Homology-Preserving Multi-Scale Graph Skeletonization Using Mapper on Graphs

Paul Rosen, Mustafa Hajij and Bei Wang

Abstract

Node-link diagrams are a popular method for representing graphs that capture relationships between individuals, businesses, proteins, and telecommunication endpoints. However, node-link diagrams may fail to convey insights regarding graph structures, even for moderately sized data of a few hundred nodes, due to visual clutter. We propose to apply the mapper construction—a popular tool in topological data analysis—to graph visualization, which provides a strong theoretical basis for summarizing the data while preserving their core structures. We develop a variation of the mapper construction targeting weighted, undirected graphs, called mapper on graphs, which generates homology-preserving skeletons of graphs. We further show how the adjustment of a single parameter enables multi-scale skeletonization of the input graph. We provide a software tool that enables interactive explorations of such skeletons and demonstrate the effectiveness of our method for synthetic and real-world data.

Index Terms—Graph visualization, topological data analysis, mapper, skeletonization, multi-scale visualization

1 INTRODUCTION

Graphs are often used to model relationships in social, biological, and technological systems. In recent years, our ability to collect and archive such data has far outpaced our ability to understand them. The challenges for graph visualizations are two-fold: how to effectively extract features from such large and complex data; and how to design effective visualizations to communicate these features to the users?

One approach to addressing this problem has been graph aggregation, which creates a skeleton of a graph via node [94] and edge clustering [29, 43]. Common node clustering techniques include spectral methods [34, 45, 67, 106], similarity-based aggregation [100], community detection [81, 82], random walks [60, 89], and hierarchical clustering [16, 20]. There are two limitations to these existing approaches. First, they make no guarantees about the preservation of homological features (i.e., components and tunnels) in the original graph (see Fig. 1). Second, they provide only a single perspective on the aggregation of the graph, driven by the objective of the clustering algorithm.

We propose to address these challenges by adapting the mapper construction [98]—a tool from topological data analysis (TDA)—to develop homology-preserving skeletonizations of graphs for visualization. The original mapper algorithm has enjoyed tremendous success
in data science, e.g., cancer research [84], phonemics [62, 108], and others [76, 101].

Our approach works by first applying a topological lens to the input graph that captures a certain property of the graph. We consider topological lenses that preserve graph-theoretic properties such as symmetries, density, and centrality. We then perform a series of operations to skeletonize the graph, producing a topological summary. The benefits of our approach are three-fold. First, it provides guarantees on preserving the homology of the graph it is summarizing, assuming certain sampling conditions [19, 79]. Second, the topological lens allows observing the graph from multiple perspectives, each highlighting different aspects of the graph being studied. Finally, the mapper construction connects naturally with visualization by providing a strong theoretical basis for multiscale simplifications of the input graph while preserving its core homological structures. In this paper, we develop and evaluate a variation of the mapper construction targeting weighted undirected graphs; the method is referred to as mapper on graphs, which produces mapper graph as output. Specifically:

- We describe a series of modifications required to make the mapper algorithm effective on graph data;
- We study the utility of live graph-specific topological lenses;
- We demonstrate the effectiveness of our method on synthetic and real-world data;
- We perform a human-subject study showing mapper on graphs produces good quality skeletons.

Finally, we provide an open-sourced implementation together with our experimental datasets via GitHub.

2 RELATED WORK

2.1 Graph Visualization

We limit our review to node-link diagrams, which are utilized by many visualization software tools, including Gephi [8], GraphViz [42], and NodeXL [56]. For a comprehensive overview of graph visualization techniques, see [105]. One of the biggest challenges with node-link diagrams is visual clutter, which has been extensively studied in graph visualization [41]. It is addressed in several ways: improved node layouts, edge bundling, lenses, and alternative visual representations.

Tutte [102] provided the earliest graph layout method for node-link diagrams, followed by methods driven by linear programming [50], force-directed embedding [46, 59], graph metric embedding [48], and connectivity structures [17, 63, 65, 66]. TopoLayout [2] creates a hybrid layout by decomposing a graph into subgraphs based on their topological features, including trees, complete graphs, bi-connected components, and clusters, which are grouped and laid out as meta-nodes. The difference between TopoLayout and our work is that we use functions defined on the graph to automatically guide decomposition and feature extraction among subgraphs, and multiple functions induce graph skeletalizations, capturing different properties of the graph.

