Connectivity-Faithful Graph Drawing

Amyra Meidiana
University of Sydney, Australia

Seok-Hee Hong
University of Sydney, Australia

Yongcheng Jing
University of Sydney, Australia

Abstract

Connectivity is one of the important fundamental structural properties of graphs, and a graph drawing D should faithfully represent the connectivity structure of the underlying graph G. This paper investigates connectivity-faithful graph drawing leveraging the famous Nagamochi-Ibaraki (NI) algorithm, which computes a *sparsification* G_{NI}, preserving the k-connectivity of a k-connected graph G.

Specifically, we first present *CFNI*, a divide-and-conquer algorithm, which computes a sparsification G_{CFNI}, which preserves the *global* k-connectivity of a graph G and the *local* h-connectivity of the h-connected components of G. We then present *CFGD*, a connectivity-faithful graph drawing algorithm based on CFNI, which faithfully displays the global and local connectivity structure of G. Extensive experiments demonstrate that CFNI outperforms NI with 66% improvement in the connectivity-related sampling quality metrics and 73% improvement in proxy quality metrics. Consequently, CFGD outperforms a naive application of NI for graph drawing, in particular with 62% improvement in stress metrics. Moreover, CFGD runs 51% faster than drawing the whole graph G, with a similar quality.

2012 ACM Subject Classification Human-centered computing → Graph drawings

Keywords and phrases Graph connectivity, Faithful graph drawing, Graph sparsification

Digital Object Identifier 10.4230/LIPIcs.GD.2024.17

Funding This work was funded by ARC (Australian Research Council) DP (Discovery Project) grant DP190103301.

1 Introduction

Connectivity is a fundamental structural property of graphs due to its wide communication, transportation, and production applications. Consequently, tremendous progress has been made in algorithms and complexity theory related to graph connectivity [40]. For example, algorithms for various aspects of the connectivity have been presented, ranging from computing the connectivity of a graph [27], increasing the connectivity of a graph through edge augmentation [41], and decomposing a graph into connected components [18].

One notable problem is finding the minimum k-connected spanning subgraph of a k-connected graph, which is NP-complete [13]. Nevertheless, an efficient linear-time algorithm for finding a k-connected spanning subgraph of a k-connected graph with an upper bound of a linear number of edges has been presented [39]. Specifically, the NI (Nagamochi and Ibaraki) algorithm computes a k-connected spanning subgraph with $O(kn)$ edges for a k-connected graph $G = (V, E)$ in $O(m)$ time, where $n = |V|$ and $m = |E|$.

In graph drawing, the *faithfulness* is an important quality metric to measure how the drawing faithfully represents the ground truth structure of a large and complex graph. Examples include distance-faithful metrics known as stress [8], shape-based metrics [9], cluster-faithful metrics [31], symmetry-faithful metrics [32, 33], neighborhood-faithful metrics [26],

32nd International Symposium on Graph Drawing and Network Visualization (GD 2024).
Editors: Stefan Felsner and Karsten Klein; Article No. 17; pp. 17:1–17:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(a) G. **(b)** G_1 (computed by NI). **(c)** G_3 (computed by CFNI).

Figure 1 Comparison of NI and CFNI: the sparsification computed by CFNI preserves both the global sparse connectivity and the local dense connectivity structures of the graph G, better than the sparsification computed by NI.

and change-faithful metrics [29]. In accordance, graph drawings that aim to optimize such faithfulness metrics have been investigated, such as stress minimization layouts [12, 22, 52], and the layouts to optimize shape-based metrics [30] and cluster faithfulness [3]. However, connectivity-faithful graph drawing has not yet been investigated.

In this paper, we present the first study on connectivity-faithful graph drawings by leveraging the NI algorithm, which can compute a sparse subgraph preserving the k-connectivity of a graph. Specifically, we first note that simply preserving the global k-connectivity of a graph may not be effective for connectivity-faithful graph drawing. For example, while a spanning tree preserves the global connectivity of a one-connected graph G, a drawing solely based on the spanning tree may fail to faithfully represent the local connectivity of dense subgraphs of G.

Therefore, we first present *CFNI (Connectivity-Faithful NI)*, a divide-and-conquer approach utilizing NI, which preserves both the global k-connectivity of a graph G and the local h-connectivity of each connected component of G. We then present *CFGD (Connectivity-Faithful Graph Drawing)*, which leverages CFNI to compute connectivity-faithful graph drawings. Our main contributions can be summarized as follows:

1. We present *CFNI (Connectivity-Faithful NI)*, a divide-and-conquer approach for graph sparsification utilizing NI, to compute a connectivity-faithful sparsification, preserving not only the global k-connectivity of a graph G but also the local h-connectivity of each h-connected component of G, for $h > k$. Extensive experiments demonstrate that CFNI achieves, on average, 66% better connectivity-related sampling quality metrics and 73% better proxy quality metrics [44] than NI, which outperforms the state-of-the-art *SS (Spectral Sparsification)* [49] with 52% better connectivity-related sampling quality metrics.
2. We present *CFGD (Connectivity-Faithful Graph Drawing)*, which leverages CFNI for connectivity-faithful graph drawing to faithfully represent both the global and local connectivity structures in a graph. Experiments show that CFGD obtains better quality metrics than a naive application of NI to graph drawing, particularly at up to 62% lower stress on average. Furthermore, CFGD runs faster than directly drawing the whole graph, at 51% faster, with a similar quality.

Figure 1 compares CFNI and NI for a one-connected graph G. The spanning tree G_1 in Figure 1b is computed by NI, while G_3 in Figure 1c is computed by CFNI with $h = 3$ (preserving the triconnectivity of triconnected components of G). Clearly, G_3 better preserves both the global mesh-like structure and the locally dense structures than G_1, which loses the local connectivity.

2 Related Work

2.1 NI (Nagamochi-Ibaraki) Algorithm

Nagamochi and Ibaraki [39] presented a linear time algorithm to find a sparse k-connected spanning subgraph of a k-connected graph, based on the following main lemma:

▶ **Lemma 1.** *For graph $G = (V, E)$, let $F_i = (V, E_i)$ be a maximal spanning forest in $V - E_1 \cup E_2 \cup \ldots \cup E_{i-1}$ for $1 \leq i \leq |E|$ where possibly $E_i = E_{i+1} = \ldots = E_{|E|} = \{\}$ for some i. Each spanning subgraph $G_i = (V, E_1 \cup E_2 \cup \ldots \cup E_i)$ satisfies $\lambda(x, y, G_i) = min(\lambda(x, y, G), i)$ for all $x, y \in V$ where $\lambda(x, y, G)$ is the local connectivity between x and y in graph G.*

Based on Lemma 1, a subgraph $G_{NI} = (V, E')$ where $E' = E_1 \cup E_2 \cup \ldots \cup E_k$ is k-connected if $k \leq \lambda(x, y, G)$. To compute G_{NI}, one must compute the disjoint edge subsets $E_1, E_2, \ldots, E_m, m = |E|$, where each E_i is a maximal spanning forest in $G \setminus (E_1 \cup \ldots \cup E_{i-1})$. G_{NI} is then constructed using the union of E_1 to E_k, i.e., $G_k = (V, E_1 \cup E_2 \cup \ldots \cup E_k)$.

