0-Dimensional Barcode

Ahmedur Rahman Shovon
Blazer ID: ashovon (ashovon@uab.edu)

October 18, 2021

1 Calculation of 0-dimensional barcode in the point cloud

We are given an adjacency matrix **M** comprised of pairwise distances of **n** vertices in a point cloud. We need to identify the 0-dimensional barcode for this given data [1].

1.1 Algorithm

Algorithm 1 consists of two procedures. The procedure *GetNumberOfComponents* calculates the number of disconnected components for a given minimum distance of a graph. This procedure uses a modified breadth-first search to find the number of disjoint components. The procedure *GetBarCodes* computes the barcode for each of the unique values of the pairwise distances from the number of components calculated by the previous procedure.

The running time of Algorithm 1 is $O(|V| + |E|)$.

1.2 Explanation

Figure 1 demonstrates 0-dimensional barcodes for a small dataset consisting of a $4 * 4$ adjacency matrix [2].

The barcode can be explained as following iterations:

1. At $\delta = 0$, there are four disconnected vertices. In other words, there are three connected components in the simplicial complex. So, four bars are born at this stage.
2. At $\delta = 0.25$, one edge is grown. So, the number of connected components decreases, and also one bar is finished.
3. At $\delta = 0.33$, another edge is grown. So, the number of connected components again decreases, and also one bar is finished.
4. At $\delta = 1$, another edge is grown. So, the number of connected components again decreases, and also one bar is finished. At this stage, there is one single component in the point cloud, and thus the upper bar is grown to infinity.

The same technique can be used to generate 0-dimensional barcodes for a large dataset shown in figure 2.

The above barcodes are isomorphic to the barcodes generated by the Ripser library shown in figure 3 and figure 4 [3].

1.3 Implementation

Implementation of the above algorithm is listed in Listing 1.

Listing 1: Implementation of 0 dimensional barcode

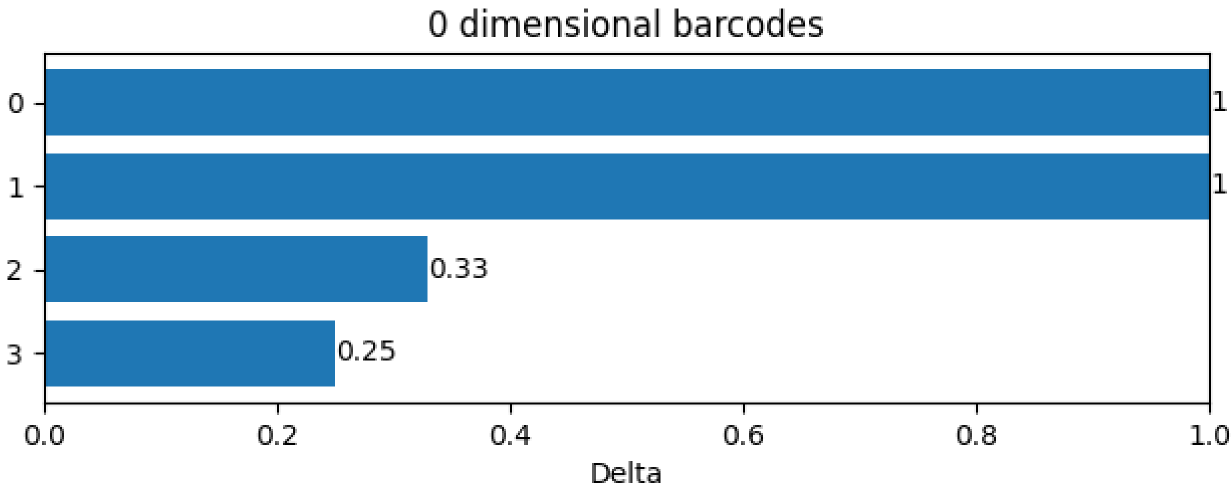
```python
def get_unique_distances(matrix):
    return list(sorted(set([j for row in matrix for j in row])))


def get_number_of_components(matrix, min_distance):
    v = len(matrix[0])
    group = 0
    seen = set()
    for i in range(v):
        if i not in seen:
            visited = [False for _ in range(v)]
            start_node = i
            q = [start_node]
            visited[start_node] = True
            while len(q) != 0:
                current = q.pop(0)
                seen.add(current)
                for node in range(v):
                    if visited[node] is False and matrix[current][node] <=
                       min_distance:
                        q.append(node)
                        visited[node] = True
            group += 1
    return group


def get_barcodes(matrix, max_val=1.0):
    unique_distances = get_unique_distances(matrix)
    barcodes = list()
    number_of_components = None
    for distance in unique_distances:
        components = get_number_of_components(matrix, distance)
        if components == 1:
            barcodes.append([0, distance])
            break
        if number_of_components is None:
            number_of_components = components
            continue
        if components < number_of_components:
            for i in range(number_of_components - components):
                barcodes.append([0, distance])
            number_of_components = components
    remaining_bars = len(matrix[0]) - len(barcodes)
    for i in range(remaining_bars):
        barcodes.append([0, max_val])
    barcodes = sorted(barcodes, key=lambda x: x[1], reverse=True)
    return barcodes
```