Edge bundling, which bundles adjacent edges together, is commonly used to reduce visual clutter on dense graphs [57]. For massive graphs, hierarchical edge bundling scales to millions of edges [47], while divided edge bundling [95] tends to produce higher-quality visual results. Nevertheless, these approaches only deal with edge clutter, not node clutter, and they only support limited types of analytic tasks [6, 78].

The concept of applying lenses to view large and complex data has also been used in graph visualization. An early example is the topologically-driven “fish-eye views” for visualizing large graphs [49]. Another example is the TugGraph used to explore neighborhoods and paths extending from the foci of interest within a large graph [4, 51].

Finally, alternative visual representations have been used to remove clutter, ranging from variations on node-link diagrams, such as replacing nodes with modules [40] and motifs [39], to abstract representations, such as matrix diagrams [36] and graph statistics [61].

2.2 Graph Clustering

The objective in node clustering (or graph clustering) is to group the nodes of the graph by taking into consideration their edge structure [94]. Vehlow et al. provided a recent survey on the visualization of clustered structures [104] (see Sec. 1 for examples). Edge clustering has also been studied [29, 43]. Importantly, in their assessment of graph readability, Archambault et al. [5] concluded that only clustering could be efficiently performed on large graphs. Broadly speaking, our approach is a type of graph clustering that simultaneously preserves relationships between clusters to retain the topological features of the graph.

Several approaches have leveraged clustering and hierarchical relationships to improve exploration within graphs. For example, van Ham and van Wijk provided methods for interacting with clusters in graph visualizations [103]. Abelino et al. built a system, ASK-GraphView, which enables interacting with very large graphs using clustering, among other techniques [1]. Grouse and GrouseFlocks are systems designed to ease the exploration through a feature topology-based hierarchy of the graph [3]. Batagelj et al. created the (x,y)-clustering, with the goal of generating visualizations where intercluster (x) and intracluster (y) graphs retained desired topological properties [9].

2.3 TDA in Graph Analysis and Visualization

Persistent homology (the study of topological features across multiscales) and mapper construction are two of the most widely used tools in TDA. Persistent homology has been used to analyze graphs [37, 58, 86, 87], with applications to collaboration networks [7, 24] and brain networks [25, 30, 69–72, 88]. In terms of graph visualization, persistent homology has been used in capturing changes in time-varying graphs [55], as well as supporting interactive force-directed layouts [38, 99]. It has also been used to simplify and visualize hypergraphs [109].

The mapper construction [98], which provides a multiscale skeletonization of point clouds has been widely utilized in TDA for a number of applications [21, 75, 76, 84, 101]. It has witnessed recent major theoretical developments (e.g., [23, 33, 79]) that further adjudicate its use in data analysis and provide mathematical foundations for its homology-preserving properties. Recently, Bodnar et al. considered learning the lens of a mapper graph using graph neural networks [13], which is based upon a preprint draft of this work on applying mapper to graphs [54].

Table 1: Topological lens comp. scalability and properties preserved
Density estimation helps to differentiate dense regions from sparse regions and outliers. The density estimation function [97] is given by:

$$D_\delta(v) = \sum_{u \in V} \exp\left(-\frac{d(u, v)^2}{\delta}\right),$$

where $d(u, v)$ is the geodesic distance between two nodes in the graph and $\delta > 0$. $D_\delta$ tends to differentiate dense regions from sparse regions and outliers, and interestingly, it tends to negatively correlate with the AGD. We set $\delta = 0.5$ for all examples. See Fig. 2b for an example.

### 3.1.3 Eccentricity

Eccentricity measures the maximum distance from one node to all others. The eccentricity of a graph node is given by:

$$\text{Ecc}(v) = \max_{u \in V}(d(u, v)),$$

where $d(u, v)$ is the geodesic distance between two nodes in the graph. Eccentricity measures centrality on the graph, since nodes that are towards the center need to travel a shorter distance to reach all other nodes than those on the periphery. See Fig. 2c for an example.

