Homology, Homotopy and Applications, vol. 14(1), 2012, pp.221–238

A FAST ALGORITHM FOR CONSTRUCTING TOPOLOGICAL STRUCTURE IN LARGE DATA

XU LIU, ZHENG XIE AND DONGYUN YI

(*communicated by Gunnar Carlsson*)

Abstract

Discovering and constructing the topological structure in data has attracted the attention within the community of data analysis. However, most methods developed so far are unsuitable for very large sets of data because of their computational difficulties. This paper presents a fast algorithm for constructing the inherent topological structure in large sets of data that might be noisy in order to enhance the MAPPER algorithm introduced by Singh, Mémoli and Carlsson. The limitation of our method, as shown by our experiments, lies with the storage in the main memory rather than the computing time.

1. Introduction

An important feature of modern science and engineering is that data of various kinds are collected at an unprecedented rate. This is particularly true with different sensors installed to collect data streams. At the same time, data obtained currently tend to contain missing values and are noisier than those collected before sensors were used. The development of the relevant methods of analyzing such data, in terms of both the quantity and nature of the available data, has clearly not kept with the pace of how fast the data are collected. To this end, there is a growing interest about how geometry and topology can be applied to the analysis of data [**4, 5, 6, 7, 9, 8, 10, 11, 12, 13, 14, 15, 23, 24, 26, 31, 32**], which has been known as topological data analysis (TDA). A lot of attention has been given to the problem of how to uncover and construct the topological structure in point cloud data, which means such a set of data points that are separated by distance. This problem is ill-posed so far, because different topological structures can be found in the same set of data when different scaling is used. To possibly resolve this problem, Carlsson et al. [**25, 30**] developed a good method along with their software MAPPER.

MAPPER employs a filter function to map a point cloud data set $(X, d(\cdot,\cdot))$ into some well-structured space $(Y, \rho(\cdot,\cdot))$, such as $2D$-Euclidean space, and then splits the space Y into overlapping pieces by covers and clusters the embedded data inside each cover by some clustering algorithm, such as the single linkage method [**30**]. These

Received January 18, 2011, revised February 26, 2012; published on June 7, 2012.
2000 Mathematics Subject Classification: 11F23, 68U05, 62H30.
Key words and phrases: data analysis, topological structure, complex network, clustering, Betti number.
Article available at `http://intlpress.com/HHA/v14/n1/a11` and `doi:10.4310/HHA.2012.v14.n1.a11`

clusters are then taken as a node set for building a simplified network with links generated between clusters with common points in the original data set.

The merit of MAPPER is that different filter functions can be employed to gain insight into any specific aspect of the data, while drawbacks include that one needs to know the topological structure of the filtered space or even the distribution of the filtered data in order to generate an appropriate cover; the cover and the filtered space limit the topological structure extracted by MAPPER. These two drawbacks control the resolution, namely the size of the small patches in the cover and the percentage overlap between successive patches. A different choice of either the topological structure or the distribution make the results of MAPPER very different. The decision of how to choose these two factors is in principle difficult to make. Adding salt to the wound, there is no guidance developed for the general use.

Inspired by the published methods, this paper focuses on extracting topological structures from point cloud data quickly and directly without constructing covers, without any knowledge of the topology of the embedded space, by making use of the distance function only. In other words, topological structures can be extracted from point cloud data even without knowing the explicit representation of the data. Our method uses only a distance matrix and a choice of scale threshold as input to extract the hidden topological structure of the data. A network will be constructed from the data and clustered directly. In our algorithm the data points are treated as abstract objects, as nodes of the network. The corresponding graph of the resultant clustering structure provides us a view of the same topological structure derived from the data as MAPPER [**25, 30**] does. The clustering structure in the network is known as the community structure [**21**] in network analysis.

In general, given a network $NW = (V, E)$, where V is the node set and E the edge set, by community it means such a subset C of V, whose nodes are more closely connected to one another than to the nodes outside C. The major difference of community structures from the traditional clusters as used by MAPPER is that communities can be connected to each other but clusters cannot. Thus simplified connected graphs from community structures can be generated; a topological structure of the original network, developed from the point cloud data with some given connection threshold, can be constructed.

Constructing the community structure in a given network, before anything else, requires a partition of the network into clusters of densely connected nodes so that nodes in different clusters are only sparsely connected. Precise formulations of this optimization problem are known to be NP-complete [**3**]. Several approximation algorithms have therefore been proposed to find reasonably good partitions within a reasonable time frame [**1, 2, 17, 19, 20, 22, 29**]. In this paper, we apply a label propagation approach to detect the topological structures of large point cloud data sets containing probably noisy values. Here is a list of some advantages of our method:

1. The algorithm extracts the topological structure of a large data set and is shown to outperform all other known structure detection methods in terms of computation time. More importantly, the quality of the detected topological structure is very good.
2. The algorithm is extremely time efficient; the algorithm's complexity is $\mathcal{O}(n)$. This is due to the facts that the possible gains in modularity are easy to compute and we typically only need to scan the data just a few rounds.

3. The algorithm is insensitive to the topological structure of the embedded space of the data set. It only depends on the scale parameter and the processing ordering of the data points.

Comparing with MAPPER, our algorithm has two differences:

1. Our algorithm does not need any filter and cover at all. The essential reason for building filters and covers is to keep the information of cluster connections. Our algorithm clusters the data points and maintains their connections simultaneously so that there is no need to generate covers of the embedded space and/or to check the intersection between clusters.
2. Instead of using filters and the associated covers, our algorithm uses the Vietoris-Rips complex as a starting point for a modularity based clustering algorithm.

The rest of this paper is organized as follows. We begin with a discussion on fast methods, which provides the mathematical formalism that permits us to quickly infer topological information from a sample of data of a geometric object. Then we show how this formalism can be applied to particular data sets with noise. After that, we show how our method can decrease the number of the origin data vertices and produce a new data set, not by embedding in a Euclidean space but instead by producing a simplicial complex associated with certain initial information about the data set. This presentation is concluded with how this work might be developed more generally.

Acknowledgements

The authors would like to express their deep gratitude to Professors Yi Wu of National University of Defense Technology, Yi Lin of Slippery Rock University and Yuan Yao of Peking University for their constructive discussions, suggestions and the useful information concerning the content of this paper. This work was funded in part by NSFC (No. 11001237) and NUDT Preparing Research Project (No. JC110204).