In other words, given a k-connected graph $G = (V, E)$, the NI algorithm computes an ordered list of disjoint edge subsets $E_1, E_2, \ldots, E_m$, such that (V, E_1) is one-connected, $(V, E_1 \cup E_2)$ is biconnected, $(V, E_1 \cup E_2 \cup E_3)$ is triconnected, and so on, up to E_k.

The NI algorithm takes a k-connected graph $G = (V, E)$ and starts by marking all $v \in V$ and $e \in E$ as "unscanned", and assigning a counter r to each $v \in V$, where all $r(v)$ starts at 0. The algorithm loops through every unscanned vertex, selecting a vertex with the highest r for each iteration. The algorithm then iterates through all unscanned edges $e = (x, y)$ incident on x, and adds e to the subset $E_{r(y)+1}$. If $r(x)$ is equal to $r(y)$, $r(x)$ is incremented by 1; otherwise, $r(y)$ is incremented by 1. e is then marked as "scanned", and once all unscanned edges incident on x has been scanned, x is marked as "scanned". The algorithm finally returns the k-connected spanning subgraph $G_{NI} = (V, E_1 \cup E_2 \cup \ldots \cup E_k)$. The following theorem describes the main results:

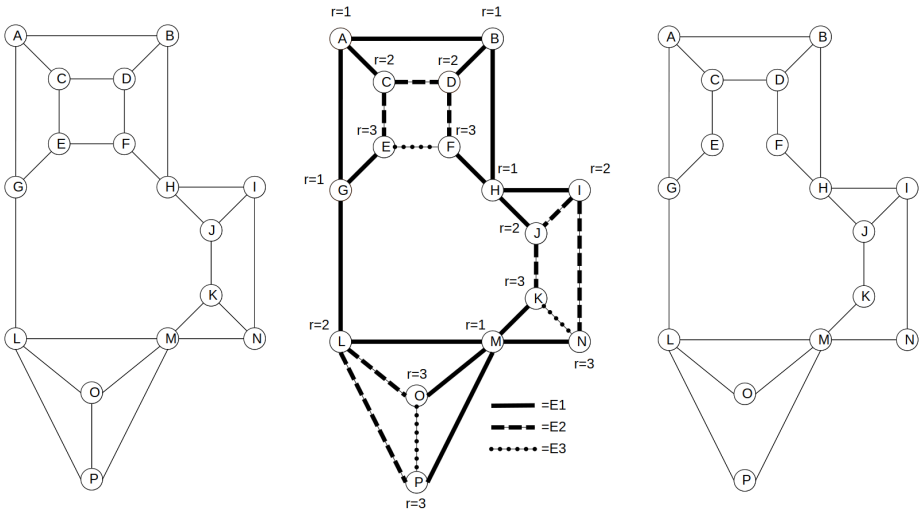
▶ **Theorem 2.** *Given a simple graph $G = (V, E)$, partition $E_i \subset E$ satisfying Lemma 1 can be found in $O(n + m)$ time, where $|E_i| \leq n - i$ for $i < n$ and $|E_i| = 0$ for $n \leq i \leq m$.*

The linear runtime comes from each vertex and edge being scanned once. As G is simple, $r(v)$ for $v \in V$ increases at most 1 when an incident vertex is scanned, thus $r(v) \leq n - 1$. Meanwhile, $|E_i| \leq n - 1$, as $|E_i| = n - 1$ implies that no more vertices have $r(v) < i$. Thus, the k-connected spanning subgraph $G_{NI} = (V, E_1 \cup E_2 \cup \ldots \cup E_k)$ can have at most $k(n - 1)$ edges.

Figure 2 shows an example of running NI on a graph, in this case, a biconnected graph G shown in Figure 2a. Figure 2b shows the result of running NI on G, in particular showing which edges belong to each edge set E_1, E_2, E_3 as well as the r values of each vertex at the end of the NI algorithm. Figure 2c shows the biconnected spanning subgraph G_k, obtained using the union of the edge sets $E_1 \cup E_2$ from the results in Figure 2b.

2.2 Graph Sampling and Spectral Sparsification

Graph sampling has been extensively studied within graph mining, where complex analysis can be computed more efficiently on the smaller sample graph G' than on the original large and complex graph G [21, 23]. The main challenge for graph sampling is to compute G', which is a good representative of G, preserving the structural properties of G. However, the most popular simple random sampling methods, such as Random Vertex (RV) or Random



(a) Biconnected graph G. **(b)** G after running NI. **(c)** Biconnected subgraph G_k.

Figure 2 Example of running NI on a biconnected graph.

Edge (RE), often produce disconnected samples, failing to preserve the connectivity of G [51]. Recent random sampling methods improve the connectivity of G' and reduce the computation time of G' using the BC (Block-Cut vertex) tree decomposition [16].

Spectral sparsification [48] computes G' preserving the spectrum of G, which is closely related to important structural properties such as clustering [50] and connectivity [6]. Every n-vertex graph G has a spectral sparsification G' with $O(n \log n)$ edges, which can be computed in near-linear time [49].

More recent work on graph sampling utilizes spectral sparsification to compute G', preserving the structural properties of G. For example, DSS (Deterministic SS) computes G' by selecting edges in decreasing order of effective resistance values [10]. Similarly, the SV (Spectral Vertex) sampling computes G' by selecting vertices in decreasing order of the sum of effective resistance values of their incident edges [20]. Both DSS and SV have been shown to perform significantly better than RE and RV, respectively, on various sampling quality metrics [10, 20].

Furthermore, spectral sparsification has been integrated with graph connectivity, such as the decomposition into biconnected (resp., triconnected) components using the BC (resp., SPQR) tree to reduce the computation time of G' and to improve the quality of G' including the connectivity, see [19, 34].

2.3 Fast Graph Drawing Algorithms using Sampling

Graph sampling methods have been successfully integrated with the most popular graph drawing methods, such as force-directed algorithms and stress minimization methods, to reduce the runtime complexity of the algorithms from quadratic time to linear time [14, 46, 52].

For example, the sparse stress-based algorithms [46, 52] sample a *pivot* set $P \subset V$ of constant size to reduce the stress computation from quadratic to linear time. Similarly, the RVS algorithm [14] uses a random vertex sampling method with a sliding window to reduce the runtime of repulsion force computation to linear time.

More recently, the fastest graph drawing algorithms using the sublinear-time force computation and stress computation have been presented [28, 36, 38]. For example, the SublinearForce framework [28] utilizes both vertex and edge sampling based on spectral sparsification to reduce the computation of both repulsion and attraction forces from linear to sublinear, while obtaining better quality than the linear-time RVS.

