Untangling Force-Directed Layouts Using Persistent Homology

Bhavana Doppalapudi, Bei Wang, and Paul Rosen

Fig. 1: Our approach in using persistent homology to untangle force-directed layouts has two main functionalities. First, as demonstrated via the HIC_1K_NET dataset, we use persistent homology to formulate an initial graph layout (column 1) that improves both the convergence rate of the graph layout and the final layout quality. Second, our approach comes with interactive capabilities, as illustrated in Fig. 2. We use the local continuity meta criterion ($Q_{LCMC}$) for layout evaluation (larger is better). In terms of convergence, our approach (column 2, rows 2 and 3) shows the formation of major graph structures as early as 5 iterations, whereas the standard approach (column 4, row 1) takes 25+ iterations. In terms of final layout quality, the $Q_{LCMC}$ scores for the final layout (column 5) show that our approach significantly exceeds the standard one. Three static layout methods, neato, fdp, and sfdp, are also included for comparison (column 6), with only sfdp (column 6, row 3) showing a comparable $Q_{LCMC}$ score.

Abstract—Force-directed layouts belong to a popular class of methods used to position nodes in a node-link diagram. However, they typically lack direct consideration of global structures, which can result in visual clutter and the overlap of unrelated structures. In this paper, we use the principles of persistent homology to untangle force-directed layouts thus mitigating these issues. First, we devise a new method to use 0-dimensional persistent homology to efficiently generate an initial graph layout. The approach results in faster convergence and better quality graph layouts. Second, we provide a new definition and an efficient algorithm for 1-dimensional persistent homology features (i.e., tunnels/cycles) on graphs. We provide users the ability to interact with the 1-dimensional features by highlighting them and adding cycle-emphasizing forces to the layout. Finally, we evaluate our approach with 32 synthetic and real-world graphs by computing various metrics, e.g., co-ranking, edge crossing, etc., to demonstrate the efficacy of our proposed method.

Index Terms—Force-directed layout, persistent homology, graph clustering, graph cycles

1 INTRODUCTION

Force-directed layouts remain one of the most popular methods for drawing graphs. Their popularity stems from several desirable qualities: generally, they are simple to implement, they are fast for small graphs, they produce aesthetically pleasing layouts, and their iterative algorithms make progressive visualization and interaction natural. Nevertheless, force-directed layouts also suffer from numerous limitations, including poor initialization and over-constraint, leading to poor local minima and limited robustness to noise.

This paper addresses two of these limitations by utilizing and highlighting important topological features of the graph. First, force-directed layouts are strongly influenced by the initial layout of graph nodes, which is often generated randomly. After the initialization, successive application of the forces among nodes causes the layout to settle in a locally minimal energy state, which hopefully shows the graph's topological structure. Unfortunately, the random initial layout does not consider the global topology, which potentially slows convergence and can lead to unrelated structures overlapping in the visualization. Second, since force-directed layouts are over-constrained systems, even without the overlap of unrelated structures, certain topological features have their shape distorted, tangled, or hidden by noise (i.e., low weight edges) making it difficult to visually identify topological features.

- B. Doppalapudi was with the University of South Florida, Tampa, FL, 33610. E-mail: bdoppalapudi@usf.edu
- B. Wang was with the University of Utah, Salt Lake City, UT, 84112. E-mail: beiwang@sci.utah.edu
- P. Rosen was with the University of Utah, Salt Lake City, UT, 84112. E-mail: rosen@sci.utah.edu

Manuscript received xx.xxx. 201x; accepted xx.xxx. 201x. Date of Publication xx.xxx. 201x; date of current version xx.xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxx
consider graphs as mechanical systems and apply forces to the nodes. In general, repulsive forces, similar to those of electrically charged particles, exist between all the vertices, and attractive forces, similar to spring-like forces, exist between connected vertices or between neighbors. The Eades model [24] was the first to apply spring forces on the initial layout to achieve a minimal energy position. Later, Fruchterman and Reingold [28] modified the Eades model to achieve a system that distributes the vertices evenly, and has uniform edge lengths and symmetry. Kamada and Kawai [45] developed another variant on Eades’ work. Instead of just applying attractive forces between neighboring vertices, they applied the concept of ideal distance, which is proportional to the length of the shortest path. Although computational costs are high for this method, speed-ups have been achieved using heuristics [27] and the GPU [33]. Meidanis et al. [50] presented a sublinear time model that pairs a radial tree drawing of a breadth first spanning tree with random sampling of repulsive forces. While their approach has technical similarities with our initial layout approach, there are important implementation differences and theoretical guarantees that come from our use of 0-dimensional persistent homology.

Constraint-Based Layouts. Constraint-based layouts are a more sophisticated version of force-directed layouts. Dwyer et al. [23] archived a high-quality, topology-preserving visualization by implementing a constraint-based layout for a detailed view and a forcedirected layout for the overview. Archambault et al. [3] proposed the TopoLayout algorithm that dynamically adapts the graph layout method based upon the topology detected within subgraphs.