2. Elements of our algorithm

In this section, we develop a mathematical formalism useful for constructing topological structures in point cloud data using finite sets of points and distance functions without assuming any knowledge on the topology of the embedded space of the data. In this analysis, the point cloud data are thought to be a finite number of samplings of a geometric object and may contain noise. Tools from various branches of topology are employed to analyze point clouds. Our fundamental method to guarantee that the simplicial complex computes the homology of the geometric object is to produce a homotopy equivalence between the object and the simplicial complex. Since the homology of simplicial complexes is algorithmically computable, it is frequently desirable to construct simplicial complexes that compute or at least have a strong relationship with the homology of the object. There are a number of simplicial complexes that can be constructed from the available data. In our fast method, we begin with the construction of a network for the cloud data. This mathematical formalism permits us to infer topological information from the data set. We then apply the network clustering method to drastically decrease the number of vertices in the cloud point data set. Then, we produce a simplicial complex for the data set by using the Vietoris-Rips complex, a topological concept, by associating with certain initial information, instead of by embedding in a Euclidean space. We then compute the homology that in turn provides the desired structure and information about the data set in question. Here are some of the key steps that come up when applying our fast method to data analysis.

2.1. Building ε-network from data

Denote the point cloud data as $(X, d(\cdot,\cdot))$, where $X = \{X_i \mid i = 1, 2, \ldots, n\}$, and $d(\cdot,\cdot)$ is a distance function defined on the data set X satisfying

$$d(X_i, X_j) = d_{ij} \geqslant 0, 0 < i \neq j < n, n \in \mathbb{N}.$$

We build the skeleton of complexes, namely the network of data, by using a very simple rule: If the distance of two points is no more than some given threshold value $\varepsilon \in \mathbb{R}^+$, then we connect the points with an edge. Since a topological structure only concerns the connection information inherent in the data, the network constructed is undirected. So an edge from V_i to V_j means an edge from V_j to V_i, for any $0 < i \neq j < n$. We thus construct a network $NW_\varepsilon(X) = (V, E)$, where $V = \{V_i \mid i = 1, 2, \ldots, n\}$ is the set of 0-simplexes or nodes of the network and $E = \{(V_i, V_j) \mid d(X_i, X_j) \leqslant \varepsilon, 0 < i \neq j < n\}$ is the set of 1-simplexes or edges of the network. Here we do not allow self-loops because they do not mean anything interesting for the extraction of topological structures.

Our construction of the network depends on the parameter ε and the point cloud data set $(X, d(\cdot,\cdot))$. As for how to choose an ε value, it is in principle a difficult concern, and there is currently not any ready guidance. However, one may run our algorithm multiple times with different ε values, i.e., $0 < \varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_n$ for some $n \in \mathbb{N}$, to see the potential impact of this threshold. This is similar to the strategy adopted by persistent homology [**5**]. Because each computation for a fixed ε runs very fast, the entire procedure can be finished quickly.

2.2. Network clustering by local relabeling

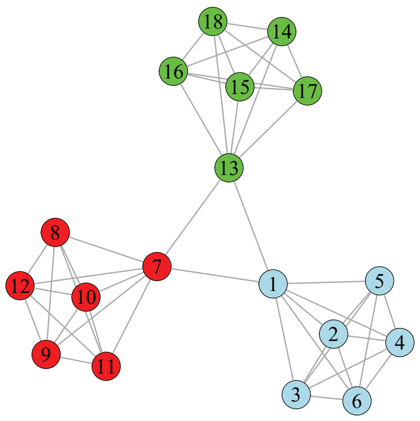
Clustering a network can be seen as assigning labels to the nodes such that the nodes with the same label belong to the same cluster. In the case of point cloud data, we may consider points in one cluster as close enough that the cluster can then be seen as one single node. These higher level nodes together with their links actually provide a sketch of the topological structure hidden in the data. The proposed algorithm is shown in Figure 2, and we are now going to explain it in detail.

A network $NW_\varepsilon(X) = (V, E)$ can be represented by its adjacency matrix $A = (A_{ij})_{n\times n}$ defined as follows:

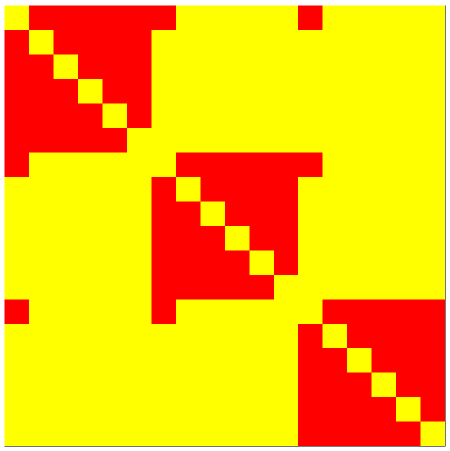
$$A_{ij} = \begin{cases} 1, & \text{if edge } V_iV_j \text{ exists;} \\ 0, & \text{otherwise.} \end{cases}$$

For any given node $V_i \in V$, the total number of edges incident with V_i is called the degree of the node V_i , denoted by $\deg(V_i)$ or k_i for short, which can be computed as

$$k_i = \sum_{j=1}^{n} A_{ij}.$$



(a) A simple network with three communities.



(b) Adjacency matrix.

Figure 1: Example of network with community structure. (a) is the original network, and (b) is the image for its adjacency matrix. See text for details.

Denoting the total number of edges in the network by m leads to

$$m = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}.$$

Assuming that there is no community structure in the network and the edges between the nodes are totally random connections, let us disconnect all the edges and rewire the network. Then the probability of a node with degree k_i connected to a node with degree k_j is [**1**]

$$P_{ij} = \frac{k_i k_j}{2m}.$$

The key point of network clustering is that each community is a subgraph or cluster that is more densely connected than a cluster that is randomly connected. This difference can be measured by comparing the number of the actual connections and the number of expected connections with random rewiring:

$$Q_l = \frac{1}{2m}\sum_{L_i=L_j=l}(A_{ij} - P_{ij}); \tag{1}$$

here l is the identity of the given cluster, and L_i is the label of node $V_i \in V, i = 1, 2, \ldots, n$.

