Segmenting Cubical Complexes with Region Growing

Soham Mukherjee

December 11, 2018

Abstract

In this project I used region growing algorithm to segment a Cubical Complex and look at the boundary faces of the regions that are adjacent.

1 Introduction

Recovering hidden data from noisy model is a fundamental task in modern Data Science. In [2] Dey et al. gave an algorithm to recover hidden graph from noisy data. In [1] Dey et al. described region growing for 2-manifolds. But all the algorithms take simplicial complexes as input. In this project I combined the two algorithms together to segment a cubical complex [3]. Handling cubical complexes as opposed to standard simplicial complexes is much intuitive as the bottleneck for triangulation gets removed.

2 Brief Theory

2.1 What is Happening Actually

It is known that for a piecewise-linear function defined on a cubical complex, the persistence pairings can be computed between maxima and saddles (and between minima and saddles) by sweeping the function value from large to small and maintaining the connected regions throughout the course via a simple union-find data structure. The spanning tree algorithm takes $O(n \log n)$ time ($orO(n\alpha(n))$ time if vertices are sorted), where $n$ is the number of vertices and $\alpha(n)$ is the inverse Ackermann’s function, a function that grows extremely slowly. We are interested only in important maxima, that is, those corresponding triangles whose persistence values are more than a prescribed threshold. Instead of computing the complete list of persistence pairings, a region merging algorithm is employed that computes these pairings up to a level dictated by an input threshold parameter $\delta$ as described in [1]. Specifically, this algorithm cancels maxima-saddle or equivalently face-edge pairs (for 3D its cube-face pairs) till only those maxima whose persistence values are more than a threshold are left. As a by product, the merging algorithm also produces an explicit segmentation of the surface.
2.2 Cubical Complex
An elementary interval is a subset $I \subset \mathbb{R}$ of the form $I = [l, l+1]$ or $I = [l, l]$. An elementary cube is the finite product of elementary intervals, i.e. $Q = I_1 \times I_2 \times I_3 \times \ldots \times I_d \subseteq \mathbb{R}^d$. A Set $X \subseteq \mathbb{R}^d$ is a Cubical Complex if it can be written as a union of elementary cubes. An interval is said to be degenerate if it is of the form $[l, l]$ and non-degenerate otherwise.

3 Workflow and Algorithm

3.1 Building Cubical Complex from Data
This step is done by reading a raw binary data and writing in a Perseus [4] style file. GUDHI [5] creates a cubical complex out of it with functional values defined at the vertices. Note that GUDHI takes scalar values defined at top dimensional cells only. I tweaked source code of GUDHI such that it takes values defined at the vertices. For more technical details readers are referred to [5].

![2D Cubical Complex](image1.png) ![3D Cubical Complex](image2.png)

Figure 1: 2D and 3D Cubical Complexes as Input

3.2 Region Growing
As the pairings of top dimensional cells are of interest, the region merging algorithm can be simplified with Kruskal like maximum spanning tree method. If the algorithm is ran with a tunable parameter $\delta$, the output would be a spanning forest as described in [1].

The algorithm partitions the entire complex into regions which are grown iteratively. Each region is denoted by a central cell. As followed from Alexander Duality, the nodes in the spanning forest correspond to top dimensional cells and edges correspond to facets of the complex. Initially all cells are critical. At the initial step we get a bunch of nodes of a graph whose edges are yet to be added.
Algorithm 1 RegionGrow \((K, \rho, \delta)\)