Layered Layouts. Layered layouts, in general, are used for directed graph layouts. Sugiyama et al. [62] used a 4-phase approach to layout graphs: (1) removing cycles, (2) assigning nodes to layers, (3) reducing edge crossings, and (4) assigning coordinates to nodes. Buchmaier et al. [6] proposed to visualize directed cyclic graphs by skipping the cycle removal step.

Algorithmic Methods. Koren et al. [46] developed the Algebraic Multigrid Method (ACE) algorithm that minimizes quadratic energy. Harel and Koren introduced High-Dimensional Embedding (HDE) [41] that projects a high-dimensional representation of a graph with PCA.

Multiscale Layouts. Multiscale layouts start with a coarse layout and refine it in phases. Hachul and Jünger [37] proposed Fast Multilevel Method (FM), a force-directed method that incorporates a multiscale approach in a system to calculate repulsive forces in rapidly evolving potential fields. By comparing various algorithms, Hachul and Jünger [38] showed that multiscale methods, including FM, ACE, and HDE, were significantly faster than regular force-directed layouts. They also found that FM produced the best quality graphs in the group.

Node Congestion. Node co-location is a challenging problem, particularly in multiscale layouts [68]. Space-filling curves have been used to avoid node co-location [51]. However, the approach works only for datasets with clear clustering, and for dense graphs, the visualizations are not of good quality or aesthetically pleasing. Gansner and North [31] proposed to improve the force-directed layout by moving the overlapping nodes within cells of a constructed Voronoi diagram. By selecting good starting positions for nodes, Gajer et al. [30] developed a multiscale approach that improved computation time and better preserved the graph’s structure. Adai et al. [2] introduced the Large Graph Layout (LGL) algorithm, which uses a minimum spanning tree to guide the force-directed iterative layout to visualize large protein map networks. However, they did not consider datasets with different characteristics. Dunne and Shneiderman [22] proposed to use motifs for node and edge bundling. Their technique replaced common graph patterns of nodes and links with intuitive glyphs. They showed that the approach required less screen space and effort, while preserving the underlying relations. However, the glyphs required additional learning from users, and charts with many large glyphs added clutter to the display and increased the possibility of overlap.

Edge Congestion. Node-link diagrams frequently suffer from edge crossings. Carpendale et al. [12] proposed displacing edges running through the area of interest. However, certain questions were left unanswered (e.g., the amount of edge displacement to use). For graphs without hierarchy, Holten and van Wijk proposed a self-organizing bundling method, where edges act as flexible springs attracting each...
other [42]. ASK-Graph [1] addresses the issues for highly dense graphs with node counts approaching 200k. Bach et al. [5] proposed to use confluent drawings (CDs) for edge bundling based on network connectivity, which showed some promising results. Nevertheless, CDs worked only for sparse graphs where node counts were less than 20 and the edge density was less than 50. Such an approach also showed low participant confidence, indicating that CDs require significant learning and may be misleading. Zinnsmaier et al. [74] proposed a level-of-detail technique that performs density-based nodes aggregation and edge accumulation.

**Interaction.** Research into interactive visualization has been done to assist with efficient data explorations. Commonly used interaction techniques for graphs include panning and zooming [67] and fisheye views [29, 61, 69] that focus on areas of interest.

### 2.2 Persistent Homology

Persistent homology studies the topological features of data that persist across multiple scales. Weinberger gave a brief explanation in his work titled “What is ... persistent homology?” [72], while Harer and Edelsbrunner [25] detailed the concept and history of persistent homology. Persistent homology has shown great promise to assist in the analysis of complex graphs due to the types of features it extracts and its ability to differentiate signal from noise [18, 43, 54, 55, 71]. For example, Rieck et al. [58] used persistent homology to track the evolution of clique communities across different edge weight thresholds. Persistent homology has also led to notable results in the study of brain networks [13, 48, 49] and social networks [7, 40, 43]. Although persistent homology has been used mainly for analysis tasks within these prior methods, it has not been used to improve the visualization of graphs.

Recently, Suh et al. [63] used 0-dimensional persistent homology features to create a persistence barcode visualization, which was then used to manipulate a force-directed graph layout. Their framework was limited to extracting 0-dimensional topological features and utilizing those features for interactive manipulation of the graph. Our work extends and complements the work of Suh et al. by utilizing the 0-dimensional features to preprocess the initial layout of graphs, thus improving the rate of convergence and quality of layouts, and by providing an algorithm to efficiently extract, interactively highlight, and manipulate the graph layout with 1-dimensional topological features of a graph.

### 3 Methods

We first describe how the persistent homology information is extracted from an input graph (Sect. 3.1). We then describe the use of this information for building fast initial graph layouts (Sect. 3.2) and for highlighting important structures in the visualization (Sect. 3.3).