Clustering a set of nodes is a procedure that gives each node a unique label such that the nodes with the same label belong to the same community, while its labeling scheme optimizes a certain quality function used to measure how good the partition is. The most prominent quality function used for network clustering is called

```
1: **Input**:
     $G_{raw} = (V, E)$//a network for analysis
     $max\_loops \in \mathbb{N}$//maximum number of loops supplied by user
     $min\_modularity\_gain \in \mathbb{R}^+$//minimal modularity gain supplied by user
2: **Output**:
     $G_{simple}$//a simplified network
3: for $V_i \in V$ do
4:     $L_i = i$//initiate labeling,each vertex is labeled uniquely
5: end for
6: $num\_of\_loops = 0$//loop counter
7: $modularity\_gain = 1000.0$//a very big value
8: $current\_modularity = modularity(G, L)$//modularity of the initial labeling
9: while $modularity\_gain > min\_modularity\_gain$ or $num\_of\_loops < max\_loops$ do
10:     $vertex\_order = shuffle(V)$//random permutation
11:     for $vertex \in vertex\_order$ do
12:         $NbrLabelSet_{vertex} = \{L_{vertex}\}$//record current label,the current label may be the best label
13:         for $nbr \in Neighbor(vertex)$ do
14:             $NbrLabelSet_{vertex} = NbrLabelSet_{vertex} \bigcup \{L_{nbr}\}$//collect neighbors' labels
15:         end for
16:         $L^{new}_{vertex} = \arg\max_{L \in NbrLabelSet} \sum_{j \in Neighbor(vertex)} (A_{vertex,j} - P_{vertex,j})\,\delta(L, L_j)$//relabeling
17:         $L_{vertex} = L^{new}_{vertex}$//update the label set
18:     end for
19:     $new\_modularity = modularity(G, L)$//modularity of the new labels
20:     $modularity\_gain = new\_modularity - current\_modularity$
21:     $current\_modularity = new\_modularity$
22:     $num\_of\_loops = num\_of\_loops + 1$
23: end while
24: $V_{simple} = unique(L)$//unique labels are used to index the nodes in the simplified network
25: $E_{simple} = \Phi$//empty edge set
26: for $vertex \in V$ do
27:     for $nbr \in Neighbor(vertex)$ do
28:         if $L_{vertex} \neq L_{nbr}$ then
29:             $E_{vertex,nbr} = makeEdge(L_{vertex}, L_{nbr})$//add edge between different clusters
30:             $E_{simple} = E_{simple} \bigcup \{E_{vertex,nbr}\}$
31:         end if
32:     end for
33: end for
34: $G_{simple} = makeGraph(V_{simple}, E_{simple})$
35: return $G_{simple}$
```

Figure 2: Pseudocode for building simplified network

modularity [**21**], which can be written as follows:

$$Q = \frac{1}{2m} \sum_{i=1}^{n} \sum_{j=1}^{n} (A_{ij} - P_{ij}) \delta(L_i, L_j) = \sum_{l \in L} Q_l, \tag{2}$$

where L is the set of all possible labels, i.e., $L = \{L_i \mid i = 1, \ldots, n\}$, and δ is Kronecker's delta function, i.e.,

$$\delta(L_i, L_j) = \begin{cases} 1, & \text{if } L_i = L_j; \\ 0, & \text{otherwise.} \end{cases}$$

An intuitive insight of the community structure in networks is that nearby nodes tend to have the same label in order to keep a high value of the modularity computed out of equation (2). The simple example in Figure 1(a) represents a network with three communities indicated by different colors. Each community is a full connected subgraph, and there are sparse links between the communities. This feature turns out to be the obvious block pattern in Figure 1(b), the image of the adjacency matrix.

This matrix is symmetric, where “red” denotes 1 and “yellow” denotes 0. It can be seen that, except for some nodes that lay between two or more different communities, like nodes 1, 7, and 13 in Figure 1, all other nodes belong exclusively to just one community. For the general network, this pattern may exist but not as sharply as it shows here, and algorithms are needed to extract this pattern.

Here we develop a simple strategy that repeatedly re-labels each given node the same label as its neighbors that best optimizes the modularity. The merit of this method is that all operations can be done locally, saving time and making it suitable for very large data sets. Our algorithm is much like the two phase method as proposed by Vincent et al. [**2**]. The algorithm begins to label every node with a unique label. That is, we have as many communities as the number of nodes in the network. Then the algorithm re-assigns the labels of each nodes in turn by maximizing the modularity contribution computed using equation (1).

In our method, we first randomly order the nodes in the network, i.e., a random permutation of the given order, for instance $\mathit{shuffle}(\{V_1, V_2, V_3, V_4\}) = \{V_4, V_2, V_1, V_3\}$ means that the algorithm will deal with nodes V_4, V_2, V_1, V_3 one after another in order. At each step the procedure will re-label, say V_i, by denoting its neighbors as $\mathit{Nbrs}(V_i) = \{V_j \mid (V_iV_j) \in E\}$. Let $\mathit{NbrLabelSet}_i = \{L_k \mid V_k \in \{V_i\} \cup \mathit{Nbrs}(V_i)\}$ be all possible labels of V_i and its neighbors. The new label of node V_i is chosen from $\mathit{NbrLabelSet}_i$ to satisfy

$$L_i^{new} = \arg \max_{L \in \mathit{NbrLabelSet}_i} \sum_{j \in \mathit{Nbrs}(V_i)} (A_{ij} - P_{ij})\delta(L, L_j). \tag{3}$$

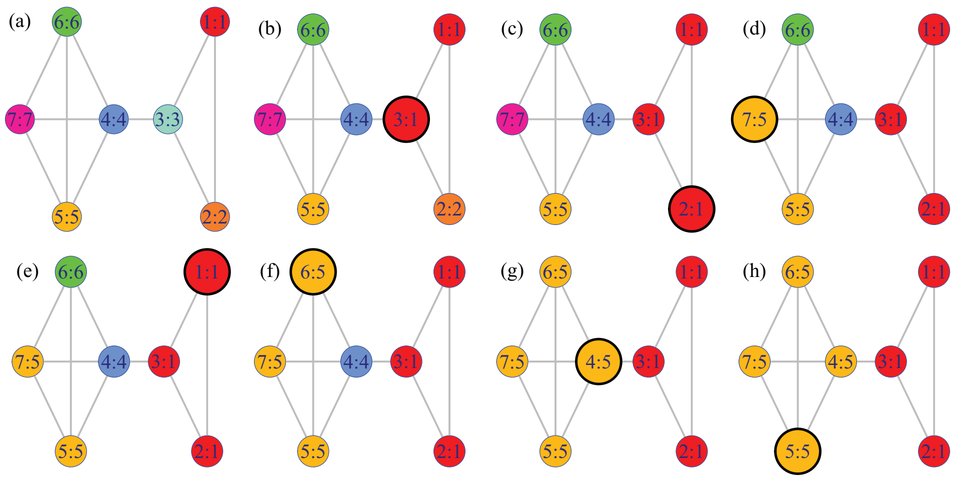


Figure 3: Relabeling of a toy network. Each node is marked by its id and label in the form of $id : label$, i.e., $V_i : L_i$. (a) is the original network with initial labels, where each node belongs to exact one cluster. (b)–(h) indicate the relabeling process. The order of the nodes being relabeled is generated randomly as $V_3, V_2, V_7, V_1, V_6, V_4, V_5$, and the currently relabeled node is enlarged. After one pass of relabeling, the network is split into two clusters. See text for details.