Sublinear-time stress computation algorithms have also been presented [38], based on the Stress Majorization and Stochastic Gradient Descent, integrating vertex sampling using spectral sparsification to reduce the stress computation from linear to sublinear time while producing drawings similar to SM and SGD.

2.4 Faithfulness Metrics and Faithful Graph Drawing

Faithfulness metrics are designed for evaluating drawings of large and complex graphs, by measuring how faithfully the ground truth structure of the graph is represented in a drawing [43]. Various faithfulness metrics have been presented based on the definition of the ground truth structure of the graph:

- *Stress* measures how proportional the geometric distances between vertices in a drawing are to the shortest path distance between the vertices in the graph [8].
- *Shape-based metrics* measure how faithfully the "shape" of the drawing, computed using the *proximity graph*, represents the ground truth structure of a graph [10, 15].
- *Proxy quality metrics* [44] measure how faithfully the drawing of a sample graph represents the ground truth structure of the original graph by computing the similarity between a graph G and the "shape" of the drawing D' of a sample graph $G' \subset G$.
- *Cluster faithfulness* [31] measures how faithfully the ground truth clustering of vertices is represented as the geometric clustering in the drawing.
- *Automorphism faithfulness* [32, 33] measures how faithfully the automorphisms of a graph are represented as symmetries in the drawing of a graph.
- *Change faithfulness* [4, 29] metrics are designed for *dynamic* graphs, measuring how proportional the change in the dynamic graph drawings is to the ground truth change of the structure of the dynamic graph.

Consequently, a number of graph drawing algorithms for optimizing faithfulness metrics have been investigated, such as stress minimization layouts [12, 22, 52], ShFR and ShSM algorithms to maximize shape faithfulness [30], and the ClusterKmeans and ClusterHAC algorithms to maximize cluster faithfulness [3].

3 CFNI: Connectivity-Faithful NI

While the NI algorithm successfully computes a spanning subgraph preserving the global k-connectivity of a graph, it may not always be sufficient to preserve the dense local connectivity structures of the graph for connectivity-faithful graph drawing. This may be an issue, especially for the graphs with a "globally sparse, locally dense" structure, such as the scale-free graphs often found in real-world social networks and biological networks [1].

To address this issue, we present CFNI, a divide-and-conquer approach leveraging NI for graph sparsification, which preserves both global and local connectivities. Given a k-connected graph G, CFNI takes as parameter a target connectivity h, and returns a subgraph preserving both the global k-connectivity of G and the local h-connectivity of each h-connected component of G.

Algorithm 1 CFNI.

1: **Input:** Graph $G = (V, E)$, target connectivity h
2: k: connectivity of G
3: **if** $h > k$ **then**
4: Decompose G into $k+1$-connected components $C_1, C_2, \ldots C_c$
5: **for** each $k+1$-connected component $C_i = (V_{C_i}, E_{C_i})$ **do**
6: $G_{h_i} = (V_{C_i}, E'_{C_i}) = \text{CFNI}(C_i, h)$
7: **end for**
8: $G_{CFNI} = (V, E'_h = E'_{C_1} \cup E'_{C_2} \cup \ldots \cup E'_{C_c})$
9: **return** G_{CFNI}
10: **end if**
11: $E_1 = E_2 = \ldots = E_m = \{\}$
12: $V_u = V$, $E_u = E$ // unscanned vertices and edges
13: $r(v) = 0$ for all $v \in V_{C_i}$
14: **while** $|V_u| > 0$ **do**
15: $x =$ vertex in V_u with largest r
16: **for** $\{e \in E | e = (x, y)\}$ **do**
17: $E_{r(y)+1} = E_{r(y)+1} \cup \{e\}$
18: **if** $r(x) == r(y)$ **then**
19: $r(x) += 1$
20: **end if**
21: $r(y) += 1$;
22: E_u.remove(e)
23: **end for**
24: V_u.remove(x)
25: **end while**
26: $G_{NI} = (V, E' = E_1 \cup E_2 \cup \ldots \cup E_k)$
27: **return** G_{NI}

Roughly speaking, the main idea of CFNI is to divide a k-connected graph into $k+1$-connected components, and then, for each $k+1$-connected component, recursively decompose it into $k+2$-connected components, and so on, until a decomposition into h-connected component is obtained. Finally, we run the NI algorithm for each connected component to preserve the local connectivity structure.

Algorithm 1 describes the steps of CFNI, which takes as input a graph $G = (V, E)$ and a target local connectivity h. h can be selected as any positive integer, not necessarily equal to the k-connectivity of G.

The algorithm first checks for the connectivity k of G. If $h > k$, the algorithm decomposes G into $k+1$-connected components, and recursively calls CFNI for each $k+1$-connected component (lines 3-7). The recursion stops when CFNI is called on a graph whose k-connectivity is no lower than h; at this step, NI is run on G (lines 11-26).

Once the recursive calls finish for all $k+1$-connected components, the local h-connectivity-preserving k-connected subgraph G_{CFNI} is finally computed using the union of the edge sets of the $k+1$-connected subgraphs of the $k+1$-connected components (line 8).

The runtime complexity of CFNI depends on the runtime of the h-connected component decomposition, while running NI on each component takes linear time. For example, the decomposition of one-connected (resp., biconnected) graphs into biconnected (resp., tricon-

nected) components takes linear time [17, 18]. Running NI on each h-connected component G_{hi} takes linear time in the number of edges in G_i, which sums up to $O(m)$ due to the number of edges in all of the connected components adding up to m.

The number of edges in G_{CFNI} is bounded by $O(hn)$. At the lowest level of recursion, CFNI decomposes a graph G into h-connected components, where NI is run on each h-connected component $G_{h_i} = (V_{h_i}, E_{h_i})$ to produce a h-connected subgraph $G'_{h_i} = (V_{h_i}, E'_{h_i})$ with $O(h|V_{h_i}|)$ edges. As G_{CFNI} is formed using the union of all G'_{h_i}, and given that the sum of all $|V_{h_i}|$ is n, the number of edges in G_{CFNI} is bounded by $O(hn)$.

4 CFGD: Connectivity-Faithful Graph Drawing

One popular method commonly used to draw big complex graphs is by utilizing *graph sparsification* [10, 19, 28, 34, 37, 38]. Namely, given a graph G, first compute a much smaller sparsified graph G', then compute a drawing D' of G'. Finally, the sparsified edges are added back to D', to obtain a drawing D of the whole graph G. While this approach is efficient (i.e., it has a much faster runtime than drawing the whole graph G), the effectiveness (i.e., the quality of the drawing D) depends on how well the sparsification G' preserves the structure of G.