**The input is an undirected graph** $G = (V,E)$ equipped with an edge weight $w : E \rightarrow \mathbb{R}$. $w$ can be any real function that quantifies the edge importance. Recall the Jaccard index between a pair of sets $A$ and $B$ is defined to be $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. In this paper, if $w$ is not known 

**a priori**, $w(e)$ for an edge $e$ is assigned the Jaccard index between the 1-neighborhood of its nodes, as was also done in [63].

#### 3.1 Persistent Homology of a Graph

We describe a novel approach to extract persistent homology features from a graph, leaving the discussions of the general theory to prior works (e.g., [25]). Previous approaches (e.g., [40]) have relied upon mapping a graph to a metric space, computing a Vietoris-Rips complex, and extracting its 0-dimensional and 1-dimensional persistent homology as topological features. However, these approaches may be costly, $O((|V| + |E|)^3)$ in the worse case, making them impractical on larger graphs. Furthermore, persistent homology identifies a class of cycles and while identifying the existence of such a class is well defined, determining which nodes specifically contribute to the cycle in the context of graphs is ambiguous [17]. In the following section, we provide an alternate strategy that resolves both of these issues.

Homology deals with the topological features of a space. In particular, given a space $\mathcal{X}$, we are interested in extracting the $0$-dimensional, $H_0(\mathcal{X})$, and $1$-dimensional, $H_1(\mathcal{X})$, homology groups of $\mathcal{X}$, which are the connected components and tunnels/cycles of the space, respectively.

To identify the homology of a graph, we begin by describing the Edge complex of a graph. Given a threshold $t$, for each edge $e_i$ in $G$ with a weight $w_i$, the Edge complex is $\text{Edge}(t) = \{ e_i | w_i \geq t \}$. In other words, the Edge complex is the set of all edges whose weight is greater than or equal to the given threshold. For example, Fig. 3b shows $\text{Edge}(4)$ and $\text{Edge}(2)$ of the graph, in Fig. 3a and 3c, respectively.

From an Edge complex, its connected components ($H_0$) and cycles ($H_1$) can be efficiently extracted by a process that will be discussed in the forthcoming sections. However, extracting the homology of the graph from a single Edge complex may fail to capture homology visible at different thresholds (e.g., see Fig. 3) and, therefore, requires careful selection of the threshold $t$.

Instead of selecting a single threshold $t$, we extract $H_0$ and $H_1$ features of the graph across all thresholds using a multiscale notion of homology, called persistent homology. Persistent homology is calculated by extracting a sequence of Edge complexes, referred to as a filtration. We consider a finite sequence of decreasing thresholds, $\infty \geq t_0 \geq t_1 \geq \cdots \geq t_m = 0$. A sequence of Edge complexes, known as an Edge filtration, is calculated and connected by inclusions,

$$\text{Edge}(t_1) \to \text{Edge}(t_1) \to \cdots \to \text{Edge}(t_m).$$

In other words, the Edge complexes are subsets of one another, $\text{Edge}(t_i) \subseteq \text{Edge}(t_{i+1})$ for $0 \leq i \leq m - 1$. The Edge filtration can
Theorem 1. Given any spanning tree $S$ of a connected graph $G$, in-
H
further reduce the number of
H
compute. Nevertheless, the shortest path needs to be calculated
O
puted using Dijkstra's algorithm with worst-case time complexity
Edge
est path is calculated between the endpoints of each edge,
the existence of an
Edge
tures with the
H
3.1.2 Efficient Identification of
H
results maximal spanning tree captures exactly the same structure
O
maximal spanning tree using Kruskal's algorithm \cite{47}, selecting the
Edge
weights greater than or equal to every other possible spanning
maximal spanning tree
Edge
to capture features and increase efficiency. Using the
graphs, taking $O(|V|^2 \log |V|)$.

Our choice of the Edge filtration is a very specific one designed
to capture features and increase efficiency. Using the Edge filtration,
the $H_0$ information for the graph can be obtained by calculating the
maximal spanning tree of the graph, which is the spanning tree with
edge weights greater than or equal to every other possible spanning
Tree. In calculating the maximal spanning tree of the graph, as edges
are added to the tree, each edge $e_i$ represents an $H_0$ death event at $w_i$ (i.e.,
merging of two connected components). We calculate the maximal spanning tree using Kruskal's algorithm \cite{47}, selecting the
largest weight edge instead of the smallest. The algorithm has a worst-
case time complexity of $O(|E| \log |E|)$. For non-negative weights, the
resulting maximal spanning tree captures exactly the same structure
as the prior minimal spanning tree of the metric space approach, only
more efficiently. In addition, our maximal spanning tree approach
captures meaningful features for negative and zero weight edges.

3.1.2 Efficient Identification of $H_1$ Cycles
In general persistent homology calculations, both detecting the existence
of $H_1$ features and extracting a representative cycle are computationally expensive, roughly $O(|V|^3)$ \cite{17}. The use of the Edge
filtration enables both detecting and extracting the $H_1$ features much
more efficiently within the limited context of graphs. To do this,
we begin with an interesting observation in Theorem 1.

