Computers & Graphics (2020)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Spectrum-Preserving Sparsification for Visualization of Big Graphs

Martin Imre[a], Jun Tao[b], Yongyu Wang[c], Zhiqiang Zhao[c], Zhuo Feng[c], Chaoli Wang[a]

[a]*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA*
[b]*School of Data and Computer Science, Sun Yat-sen University, Guangzhou, Guangdong 510006, China*
[c]*Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931, USA*

ARTICLE INFO

Article history:
Received February 20, 2020

Keywords: Big graph visualization, spectral graph sparsification, node reduction, spectral clustering, edge bundling.

ABSTRACT

We present a novel spectrum-preserving sparsification algorithm for visualizing big graph data. Although spectral methods have many advantages, the high memory and computation costs due to the involved Laplacian eigenvalue problems could immediately hinder their applications in big graph analytics. In this paper, we introduce a practically efficient, nearly-linear time spectral sparsification algorithm for tackling real-world big graph data. Besides spectral sparsification, we further propose a node reduction scheme based on intrinsic spectral graph properties to allow more aggressive, level-of-detail simplification. To enable effective visual exploration of the resulting spectrally sparsified graphs, we implement spectral clustering and edge bundling. Our framework does not depend on a particular graph layout and can be integrated into different graph drawing algorithms. We experiment with publicly available graph data of different sizes and characteristics to demonstrate the efficiency and effectiveness of our approach. To further verify our solution, we quantitatively compare our method against different graph simplification solutions using a proxy quality metric and statistical properties of the graphs.

1. Introduction

Spectral methods are playing an increasingly important role in many graph-based applications [1], such as scientific computing [2], numerical optimization [3], image processing [4], data mining [5], machine learning [6], and graph analytics [7]. For example, classical spectral clustering algorithms leverage the eigenvectors corresponding to a few smallest nontrivial (i.e., nonzero) eigenvalues of Laplacians for low-dimensional spectral graph embedding, which is followed by a k-means clustering procedure that usually leads to high-quality clustering results. Although spectral methods have many advantages, such as easy implementation, good solution quality, and rigorous theoretical foundations [8, 9, 10], the high memory and computation cost due to the involved Laplacian eigenvalue problems could hinder their applications in many emerging big graph analytical tasks [11, 7, 12].

Graph sparsification refers to the approximation of a large graph using a sparse graph. Compared to the original graphs, sparsified graphs provide a number of advantages for subsequent analysis and visualization. For example, sparsified transportation networks allow for developing more scalable navigation or routing algorithms for large transportation systems; sparsified social networks enable more effective understanding and prediction of information propagation in large social networks; and sparsified matrices can be leveraged to efficiently compute the solution of a large linear system of equations. Recent research efforts on *spectral graph sparsification* allow computing nearly-linear-sized subgraphs or sparsifiers (i.e., the number of edges is similar to the number of nodes in the subgraph) that can robustly preserve the spectrum (i.e., eigenvalues and eigenvectors) of the original graph Laplacian. This leads to a series of "theoretically nearly-linear-time" numerical and graph algorithms for solving sparse matrices, graph-based semi-supervised learning, spectral graph clustering, and max-

flow problems [13, 14, 15, 16, 17, 18, 19, 3, 2]. However, the long-standing question of whether there exists a practically efficient spectral graph sparsification algorithm for tackling general large-scale, real-world graphs still remains. For instance, the state-of-the-art nearly-linear time spectral sparsification methods leverage Johnson-Lindenstrauss Lemma to compute effective resistances for the edge sampling procedure [14]. This requires solving the original graph Laplacian multiple times, thus making them impractical for handling real-world big graph problems.

In this paper, we present spectrum-preserving sparsification (SPS), a spectrum-preserving framework for sparsification and visualization of big graph data. For sparsification, we realize the nearly-linear time, yet practically scalable spectrum-preserving big graph sparsification by leveraging a generalized eigenvalue perturbation analysis framework. Our spectral graph sparsification framework will guarantee the preservation of the key eigenvalues and eigenvectors within nearly-linear-sized spectrally-similar graph sparsifiers, achieving more efficient and effective compression of arbitrarily complex big graph data. Furthermore, based on intrinsic spectral graph properties, we propose a multilevel scheme for node reduction at varying levels of detail, enabling interactive hierarchical visualization of big graph data at runtime. For visualization, we develop a framework that fluidly integrates edge and node reduction, spectral clustering, and level-of-detail exploration to support adaptive visual exploration of big graph data. This provides users previously unavailable capabilities to navigate the large graphs toward effective visual exploration and reasoning.