Due to the limitation of NI in preserving the local connectivity of highly connected components of a graph G, a naive application of NI for graph drawing may not be sufficient to represent all important connectivity structures of a graph faithfully. For example, a drawing of a one-connected graph G based on the spanning tree may fail to depict cycles or misrepresent locally dense subgraphs. We, therefore, present *CFGD*, which leverages CFNI for connectivity-faithful graph drawing to overcome the weakness of NI in preserving the local connectivity of highly connected components.

Algorithm 2 CFGD.

Step 1: Compute subgraph $G_{CFNI} = (V, E'_h)$ preserving global k-connectivity and local h-connectivity of k-connected graph G using CFNI.
Step 2: Compute a drawing $D_{G_{CFNI}}$ of G_{CFNI} using a graph drawing algorithm.
Step 3: Add all edges in $E_{r_h} = E \setminus E'_h$ to $D_{G_{CFNI}}$ to obtain a drawing D of G.

We expect CFGD to be able to compute high-quality connectivity-faithful drawings due to CFNI preserving not only the global k-connectivity of a graph G but also the local h-connectivity of each h-connected component of G, while still obtaining a fast runtime due to the efficient runtime of CFNI.

5 CFNI Experiment

5.1 NI Experiment

We first evaluate the baseline performance of NI for graph sparsification by comparing NI to *SS (Spectral Sparsification)*, which has been shown to outperform stochastic sampling methods [20, 19, 10, 34]. In summary, NI outperforms SS on several connectivity-related sampling quality metrics, most notably on the connectivity-related metrics: Closeness Centrality at 52% better, and Betweenness Centrality at 20% better. The visual comparison also demonstrates the strengths of NI in preserving the overall connectivity structures that SS often fails to preserve, for *biconnected* graphs. Thus, both the quality metrics and visual comparisons demonstrate the strengths of NI over SS for connectivity-faithful sampling. For details of the experiment, see the journal version of this paper [35].

Although NI shows a good performance on biconnected graphs, for one-connected graphs, the performance of NI deteriorates since the spanning tree misses the local connectivity structures of graphs, such as cycles and clusters. We, therefore, conduct experiments to evaluate how CFNI improves upon NI for globally sparse and locally dense one-connected graphs.

5.2 CFNI Experiment Design

We now present comparison experiments to evaluate the strengths of CFNI over NI. Specifically, we use one-connected graphs as inputs with $h = 2, 3$, since efficient linear-time algorithms are known for computing biconnected components and triconnected components [18, 17]. We denote the sparsification of a graph G computed by NI as G_1, as $k = 1$ for the one-connected graphs. We then denote the sparsification computed by CFNI with $h = 2, 3$ as G_2 and G_3.

We use a mix of real-world and synthetic graphs with various connectivity structures: 1) real-world benchmark *scale-free* graphs, with globally sparse, locally dense clusters and small diameters [24]; 2) *GION* graphs, biochemical networks with globally sparse, locally dense clusters and long diameters [25]; 3) *mesh* graphs, with regular grid-like structures [7]; and 4) *black-hole* graphs, synthetic graphs with globally sparse mesh- or cycle-like structures with locally dense "blobs" attached [10]. See Table 1 for details.

Table 1 Data sets for the CFNI experiments.

(a) Scale-free.

G	$\|V\|$	$\|E\|$
soc_h	2000	16097
block_2000	2000	3992
oflights	2905	15645
tvcg	3213	10140
facebook	4039	88234
CA-GrQc	4158	13422
EVA	4475	4652
us_powergrid	4941	6594
as19990606	5188	9930
migrations	6025	9378
lastfm_asia	7624	27806

(b) Mesh.

G	$\|V\|$	$\|E\|$
dwt_1005	1005	4813
cage8	1015	4994
bcsstk09	1083	8677
nasa1824	1824	18692
plat1919	1919	15240
sierpinski3d	2050	6144
data	2851	15093
3elt	4720	13722

(c) GION.

G	$\|V\|$	$\|E\|$
2_gion	1159	6424
5_gion	1748	13957
6_gion	1785	20459
7_gion	3010	41757
8_gion	4924	52502
4_gion	5953	186279
1_gion	5452	118404
3_gion	7885	427406

(d) Black-hole.

G	$\|V\|$	$\|E\|$
G443	285	2009
Cycle759	377	4790
G462	733	62509
Cycle907	823	14995
Cycle896	1031	22638
G500	1080	17636
G887	4784	38135

5.3 Quality Metrics Comparison

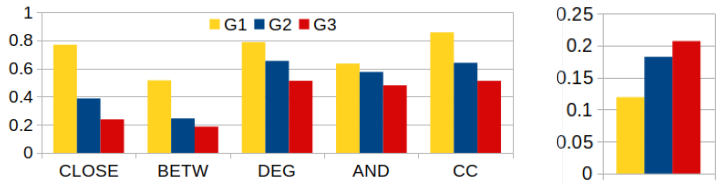
We use the well-known *sampling quality metrics* to measure how well the sparsifications preserve the following properties of the original graphs [21, 23]:

- Closeness Centrality (CLOSE) computes the "closeness" of a vertex to other vertices by summing up the length of all shortest paths between a vertex and all the other vertices [11].
- Betweenness Centrality (BETW) measures the ratio of all shortest paths between each pair of vertices that pass through a certain vertex [11].
- Degree Correlation Associativity (DEG) computes the likelihood that vertices link to other vertices of similar degrees [42].
- Average Neighbor Degree (AND) computes the average degree of a vertex's neighbors [2].
- Clustering Coefficient (CC) measures the clustering of edges into tightly connected neighborhoods and represents the extent of clustering tendency between vertices [47].

More specifically, we measure the sampling quality metrics using the *Kolmogorov-Smirnov (KS)* goodness-of-fit-test [5], to compare the similarity of the CDF (Cumulative Distributive Function) of each graph metric of the original and sparsified graphs. The KS distance has a value between 0 and 1, where 0 means completely identical CDFs.

We compute the percentage ratio of the difference to compare the metrics computed by G_1 (i.e., computed by NI) and G_2, G_3 (i.e., computed by CFNI). For example, to compute the percentage difference of AND computed by G_1 and G_3, we use the formula $\frac{AND(G_1)-AND(G_3)}{AND(G_1)}$.

Figure 3a shows the sampling quality metrics computed on G_1, G_2, and G_3, averaged over all data sets. Clearly, G_2 and G_3 achieve notably better sampling quality metrics over G_1, and G_3 further obtains better metrics over G_2. The largest improvements are seen on the connectivity-related metrics Closeness centrality and Betweenness centrality: averaged over both, G_2 and G_3 obtain 51% and 66% improvements, respectively, compared to G_1. Improvements can also be seen over the other three metrics, with G_2 and G_3 obtaining 17% and 33% improvements, respectively, over G_1.