Theorem 1. Given any spanning tree $S$ of a connected graph $G,$ inserting any additional graph edge into $S$ creates a cycle.

Proof. Since $S$ is a tree, it is acyclic, and any two nodes have a unique
simple path between them. Therefore, if an edge is added between any
two nodes in $S$, those nodes will now have two non-overlapping paths
between them. Concatenating the edges of the two paths will create
closed trail, i.e., a cycle, between them. $\blacksquare$

As it turns out, this property enables efficient extraction of $H_1$ features
with the Edge filtration. As the maximal spanning tree is calculated,
an edge $e_i$ that would be excluded from the spanning tree signifies the
existence of an $H_1$ cycle feature with a birth at $w_i$.

To extract the $H_1$ cycle paths themselves, the unweighted shortest
path is calculated between the endpoints of each edge, $e_i$, in the
associated Edge complex, $\text{Edge}(w_i)$. The shortest path is computed
using Dijkstra's algorithm with worst-case time complexity
$O((|V| + |E|) \log |V|)$. In practice, paths are short and generally fast
to compute. Nevertheless, the shortest path needs to be calculated
for every $H_1$ cycle. Therefore, in practice, we defer calculating the complete cycle paths until the visualization needs the information. To
further reduce the number of $H_1$ features considered, cycles of length
three are considered to be trivial and discarded\footnote{Since 2-simplices (i.e., triangles) are not used, we do not track cycle death.}.

While this approach will extract all $H_1$ features of the Edge complex,
it will not extract all cycles of the graph. Our $H_1$ features are a specific
type of cycle, where no chord within the cycle has a weight greater than or
equal to the weight of all edges of the cycle. This type of cycle has a
strong theoretical basis that is useful for many analysis tasks, but it may
not be relevant for all such tasks. As we will show in our evaluation
(see Sect. 5.2), these cycles are useful for many graphs. Nevertheless,
adapting our approach to extracting other representative cycle types
would further extend the utility of the approach.

3.2 Using $H_0$ Features to Untangle Initial Graph Layouts
Recent works (e.g., \cite{19, 63}) have demonstrated the value of using $H_0$ information in generating high-quality layouts of graphs and high-dimensional data, respectively. In contrast, we focus on quickly produc-
ing a good-quality layout that reflects the most important structures of
the graph, as defined by persistent homology. We then utilize a D3.js's
force-directed layout capabilities \cite{11} to optimize the final layout.

Our algorithm works by laying out the graph using the maximal
spanning tree. Inspired by early works on tidy tree drawing \cite{57, 64,
73}, our approach has two main steps, with a focus on simplicity
and efficiency. First, we generate an abstract layout of the maximal
spanning tree that determines the distribution of space to nodes and
subtrees of the graph. Then, we embed the tree into the drawing canvas
using either a layered or radial scheme.

3.2.1 Abstract Layout
The first phase of the algorithm forms an abstract layout of the tree.
The algorithm begins by selecting a node at random to serve as the root
of the tree\footnote{We tested several other strategies, e.g., finding the most central node or the node with the most children. However, improvements were minimal, sometimes with a high additional cost over a randomly chosen node.}. The children of the selected root are then laid out hierar-
chically. The algorithm recursively processes subtrees, subdividing the
available space until all nodes have been visited. The available space is
divided between children at each level based on the number of nodes in
their respective subtrees. For example, in Fig. 5b, the orange node is
selected as the root. The leftmost subtree is allocated more space since
it contains more nodes.

3.2.2 Graph Embedding
In the second part of the algorithm, the abstract layout is used to embed
nodes into the drawing canvas using either a layered or radial layout.
Layered Layout. The first version of our layout algorithm maps the
abstract layout directly to the available drawing canvas in a layered
tree visualization. Specifically, the horizontal space on the canvas is
mapped to the width of the tree in the abstract layout, and the vertical
space is mapped to the height of the tree in the abstract layout.
For example, in Fig. 5c, the abstract layout from Fig. 5b is mapped to the
available drawing canvas.

![Fig. 5: An illustration of the initial layout schemes. The input graph (a) first has an abstract layout formed in (b) and is then mapped into a layered layout (c) or a radial layout (d). After the initial layout, any force-directed layout can be used.](image-url)
of their normalized node valence. The only exception to this approach is the visualization. Those nodes serve as the major axis of the ellipse and are parameterized (i.e., ordered) and their target locations on the ellipse are identified. (d) Forces are applied to the nodes on the cycle to move them toward their target locations.

Radial Layout. The second version adopts a radial layout for the tree. The width of the abstract tree is mapped to an angle in the unit circle, and each layer of the tree occupies an increasing radius. For example, in Fig. 5d, the abstract layout from Fig. 5b is mapped to polar coordinates in the drawing canvas.

For either layered or radial layout, after the initial layout is formulated, a standard force-directed layout is applied to the entire graph.