If there are multiple equal maximum labels in the right-hand side of equation (3), then the L_i^{new} is chosen randomly from among these labels. That is, we choose a new label for V_i to maximize its contribution to the modularity on the condition that all other labels stay unchanged. Continue this operation with each node in the network in a random order. After all nodes in the network are treated precisely one time, all labels are adjusted to maximize their contributions to the total modularity in a local fashion. Multiple runs may be taken to make the labeling stable. Or when the modularity stops growing significantly, we stop any further run and produce a labeling of the network with an optimal modularity score.

In order to clearly illustrate our relabeling process, we generate a toy network with seven nodes, consisting of one 3-clique, one 4-clique, and one edge connection (see Figure 3(a)). All nodes are marked by its id and label in the form of $V_i : L_i, i = 1, \ldots, 7$, and colors are used to indicate the labels. So the network is initiated with seven clusters and is shown in seven different colors. Then a random order as $3, 2, 7, 1, 6, 4, 5$ of the nodes is generated. So, the algorithm will re-label node V_3 first, followed by node $V_2, \ldots$ until finally node V_5. Figure 3(b) shows the result of relabeling node V_3. It has three neighbors, i.e., $\mathit{Neighbor}(V_3) = \{V_1, V_2, V_3, V_4\}$, together with three different labels, i.e., $\mathit{NbrLabelSet}_{V_3} = \{1, 2, 3, 4\}$. The respective modularity contributions are $0.7, 0.7, 0.0, 0.4$, and the optimal new label for V_3 is L_1 or L_2 (see equation (3)). As a matter of fact, the algorithm runs a simple random drawing and chooses $L_1 = 1$ as the new label for V_3, i.e., $L_3^{new} = 1$. Figure 3(c) shows the result of relabeling node V_2. We have $\mathit{Neighbor}(V_2) = \{V_1, V_2, V_3\}$ and $\mathit{NbrLabelSet}_{V_2} = \{1, 2\}$, and the corresponding modularity contributions are respectively 1.5 and 0. So $L_2^{new} = 1$. In Figure 3(d) we have $L_7^{new} = 5$ as the result of relabeling V_7. Again double optimal modularity contributions 0.55 are obtained at $L = 5$ and $L = 6$ due to the symmetry of the network. In Figure 3(e) V_1's neighbor has only one unique label that stays the same after the relabeling. In Figure 3(f) we see that the optimal new label for V_6 is 5, because the modularity contributed by relabeling this node as 5 is 1.1, comparing to 0.4 when labeling the node as 4. In Figure 3(g) V_4 has an optimal label with the modularity gain of 1.2 with $L = 5$, and the runner up is only 0.4 with the label $L = 1$. For the last step we get a new label for V_5 as $L_5^{new} = 5$, because $\mathit{NbrLabelSet}_{V_5} = \{5\}$ has only one member. After this relabeling procedure is finished, the toy network is split into two groups, and each group consists of a clique (see Figure 3(h)).

The second phase of our algorithm consists of building a simplified network whose vertices are now the clusters found in the first phase. To do so, the weights of the links between the new vertices can be given by the sums of the weights of all links between vertices in the corresponding clusters. Since the main focus here is to extract connections, we neglect the weight of these links and only build a binary network from the previous clustering result. The weighted simplified network is left for future work. Links between vertices of the same cluster lead to self-loops for this cluster in the new network. Thus they are omitted. (Please see Figure 2 for details.) Once this second phase is completed, it is then possible to reapply the first phase of the algorithm to the resulting weighted network and the iteration continues. However, for some data sets, one application of the procedure is enough to extract the topological structure from the data. The previously detailed procedure is summarized in Figure 4.

Our algorithm takes as input a finite set of data and a parameter ε which is used to generate the ε-network. It then constructs a much smaller simplified network than the

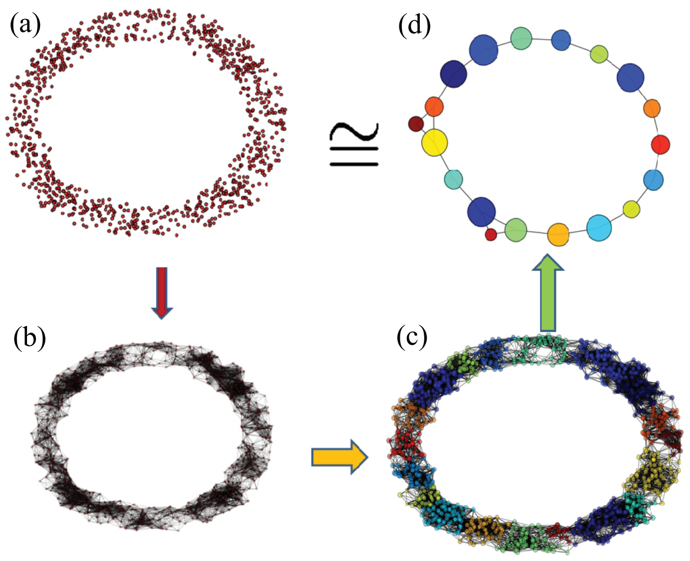


Figure 4: Diagrammatic sketch of clustering of data by relabeling. (a) Point cloud data set drawn randomly from a torus surface. (b) An ε-network built from the data set for some given threshold. (c) Clustering result of the ε-network by relabeling to maximize modularity. Clusters are identified by colors. (d) Simplified network extracted from the clustering result shows the same topological structure hidden in the original data set.

original ε-network, while retaining the most significant topological structure of the point cloud data set. Throughout the computation, the number of meta-communities decreases with each iteration. So, consequently, it can be imagined that most of the computing time is consumed during the first pass. Passes are iterated until there is no more improvement in the modularity so that a maximum modularity is reached or the pre-determined maximum number of allowed passes is used up. The output of our algorithm depends on the order in which the vertices are considered. Thus we shuffle the vertices in each pass in order to avoid any bias possibly introduced by a fixed ordering of the nodes. In the following experiments, the results of several test cases seem to indicate that the idea of shuffling works well and does not influence the final modularity significantly.