Algorithm 1 0-dimensional bar code calculation algorithm

```
procedure GetNumberOfComponents(M, minDistance)
    v ← M[0].length
    group ← 0
    seen ← emptyset()
    for i ← 0, v do
        if i not in seen then
            visited ← []
            for k ← 0, v do
                visited[k] ← false
            end for
            startNode ← i
            q ← [startNode]
            visited[startNode] ← True
            while q.length! = 0 do
                current ← q.pop(0)
                seen.add(current)
                for node ← 0, v do
                    if visited[node] == false &&M[current][node] ≤ minDistance then
                        q.append(node)
                        visited[node] ← true
                    end if
                end for
            end while
            group ← group + 1
        end if
    end for
    return group                                    ▷ return the number of components
end procedure
procedure GetBarCodes(M, maxVal)
    uniqueDistances ← Unique values of the matrix M
    barcodes ← []
    numberOfComponents ← None
    for distance in uniqueDistances do
        components ← GetNumberOfComponents(M, distance)
        if component == 1 then
            barcodes.append([0, distance])
            break
        end if
        if numberOfComponents == None then
            numberOfComponents ← components
            continue
        end if
        if components < numberOfComponents then
            for i ← 0, numberOfComponents - components do
                barcodes.append([0, distance])
            end for
            numberOfComponents ← components
        end if
    end for
    remainingBars ← M[0].length - barcodes.length
    for i ← 0, remaingBars do
        barcodes.insert(0, ([0, maxVal]))
    end for
    barcodes ← Sort barcodes by values in descending order
    return barcodes                                 ▷ return the list of barcodes
end procedure
```