3.3 Interactive Untangling with Persistent Homology

3.3.1 Visualization of Persistent Homology Features

We visualize the $H_0$ and $H_1$ persistent homology using a visualization based upon a persistence barcode (see Fig. 2), a standard tool of persistent homology. For this visualization, a barcode is associated with a set of $H_0$ or $H_1$ features. For each barcode, a bar represents a single topological feature. Its length is proportional to the death or birth time of the associated $H_0$ or $H_1$ features, respectively.

3.3.2 Interacting with $H_0$ Components

Similar to the $H_0$ interactions in [63], when $H_0$ bars are selected in the barcode, a strong attractive force is created (i.e., a spring-like force) between the nodes of the associated edge from the spanning tree. Our selection offers a filtering slider (see demo\(^5\)) to select multiple features simultaneously.

3.3.3 Interacting with $H_1$ Cycles

For interacting with $H_1$ cycles, we offer two modalities. The first is a highlighting modality. As the user’s mouse goes over the bar for a given cycle, that cycle is extracted and highlighted in the graph. Fig. 2 shows two examples, each highlighting two cycles.

The second modality, triggered when a user clicks a feature in the barcode, uses information about the cycle to add a new elliptical force to the cycle nodes in the force-directed layout. The approach (see Fig. 6 for an illustration) first takes the nodes of the cycle and identifies the two nodes with the largest Euclidean distance from one another in the visualization. Those nodes serve as the major axis of the ellipse and determine the diameter of the major axis. The minor axis diameter is determined by a user-selectable aspect ratio. Once the elliptical shape is calculated, the cycle nodes are parameterized, i.e., ordered around the ellipse, to select a target location. The ordering step is quite important, as it enables powerful modifications, e.g., untangling cycles. Finally, a strong force is added to attract the nodes to their target locations. However, due to the over-constraint of force-directed layouts, this new force does not guarantee nodes will end up on the ellipse. Fig. 2 shows examples of imposing the elliptical shape on the cycles and examples of the forces untangling cycles in the graph.

4 Evaluation

To demonstrate the efficacy of our approach, we provide a two-phase evaluation. In Sect. 5.1, we first evaluate our graph initialization approach using $H_0$ persistent homology in terms of layout quality and rate of convergence. Second, in Sect. 5.2, we evaluate the layout quality of using $H_1$ persistent homology to modify the forces of a force-directed layout. For all comparisons, we primarily compare to the state-of-the-practice force-directed layout provided by D3.js [11], which uses a random initialization. Furthermore, in Sect. 5.1.4, we compare to static graph visualizations, including the neato [45], fdp [24], and sfdp [44] algorithms coming from Graphviz [26].

4.1 Implementation

We have implemented our approach in JavaScript and D3.js v5 using the base implementation of D3.js force-directed layout with all standard settings. To initialize the layout, our code provides xy-coordinates to all nodes before the D3.js force-directed layout simulation takes control of the data. Modifications to the graph forces are done by adding new forces to the D3.js layout simulation. All experiments use the default D3.js stopping criteria for computation. A demo version of our approach is at https://usfdatavisualization.github.io/UntangleFDL/, and our source code is available at https://github.com/USFDataVisualization/UntangleFDL/.

4.2 Datasets

We have tested 32 datasets that include a mix of synthetic and real-world datasets, acquired from sources including the Network Repository [59], NetworkX [39], BioSNAP [75], and the UF Sparse Matrix Collection [15]. The graphs are evenly divided into 16 dense and 16 sparse graphs, based upon their average node eccentricity (ECC)\(^5\). A summary of graphs found in the paper can be seen in Table 1 and Table 2. Further, as a practical matter, interactivity of graph visualizations begins to degrade at ~1000 nodes in D3.js. Therefore, we differentiate larger graphs as those where $|N| > 1000$. Graphs not in the paper can be found in a comprehensive table of results included in our supplemental materials and in our demo.

All graphs are colored using the D3.js plasma color map (0 to 1) of their normalized node valence. The only exception is the MAP OF SCIENCE dataset (see Fig. 9e), which is colored using a categorical color map.

4.3 Evaluation Metrics

Layout algorithms are often optimized considering aesthetic criteria. Purchase [56] worked on various aesthetic criteria of importance and priority and showed that minimizing the number of edge crossings serves as critical aesthetic quality. Beck et al. [9] defined several aesthetic criteria that ease designing, comparing, and evaluating different dynamic visualizations, including general aesthetic criteria, dynamic aesthetic criteria, and aesthetic scalability criteria. We use several criteria, including time ($T_\text{avg}$), convergence ($C_\jmath$), and layout quality ($Q_\text{avg}$).

Our main goal is to measure whether global structures overlap with one another. To identify those overlaps, the primary measure we consider is that of co-ranking. Co-ranking compares the $k$-neighborhoods of a high-dimensional space, in our case defined by the unweighted shortest path distance in the graph, with a low-dimensional embedding, the Euclidean distance between nodes on the image. We use several meta-criteria on the co-ranking.