In summary, we conclude that the algorithm proposed in this paper has three main advantages:

1. The algorithm can run unsupervised and only a few parameters are needed. Especially, the number of clusters can be automatically decided by the algorithm during the optimization procedure in the form of relabeling.
2. The number of clusters decreases drastically after just a few passes so that most of the computing time is spent on the first pass.
3. The algorithm is extremely fast and its time complexity is linear against the number of links in the ε-network. For a typical sparse data set, the number of

links is linear against the number of data points, leading to a linear computing time.

3. Results

In this section we will test the proposed method method on computer generated point cloud data sets, both with and without noise, in order to verify the effectiveness of the network clustering procedure, and a real world point cloud data set will also be tested. The results of the tests show that the proposed method efficiently enhances MAPPER in terms of extracting topological structure. Our implementation is in GNU C and utilizes GraphViz [**16**] for visualization. All tests run on a laptop with Intel®Core i5 M 430 CPU @2.27 GHz and 2.9GB available memory, using a single thread.

3.1. Experiments on point cloud data set without noise

In this subsection we generate random data from three overlapping circles and apply the proposed method to reconstruct it. We generate three circles together (see Figure 5) and use MAPPER together with the proposed method to reconstruct it. The result shows that our method performs better than MAPPER. (We have conducted a similar experiment on data from a simple unit circle. Both MAPPER and our method perform equally well on the reconstruction of a single unit circle, and we omit it here due to space limitations.)

The data set X consists of 900 points embedded in the $2D$-Euclidean plane and each circle consists of 300 points. The usual Euclidean distance $d(\cdot,\cdot)$, together with X, composes the point cloud data set $(X, d(\cdot,\cdot))$.

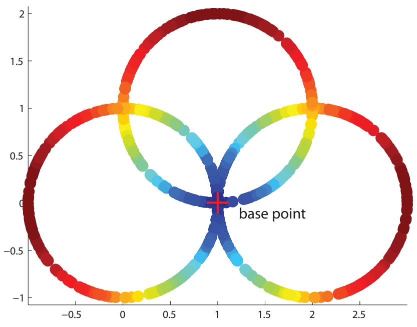
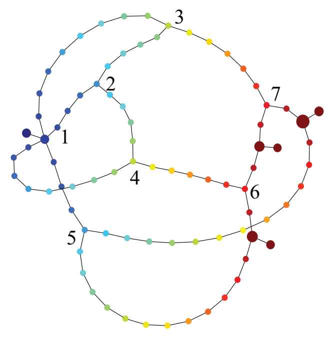
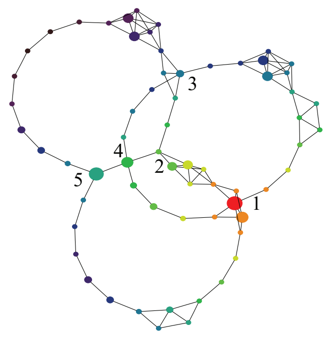


Figure 5: Samples from three circles. The base point in the center used by MAPPER is marked as a red cross. The colors of the points indicate their distances to the base point.

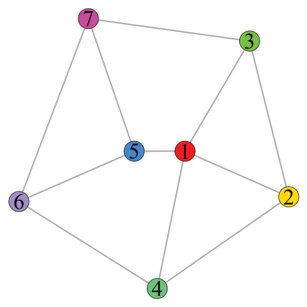
The result of MAPPER is shown in Figure 6(a). The filter function for a node in the data set is chosen as its distance from the predetermined base point, which is marked as a red cross in the center. So, the interval $[0, 2.2] \in \mathbb{R}$ is the mapped space. We choose the cover of this mapped space as 21 overlapping intervals such that the percentage of overlap for each pair of adjacency intervals is 50%, i.e., $c_k = [0.01(k-1), 0.01(k+1)], k = 1, \ldots, 21$. We also try other parameter settings and run MAPPER multiple times. The above parameters setting gets the most prominent performance, shown in Figure 6(a).



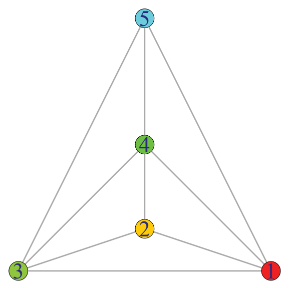
(a) The result by MAPPER.



(b) The result by our method.



(c) Connecting graph of the generators of the first homology classes for simplicial complex in (a).



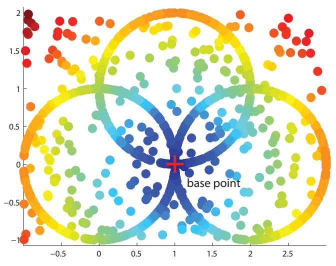
(d) Connecting graph of the generators of the first homology classes for simplicial complex in (b).

Figure 6: Experimental results for the three toruses in Figure 5. The nodes with numbers are the generators of the first homology classes of the simplicial complex, i.e., the simplified network.

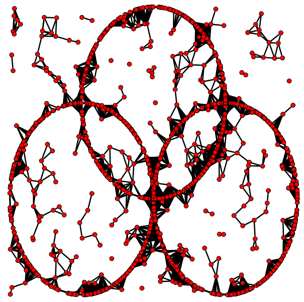
When applying the proposed method, we set $\varepsilon = 0.2$ to generate the ε-network and run the relabeling procedure with only one pass, where the minimal modularity gain is set to a very small value: $0 : 000001$. Because the procedure is run with only one pass, this parameter takes no actual effects. The simplified network, see Figure 6(b), contains the same first simplicial homology group as the three overlapping circles from where the data is sampled. To see this end clearly, we further simplify the result in Figure 6(a) and (b) into that shown in Figure 6(c) and (d), respectively. Both of them have the same homological group and the 1-dimensional Betti number: $\beta_1 = 5$. However, if one is not content with the recovery of the Betti number of the homological groups of the point cloud data, but also likes to honestly recover other topological features of the data, then our proposed method outperforms MAPPER in terms of generating the optimal generator of the homology classes. The three overlapping circles have five generators, i.e., the five cross points between the circles; see Figure 5. However, for the 1-dimensional homological group, MAPPER extracts seven generators. In comparison, our method extracts five generators. Our method not only recovers the correct homology group of the original point cloud data, but it also correctly extracts the homology generators. This simple experiment shows that our method indeed enhances the accuracy of MAPPER, and it can better express the topological property of the original geometric object.