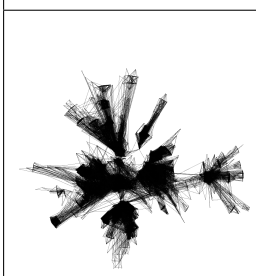
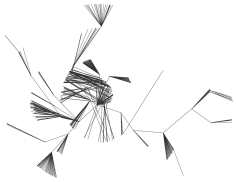
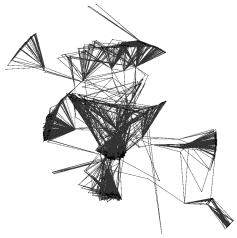
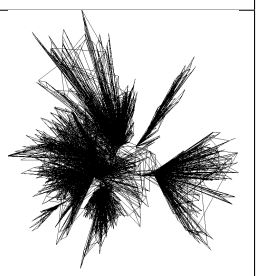
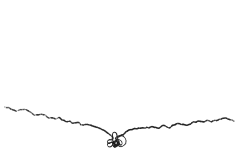
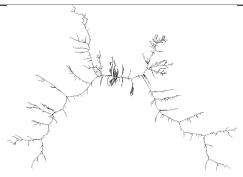
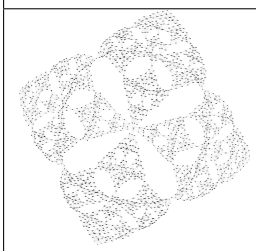
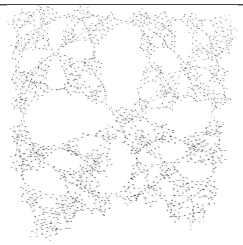
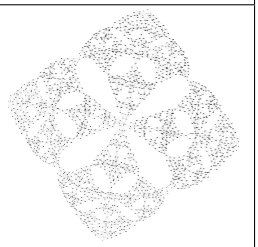
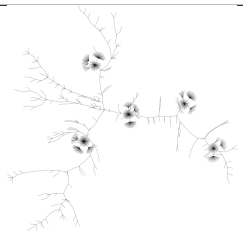
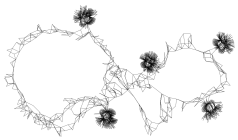
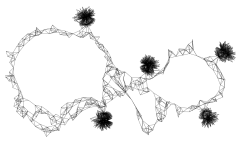
(a) Sampling quality metrics. **(b)** Prox. qual.

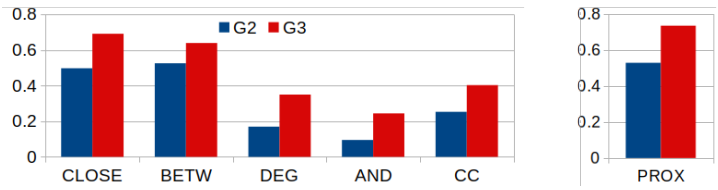
Figure 3 Average sampling (lower = better) and proxy quality metrics (higher = better) for G_1, G_2, and G_3. On average, G_3 obtains significantly better metrics than G_1 (i.e., NI), especially on connectivity-related metrics CLOSE and BETW at 66% better on average.

To evaluate the effectiveness of the sparsifications for the purpose of graph drawing, we compute the *proxy quality metrics* [44], for measuring how faithfully the drawing of the sparsifications represents the ground truth structure of the original graph. We use the *Backbone* layout, specifically designed to untangle "hairball" drawings of large complex graphs [45], to draw G_1, G_2, and G_3.

Figure 3b shows the proxy quality metrics computed on G_1, G_2, and G_3, averaged over all data sets. Similar to the results for sampling quality metrics, G_2 and G_3 obtain notably better proxy quality metrics than G_1, on average 53% better by G_2 and 73% better by G_3.

Table 2 Visual comparison of the sparsified graphs computed by CFNI. G_2 and G_3 (by CFNI) consistently preserve the structure of the graph G better than G_1 (by NI), with G_3 significantly outperforming G_2 on some graphs, e.g., Facebook and Sierpinski3d.

G	G_1	G_2	G_3
facebook			
			
GION_1			
			
sierpinski3d			
			
Cycle896			
			



(a) Sampling metrics. **(b)** Proxy metrics.

Figure 4 Average improvements by G_2 and G_3 (computed by CFNI) over G_1 (computed by NI). CFNI obtains improvement over NI on all metrics, most significantly on connectivity-related sampling quality metrics CLOSE and BETW.

5.4 Visual Comparison

Table 2 shows example visual comparisons between the drawings of sparsifications computed by NI and CFNI, with the drawings of the whole graph G added as reference. Clearly, the drawings of G_2 and G_3 are very similar to those of G, i.e., they faithfully represent the original graph's structure, while G_1 often fails to preserve the structure of G. For example, see the GION graph GION_1, where G_1 misleadingly shows four "branches" expanding from the middle cluster while G_2 and G_3 show only two, more faithful to the original G.

Moreover, sometimes only G_3 is highly similar to G, while G_2 also fails to preserve the structure of G. For example, see the mesh graph Sierpinski3d, where G_1 completely fails to maintain the mesh structure of the original graph, and while G_2 manages to maintain the structure better, it is still distorted compared to G. Meanwhile, G_3 displays almost the same structure as G.

5.5 Discussion and Summary

Extensive experiments have demonstrated the strengths of CFNI over NI, preserving both the global and local connectivity structures of graphs. Figure 4 shows the average improvements obtained by G_2 and G_3 (i.e., running CFNI with $h = 2$ and $h = 3$ respectively) over G_1 (i.e., running NI). The largest improvements are seen in Closeness centrality and Betweenness centrality, which are both distance-based centralities highly related to connectivity. On average, these improvements are 51% better for G_2 and 66% better for G_3. Significant improvements are also seen in proxy quality metrics, at 53% better for G_2 and 73% for G_3. In addition, G_3 further obtains an average 31% improvement over G_2 averaged between Closeness centrality and Betweenness centrality, and 13% improvement for proxy quality metrics over G_2.

The visual comparisons in Table 2 validate the quality metrics, showing that G_3 (computed using CFNI with $h = 3$) represents both global and local connectivity structures of graphs much more faithfully than G_1 (computed by NI). In particular, for globally sparse and locally dense graphs such as the scale-free and black-hole graphs, G_3 faithfully represents both the overall global shape and the locally dense clusters better than G_1, improving the limitations of NI. Furthermore, G_3 also outperforms G_2 in cases where G_2 still has limitations capturing the structure of G, such as seen in the Facebook graph, where the drawing of G_3 is much more similar to G compared to that of G_2.

6 CFGD Experiment

6.1 Experiment Design

We now present experiments to evaluate the effectiveness of the CFGD approach, over a naive application of NI to graph drawing. We performed an initial experiment for the naive application of NI to graph drawing: given a k-connected graph G, we first compute the k-connected subgraph $G_{NI} = (V, E')$ using NI, then compute a drawing $D_{G_{NI}}$ of G_{NI}, and finally add back the edges in $E_r = E \setminus E'$ to produce the drawing $D_{G_{NI}+E_r}$ of G. On average, computing $D_{G_{NI}+E_r}$ is 30% faster than directly computing a drawing D of G (i.e., applying a graph drawing algorithm directly on G), with on average 11% better edge crossing and only 15% lower shape-based metrics and neighborhood preservation. However, stress is significantly higher, at 55% higher on average. For details, see the journal version of this paper [35].