Local Continuity Meta Criterion ($Q_{LCMC}$) measures the ranked order overlap of $k$-neighborhoods for a range $[1, k]$ and averages them [14]. To ease comparisons, we fix $k = 20$. $Q_{LCMC}$ is normalized such that $Q_{LCMC} \in [-1, 1]$, where larger is better and negative values imply opposite ordering. We also utilize the convergence $C_{LCMC}$, which is the iteration number when $Q_{LCMC}$ is within 0.01 of the final value (after 300 iterations of force calculations).

Trustworthiness ($Q_{\text{trust}}$) and Continuity ($Q_{\text{cont}}$) co-ranking meta criteria [66], whose conclusions parallel LCMC, are also provided.

We next consider three measures of performance that quantify the processing time.

\(^{4}$Demo at https://usfdatavisualization.github.io/UntangleFDL/.

\(^{5}$Eccentricity is the maximum shortest path distance from a given node.
Table 1: Table of dense datasets. See Sect. 4.3 for details about metrics. Our discussion focuses on LCMC metrics (in blue). In these cells, bold indicates a smaller time for T_{LCMC}; a lower improvement count for LCMP; or a value 0.005 larger for Q_{LCMC}.

<table>
<thead>
<tr>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIRPORT</td>
</tr>
<tr>
<td>BALB</td>
</tr>
<tr>
<td>BCS</td>
</tr>
<tr>
<td>BIO-DISEASE</td>
</tr>
<tr>
<td>CIRCULAR LADDER GRAPH (100)</td>
</tr>
<tr>
<td>CONNECTED CAVEMEN (10,20)</td>
</tr>
<tr>
<td>ENEMY-123</td>
</tr>
<tr>
<td>LADDER</td>
</tr>
<tr>
<td>LOBSTER</td>
</tr>
<tr>
<td>LOLLIP (10,50)</td>
</tr>
<tr>
<td>MAP OF SCIENCE</td>
</tr>
<tr>
<td>RANDOM GEOMETRIC (0.001,0.1)</td>
</tr>
<tr>
<td>RESTW</td>
</tr>
<tr>
<td>SCIENCE COLLABORATION NETWORK</td>
</tr>
<tr>
<td>WATTS STROGAZE (100,0.05)</td>
</tr>
</tbody>
</table>

Table 2: Table of sparse datasets. See Table 1 for a description.

<table>
<thead>
<tr>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIRPORT</td>
</tr>
<tr>
<td>BALB</td>
</tr>
<tr>
<td>BCS</td>
</tr>
<tr>
<td>BIO-DISEASE</td>
</tr>
<tr>
<td>CIRCULAR LADDER GRAPH (100)</td>
</tr>
<tr>
<td>CONNECTED CAVEMEN (10,20)</td>
</tr>
<tr>
<td>ENEMY-123</td>
</tr>
<tr>
<td>LADDER</td>
</tr>
<tr>
<td>LOBSTER</td>
</tr>
<tr>
<td>LOLLIP (10,50)</td>
</tr>
<tr>
<td>MAP OF SCIENCE</td>
</tr>
<tr>
<td>RANDOM GEOMETRIC (0.001,0.1)</td>
</tr>
<tr>
<td>RESTW</td>
</tr>
<tr>
<td>SCIENCE COLLABORATION NETWORK</td>
</tr>
<tr>
<td>WATTS STROGAZE (100,0.05)</td>
</tr>
</tbody>
</table>

**Initialization Time** ($T_{IT}$) is the time taken to initialize the force-directed layout system. For our approach, the timing includes the overhead to calculate the spanning tree and position nodes.

**Average Iteration Time** ($T_{IT}$) is the average time required to calculate one iteration of the force-directed layout.

**Total Time to LCMC Convergence** ($T_{LCMC}$) is the total time ($T_{IT}$ + $T_{IT}$) required to reach the LCMC convergence criteria.

Finally, we produce a set of well-established graph readability metrics [21,36]. Our evaluation does not discuss them, but they are included for completeness.

**Edge Crossing** ($Q_{EC}$) measures the ratio of non-intersecting edges to total possible intersections. Graphs are generally more readable with fewer crossings. $Q_{EC}$ is normalized, such that $Q_{EC} \in [0,1]$, where larger is better.

**Crossing Angle** ($Q_{CA}$) is the average deviation from the ideal crossing angle. If edges cross, it is preferable they cross at an ideal angle of 70 degrees that makes their individual paths most visible. $Q_{CA}$ is normalized, such that $Q_{CA} \in [0,1]$, where larger is better.

**Minimum Angular Resolution** ($Q_{MAR}$) measures the average deviation of adjacent edge angles from the ideal angle (360/degree)(v) for any $v \in V$). For nodes with multiple edges, it is preferable to their edges be distributed around the node as much as possible. $Q_{MAR}$ is normalized, such that $Q_{MAR} \in [0,1]$, where larger is better.