3.2. Dealing with noisy data

In this subsection we proceed to evaluate the performance of the proposed method on random point cloud data with noise. The results indicate that although MAPPER fails to deal with noisy data, our proposed method can reconstruct the topology of the data set if the data contains only a mild degree of noise. To this end, one may argue that the noise could be filtered out before a topological reconstruction algorithm is applied. However, in the following experiments we apply our method directly to the noisy point cloud data without first employing any specialized outlier detection and/or smoothing treatment. Even so, the result shows that our method is quite robust and can significantly alleviate the effects of the noise, at least for the data set used here.



(a) Sample on three circles with noise.



(b) ε-network built from (a).

Figure 7: The base point in the center used by MAPPER is marked as a red cross. Colors of the points indicate the points' distances to the base point.

We add 300 randomly sampled data points from $[-1, 3] \times [-1, 2] \in \mathbb{R}^2$ to the three crossed circles used in the last experiment to generate a noisy point cloud data set; see Figure 7. The global noise imposes quite a big challenge to the topology reconstruction algorithm, because the noise may lead to the generation of many fake connections and mislead the algorithm. If the noisy points are removed, then MAPPER will certainly cluster the data correctly, as we can see from the previous experiment. However, in this experiment the noisy points act as bridges and actually connect the separated clusters to form a single cluster. MAPPER produces the result in Figure 8(a). Like the previous experiment, we have tried several different parameters and only report the most meaningful result here. The most prominent cover consists of 21 equal-length intervals with 50% overlapping percentage between adjacent intervals. From the result it can be seen that the true circles disappear while some fake circles emerge. So, we

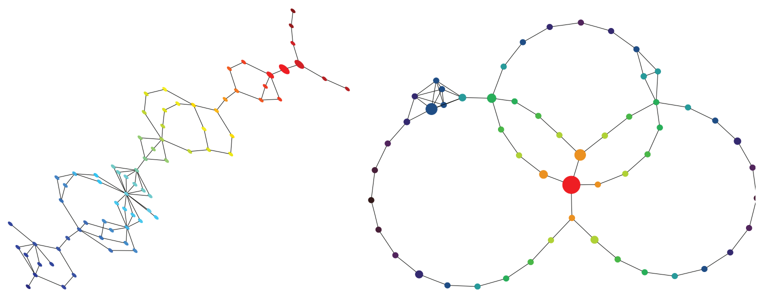

(a) Result by MAPPER. (b) Result by the proposed method.

Figure 8: Experimental results for the noisy data.

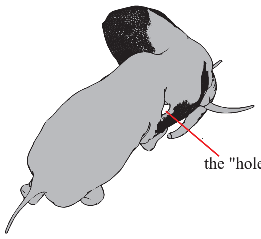
may conclude that MAPPER totally fails to find the most significant topological structure due to the large amount of noise, at least for the data set tested here.

We apply our method to generate the ε-network with $\varepsilon = 0.2$, see Figure 7(b), and run the relabeling procedure for just one pass. Here the minimal modularity gain is set to a very small value: 0.000001. The resultant simplified network is shown in Figure 8(b). From the result, it can be seen that our method preserves the most significant topological structure hidden in the original data set.

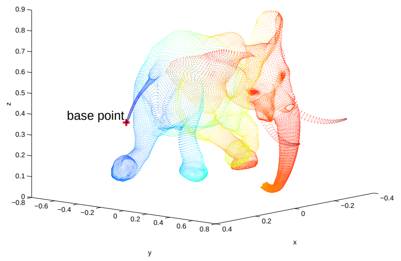
3.3. Test on 3D shape data

In this subsection we proceed to evaluate the performance of the proposed method on point cloud data sampled from a "real-world" 3D shape, which is one of the seven objects from a publicly available database [**27**]. We only test one of the elephant shapes shown in Figure 9, and tests on the rest of the objects in the database are very similar and we omit them here. Roughly speaking, the shape data is topologically equal to a 2D-spherical surface in the sense that it has $\beta_0 = 1$, $\beta_1 = 0$ and $\beta_2 = 1$ as the first three Betti numbers. It is very interesting that our experiments tell us that the elephant is slightly different from the 2D-spherical surface. When we apply the proposed method to it, we get an estimation $\hat{\beta}_1 = 1$, which greatly violates our intuition. Fortunately, we carefully check the elephant shape by actually rendering it by *Ink9000*. (This is an open source mesh renderer available at `http://inkulator.sourceforge.net/`); see Figure 9(a). We find that the end of its right ear is actually attached together with its right shoulder and creates a 1-dimensional hole in the shape. Thus we have $\beta_1 = 1$, and our method gets the right answer. We will see this "hole" later in the simplified network.

This 3D elephant shape is in the form of triangulated meshes, and we only take the Cartesian coordinates of the points. There are 42,321 points and the distance matrix of them cost about 13GB memory, so we cannot use all of them as input for MAPPER since its implementation relies on the distance matrix. Fortunately, we do not need to use all these data, and by down sampling we will get the same answer [**23**]. So we randomly choose 10,000 nodes. Like the previous tests, we choose one node as the base point and run MAPPER to get the simplified network. The filter used by MAPPER is chosen as the distance to the base point, and the base point is the point that is the farthest one from all the reset points in the 10,000 sampled points. That



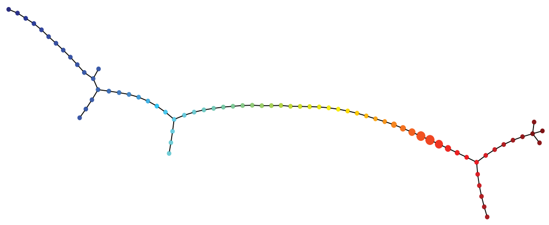
(a) The elephant shape rendered by *Ink9000*.



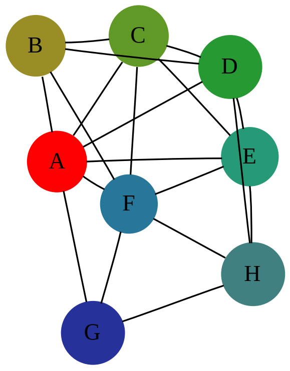
(b) Samples from surface.