We thus present experiments to evaluate how CFGD improves over a naive application of NI to graph drawing. For the CFGD experiments, we use *one-connected* graphs, with $h = 2, 3$, with the same data set as used in Section 5, and use the Backbone layout [45] for its strengths in untangling "hairballs" in drawings of large, complex graphs. We denote the drawing obtained using the sparsified graph computed by NI as D_1, corresponding to the notation G_1 for the result of running NI on a 1-connected graph G used in Section 5. Similarly, we use D_h to simplify the notation $D_{G_{CFNI}+E_{r_h}}$ used to denote the resulting drawing from running CFGD on a graph G, i.e., we denote the drawing computed by CFGD using $h = 2$ and $h = 3$ as D_2 and D_3 respectively.

6.2 Runtime Comparison

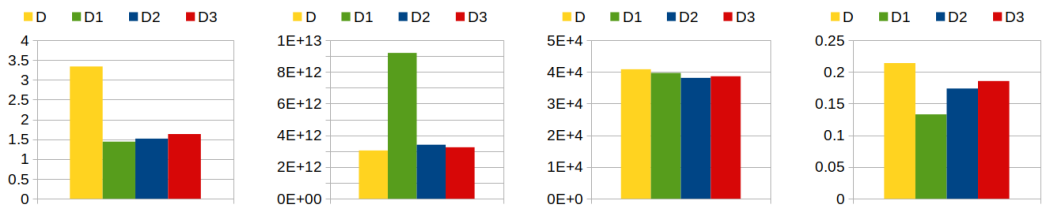
Figure 5a shows the average runtimes of computing D_1, D_2, D_3 compared to computing a drawing D directly from G. CFGD always runs significantly faster than drawing G directly, with over 50% runtime improvement on both D_2 and D_3. On average, the runtime of computing D_3 is still very similar to D_1, at only around 5% difference in runtime improvement over D, showing that CFGD still preserves much of the runtime improvement obtained by a naive application of NI for graph drawing.

6.3 Quality Metrics Comparison

To evaluate the performance of CFGD, we compare its results to those obtained from drawing a graph directly, using graph drawing quality metrics. We use a selection of commonly-used graph drawing quality metrics: stress [8], edge crossing, and shape-based metrics [10, 15]. See Section 2.4 for details on the metrics.

Stress. Figure 5b shows the average stress of D, D_1, D_2, and D_3. On stress, we see the largest improvement obtained by CFGD over a naive application of NI for graph drawing: D_2 and D_3 obtain much lower stress than D_1, at over 62% lower on average. This also brings the stress to be relatively similar to that of D, at only about 7% difference for D_3, in contrast to D_1 obtaining over two times higher stress than D.

Edge crossing. Figure 5c shows the average edge crossing metrics on D, D_1, D_2, and D_3. Surprisingly, even on D_1, edge crossing is already almost the same as D, at only 3% lower on average. D_2 and D_3 also show good performance, both at around 3% lower than D_1 on average, and furthermore even slightly better than D at around 6% better on average.

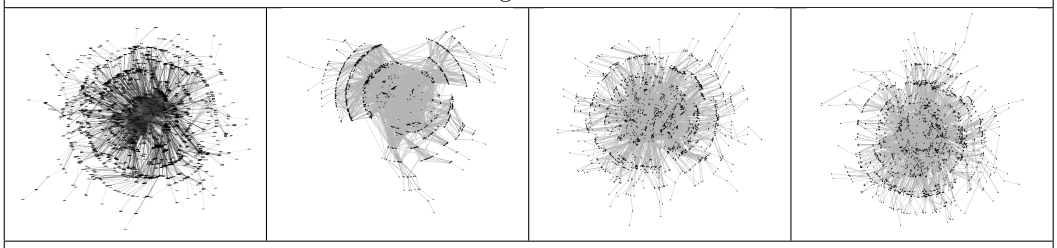
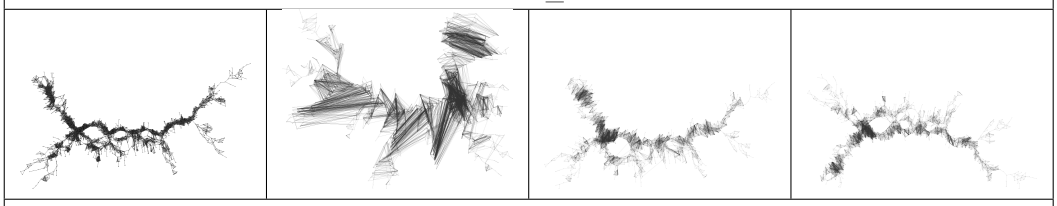
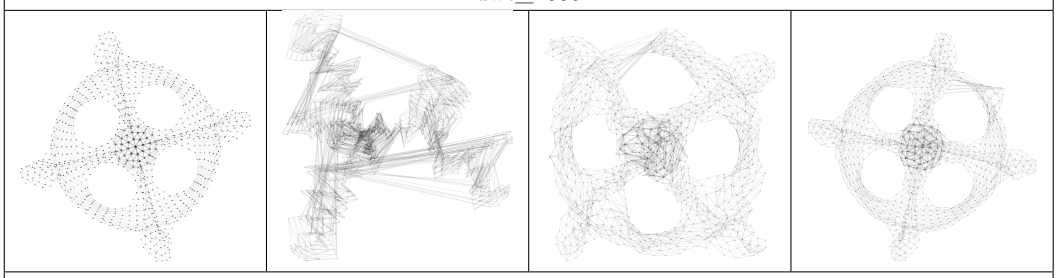
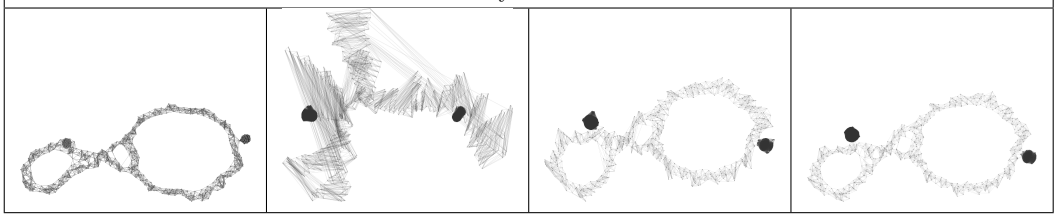


(a) Runtime. **(b)** Stress. **(c)** Edge crossing. **(d)** Shape-based.