**Input:** Cubical Complex \(K\), density function \(\rho : M \rightarrow \mathbb{R}\), threshold \(\delta\)

**Output:** List of Maximas that correspond to regions

1: \(\mathcal{R} \leftarrow n\) dimensional faces of \(K\) \(\triangleright n\) is the top dim.
2: \(\mathcal{F} \leftarrow n - 1\) dimensional faces of \(K\)
3: Sort \(\mathcal{F}\) in Nondecreasing order of \(\rho(f)\) with \(f \in \mathcal{F}\)
4: for all \(f \in \mathcal{F}\) do
5: \((t_0, t_1) \leftarrow\) Cells sharing \(f\) as their boundary
6: \(\text{Root}_0 \leftarrow \text{Find}(t_0)\)
7: \(\text{Root}_1 \leftarrow \text{Find}(t_1)\)
8: if \(\text{Root}_0 \neq \text{Root}_1\) then
9: \(\text{pers}(f) \leftarrow f - \min\{\rho(\text{Root}_0), \rho(\text{Root}_1)\}\)
10: if \(\text{pers}(f) < \delta\) then
11: \(\text{Root}_0 \leftarrow \text{FindinRegion}(t_0)\)
12: \(\text{Root}_1 \leftarrow \text{FindinRegion}(t_1)\)
13: \(\text{Root} \leftarrow \text{argmax}(\rho(\text{Root}_0), \rho(\text{Root}_1))\)
14: \(R_0 \leftarrow \text{Region belonging to Root}_0\)
15: \(R_1 \leftarrow \text{Region belonging to Root}_1\)
16: \(\text{Merge}(R_0, R_1)\)
17: end if
18: end if
19: \(\text{Merge}(\text{Root}_0, \text{Root}_1)\)
20: end for
21: end if
22: return \(\mathcal{R}\)
23: end

Equivalently we are maintaining two spanning forests. One for computation of persistence (henceforth termed as \(\text{persistence\_tree}\)) and another one for regions (henceforth termed as \(\text{region\_tree}\)). At a generic step when an edge \(e\) comes it joins two forests of \(\text{persistence\_tree}\) rooted at two different vertices, say \(v_0\) and \(v_1\). The new root is chosen to be \(\text{argmax}\{\rho(v_0), \rho(v_1)\}\). Then we can define the persistence of the edge \(e\) as \(\text{pers}(e) = \rho(e) - \min\{\rho(v_0), \rho(v_1)\}\). If the persistence is less than threshold \(\delta\), we join the two forests of \(\text{region\_tree}\) representing regions with the merged region having root at \(\text{argmax}\{\rho(v_0), \rho(v_1)\}\). Should it be greater than \(\delta\), the forests remains separated but nevertheless we join the forests in \(\text{persistence\_tree}\). The complete algorithm is described in Algorithm 1.

However Algorithm 1 is for complexes without boundary. Boundary edges must be taken care of by adding a special vertex with infinite functional value.
Algorithm 2 CollectOutput($\mathcal{F}$)

**Input:** Facets of $K$

**Output:** List of facets separating regions

1: begin
2: $\mathcal{E} = \emptyset$
3: for all $f \in \mathcal{F}$ do
4:   $(t_0, t_1) \leftarrow$ Cells sharing $f$ as their boundary
5:   $\text{Root}_0 \leftarrow \text{Find}(t_0)$
6:   $\text{Root}_1 \leftarrow \text{Find}(t_1)$
7:   if $\text{Root}_0 \neq \text{Root}_1$ then
8:     $\mathcal{E} \leftarrow \mathcal{E} \cup \{f\}$
9: end if
10: end for
11: return $\mathcal{E}$
12: end

Algorithm 2 depicts how to obtain the region boundaries when Algorithm 1 terminates. As the main algorithm is essentially a Kruskal like Spanning Tree Algorithm, data structure required is simply Union-Find. However to store all the connectivity and functional values defined at the cells GUDHI uses a Bitmap like data structured originally proposed by Wagner et.al. in [6].

### 3.3 Output

Output of the program is a bunch of 2-Cubes(faces) each of which is shared by two 3-Cubes that belong to two different regions for 3D cubical complexes. For 2D it is a bunch of edges that separates two regions. Figure 2a and 2b shows the output of the algorithm when viewed through Paraview. The empty regions, i.e. those without borders, are the region that belongs to the special maxima with infinite value which was added to take care of the boundary edges.

(a) Segmentation of 2D Image  
(b) Segmentation of 3D Volumetric Data

Figure 2: Regions in 2D and 3D cubical complex
3.4 Visualizing The Output

For visualization Paraview is used. At the termination of the program .vtp files are generated which can be viewed directly in Paraview. For the raw data dimensions of the data in x,y and z must be known prior to load the raw file.

4 Observation

- The regions are elongated. The sole reason is degeneracy and how GUDHI handles the degeneracy. Better indexing scheme must be looked for to handle such cases.

5 Further Works

Due to time constraints hairs for 2D and the hair sheets for 3D was not shown as the output as promised in the project proposal. I believe the hair structures along with the regions will reveal more information. As mentioned before GUDHI takes function value defined at top dimensional cells and assigns minimum functional value of the cells that share a lower dimensional cell. However this is not a lower-star filtration. For this project I tweaked source code of GUDHI so that it takes scalar values defined at vertices and linearly extrapolating them to higher dimensional cells. I believe the change in the code not efficient.
References