Figure 9: The elephant shape.The base point in the elephant's tail used by MAPPER is marked as a red cross. Colors of the points indicate their the distances to the base point.

is to say, for each point, we calculate the sum of its distances between all the other points, and the base point is chosen as the node where the sum attains the maximum. The base point turns out to be the very point at the end of the tail of the elephant; see Figure 9(b). This filter function maps all the points into the interval $[0, 1.39]$, and we use 61 overlapping intervals to cover it. Again the percentage of overlap for each pair of adjacency intervals is 50%. (Please see Figure 10(a) for the result returned by MAPPER.) Using the simplified network, we get $\hat{\beta}_0 = 1$, $\hat{\beta}_1 = 0$ and $\hat{\beta}_2 = 0$ as the estimate of the first three Betti numbers from which we can see that only the first order homology group is preserved by MAPPER.



(a) Result by MAPPER.



(b) Result by the proposed method.

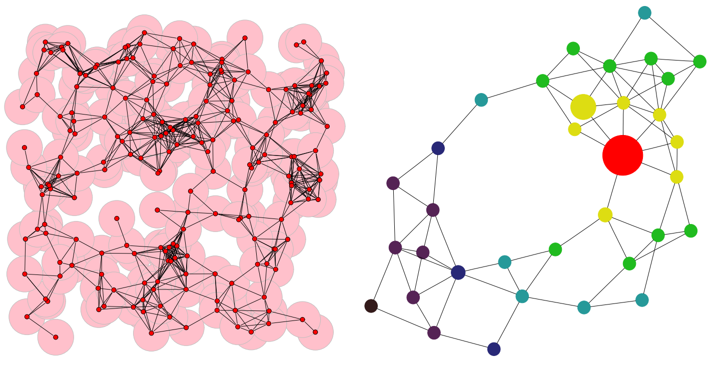
Figure 10: Experimental results for the elephant shape.

When we apply our proposed method, we use all 42,321 points since the proposed method is more memory efficient due to the fact that it does not explicitly store the distance matrix. So all the data can be processed and no sampling is needed here. Also, our method runs much faster than MAPPER, and we soon find the optimal scale parameter. The result shown in Figure 10(b) is obtained by $\varepsilon = 0.05$, and the minimal modularity gain is 0.000001. The relabeling procedure runs 35 pass, and the modularity attains its maximum. The extra relabeling procedures are caused by the fact that the points in the elephant shape are much more numerous than the points in the previous tests. Thus, it needs more rounds to reduce the number of nodes in the simplified network. From the simplified network we build the Vietoris-Rips complex and get $\hat{\beta}_0 = 1$, $\hat{\beta}_1 = 1$ and $\hat{\beta}_2 = 1$ as the estimations of the first three Betti numbers, where $\hat{\beta}_0 = 1$ means the simplified network is connected and $\hat{\beta}_1 = 1$ is caused by the quadrangle ADHF in Figure 10(b). Interestingly, one may see that it corresponds to the right ear "hole" of the elephant; see Figure 9(a). Another interesting fact is that $\hat{\beta}_2 = 1$ is caused by the irregular convex octahedron ABCDEF. To see it clearly, we can classify the eight faces into two categories: the four *front* side ones and the four *back* side ones, all of them are triangles.The four front side ones are $\triangle$ABF, $\triangle$BCF, $\triangle$CEF and $\triangle$CDE. They cover the hexagon ABCDEF completely. The four back side ones are: $\triangle$AEF, $\triangle$ADE, $\triangle$ACD and $\triangle$ABC. They again cover the hexagon ABCDEF completely. So the above eight triangle faces actually seal an irregular convex octahedron inside and thus create a void in the Vietoris-Rips complex. Thus we have $\hat{\beta}_2 = 1$. To sum up, we can see that our method recovers the first three homology groups of the elephant shape, and MAPPER only retains the first one.

4. Conclusion and future work

In this paper we conduct a clustering operation on the network or simplicial complex, constructed from a point cloud data set, in order to generate a simplified simplicial complex that preserves the most significant topological structure of the data. This study enhances the previous work by Carlsson et al. [**4**], namely the MAPPER framework. MAPPER uses a filter function, typically the distance function, to map the data into a well-structured space. Then it applies a cover to split the mapped space into overlapping patches. A simplified simplicial complex is then obtained by constructing connections between the clusters with common points in the original space. However, the choices of the filter function and the cover of the mapped space are not easy, and there is no general guidance for users of MAPPER to follow. Any inappropriate choice of the filter function and cover will lead to the failure of reconstructing the topological structure hidden in the data. Moreover, MAPPER needs special pre-processes when noise exists in the point cloud data. Without the necessary pre-processes, MAPPER will produce totally misleading results.

The method proposed in this paper greatly remedies these drawbacks of Mapper. The only key parameters needed for our algorithm is the connection threshold of the ε-network and the ordering of the nodes being processed in the clustering procedure. Comparing to Mapper, these two limitations are much less troublesome since they can be alleviated fairly easy. For the choice of scale parameter ε, one may run our fast algorithm multiple times for different scales to choose the best one. Also, the effects of the ordering can be alleviated by running the algorithm multiple times, using totally random ordering independently. For example, for some circumstances,



(a) Simple sensor network. (b) The simplified coverage network.

Figure 11: Detecting coverage holes in a simple sensors network. (a) A simple network consists of 200 sensors. Sensors with overlapping coverage are connected to form a ε-network. (b) The simplified network produced by using our method shows the most significant holes of the sensors network.

such as the simple sensors (see Figure 11), the scale parameter is fixed; the user does not need to set it at all.

A simple sensors network consists of thousands of inexpensive mobile sensors [**18, 28**]. The problem of detecting coverage holes in such a network with no location information is a crucial task of engineering concern. It reflects how well a region of interest is monitored or tracked by sensors. Each sensor broadcasts within a nearby disk region of a given radius. So this radius gives rise to a ε-network; see Figure 11(a). We may refer to this ε-network as the coverage network of the sensors, in which any subset of nodes that are in pairwise communication cover their entire convex hull. The Rips complex corresponding to this communication graph of the sensor network is then used to detect coverage holes, and the first homology group of this simplicial complex provides sufficient information about the coverage [**28**]. The most significant holes of the coverage network are preserved by our network clustering method; see Figure 11(b). Because the network shown here is quite small with only 200 nodes, the relabeling procedure only needs to be run once to converge. Only when the network of concern is sufficiently large, i.e., with tens of thousands nodes, one may need to run the relabeling procedure several rounds.

Another issue in our proposed method is that the label propagation aims at maximizing the modularity of the network clusters and is not in direct service of preserving the topological feature of the simplicial complex. So, one possible future work will be looking at how to develop a homology preserving relabeling algorithm.