### 3.1.4 Eigenfunctions of the Graph Laplacian

Eigenfunctions of the graph Laplacian $L$ capture spectral properties of the graph [68]. In particular, the gradient of the eigenfunctions of $L$ tends to follow the overall shape of the data [74], and these functions have been used in applications, such as graph understanding [96], segmentation [92], spectral clustering [83], and min-cut problems [77].

Let $C(G)$ be the vector space of all functions, $f : V \to \mathbb{R}$. The unnormalized graph Laplacian of $G$ is the linear operator $L : C(G) \to C(G)$ defined by mapping $f \in C(G)$ to $Lf$, with weight, $w$, where

$$(Lf)(v) = \sum_{u \sim v} w_{u,v}(f(v) - f(u)).$$

Sorting the eigenvectors of $L$ by increasing eigenvalues, we can use eigenvectors of the second or the third smallest eigenvalues of $L$ as the lens, denoted as $l_2$, $l_3$, and so on. We focus in particular on $l_2$ (both unnormalized and normalized), commonly referred to as the Fiedler vector [74], since it has desirable geometric properties [35]. For example, the minimum and maximum of the Fiedler vector tend to occur at nodes with maximum geodesic distances [27]. Efficient computations of the Fiedler vector also exist, making it feasible to use in larger graphs. See Fig. 2d for an example.

### 3.1.5 PageRank

Except for the Fiedler vector, prior topological lenses are computationally expensive, making them prohibitive to use on large graphs. To

![Fig. 2: Examples of the topological lenses applied to the US AIR 97 input graph (top) and the resulting mapper graph (bottom) using $n = 8$, $\epsilon = 0$, and modularity clustering. Each of the topological lenses provides a slightly different perspective on the input graph, which is reflected in the output mapper graph.](image-url)
address the scalability issue, we utilize PageRank [18] as a computationally efficient lens that measures the importance of nodes. We use a version of the PageRank algorithm applicable to undirected graphs [52]. A PageRank vector $R: V \to \mathbb{R}$ is defined for every node $v \in V$, $$R(v) = \frac{(1 - d)}{|V|} + d \sum_{u \in N(v)} \frac{R(u)}{|N(u)|},$$ where $N(v)$ is the set of neighbors of $v$; $0 < d < 1$ is the damping factor, which is set at 0.85 for our examples. Using this formation of $R(v)$, PageRank yields an iterative algorithm that can be computed efficiently in practice [18, 85]. A high PageRank score at $v$ typically means that $v$ is connected to many nodes that also have high PageRank scores. PageRank has been shown to have many properties similar to density [90].

### 3.2 Cover of $f$ (Fig. 3b)

In the second step, a cover $U$ is formed over the range of $f$, $f(V) \subseteq [a, b]$ (see Fig. 3b). The cover consists of a finite number of open intervals called cover elements, $U = \{U_n\}_{n \in A}$. To obtain the cover, we use a common strategy, uniformly sized overlapping intervals. Given a graph $G$ equipped with a topological lens $f: V \to \mathbb{R}$, suppose the range of the function $f(V)$ is normalized to be within $[0, 1]$. To obtain an initial cover, we split $[0, 1]$ into $n$ (the resolution parameter) intervals $[c_1, c_2], \ldots, [c_{n-1}, c_n]$ with equal length, such that $c_1 = 0$ and $c_n = 1$. Adjusting the resolution parameter increases or decreases the amount of aggregation over graph subnodes. Broadly speaking, smaller $n$ leads to a more compact representation of the graph. Fig. 4a-4d show an example by varying the number of cover elements $n$.

The overlap parameter $\epsilon$ is used to obtain the cover $U$ consisting of cover elements $(c_i - \epsilon, c_{i+1} + \epsilon)$ for $1 \leq i \leq n - 1$. Fig. 4e-4h show an example of increasing $\epsilon$. Increasing the overlap causes the number of output nodes and density of edges in the mapper graph to increase, which is caused by the duplication of data in the overlapped cover elements. The necessity for the overlapping cover elements in the classic mapper is partially attributable to the lack of connectivity in point clouds. Importantly, in a significant departure from classic mapper, we utilize the edges of the graph for the connectivity, as will be discussed shortly. This modification eliminates the need for overlap. Therefore, we use $\epsilon = 0$ in all of our experiments.