To demonstrate the effectiveness of our approach, we conduct extensive experiments using large graphs publicly available at the Stanford Large Network Dataset Collection [20] and the University of Florida Sparse Matrix Collection [21]. The Stanford collection includes data sets from various applications (e.g., social networks, communication networks, citation networks, collaboration networks, road networks) with data gathered from different platforms (e.g., Amazon, Flickr, Reddit, Twitter, Wikipedia). The Florida collection includes a growing set of sparse matrices that arise in real applications such as social networks, web document networks, and geometric meshes. Graph data sets of different characteristics are selected to showcase the scalability and robustness of our spectral graph sparsification and visualization techniques. In summary, the contributions of our work are the following:

- First, we present an efficient spectral edge sparsification (SES) algorithm that preserves the most important spectral and structural properties within ultra-sparse graph sparsifiers, achieving superior speed performance compared to the state-of-the-art algorithms.
- Second, we propose a multilevel node reduction (MNR) scheme to further simplify the spectrally-sparsified graph, enabling level-of-detail exploration and speeding up the subsequent layout computation.
- Third, we integrate spectral clustering and edge bundling into graph drawing for effective visualization and exploration of the underlying big graph data.
- Fourth, we demonstrate the effectiveness of our solution against other graph simplification solutions through an objective evaluation using a proxy quality metric derived from the graphs and statistical properties of the graphs.

2. Related Work

2.1. Spectral Methods for Graph Application

To address the computational bottleneck of spectral methods in graph-related applications, recent research efforts aimed to reduce the complexity of the original graph Laplacian through various kinds of approximations. For example, k-nearest neighbor (kNN) graphs maintain k nearest neighbors for each node, whereas ε-neighborhood graphs keep the neighbors within the range of distance ε [22]. Williams and Seeger [23] introduced a sampling-based approach for affinity matrix approximation using the Nyström method, while its error analysis has been proposed in [24]. Chen and Cai [25] presented a landmark-based method for representing the original data points for large-scale spectral clustering. Yang et al. [26] proposed a general framework for fast approximate spectral clustering by collapsing the original data points into a small number of centroids using k-means or random-projection trees. Liu et al. [27] introduced a method for compressing the original graph into a sparse bipartite graph by generating a small number of "supernodes". Satuluri et al. [28] proposed a graph sparsification method for scalable clustering using a simple similarity-based heuristic. However, existing graph approximation methods cannot efficiently and robustly preserve the spectrums of the original graphs, and thus may lead to degraded or even misleading results. Recently, spectral perturbation analysis was applied to spectral graph sparsification and reduction in order to reduce the graph to nearly-linear-sized with high spectral similarity [29, 30, 31]. This progress makes it possible to develop much faster algorithms such as the symmetric diagonally dominant (SDD) matrix solvers [32] as well as spectral graph partitioning algorithm [30]. Note that these recent works on graph sparsification [29, 31, 32] only address spectral graph simplification but not spectral graph drawing using a multilevel approach. To our best knowledge, the integration of spectral sparsification, multi-level spectral clustering, graph layouts, and state-of-the-art edge bundling has not been attempted and thus poses a valid scientific contribution.

2.2. Spectral Graph Drawing

Among the spectral methods for graph drawing, the *eigen-projection* method uses the first few nontrivial eigenvectors of the graph Laplacian matrix or the top dominant eigenvectors of the adjacency matrix. Hall [33] used the eigenvectors of the Laplacian to embed graph vertices in a space of arbitrary dimension. The entries of the k eigenvectors related to the smallest nonzero eigenvalues are used as a node's coordinates. This is referred to as k-dimensional graph spectral embedding. Pisanski and Shawe-Taylor [34] took Hall's method to generate pleasing drawings of symmetrical graphs such as fullerene molecules in chemistry. Brandes and Willhalm [35] used eigenvectors of a modified Laplacian to draw bibliographic networks. Note that for *regular* graphs (where every node has the same degree), the eigenvectors of the Laplacian equal those of the

adjacency matrix, but in a reversed order. This is not the case for *non-regular* graphs. Using the Laplacian is advantageous as it is rooted in a more solid theoretical basis and gives better results than those obtained using the adjacency matrix.