References

[**1**] R. Albert and A.-L. Barabási, Statistical mechanics of complex networks, *Rev. Mod. Phys.* **74** (2002), no. 1, 47–97.

[**2**] V.D. Blondel, J.-L. Guillaume, R. Lambiotte and E. Lefebvre, Fast unfolding

of communities in large networks, *J. Stat. Mech.* (2008), P10008.

[3] U. Brandes, D. Delling and M. Gaertler et al., On modularity clustering, *IEEE Trans. on Knowledge and Data Engineering* **20** (2007), no. 2, 172–188.

[4] E. Carlsson, G. Carlsson and V. de Silva, An algebraic topological method for feature identification, *Internat. J. Comput. Geom. Appl.* **16** (2006), no. 4, 291–314.

[5] G. Carlsson, Topology and data, *Bull. Amer. Math. Soc. (N.S.)* **46** (2009), no. 2, 255–308.

[6] G. Carlsson, T. Ishkhanov, V. De Silva and A. Zomorodian, On the local behavior of spaces of natural images, *Int. J. Computer Vision* **76** (2008), no. 1, 1–12.

[7] G. Carlsson and F. Mémoli, Persistent clustering and a theorem of J. Kleinberg `arXiv:0808.2241`.

[8] G. Carlsson, A. Zomorodian, A. Collins and L.J. Guibas, Persistence barcodes for shapes, *Int. J. Shape Modeling* **11** (2005), no. 2, 149–187.

[9] G. Carlsson, G. Singh and A. Zomorodian, Computing multidimensional persistence, *Algorithms and computation*, Lecture Notes in Comput. Sci. **5878** (2009), 730–739.

[10] G. Carlsson and A. Zomorodian, The theory of multidimensional persistence, *Discrete Comput. Geom.* **42** (2009), no. 1, 71–93.

[11] D. Cohen-Steiner, H. Edelsbrunner and J. Harer, Stability of persistence diagrams, *Discrete Comput. Geom.* **37** (2007), no. 1, 103–120.

[12] D. Cohen-Steiner, H. Edelsbrunner, J. Harer and Y. Mileyko, Lipschitz functions have L_p-stable persistence, *Found. Computat. Math.* **10** (2010), no. 2, 127–139.

[13] A. Collins, A. Zomorodian, G. Carlsson and L.J. Guibas, A barcode shape descriptor for curve point cloud data, *Computers & Graphics* **28** (2004), no. 6, 881–894.

[14] J.G. Dumas, F. Heckenbach, D. Saunders and V. Welker, Computing simplicial homology based on efficient Smith normal form algorithms, in *Algebra, geometry, and software systems*, 177–206, Springer-Verlag, New York, 2003.

[15] H. Edelsbrunner, D. Letscher and A. Zomorodian, Topological persistence and simplification, *Discrete Comput. Geom.* **28** (2002), no. 4, 511–533.

[16] J. Ellson, E. Gansner, L. Koutsofios, S. North and G. Woodhull, Graphviz—open source graph drawing tools, *Lecture Notes in Comput. Sci.* **2265** (2002), 594–597.

[17] S. Fortunato, Community detection in graphs, *Phys. Rep.* **486** (2010), no. 3-5, 75–174.

[18] R. Ghrist and A. Muhammad, Coverage and hole-detection in sensor networks via homology, *Proc. of the 4th International Symposium on Information Processing in Sensor Networks*, 2005, 254–260, IEEE Press, Piscataway, NJ.

[19] R. Lambiotte, V.D. Blondel, C. De Kerchove, E. Huens, C. Prieur, Z. Smoreda and P. Van Dooren, Geographical dispersal of mobile communication networks, *Phys. A* **387** (2008), no. 21, 5317–5325.

[20] A. Lancichinetti, S. Fortunato and J. Kertész, Detecting the overlapping and hierarchical community structure in complex networks, *New J. Phys.* **11** (2009), no. 3, 033015.

[21] M.E.J. Newman, Modularity and community structure in networks, *Proc. Nat. Acad. Sci. USA* **103** (2006), no. 23, 8577–8582.

[22] M.E.J. Newman, A.-L. Barabási and D.J. Watts, *The structure and dynamics of networks*, Princeton Univ. Press, Princeton, NJ, 2006.

[23] V. de Silva and G. Carlsson, Topological estimation using witness complexes, *Eurographics Symposium on Point-Based Graphics*, ETH-Zürich, Switzerland, June 2–4, 2004.

[24] V. de Silva and R. Ghrist, Coverage in sensor networks via persistent homology, *Algebr. Geom. Topol.* **7** (2007), 339–358.

[25] G. Singh, F. Mémoli and G. Carlsson, Topological methods for the analysis of high dimensional data sets and 3D object recognition, *Eurographics Symposium on Point-Based Graphics* (2007), 91–100.

[26] G. Singh, F. Mémoli, T. Ishkhanov, G. Sapiro, G. Carlsson and D.L. Ringach, Topological analysis of population activity in primary visual cortex, *J. Vision* **8** (2008), no. 8, article 11.

[27] R.W. Sumner and J. Popovic, Mesh data from deformation transfer for triangle meshes, `http://people.csail.mit.edu/sumner/research/deftransfer/data.html`.

[28] A. Tahbaz-Salehi and A. Jadbabaie, Distributed coverage verification in sensor networks without location information, *IEEE Trans. Automat. Control* **55** (2010), no. 8, 1837–1849.

[29] K. Wakita and T. Tsurumi, Finding community structure in mega-scale social networks, *Proc. of the 16th International Conference on World Wide Web*, 1275–1276, 2007, ACM, New York.

[30] Y. Yao, J. Sun, X. Huang, G.R. Bowman, G. Singh, M. Lesnick, L.J. Guibas, V.S. Pande and G. Carlsson, Topological methods for exploring low-density states in biomolecular folding pathways, *J. Chem. Phys.* **130** (2009), no. 14, article 144115.

[31] A. Zomorodian and G. Carlsson, Computing persistent homology, *Discrete Comput. Geom.* **33** (2005), no. 2, 249–274.

[32] A. Zomorodian and G. Carlsson, Localized homology, *Comput. Geom.* **41** (2008), no. 3, 126–148.

Xu Liu `liuxuwork@gmail.com`
Zheng Xie `lenozhengxie@yahoo.com.cn`
Dongyun Yi `dongyunyi@nudt.edu.cn`

Department of Mathematics and Systems Science, National University of Defense Technology, Changsha, Hunan 410073, China