### 3.3 Inverse Image (Fig. 3c)

In the third step, the inverse image of the cover is calculated, as shown in Fig. 3c. For each cover element $U_n$, the inverse image $f^{-1}(U_n)$ is the subset of nodes $V_{n} \subseteq V$, where $f(V_n) \in U_n$. In other words, a node is associated with a cover element whose interval cover the node’s function value. In contrast with the classic mapper, for each element of the cover ($U_n$) we find the nodes $V_n = f^{-1}(U_n)$ and extract its induced subgraph $H_n \subseteq G$, where $f(V_n) \in U_n$. It is important to note that when the overlap is used in cover elements (i.e., $\epsilon \neq 0$), some nodes from the input graph will be replicated across multiple inverse images.

### 3.4 Mapper Graph Nodes and Edges (Fig. 3d)

The final step of the process is forming nodes and edges of the output mapper graph, as shown in Fig. 3d.

#### 3.4.1 Forming Mapper Graph Nodes

To form the nodes of the output, classic mapper construction applies a clustering algorithm to a set of points in the inverse image of a cover element (e.g., agglomerative clustering or DBSCAN [44]). In our setting, the subgraphs $H_n$ extracted with mapper on graphs have additional structures coming from the connectivity of the graph. Therefore, to form the nodes of the mapper graph, we apply a clustering algorithm to each subgraph $H_n$. Every cluster within $H_n$ is returned as a node of the output mapper graph.

We apply three clustering algorithms that preserve different aspects of the subgraph. The first identifies connected components in each subgraph (see Fig. 5a), that is, groups of nodes in $H_n$ that are reachable to each other. The second algorithm is modularity clustering [15] (see Fig. 5b), which identifies communities with dense connections between nodes and sparse connections between communities. Finally, we consider label propagation (see Fig. 5c), which identifies communities by iteratively assigning node labels and propagating the labels through the graph. We tested two label propagation algorithms (semi-synchronous [28] and asynchronous [91]) and obtained similar results (all presented results use asynchronous label propagation).

Fig. 5 show the results of these clustering algorithms, which preserve similar structures. However, the graph details they retain vary based on the properties of the underlying algorithm.

#### 3.4.2 Forming Mapper Graph Edges

Finally, we form the edges of the mapper graph. Initially, we test a classic (mapper) approach that identifies the overlapping node sets between clusters. For all mapper graph nodes created in the prior step, if they share any node from the original graph, they are connected with a weight equal to the overlap between them. In other words, consider two mapper nodes $N_i$ and $N_j$ ($i \neq j$), they are connected if $N_i \cap N_j \neq \emptyset$ with $w_{ij} = |N_i \cap N_j|$. Unfortunately, this method does not work as well as we have hoped. As seen in Fig. 5d, the method has two undesirable problems. First, the required overlap creates a few dense connections. Second, the method does not guarantee connectivity. Instead, we use the connectivity of the input graph to determine the connectivity of the output graph. We utilize the number of edges between the clusters.
in the input graph that go between any two mapper graph nodes $N_i$ and $N_j$ ($i \neq j$). We count the number of edges between the nodes of $N_i$ and $N_j$, which can be efficiently computed using a graph-cut. The nodes are connected if $\text{graph-cut}(N_i, N_j) \neq \emptyset$, with $w_{ij} = \sum_{e \in \text{graph-cut}(N_i, N_j)}$. Fig. 5e shows an example. Alternative approaches could sum $w_{ij} = \sum_{e \in \text{graph-cut}(N_i, N_j)}$ or apply a statistic, such as the mean, to the weights of the edges of the graph-cut. The utility of such approaches depends upon the analytical goals of the visualization. In comparison with the initial approach, the graph-cut-based method better preserves graph details and eliminates the need for enforcing overlaps among cover elements. Unless otherwise specified, all examples in the paper use the graph-cut method to form edges in the mapper graph.