Adjacency matrix

0.00	0.33	0.50	1.50
0.33	0.00	0.25	1.25
0.50	0.25	0.00	1.00
1.50	1.25	1.00	0.00

Figure 1: 0-dimensional barcode for small dataset

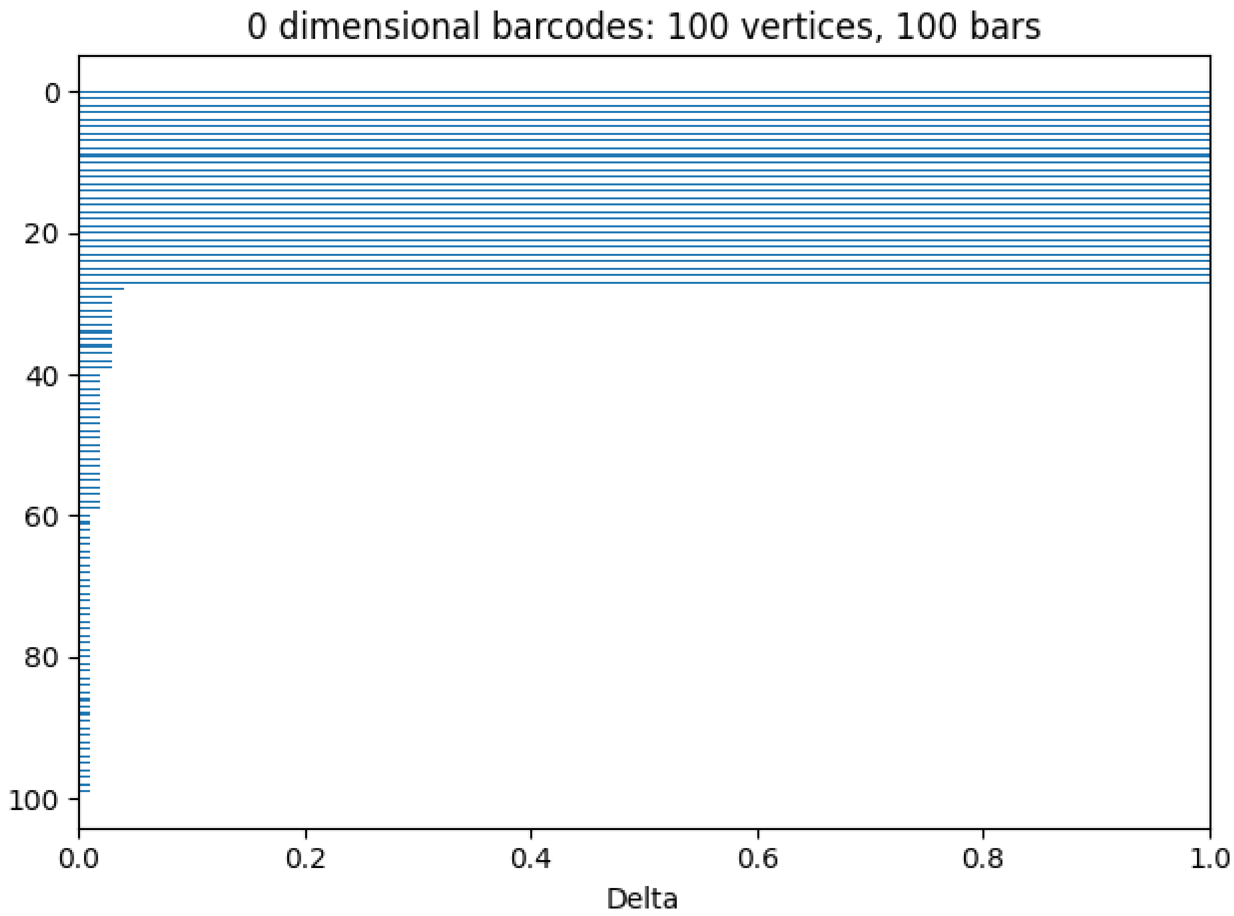


Figure 2: 0-dimensional barcode for large dataset

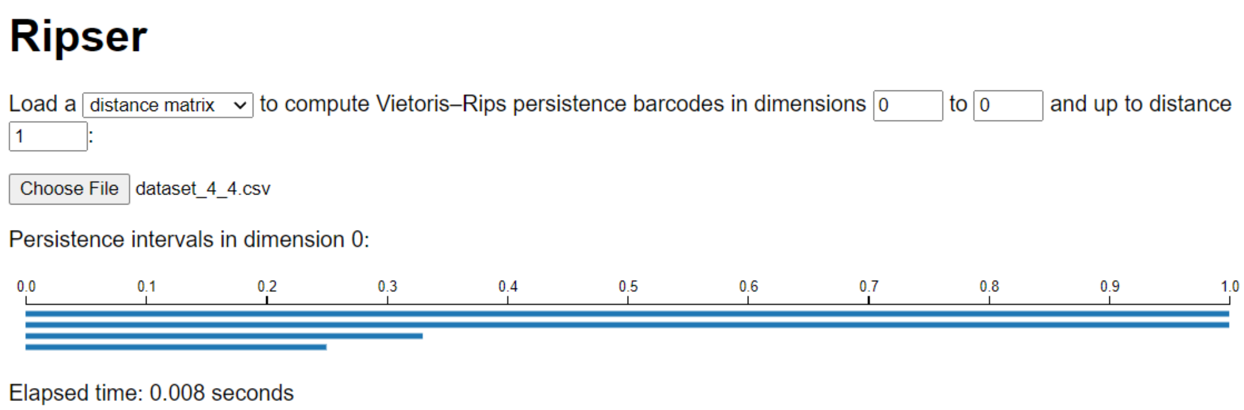


Figure 3: Ripser generated 0-dimensional barcode for same small dataset

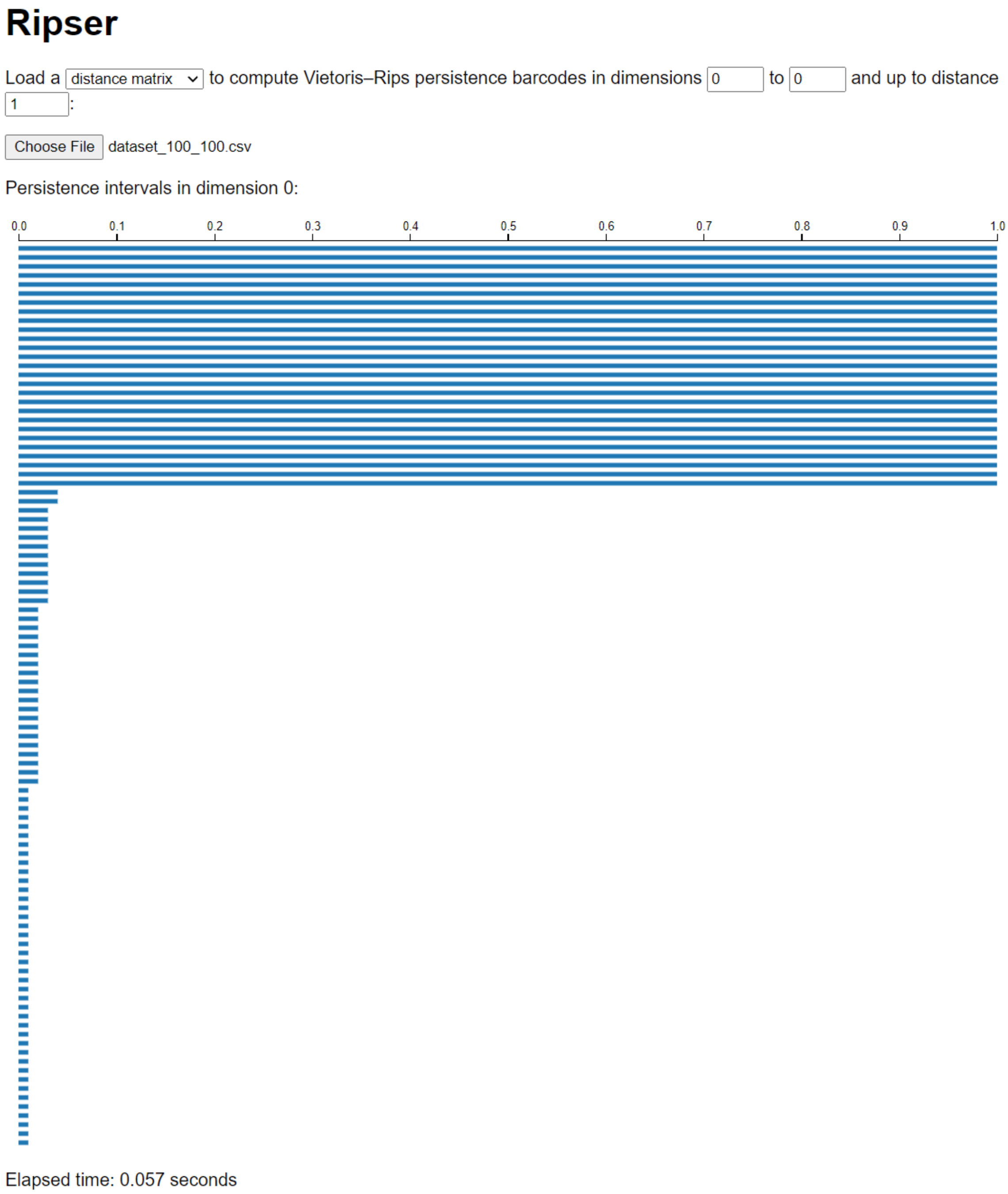


Figure 4: Ripser generated 0-dimensional barcode for same large dataset

References

[1] Mehmet E. Aktas, Esra Akbas, and Ahmed El Fatmaoui. Persistence homology of networks: methods and applications. 4(1), August 2019.

[2] Ashley Suh, Mustafa Hajij, Bei Wang, Carlos Scheidegger, and Paul Rosen. Persistent homology guided force-directed graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):697–707, January 2020. Funding Information: We thank the reviewers for their valuable feedback. This work was supported in part by National Science Foundation grants IIS-1513616 and DBI-1661375, CRA-W Collaborative Research Experiences for Undergraduates (CREU) program, DARPA CHESS FA8750-19-C-0002, and an NVIDIA Academic Hardware Grant. Publisher Copyright: © 2020 IEEE.

[3] Ulrich Bauer. Ripser: efficient computation of vietoris-rips persistence barcodes. *Journal of Applied and Computational Topology*, 2021.