5 RESULTS

We evaluate our method's ability to entangle initial graph layouts, followed by untangling cycle structures.

5.1 Untangling Initial Graph Layouts

We evaluate our initial graph layout approach in terms of graph quality, convergence, and time. For the experiments, we initialized the graphs with either the standard D3.js random layout or our approach and let them run until D3.js stopped force calculations (after 300 iterations using the default settings).

5.1.1 Layout Quality

Table 1 and 2 show the results for all quality metrics from Sect. 4.3, except for the readability metrics for the three largest graphs, which were skipped due to very high computational costs. Although all metrics are available, we discuss only $Q_{LCMC}$. 


Layered vs. Radial. The results in Table 1 and 2 show only the layout method, layered or radial, which produced higher $Q_{LCMC}$. In many cases, the result between both methods is effectively identical. The results show that neither method is universally better and seemed to be graph dependent. Nevertheless, the results for both layout methods are available in the supplemental material.

Dense Graphs. With dense graphs (see Table 1 and Fig. 7), our approach generally produced higher $Q_{LCMC}$ scores. However, for several graphs, our scores were similar or slightly lower, e.g., Fig. 7e and 7f. In these cases, the results are still quite similar visually. In general though for dense graphs, it seems that no matter the initial position of nodes, the layout will end in a more or less similar configuration. Importantly, even when our score is lower, our approach converges much more quickly, e.g., for the HIC 1K NET 6 dataset (see Fig. 11b), our method produced a similar $Q_{LCMC}$ in about one fifth the time of the random layout. This property is discussed more in Sect. 5.1.2.

Sparse Graphs. With sparse graphs (see Table 2 and Fig. 9), the story is a bit different, as these graphs are not so overconstrained. Using random initial layouts, quite often, their topological structures are overlapping or hidden altogether. On the other hand, our approach untangles these topological structures, leading to better final graph layouts, e.g., with ENGYMES-G123 dataset (see Fig. 9c) our approach ($Q_{LCMC} = 0.436$) produces higher co-ranking scores than the random layout ($Q_{LCMC} = 0.374$), and the cycle structures of the graph are more clearly visible. There was one sparse case, BCSSTK, where our approach performed slightly worse than random because it was unable to untangle the long cycle in the graph (see supplement). However, the interactive functionality discussed in Sect. 5.2 resolved that issue.

Larger Graphs. Improving layouts is particularly important for larger graphs (i.e., graphs of 1000 nodes, see Sect. 4.2). Fig. 1 and 11 show examples of larger graphs. Our approach shows better clustering structures for the HIC 1K NET and AIRPORT datasets, supported by the improved $Q_{LCMC}$ scores. SMITH, on the other hand, being a dense graph, shows similar clustering and identical $Q_{LCMC}$ scores.

Overall, we observe that although our approach could improve the layout quality of dense graphs, sparse graphs almost always benefited from utilizing our approach.

5.1.2 Rate of Convergence

We compute the convergence metrics (see Sect. 4.3) on all of the datasets listed in Table 1 and 2, and we focus on the convergence of the LCMC ($C_{LCMC}$). Our results show that for all except one dataset, USAIR 97, using our approach converged faster than random layouts, often significantly so. Fig. 8 shows plots of $Q_{LCMC}$ against iterations for three example datasets, including USAIR 97. In all cases, our approach starts with a much higher $Q_{LCMC}$ score and fine-tunes the results. One interesting observation is that the $Q_{LCMC}$ sometimes starts high and dips, e.g., in Fig. 8c. This results from our good initial layout being in a high energy state which is distorted by the force-directed layout before settling in a good quality low energy state.

The rate of convergence is particularly important for larger graphs, where the average time per iteration is higher. For the large datasets, AIRPORT, HIC 1K NET 6, and SMITH, our approach converged faster than random layouts, with 37 vs. 76, 55 vs. 112, and 4 vs. 28 iterations, respectively. This phenomenon is also visible in Fig. 1 and 11, where graphs layouts are shown at several intervals—initial, 5 iterations, 10 iterations, etc. In all cases, the structures shown in the final graph are visible much earlier with our approach (iteration 5 or 10) than with a random layout (iteration 25 or more).

Therefore, we conclude that our approach significantly improves convergence in most cases.

5.1.3 Compute Time

The improved rate of convergence our method offers does not come for free. An initialization time penalty (i.e., $T_{IT}$), albeit small, must be paid. Due to the imprecise time measurements offered by the web browser,
We also compared our results to other graph layout algorithms, namely, neato, \textsc{sfdp}, and \textsc{random}. For these graphs, there was an additional initialization cost ($T_{IT}$) of $15-20\%$ over a random initialization. Since we made no modifications to the force calculations, the average time per iteration, $T_{IT}$, was virtually identical.