3.5 Visualization (Fig. 3e)

The final output mapper graph (Fig. 3e) provides an overview of the homological structure of the input graph (Fig. 3a) through multiple perspectives using different topological lenses. The levels of details captured by the mapper graph may be adjusted by modifying the cover. Its visualization consists of two components (see Fig. 6): a visualization of the mapper graph, and a visualization of the cover with a histogram useful for parameter selection.

3.5.1 Output Mapper Graph Visualization

The output of mapper on graphs is a mapper graph with additional metadata, which is the primary visualization target, with flexibility in its layout and visual encodings; see Fig. 6d. For the purpose of this paper, we have chosen a simple approach. The mapper graph is laid out using a standard force-directed layout provided by D3.js. Our approach is ultimately agnostic of the graph layout algorithm, and different layouts (e.g., layered approaches) may improve the presentation under certain scenarios. For illustrative purposes, we also add several (optional) visual encodings to better understand the information captured by mapper on graphs. First, a mapper graph node $N_i$ (i.e., the cluster $N_i \subseteq \mathcal{G}$) is colored using the average function value across its members using a topological lens specific colormap (see Fig. 2). Next, the sizes of the nodes are proportional to $|N_i|$. Finally, edge thickness is drawn proportional to the edge weight (i.e., $|\text{graph-cut}(N_i, N_j)|$).

3.5.2 Histogram and Cover Visualization

To assist with parameter selection, we provide additional information about the input graph, which is particularly useful when the input graph is too large to draw.

To understand the distribution of a topological lens $f$, a histogram of the function values is laid out vertically, with the minimum value $a$ at the bottom and the maximum value $b$ at the top. Fig. 6c shows an example of a histogram for the Fiedler lens. The histogram is split into an adjustable number of bins.

The cover (see Fig. 6b) is visualized using a series of boxes, one per cover element, displayed next to the histogram of the topological lens. Each box is placed and colored based on its corresponding interval. The boxes are laid out horizontally automatically using a greedy packing approach. Observing patterns in the histogram and its alignment with the cover elements can help inform choices for $n$, $\epsilon$, and the equalization of the topological lens.

4 EVALUATION

We evaluate the computational performance of our approach (see Sec. 4.1) and its ability to provide multiscale homology-preserving skeletonizations (see Sec. 4.2, 4.3, and 4.4).

Implementation. Our approach is implemented using Python for the calculation of the mapper graph and D3.js for visualization. The code and datasets are available on GitHub [https://github.com/shape-vis/MapperOnGraphs]. A demo version of the software is available at [https://shape-vis.github.io/MapperOnGraphsDemo/].

Datasets. We evaluate our approach using over 45 graphs (see Tab. 2). We focus on a subset of small ($|V| \in [100,1000]$, $|E| \in [200,16651]$), medium ($|V| \in [1000,5000]$, $|E| \in [1092,284924]$), and large ($|V| \in [5000,1134890]$, $|E| \in [91342,2987624]$) graphs. The datasets are a mix of synthetic and real-world datasets. Graphs not in the paper are available in the demo.

Table 2: Listing of Datasets Used in Evaluation

| Dataset | | |
|---------|---|---|
| SNAP [73] | 262,111 | 899,792 |
| SNAP [73] | 1,092 | 1,092 |
| SNAP [73] | 1,032 | 378,343 |
| SNAP [73] | 1,510 | 120 |
| SNAP [73] | 120 | 120 |
| SNAP [73] | 7,197 | 1,093 |
| SNAP [73] | 91,342 | 7,197 |
| SNAP [73] | 21,363 | 334,863 |
| SNAP [73] | 632 | 162 |

Fig. 6: Visualization of the (a) MOVIES graph includes a visualization of (b) the cover elements $U$, (c) the histogram of the topological lens $f$, and (d) the mapper graph ($n=6$, $\epsilon=0$, with modularity clustering).