Koren et al. [36, 37] proposed *algebraic multigrid computation of eigenvectors* (ACE), an extremely fast algorithm for drawing very large graphs. ACE identifies an optimal drawing of the graph by minimizing a quadratic energy function, which is expressed as a general eigenvalue problem and efficiently solved using fast algebraic multigrid implementation. Harel and Koren [38, 39] designed *high-dimensional embedding* (HDE) for aesthetic drawing of undirected graphs. HDE first embeds the graph in a very high dimension and then projects it into the 2D plane using principal component analysis. This algorithm is fast, exhibits the graph in various dimensions, and supports interactive exploration of large graphs. Koren [37, 40] presented a modified approach that uses *degree-normalized eigenvectors* to achieve aesthetic graph layouts. The degree normalized eigenvectors adjust the edge weights to reflect their relative importance in the related local scale. As such, the modified solution can allocate each cluster an adequate area in the drawing and avoid drawing extremely dense clusters. Hu et al. [7] designed a spectral graph drawing algorithm that includes *node projection*, *node dispersion*, and *sphere warping*. They first projected nodes onto a k-dimensional sphere, then dispersed nodes around the sphere's surface to separate apart densely connected clustered nodes, and finally warped the k-dimensional sphere's surface to a 2D space using multidimensional scaling. Their algorithm can clearly show the topology and community structures of the graph.

Most spectrum-based graph visualization techniques [34, 35, 36, 38, 37] only place their focus on graph layout. Besides drawing the graph using spectral sparsification, we integrate spectral clustering and edge bundling to help users better examine the graph for effective visual understanding. This is particularly important when handling big graph data as visual understanding of the complex and diverse graph relationships is the key.

2.3. Quality Metrics for Graph Sampling

An important question for graph sampling is how to evaluate the quality of the simplified graph. To evaluate the similarity between the original and sampled graphs, Hu and Lau [41] employed three metrics: (1) *total variation distance* which measures all the difference between two distributions; (2) *Kullback-Leibler divergence* which captures the difference between the two distributions accounting for the bulk of the distributions; and (3) *Kolmogorov-Smirnov statistic* which captures the maximum vertical distance of the cumulative distribution function of the two distributions. Zhang et al. [42] computed seven statistical properties, namely, *degree distribution*, *betweenness centrality distribution*, *clustering coefficient distribution*, *average neighbor degree distribution*, *degree centrality distribution*, *edge betweenness centrality distribution*, and *hop distribution*, to quantitatively compare different graph sampling methods. Recently, Hong et al. [43] used five metrics, namely, *degree correlation assortativity*, *closeness centrality*, *clustering coefficient*, *largest connected component*, and *average neighbor degree*, to evaluate their graph sampling methods, which improve random-based sampling by considering the block-cut tree.

A problem with the above statistical metrics and properties is that they are not well-suited to capture the visual quality of the corresponding graph layout. This is especially the case for large social and biological networks where nodes and edges could easily become "blobs" in the drawing of dense graphs with a few hundred vertices or sparse graphs with a few thousand vertices. Wu et al. [44] pointed out that quality metrics based on statistical or topological properties do not translate to visual quality. Their study shows that three visual factors significantly influence the representativeness of sampled graphs: *cluster quality*, *high degree nodes*, and *coverage area*. Eades et al. [45] proposed a shape-based quality metric for large graph visualization by treating the quality of a drawing D of a graph G as the similarity between G and the "shape" of the set of vertex locations of D. Nguyen et al. [46] generalized this metric to compare proxy graphs using the shape-based quality metric. In this paper, we use this so-called *proxy quality metric* to evaluate the graph after spectral edge sparsification (where only edges are removed) and employ statistical metrics to further evaluate the graph after multilevel node reduction (where nodes are aggregated to form pseudo-nodes).