Figure 5 Average runtime and quality metrics (lower is better for stress and edge crossing, and higher is better for shape-based) of computing D_1, D_2, and D_3 compared to computing D directly on G. CFGD (D_2 and D_3) obtains significant runtime improvements over computing D directly on G, while obtaining significantly lower stress than D_1 and similar metrics to D.

Shape-based metrics. Figure 5d shows the average shape-based metrics of D, D_1, D_2, and D_3. D_2 and D_3 show notable improvements over D_1, on average 31% and 40% higher. Furthermore, this brings the shape-based metrics of the drawings computed by CFGD closer to those of the drawings computed directly from G: with D_3, the shape-based metrics are around 13% lower than D, significantly lower than the 51% improvement in runtime.

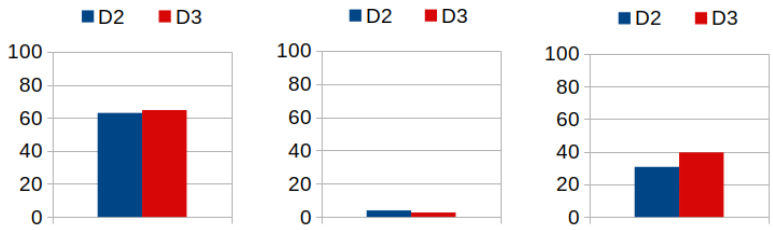
Table 3 Visual comparison of the drawings computed by CFGD. D_2 and D_3 (by CFGD) clearly depict the structures of the graphs more faithfully than D_1, with D_3 further removing some distortion issues occasionally displayed by D_2 (e.g., on dwt_1005).

D	D_1	D_2	D_3
migrations			
			
GION_5			
			
dwt_1005			
			
Cycle907			
			

6.4 Visual Comparison

Table 3 shows some example visual comparisons of CFGD to directly drawing graph G. It can be seen that in general, drawing D_1, i.e., drawing the sparsification computed by NI, often fails to maintain the structure of G, as can be seen in the direct drawing D. D_2 and D_3 are often far more successful in preserving the structures of graphs, such as those seen in the scale-free graph migrations and the GION graph GION_5.

In other cases, D_3 still succeeds in preserving the structure when even D_2 fails. For example, with the mesh graph dwt_1005, D_2 manages to maintain the overall four-pronged shape, but the drawing is still distorted compared to D. Similarly, for the black-hole graph Cycle907, although D_2 preserves both the local blobs and the overall "cycle"-like global structure, the shape is somewhat distorted with "zig-zags". Meanwhile, D_3 of both graphs mostly eliminates the distortion in the drawings compared to D_2.



(a) Stress. **(b)** Edge crossing. **(c)** Shape-based.

Figure 6 Average improvements (in %) in quality metrics computed by D_2 and D_3 over D_1, i.e., improvement of CFGD over a naive application of NI to graph drawing. CFGD obtains improvements on all quality metrics, with the largest improvement on stress at over 63%.

6.5 Discussion and Summary

Our experiments have demonstrated the effectiveness of CFGD. Figure 6 shows the average improvements in quality metrics obtained by D_2 and D_3 over D_1. In particular, the largest improvement is seen on stress: D_3 obtains on average 62% lower stress than D_1. Looking at the visual comparison, drawings D_1 often contain very long edges between vertices that neighbor each other in the original graph G but are in distant branches in the spanning tree, leading to high stress. Meanwhile, these long edges are absent in D_2 and D_3, leading to much lower stress compared to D_1.

Surprisingly, D_1, D_2, and D_3 obtain edge crossings very similar to D, even slightly better at 3%, 6%, and 6% lower on average, respectively. Most of this improvement is on scale-free and black-hole graphs, both containing graphs with globally sparse, locally dense structures. One possible explanation can be seen from the black-hole graphs, such as can be seen in graph Cycle907 in Table 3 where the locally dense blobs are drawn with a larger area in drawings D_1, D_2, D_3 compared to D. This may have removed some of the edge crossings introduced in D due to the blob being compressed into a smaller drawing area.

7 Conclusion

We present the first study of connectivity-faithful graph drawing, by leveraging the NI algorithm to graph sparsification and drawing. Specifically, we present local connectivity-preserving divide-and-conquer approaches CFNI and CFGD, to improve on the limitations of NI by not only preserving the global k-connectivity of a k-connected graph G, but also preserving the local connectivities of h-connected components of G, where $h > k$.

We demonstrate the effectiveness of CFNI over a naive application of NI, obtaining up to 66% average better connectivity-related sampling quality metrics and 73% better proxy quality metrics over NI. We also demonstrate the effectiveness of CFGD over a naive application of NI to graph drawing, most notably with 62% lower stress; CFGD also runs 51% faster than directly drawing the whole graph with similar quality metrics.

Future work includes evaluations of CFNI and CFGD using higher local connectivity.

References

1. Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
2. Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences*, 101(11):3747–3752, March 2004.
3. Shijun Cai, Seok-Hee Hong, Amyra Meidiana, Peter Eades, and Daniel Keim. Cluster-faithful graph visualization: New metrics and algorithms. In *IEEE PacificVis*, pages 192–201, 2024. doi:10.1109/PACIFICVIS60374.2024.00029.
4. Shijun Cai, Amyra Meidiana, and Seok-Hee Hong. Dnc: Dynamic neighborhood change faithfulness metrics. In *EuroVis*, pages 49–53, 2022. doi:10.2312/EVS.20221092.
5. Indra Mohan Chakravarti, Radha Govira Laha, and Jogabrata Roy. Handbook of methods of applied statistics. *Wiley Series in Probability and Mathematical Statistics*, 1967.
6. Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.
7. Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *Transactions on Mathematical Software*, 38(1):1, 2011.
8. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, 1998.
9. Peter Eades, Seok-Hee Hong, An Nguyen, and Karsten Klein. Shape-based quality metrics for large graph visualization. *JGAA*, 21(1):29–53, 2017. doi:10.7155/JGAA.00405.
10. Peter Eades, Quan Nguyen, and Seok-Hee Hong. Drawing big graphs using spectral sparsification. In *Graph Drawing*, pages 272–286, 2017.
11. Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.
12. Emden R Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *Graph Drawing*, pages 239–250, 2005.
13. Michael R Garey and David S Johnson. *Computers and Intractability*, volume 174. freeman San Francisco, 1979.
14. Robert Gove. A random sampling o(n) force-calculation algorithm for graph layouts. *Computer Graphics Forum*, 38(3):739–751, 2019. doi:10.1111/CGF.13724.
15. Seok-Hee Hong, Amyra Meidiana, James Wood, Juan Pablo Ataides, Peter Eades, and Kunsoo Park. dgg, drng, dsc: New degree-based shape-based faithfulness metrics for large and complex graph visualization. In *IEEE PacificVis*, pages 51–60, 2022. doi:10.1109/PACIFICVIS53943.2022.00014.
16. Seok-Hee Hong, Quan Nguyen, Amyra Meidiana, Jiaxi Li, and Peter Eades. BC tree-based proxy graphs for visualization of big graphs. In *IEEE PacificVis*, pages 11–20, 2018.
17. John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.
18. John E Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM Journal on computing*, 2(3):135–158, 1973. doi:10.1137/0202012.
19. Jingming Hu, Tuan Tran Chu, Seok-Hee Hong, Jialu Chen, Amyra Meidiana, Marnijati Torkel, Peter Eades, and Kwan-Liu Ma. BC tree-based spectral sampling for big complex network visualization. *Appl. Netw. Sci.*, 6(1):60, 2021. doi:10.1007/S41109-021-00405-3.
20. Jingming Hu, Seok-Hee Hong, and Peter Eades. Spectral vertex sampling for big complex graphs. In *Complex Networks*, pages 216–227, 2019. doi:10.1007/978-3-030-36683-4_18.
21. Pili Hu and Wing Cheong Lau. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865*, 2013. arXiv:1308.5865.
22. Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. doi:10.1016/0020-0190(89)90102-6.
23. Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *ACM SIGKDD*, pages 631–636, 2006. doi:10.1145/1150402.1150479.