4.1 Performance

4.1.1 Calculating Topological Lenses

Calculating the topological lens is oftentimes the most expensive part of generating a mapper graph. We recorded the compute time for all lenses on all graphs in our test set. In some cases, certain lenses cannot be computed in a reasonable amount of time. We terminated these computations after 900 seconds.

Fig. 7a shows the (completed) compute time for all lenses. They are laid out by the number of input graph nodes ($|V|$) against the compute time in seconds using a log-log scale. A power regression is plotted for each lens. The experimental results show that the methods from fastest to slowest are: (1) PageRank: $O(|V|^1.07)$, (2) Fiedler: $O(|V|^{1.17})$, (3) eccentricity: $O(|V|^2.45)$, (4) density: $O(|V|^2.45)$, and (5) AGD: $O(|V|^{2.46})$. One conclusion is that Fiedler and PageRank perform only slightly worse than linear, making them good candidates for large graphs. The other methods—AGD, density, and eccentricity—are worse than quadratic, making them impractical on large graphs.
As we investigated the performance of the mapper on graphs framework, it became clear that the performance was highly dependent upon the clustering algorithm. Fig. 7b shows the time to calculate mapper graph for three typical datasets, one small, one medium, and one large, across several lens functions and various numbers of cover elements. In addition, it differentiates the clustering algorithm by color. The connected components algorithm (in blue) is the most efficient, often by several orders of magnitude. We also observed that asynchronous label propagation and modularity clustering, the time required decreases as the number of cover elements increases. This occurs because as more cover elements are added, the number of input graph nodes that fall into each inverse image decreases.

Fig. 7c shows the time to calculate a mapper graph (excluding lens calculation) for all datasets and all lenses with \( n = 2 \) and \( n = 20 \) cover elements. The plots show the performance trend of different clustering algorithms across a variety of datasets. The power regressions show that from fastest to slowest are: (1) connected components: \( O(V^{1.15}) \) and \( O(V^{1.05}) \); (2) asynchronous label propagation: \( O(V^{1.60}) \) and \( O(V^{1.55}) \); and (3) modularity clustering: \( O(V^{1.88}) \) and \( O(V^{1.72}) \), for \( n = 2 \) and \( n = 20 \), respectively. In summary, connected component clustering is significantly more scalable than label propagation, which is somewhat more scalable than modularity clustering.

4.2 Examples

A major challenge in evaluation is that we were unable to identify appropriate quantitative metrics for comparing skeltonizations in our context [107]. As such, we relied on qualitative evaluations in this section and Sec. 4.3 and a human-subject evaluation in Sec. 4.4.

4.2.1 Multi-scale Homology Preservation

To demonstrate the homology preservation properties of our approach, we show a variety of examples. The examples were selected such that the homology of the input graph was clear, and a corresponding mapper graph was selected that reflected that homology. Fig. 1, Fig. 8, and Fig. 9 show two, two, and four examples, respectively. These examples were chosen to show diversity among topological lenses and clustering algorithms. In all examples, as the number of cover elements \( n \) increases, the mapper graph becomes more similar to the input graph. As \( n \) decreases, the mapper graph contains fewer nodes while still retaining important homological structures. These structures include the preservation of tunnels, which are particularly visible in both graphs of Fig. 1 and can also be observed in Fig. 9a, 9b, 9c, 9d, and others.

4.2.2 Large Graphs

We demonstrate the effectiveness of our approach to summarize large graphs as well. Fig. 10 shows results for three large graphs, AMAZON0302, COM-AMAZON, and COM-YOUTUBE, with different numbers of cover elements \( n \). As \( n \) increases, the topological structure remains visible with more details added. In the case of COM-AMAZON, the graph eventually becomes a hairball. The same would eventually happen for the other graphs as \( n \) and the level-of-detail increase.

4.2.3 Case Study: USAIR97

The USAIR97 dataset is an undirected graph whose nodes are 332 airports in the US, and edges encode direct flights between the airports.
in 1997. We are interested in studying the properties of the network in terms of identifying which airports have high or low access to other airports in the network. Since US airlines tend to work on the hub-and-spoke model, we chose PageRank as the topological lens, since it identifies nodes of high importance (i.e., hubs). The input graph and three levels of mapper graph skeletonizations are shown in Fig. 12a.