3. Background

Consider a graph $G = (N, E, w)$ where N and E are the node set and edge set respectively, and w is a weight function that assigns positive weights to all edges. The symmetric diagonally dominant Laplacian matrix of G can be constructed as follows

$$\mathbf{L}_G(n_i, n_j) = \begin{cases} -w_{ij} & \text{if } e_{ij} \in E, \\ \sum_{e_{ik} \in E} w_{ik} & \text{if } n_i = n_j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

where n_i is a node, e_{ij} is the edge between n_i and n_j, and w_{ij} is the weight of e_{ij}. Graph sparsification aims to find $G' = (N, E', w')$, a subgraph or *sparsifier* of G that maintains the same set of nodes but fewer edges. To tell if two graphs have similar spectra, we usually use the following Laplacian quadratic form

$$\mathbf{x}^T \mathbf{L}_G \mathbf{x} = \sum_{e_{ij} \in E} w_{ij} (\mathbf{x}(n_i) - \mathbf{x}(n_j))^2, \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^N$ is a real vector. Two graphs G and G' are $\sigma-$*spectrally similar* if the following condition holds for all real vectors $\mathbf{x} \in \mathbb{R}^N$

$$\frac{\mathbf{x}^T \mathbf{L}_{G'} \mathbf{x}}{\sigma} \leq \mathbf{x}^T \mathbf{L}_G \mathbf{x} \leq \sigma \mathbf{x}^T \mathbf{L}_{G'} \mathbf{x}. \tag{3}$$

Defining the relative condition number to be $\kappa(\mathbf{L}_G, \mathbf{L}_{G'}) = \lambda_{\max}/\lambda_{\min}$, where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest nonzero generalized eigenvalues satisfying

$$\mathbf{L}_G \mathbf{u} = \lambda \mathbf{L}_{G'} \mathbf{u}, \tag{4}$$

where $\mathbf{u}$ is the generalized eigenvector of λ. It can be further shown that $\kappa(\mathbf{L}_G, \mathbf{L}_{G'}) \leq \sigma^2$, which indicates that a smaller relative condition number or σ^2 corresponds to a higher spectral similarity.

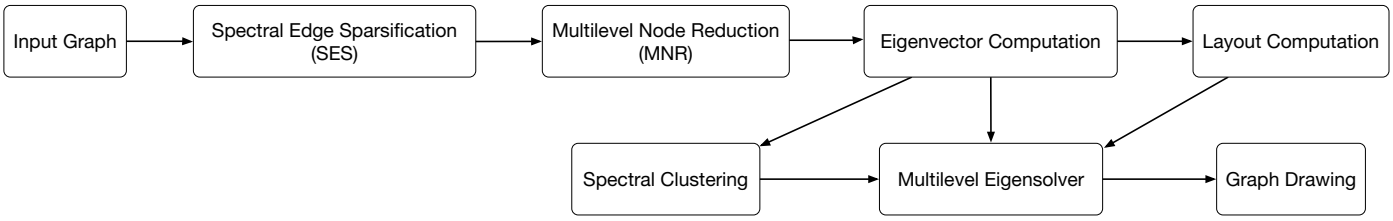


Fig. 1: The diagram of our SPS framework. Layout computation could use the eigenvector-based layout, t-SNE-based layout, or any other graph drawing algorithm.

The state-of-the-art nearly-linear time spectral sparsification algorithm leverages an edge sampling scheme that sets sampling probabilities proportional to edge *effective resistances* [14]. However, it becomes a chicken-and-egg problem since even approximately computing edge effective resistances by leveraging the Johnson-Lindenstrauss Lemma still requires solving the original graph Laplacian matrix $\log|N|$ times and thus can be extremely expensive for very large graphs, not to mention directly computing the Moore-Penrose pseudo inverse of graph Laplacians. For example, a recent work on graph drawing using spectral sparsification shows the major computational bottleneck is due to estimating edge effective resistances (by computing the Moore-Penrose pseudo inverse): even for a relatively small graph with $|N| = 7,885$, $|E| = 427,406$, the spectral sparsification procedure can take several hours to complete [12].

4. Our Approach

Figure 1 shows an overview of our SPS framework. Given the input graph, we first perform SES (Section 4.1) to reduce the number of edges. Next, based on the edge sparsification results, we perform MNR (Section 4.2) to further produce multiple levels of node simplification. This leads to a fairly small graph that preserves spectrally-important nodes and edges, allowing us to compute the eigenvectors of the graph Laplacian in an efficient manner. We then use these eigenvectors as input for dimensionality reduction using t-distributed stochastic neighbor embedding (t-SNE) [47, 48] and for spectral clustering using k-means. For spectral graph drawing (Section 4.3), we can layout the most simplified level of the graph based on the eigenvectors, t-SNE, and clustering results, where node positions are determined by either the leading eigenvectors or t-SNE projection and node colors are determined by spectral cluster labels. To obtain the graph drawing at a finer level, we can compute positions for newly-added nodes based on a multilevel eigensolver without recomputing the layout. Note that our SPS framework can readily work with other graph drawing algorithms by replacing the layout based on eigenvectors, t-SNE, or with another one.