24 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

25 Michael R Marner, Ross T Smith, Bruce H Thomas, Karsten Klein, Peter Eades, and Seok-Hee Hong. GION: interactively untangling large graphs on wall-sized displays. In *Graph Drawing*, pages 113–124, 2014. doi:10.1007/978-3-662-45803-7_10.

26 Rafael Messias Martins, Rosane Minghim, Alexandru C Telea, et al. Explaining neighborhood preservation for multidimensional projections. In *CGVC*, pages 7–14, 2015.

27 David W Matula. Determining edge connectivity in o(nm). In *SCFS*, pages 249–251, 1987. doi:10.1109/SFCS.1987.19.

28 Amyra Meidiana, Seok-Hee Hong, Shijun Cai, Marnijati Torkel, and Peter Eades. Sublinear-force: Fully sublinear-time force computation for large complex graph drawing. *IEEE TVCG*, 30(7):3386–3399, 2024. doi:10.1109/TVCG.2022.3233287.

29 Amyra Meidiana, Seok-Hee Hong, and Peter Eades. New quality metrics for dynamic graph drawing. In *Graph Drawing*, pages 450–465, 2020. doi:10.1007/978-3-030-68766-3_35.

30 Amyra Meidiana, Seok-Hee Hong, and Peter Eades. Shape-faithful graph drawings. In *Graph Drawing*, pages 93–108, 2022. doi:10.1007/978-3-031-22203-0_8.

31 Amyra Meidiana, Seok-Hee Hong, Peter Eades, and Daniel Keim. A quality metric for visualization of clusters in graphs. In *Graph Drawing*, pages 125–138, 2019. doi:10.1007/978-3-030-35802-0_10.

32 Amyra Meidiana, Seok-Hee Hong, Peter Eades, and Daniel Keim. Quality metrics for symmetric graph drawings. In *IEEE PacificVis*, pages 11–15, 2020. doi:10.1109/PACIFICVIS48177.2020.1022.

33 Amyra Meidiana, Seok-Hee Hong, Peter Eades, and Daniel A. Keim. Automorphism faithfulness metrics for symmetric graph drawings. *IEEE TVCG*, 30(7):3241–3255, 2024. doi:10.1109/TVCG.2022.3229354.

34 Amyra Meidiana, Seok-Hee Hong, Jiajun Huang, Peter Eades, and Kwan-Liu Ma. Topology-based spectral sparsification. In *LDAV*, pages 73–82, 2019. doi:10.1109/LDAV48142.2019.8944358.

35 Amyra Meidiana, Seok-Hee Hong, and Yongcheng Jing. Connectivity-Faithful Graph Sparsification for Drawing Large and Complex Graphs. *Submitted*, 2024.

36 Amyra Meidiana, Seok-Hee Hong, Marnijati Torkel, Shijun Cai, and Peter Eades. Sublinear Time Force Computation for Big Complex Network Visualization. *Computer Graphics Forum*, 39(3):579–591, 2020. doi:10.1111/CGF.14003.

37 Amyra Meidiana, Seok-Hee Hong, Marnijati Torkel, Shijun Cai, and Peter Eades. Sublinear time force computation for big complex network visualization. *Computer Graphics Forum*, 39(3):579–591, 2020. doi:10.1111/CGF.14003.

38 Amyra Meidiana, James Wood, and Seok-Hee Hong. Sublinear-time algorithms for stress minimization in graph drawing. In *IEEE PacificVis*, pages 166–175, 2021. doi:10.1109/PACIFICVIS52677.2021.00030.

39 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of ak-connected graph. *Algorithmica*, 7(1):583–596, 1992. doi:10.1007/BF01758778.

40 Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*, volume 123 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2008. doi:10.1017/CBO9780511721649.

41 Dalit Naor, Dan Gusfield, and Charles Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, 1997. doi:10.1137/S0097539792234226.

42 M. E. J. Newman. Mixing patterns in networks. *Physical Review E*, 67(2), February 2003.

43 Quan Nguyen, Peter Eades, and Seok-Hee Hong. On the faithfulness of graph visualizations. In *IEEE PacificVis*, pages 209–216, 2013.

44 Quan Hoang Nguyen, Seok-Hee Hong, Peter Eades, and Amyra Meidiana. Proxy graph: Visual quality metrics of big graph sampling. *IEEE TVCG*, 23(6):1600–1611, 2017. doi:10.1109/TVCG.2017.2674999.

45 Arlind Nocaj, Mark Ortmann, and Ulrik Brandes. Untangling hairballs - from 3 to 14 degrees of separation. In *Graph Drawing*, 2014.

46 Mark Ortmann, Mirza Klimenta, and Ulrik Brandes. A sparse stress model. In Yifan Hu and Martin Nöllenburg, editors, *Graph Drawing*, pages 18–32, 2016. doi:10.1007/978-3-319-50106-2_2.

47 Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and János Kertész. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75:027105, March 2007.

48 Daniel A Spielman. Spectral graph theory and its applications. In *IEEE FOCS*, pages 29–38, 2007. doi:10.1109/FOCS.2007.66.

49 Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. doi:10.1137/08074489X.

50 Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007. doi:10.1007/S11222-007-9033-Z.

51 Yanhong Wu, Nan Cao, Daniel Archambault, Qiaomu Shen, Huamin Qu, and Weiwei Cui. Evaluation of graph sampling: A visualization perspective. *IEEE TVCG*, 23(1):401–410, 2016. doi:10.1109/TVCG.2016.2598867.

52 Jonathan X Zheng, Samraat Pawar, and Dan FM Goodman. Graph drawing by stochastic gradient descent. *IEEE TVCG*, 25(9):2738–2748, 2018. doi:10.1109/TVCG.2018.2859997.