Using the most aggregated mapper graph version (see Fig. 12b), we identified several groups of airports with varying levels of access. At the highest level are hub nodes in orange (type 1), which contain airline hubs (e.g., LAX/Los Angeles, ATL/Atlanta, etc.) and popular vacation
destinations (e.g., LAS/Las Vegas and MCO/Orlando). Unsurprisingly, flights from these airports can directly access many other airports, as well as connecting limited access airports. Another group with high access are the yellow nodes in the center (type 2), connected to all three hub groups. These airports can reach most destinations directly or through a single layover. The next group with medium access are the few airports that are connectors between two hub groups (type 3). These airports connect to multiple hubs and mostly require a single layover. Finally, the limited access airports are the fans that connect to hub nodes (type 4). These airports require going through at least one hub to reach any destination (besides the hub itself).

4.3 Comparison With Community-based Skeletonization

To compare with existing approaches for skeletonization via clustering, we compare our approach with two community finding algorithms provided by NetworkX [35] that offer a resolution parameter, namely, modularity-based communities [15] and Louvain community detection [12]. To ease comparison, graphs are constructed using the method described in Sec. 3.4. We show several examples in Fig. 11. For each example, the mapper graph and community-based graph were chosen to have approximately the same number of nodes with the exception of Fig. 11b, which had five large fans. In many cases, the community-based methods produce graphs that are similar to the mapper on graphs approach (e.g., see Fig. 11a). However, there are some distinctions.

Multiple Perspectives. Community-driven skeletonization provides only a single perspective on the graph. The ability to use different topological lenses provides mapper on graphs with the ability to provide multiple context-specific visualizations of the homological structure of an input graph. Fig. 11b reveals some variation in the structure.

Maintenance of Homological Features. Community-drive skeletonization often preserves some homological structures, tunnels in particular. However, these methods make no guarantees of the preservation of these features. In particular, community-based methods struggle with dense graphs where extremities are lost and no structure is visible in the core of the graph. Fig. 11d and 11e are the best examples of this, where the output community-based graphs are hairballs, and the mapper on graphs graphs provide insights into the graph structure.

4.4 Human-Subject Evaluation

To test if our approach captures the overall homology of the graph, we ran an IRB-approved human-subject evaluation on Mechanical Turk.

4.4.1 Experiment Design

We generated 1140 stimuli (i.e., mapper graphs) by computing all permutations of the following parameters:

- Small graphs from Tab. 2 / Fig. 13b (19 total)
- Number of cover elements: \( n \in \{6, 10, 20, 30\} \)
- Clustering methods: \{connected components, modularity clustering, asynchronous label propagation\}
- Topological lens: \{AGD, density, eccentricity, Fiedler vector, PageRank\}
- Graphs were laid out using the approach in [38] to ensure consistency.

For each trial, subjects were exposed to one input graph and four mapper graphs (see Fig. 13a): two of the mapper graphs were generated from the input graph using different topological lenses selected at random; the other two mapper graphs were generated from randomly selected graphs and parameters. The subject was asked: “Which picture has the most similar structure to the reference picture?” The idea was that we were trying to capture if their concept of the graph shape was preserved by our approach. They could select one of the four mapper graphs or an “Unsure” option.

4.4.2 Subjects and Trials

The study was conducted using 25 subjects. The experiment took 5-10 minutes, and subjects were compensated $1.50 USD. Two subjects failed to complete the experiment and were excluded for a total of 23 subjects. In terms of age, subjects reported: 7 \( \times \) [25, 34]; 11 \( \times \) [35, 44]; 1 \( \times \) [45, 54]; and 4 \( \times \) [55, 64]. In terms of gender, 10 reported female, and 13 reported male. In terms of visualization experience, 10 reported minimal or none, 8 reported casual, and 5 reported regular or extensive experience. The 23 subjects were each exposed to one
We predicted that the subjects would select mapper graphs associated with the input graph with high accuracy. Since each subject was exposed to two stimuli from the input graph, the null hypothesis was that they would select one of the correct graphs 50% of the time.