4.1. Spectral Edge Sparsification (SES)

We outline the key steps of the proposed method for spectral graph sparsification of a given undirected graphs as follows: (1) low-stretch spanning tree extraction based on the original graph [49, 50]; (2) spectral embedding and criticality ranking of off-tree edges using approximate generalized eigenvectors leveraging the recent spectral perturbation analysis framework [29]; (3) subgraph densification by recovering a small portion of the most "spectrally critical" off-tree edges to the spanning tree; and (4) subgraph edge weight scaling via stochastic gradient descent (SGD) optimization.

In the following, we assume that $G = (N, E, w)$ is a weighted, undirected, and connected graph, whereas $G' = (N, E', w')$ is its graph sparsifier. The descending generalized eigenvalues of $\mathbf{L}_{G'}^{+}\mathbf{L}_G$ are denoted by $\lambda_{\max} = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$, where $\mathbf{L}_{G'}^{+}$ denotes the Moore-Penrose pseudoinverse of $\mathbf{L}_G$.

Spectral distortion of spanning-tree sparsifiers. Spielman [51] showed that there are not too many large generalized eigenvalues for spanning tree sparsifiers: $\mathbf{L}_{G'}^{+}\mathbf{L}_G$ has at most k generalized eigenvalues greater than $\mathrm{st}_{G'}(G)/k$, where $\mathrm{st}_{G'}(G)$ is the total stretch of the spanning-tree subgraph G' with respect to the original graph G that can be considered as the spectral distortion due to the spanning tree approximation. Recent research shows that every graph has a *low-stretch spanning tree* (LSST) such that the total stretch $\mathrm{st}_{G'}(G)$ can be bounded by [15]

$$O(|E|\log|N|\log\log|N|) \geq \mathrm{st}_{G'}(G) = \mathrm{tr}(\mathbf{L}_{G'}^{+}\mathbf{L}_G) = \textstyle\sum_{i=1}^{|N|}\lambda_i \geq \sigma^2, \tag{5}$$

where $\mathrm{tr}(\mathbf{L}_{G'}^{+}\mathbf{L}_G)$ is the trace of $\mathbf{L}_{G'}^{+}\mathbf{L}_G$. As a result, it is possible to construct an ultra-sparse yet spectrally similar sparsifier by recovering only a small portion of *spectrally critical* off-tree edges to the spanning tree. For example, σ-similar spectral sparsifiers with $O(|E|\log|N|\log\log|N|/\sigma^2)$ off-tree edges can be constructed in nearly linear time [29].

Edge embedding with generalized eigenvectors. To identify the off-tree edges that should be recovered to the spanning tree to dramatically reduce spectral distortion (the total stretch), Feng [29] introduced an off-tree edge embedding scheme using generalized eigenvectors, which is based on the following spectral perturbation framework. Considering the following first-order eigenvalue perturbation problem

$$\mathbf{L}_G(\mathbf{u}_i + \delta\mathbf{u}_i) = (\lambda_i + \delta\lambda_i)(\mathbf{L}_{G'} + \delta\mathbf{L}_{G'})(\mathbf{u}_i + \delta\mathbf{u}_i), \tag{6}$$

where a perturbation $\delta\mathbf{L}_{G'}$ is applied to $\mathbf{L}_{G'}$, which results in perturbations in generalized eigenvalues $\lambda_i + \delta\lambda_i$ and eigenvectors $\mathbf{u}_i + \delta\mathbf{u}_i$ for $i = 1, \ldots, n$, respectively. The first-order perturbation analysis shows that [29]

$$-\frac{\delta\lambda_i}{\lambda_i} = \mathbf{u}_i^T \delta\mathbf{L}_{G'}\mathbf{u}_i, \tag{7}$$

which indicates that the reduction of λ_i is proportional to the Laplacian quadratic form of $\delta\mathbf{L}_{G'}$ with the generalized eigenvector $\mathbf{u}_i$. Consequently, if the eigenvector $\mathbf{u}_1$ is applied, a significant reduction of the largest generalized eigenvalue λ_1 can

be achieved. Once all large generalized eigenvalues are dramatically reduced, the subgraph G' can serve as a very good spectral sparsifier of G.