However, the time benefit of our approach is placed in context when considering the time to convergence, $T_{LCMC}$. Due to the low additional overhead and significant reduction in number of iterations, our approach offers a speed-up of $\sim 2\times$ for AIRPORT and HIC 1K NET 6, and a speed-up of $\sim 5.6\times$ for SMITH.

Given these observations, we conclude the benefits of fast convergence far outweigh the additional initialization time required for the spanning tree calculation, particularly for larger graphs.

### 5.1.4 Comparison to Other Algorithms

We also compared our results to other graph layout algorithms, namely, neato, \textsc{sfdp}, and \textsc{random}. Due to space consideration, the majority of results are presented in our supplemental document. However, results on larger graphs can be seen in Fig. 1 and 11, and on smaller graphs in Fig. 7 and 9. Generally speaking, one or more of these methods produced graph layouts with similar or slightly better $Q_{LCMC}$ scores than our approach. Therefore, our method can be viewed as closing the gap between random initialization force-directed layout and these more advanced methods. Ultimately, we are still limited by the capacity of the D3.js force-directed layout to produce high-quality final layouts. Our method provides only a boost. Finally, an important aspect of our approach is that the layouts are intended to be interactive. Users are supposed to explore $H_0$ and $H_1$ features to learn the graph’s structure, which is a capability not necessarily offered by these other methods.

#### 5.2 Interactive Untangling with $H_1$ Features

We evaluate whether the interaction with $H_1$ features reveals anything about the graph structure previously available using a random initialization force-directed layout or by using the $H_0$ forces introduced in [63].

**Generating $H_0$ Examples.** To compare to $H_0$ persistent homology feature forces, we start with the random initial layout (i.e., the D3.js default) and allow the graph to converge to a stable configuration. In other words, they look like the random final graph layouts in the upper middle of Fig. 7 and 9. We then apply a force to all $H_0$ features and allow the graph to converge.

**Generating $H_1$ Examples.** To determine the efficacy of $H_1$ persistent homology feature forces, we configure them similarly. Starting with the random initial layout, we allow the graph to converge to a stable configuration. We then apply a force to a single $H_1$ feature, which is selected by considering $H_1$ features with longer cycle lengths, and again the graph is allowed to converge.

**Results.** The results are visible primarily in Fig. 10 and 12, and additionally in Fig. 2. First, we can see that our approach reveals cycle structures that are often hidden in standard graph layout and only sometimes revealed using $H_0$ features. In other words, our approach reveals topology of the graph that is otherwise hidden or difficult to see. The second observation we make is that whereas $H_0$ features tend to improve the overall presentation of the graph, i.e., higher $Q_{LCMC}$ scores, our approach of applying forces to $H_1$ features has a much stronger impact, resulting in even better graph layouts, sometimes dramatically so, e.g., in CIRCULAR LADDER (Fig. 12b) or WATTS STROGATZ (Fig. 12d) graphs.

An important aspect of interacting with $H_1$ features is that highlight-
Fig. 11: A comparison of random initial layout to our approach for large graphs shows the layouts at various stages of processing. Importantly, our approach shows the graph structure much earlier in the process, and the final layout is similar or better quality. We also compare to the results of non-interactive methods of neato, fdp, and sfdp (right).


6 D I S C U S S I O N A N D   C O N C L U S I O N S

In this paper, we have evaluated two new uses of persistent homology on force-directed layouts. We first investigate using $H_0$ persistent homology for initializing graph layouts. Although the implementation itself relies on maximal spanning trees, persistent homology provides a theoretical foundation for justifying its use. At the same time, our experimental results show that it indeed improves the convergence rate and quality of force-directed layouts.

Second, we investigate using $H_1$ features for highlighting and modifying the forces of a force-directed layout. Here again, in addition to the algorithmic contribution of efficiently extracting the $H_1$ features, we observe that using these features reveals hidden features and improves graph layouts in many situations.

Beyond our current work, there is potentially room for developing additional initial layout schemes or perhaps automatically identifying which scheme would work best for a given dataset. Still, a good balance between performance and final quality remains of the utmost importance. In addition, our scheme for utilizing $H_1$ features could be utilized in a more elaborate manner. Additionally, it would be interesting to study whether other simplicial complexes could be used with persistent homology to capture topological information about other graph structures, e.g., cliques, stars and trees. Finally, our approach is implemented using the D3.js force-directed layout, but we believe the approach would work with other state-of-the-art techniques and frameworks. However, the exact implementation and the ultimate performance and quality gain require additional study.

Demo: [https://usfdatavisualization.github.io/UntangleFDL/](https://usfdatavisualization.github.io/UntangleFDL/)
Source: [https://github.com/USFDataVisualization/UntangleFDL/](https://github.com/USFDataVisualization/UntangleFDL/)

ACKNOWLEDGMENTS

The authors wish to thank Ashley Suh, Mustafa Hajij, and Carlos Scheidegger for initial discussions. This work was supported in part by grants from the National Science Foundation (NSF) IIS-1513616 and IIS-1845204, and Department of Energy (DOE) DE-SC0021015.