Fig. 13b summarizes the results of the survey. Accuracy is defined as $N_{\text{correct}}/(N_{\text{correct}} + N_{\text{incorrect}} + N_{\text{unsure}})$. In the final overall column, the results show that subjects had a 79% accuracy at picking one of the two correct answers, well above the 50% null hypothesis. A binomial test, which is appropriate for testing whether the expected distribution of a binary outcome (correct vs. incorrect) matches the observed distribution, was run to determine if the difference was statistically significant with a p-value $p < 0.001$, thus confirming our hypothesis.

Fig. 13b also reports on the results for individual graphs. Participants performed well with most of the input graphs, except the CIRCULAR LADDER and the CONNECTED CAVEMAN. Upon further inspection, we discovered that several challenging examples were generated for these graphs (see Fig. 14). In these cases, the combination of the topological lens, the clustering algorithm, and the level of aggregation (n) made the relationship difficult to identify, leading to lower scores.

4.4.4 Ecological Validity

Our experimental question was intentionally vague in order to elicit a response that captured subjects’ instincts about what makes two graph visualizations similar. Within this narrow context, our experiment has high ecological validity. In other words, if the visual analytics task is to determine if two graphs have the same “structure”, our experiment showed that participants largely found our approach to satisfy that requirement. Furthermore, the layout itself, which was randomly initialized, did not significantly impact the results. Therefore, our experiment validates that our approach preserves homology and that homology-preservation is important. However, we note that this is not the overall use case for our approach, which is multi-scale homology-preserving skeletonizations of graphs.

5 Conclusion and Discussion

In this paper, we present a TDA-based approach for graph visualization using a construction called mapper on graphs. The approach is effective at creating skeletonizations of graphs that capture the structure across multiple scales and from multiple perspectives using different topological lenses.

Parameter Selection. The flexibility of the mapper on graphs framework comes at a cost. It appears to have a number of parameters (e.g., the lens f, the cover parameters n, and c) that can lead to different skeletonizations of the data. The selection of these parameters is largely data-dependent to the point where an exploration of the parameter space may be required. In our case, we accomplished this using a small-multiple interface that allowed us to examine different topological lenses and numbers of cover elements. This interface is part of the web demo linked previously.

For cover parameter selection, there are recent works on automatic parameter selection for mapper based on statistics [22] and information criteria [26]. For topological lens selection, the process is not necessarily obvious or intuitive. A general guideline is that the chosen lens reflects the geometric or topological property of interest to a user. For instance, the PageRank lens captures node importance, where input nodes that are close to each other and those of similar importance are grouped together to form a mapper graph node. As a consequence, the corresponding skeletonization encodes the distribution of input node importance. We plan to explore (semi-)automatic parameter selection methods in the future.

Graph Layout. As pointed out earlier, the mapper on graphs technique is agnostic to the graph layout method. However, further study is needed to identify the best graph layout methods and create coherency in the layout for different parameter configurations. One possible approach would be to apply our previous method [38].

Flexibility vs. Scalability. To achieve scalability, we have shown that limiting topological lenses to the Fiedler vector or PageRank and limiting clustering to connected components lead to highly scalable algorithms that are practical for large graphs. We will expand upon scalable topological lenses and clustering algorithms in the future.

Relationship to Spectral Clustering. There is a connection between mapper on graphs, spectral clustering, and graph min-cut. The Fiedler vector $l_2$ can be used to bi-partition the graph $G$ (i.e., based on $l_2(v) > 0$ or $l_2(v) \leq 0$). Such a partition could also be realized by computing the mapper graph with $l_2$ as the lens and setting $n = 2$ and $\epsilon = 0$. Such an output not only provides a generalization of spectral clustering but also preserves the connections (which form the min-cut) between the clusters (for appropriately chosen $\epsilon > 0$). Exploring such a connection in detail would be quite interesting.

Acknowledgments

This work was supported in part by grants from the NSF (IIS-1513616, DBI-1661375, IIS-2316496) and DOE (DE-SC0021015).

References

[1] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A large scale graph visualization system. IEEE Trans. on Visual. and Comp. Graph., 12(5), 2006.