To achieve effective reductions of large generalized eigenvalues, we exploit the following two key steps: (1) recover a small portion of most spectrally-critical off-tree edges into the spanning tree; (2) scale up edge weights in the subgraph G' to further improve the approximation. Additionally, the scaling factor obtained for each edge can be treated as its *spectral importance* in the subgraph: a larger scaling factor may indicate a more important role that the edge plays in mimicking the original graph.

Subgraph densification. If we denote $\mathbf{e}_{jk} \in \mathbb{R}^N$ the vector with only the j-th element being 1, the k-th element being -1, and others being 0, then the eigenvalue perturbation due to the inclusion of all off-tree edges can be expressed as follows

$$-\frac{\delta\lambda_i}{\lambda_i} = \mathbf{u}_i^T \delta\mathbf{L}_{G',\max}\mathbf{u}_i = \sum_{e_{jk}\in E\setminus E'} w_{jk}\left(\mathbf{e}_{jk}^T\mathbf{u}_i\right)^2 = \sum_{e_{jk}\in E\setminus E'} H_{jk}(\mathbf{u}_i), \tag{8}$$

where $\delta\mathbf{L}_{G',\max} = \mathbf{L}_G - \mathbf{L}_{G'}$ denotes the Laplacian including all off-tree edges, $H_{jk}(\mathbf{u}_i)$ denotes the Joule heat (power dissipation) of edge e_{jk} by considering the undirected graph G as a resistor network and $\mathbf{u}_i$ as the voltage vector. Equation (8) can also be considered as a spectral off-tree edge embedding scheme using generalized eigenvectors. It indicates that when using the first few dominant generalized eigenvectors for off-tree edge embedding, the top few generalized eigenvalues can be dramatically reduced by recovering the most spectrally-critical off-tree edges back to the spanning tree. In practice, we can leverage approximate eigenvectors computed via a few steps of generalized power iterations for good efficiency [29]:

- **Step 1:** Compute an approximate generalized eigenvector $\mathbf{h}_t$ from an initial random vector $\mathbf{h}_0$ via t-step generalized power iterations

$$\mathbf{h}_t = \left(\mathbf{L}_{G'}^{+}\mathbf{L}_G\right)^t \mathbf{h}_0 = \left(\mathbf{L}_{G'}^{+}\mathbf{L}_G\right)^t \sum_{i=1}^{|N|}\alpha_i\mathbf{u}_i = \sum_{i=1}^{|N|}\alpha_i\lambda_i^t\mathbf{u}_i; \tag{9}$$

- **Step 2:** Compute the Joule heat of all off-tree edges with $\mathbf{h}_t$ by

$$\begin{aligned}&\mathbf{h}_t^T\delta\mathbf{L}_{G',\max}\mathbf{h}_t = \textstyle\sum_{i=1}^{|N|}(\alpha_i\lambda_i^t)^2(\lambda_i-1)\\ &= \textstyle\sum_{e_{jk}\in E\setminus E'} w_{jk}\sum_{i=1}^{|N|}\alpha_i^2\lambda_i^{2t}\left(\mathbf{e}_{jk}^T\mathbf{u}_i\right)^2 = \sum_{e_{jk}\in E\setminus E'} H_{jk}(\mathbf{h}_t).\end{aligned} \tag{10}$$

Similar to Equation (8), Equation (10) also allows embedding generalized eigenvalues into the Laplacian quadratic form of each off-tree edge and thus ranking off-tree edges according to their spectral criticality levels: recovering the off-tree edges with the largest edge Joule heat values will most significantly decrease the largest generalized eigenvalues. In practice, using a small number (e.g., $0 < t < 3$) of power iterations suffices for the embedding purpose.

