illustration by rotjberg.net

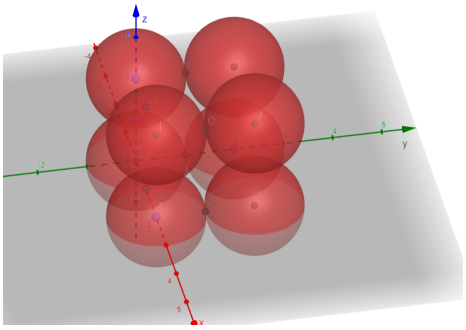

Figure 8: Balls on the corners of a cube.[3]

For this filtration, we consider the union of 8 overlapping spheres (or rather spherical-shell) placed on the corner of a cube. This configuration is reminiscent of the electronic cloud distribution in a simple cubic lattice.

- The 8 first H_2 bars correspond to the spherical-shell themselves that are qui quickly filled.
- The 5 H_1 bars correspond to the holes in the faces of the cube.
- The 1 last H_2 bar correspond to the cube itself that appears when all the cube's faces are closed

The simultaneity of the bars led us to a really structured representation.

3. **Filtration C:**

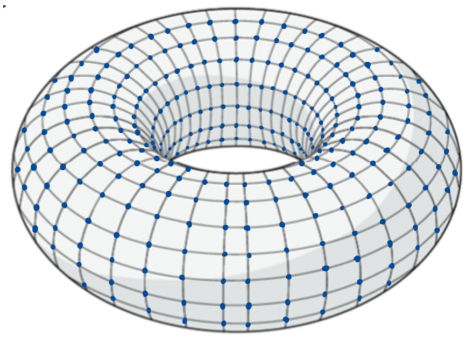

Figure 9: Points uniformly on torus.[4]

[3]Geogebra capture
[4]Original illustration by aofoundation.org.

We can consider a collection of points uniformly placed on the surface of a torus. Like the picture above, but we think of the surface being very "densely" packed with points. This gives at time $t = 0$, many connected components, so we would see a lot of H_0 bars and we do. When we increase the time a little bit we would see these points merge in to 1 connected component almost instantly. In the process, there would be a very small window where the balls have only partially merged an create little pockets or one dimensional holes, this would give a lot of H_1 bars just after the H_0, which is again what we see in the bar code. When all the balls have merged we would have a torus with a completely closed surface given the characteristic $\beta_0 = 1, \beta_1 = 2, \beta_2 = 1$ bars as seen in the barcode. Apart from being densely packed with points we also require that the torus is thin compared to its innerhole, so this would mean the when the balls grow, first the innerspace closes, this creates many small pockets inside the torus (giving many H_2 bars for a short while as seen in the barcode) and then filling the torus up completely giving a ring. Now, when these balls grow even further eventually the hole in the middle of the ring closes and we get a ball, as seen in the barcode.

4. **Filtration D:**

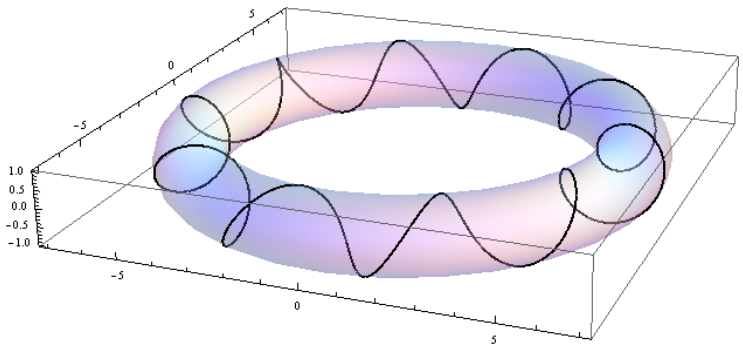

Figure 10: Spiral around torus.[5]

We can consider a collection of points on a spiral over the surface of a torus, see picture above. We imagine that these points are not uniformly distributed over this spiral but with little deviations, but we do see that the points are quite dense on the spiral itself. This gives our first part of the barcode.

When the points grow, the balls will quickly merge, which gives a quick decrease in the number of connected components until the spiral is one long connected component. This is seen in the barcode by the many H_0

[5]Illustration generated by Zev Chonoles.

bars decreasing and when there is only one H_0 bar left there also pops up one H_1 bar, which gives a circle. this connected spiral precisely gives a circle.

Now we also require this spiral to be quite densely wound around the torus which gives that when the balls grow even more the balls will slowly close the gaps and form a surface of the torus. When the surface is formed we again see the characteristic $\beta_0 = 1, \beta_1 = 2, \beta_2 = 1$.

Now we require, like before, that the torus is sufficiently thin compared to the hole in the middle, such that when the balls forming the surface grow even more they first make the torus solid creating a ring or circle and only later getting so big they close the hole in the middle and form effectively a ball. This would be similar to the filtration C, which is why we see the same bars in the barcode for this stage.