Subgraph edge scaling via SGD iterations. Once a sufficient number ($O(|E|\log|N|\log\log|N|/\sigma^2)$) of off-tree edges are selected and recovered to the spanning tree, the subgraph can already well mimic the original graph by approximating its first few Laplacian eigenvectors. To further mitigate the accuracy loss due to the missing edges in the subgraph, we introduce a novel edge scaling procedure that scales up edge weights in the subgraph so that λ_1 can be substantially reduced. To this end, we express the dominant eigenvalue perturbation $\delta\lambda_1$ in terms of edge weights perturbation δw as

$$-\frac{\delta\lambda_1}{\lambda_1} = \mathbf{u}_1^T\delta\mathbf{L}_{G'}\mathbf{u}_1 = \sum_{e_{jk}\in E'}\delta w_{jk}\left(\mathbf{e}_{jk}^T\mathbf{u}_1\right)^2, \tag{11}$$

which directly gives the sensitivity of λ_1 with respect to each edge weight w_{jk} as

$$\frac{\delta\lambda_1}{\delta w_{jk}} = -\lambda_1\left(\mathbf{e}_{jk}^T\mathbf{u}_1\right)^2 \approx -\lambda_1\left(\mathbf{e}_{jk}^T\mathbf{h}_t\right)^2. \tag{12}$$

With the weight sensitivity expressed in Equation (12), SGD iterations can be performed for scaling up edge weights: during each iteration of SGD, a random vector is first generated and used to compute the approximate dominant eigenvector ($\mathbf{h}_t$) using Equation (9) as well as edge weight sensitivities using Equation (12) for the following edge scaling step; when the edge weight sensitivities are small enough, we can terminate the SGD iterations. Since edge weights in G' will be updated during each SGD iteration, we need to solve a new subgraph Laplacian matrix $\mathbf{L}_{G'}$ for updating the approximate eigenvector $\mathbf{u}_1$ in Equation (12). This can be achieved by leveraging recent graph-theoretic algebraic multigrid algorithms that have shown highly scalable performance for solving large graph Laplacians [52, 53, 32]. Since the subgraph structure remains unchanged with only edge weights adjusted during the SGD iterations, it is also possible to incrementally update graph Laplacian solvers for achieving better computation efficiency.

4.2. Multilevel Node Reduction (MNR)

To generate the reduced graph based on the original graph (in our case, the graph after SES), our MNR framework applies a spectrum-preserving node aggregation scheme where the node affinity metric is considered [31]. Given neighboring nodes p and q, the node affinity between them is defined as [53, 54]

$$a_{p,q} = \frac{\|(\mathbf{X}_p,\mathbf{X}_q)\|^2}{(\mathbf{X}_p,\mathbf{X}_p)(\mathbf{X}_q,\mathbf{X}_q)},\ (\mathbf{X}_p,\mathbf{X}_q) = \sum_{k=1}^{K}\left(\mathbf{x}_p^{(k)}\cdot\mathbf{x}_q^{(k)}\right), \tag{13}$$

where $\mathbf{X} = (\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(K)})$ is a vector set with K test vectors which are computed by applying a few Gauss-Seidel (GS) relaxations to the linear system of equations $\mathbf{L}_G\mathbf{x}^{(i)} = 0$ for $i = 1,\ldots,K$, starting with K random vectors that are orthogonal to the all-one vector $\mathbf{1}$. If we consider $\tilde{\mathbf{x}}^{(i)}$ to be the approximate solution of $\mathbf{L}_G\mathbf{x}^{(i)} = 0$ after a few GS relaxations, and $\mathbf{x}^{(i)}$ to be the true solution, the error between $\tilde{\mathbf{x}}^{(i)}$ and $\mathbf{x}^{(i)}$ can be expressed as $\mathbf{e}_s^{(i)} = \mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}$. Due to the smoothing property of GS relaxation, $\mathbf{e}_s^{(i)}$ will only contain the smooth (low-frequency) modes of the initial error, while the oscillatory (high-frequency) modes of the initial error will be effectively removed [55]. Based on these K smoothed vectors in $\mathbf{X}$, we are able to embed each node

into a K-dimensional space such that nodes p and q are considered spectrally-close to each other if their low-dimensional embedding vectors, $\mathbf{x}_p \in \mathbb{R}^K$ and $\mathbf{x}_q \in \mathbb{R}^K$, are highly correlated. Thus, spectrally-similar nodes p and q can be aggregated together for node reduction purpose.

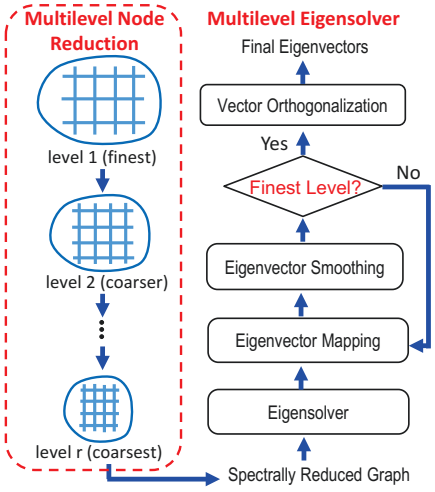


Fig. 2: The framework of multilevel node reduction and multilevel eigensolver.

The node affinity metric $a_{p,q}$ also reflects the distance or strength of the connection between nodes p and q. For example, the algebraic distance $d_{p,q}$ can be expressed by $d_{p,q} = 1 - a_{p,q}$, which can be used to represent the geometric distance in grid-structure graphs. Nodes with large affinity or small algebraic distance should be aggregated together to form the nodes in the reduced graph. Based on this node aggregation scheme, we can generate the next coarser-level graph by applying it to the original graph. To further reduce its size, we leverage a multilevel procedure by repeatedly applying the above node reduction procedure to the current-level graph until the desired size of the reduced graph at the coarsest level is reached, as shown in Figure 2. Once the node aggregation scheme for each level is determined, we can define the graph mapping operators $\mathbf{H}_i^{i+1}$ (fine-to-coarse) and $\mathbf{H}_{i+1}^i$ (coarse-to-fine), which can be further leveraged for constructing the spectrally-reduced graph. For example, given the graph Laplacian $\mathbf{L}_G$ and the defined mapping operators from the finest level 1 to the coarsest level r, we can always uniquely compute the final reduced Laplacian by $\mathbf{L}_R = \mathbf{H}_G^R \mathbf{L}_G \mathbf{H}_R^G$, where $\mathbf{H}_G^R = \mathbf{H}_1^2 \mathbf{H}_2^3 \cdots \mathbf{H}_{r-1}^r$ and $\mathbf{H}_R^G = \mathbf{H}_2^1 \mathbf{H}_3^2 \cdots \mathbf{H}_r^{r-1}$.

The computational cost of node reduction scheme based on the above spectral node affinities is linear. This allows us to preserve the spectral properties of the original graph in a highly efficient and effective manner: the node aggregation scheme will preserve the smooth components in the first few Laplacian eigenvectors well, which is key to preserving the first few eigenvalues and eigenvectors of the original graph Laplacian in the reduced graphs.

Since only the first few nontrivial eigenvectors of the original graph Laplacian are needed for graph visualization tasks, they can efficiently and effectively be calculated by leveraging a multilevel eigensolver procedure [31], as shown in Figure 2. Instead of directly solving the eigenvalue problems on the original graph G, we will first reduce G into a much smaller graph R such that the eigenvectors of the reduced graph can be easily calculated. Once we get the eigenvectors of graph R, we will map them back to next finer level using the mapping operators defined during the MNR process. To further improve the solution accuracy of the mapped eigenvectors, a weighted-Jacobi-iteration-based eigenvectors smoothing (refinement) scheme is applied. The eigenvector mapping and smoothing procedures are recursively applied until the finest level graph is reached. Finally, all the eigenvectors for the finest level will be orthonormalized using the Gram-Schmidt process.

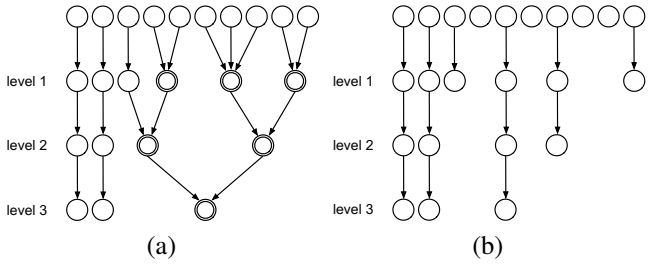


Fig. 3: Comparison of different node reduction processes. (a) shows the node reduction process taken by our SPS (or METIS) method where double-circled nodes are pseudo-nodes newly created. (b) shows the node reduction process taken by graph sampling methods (such as DSS and FF, refer to Section 5.1).

Note that the MNR process taken by SPS generates pseudo-nodes that are not in the node set of the original graph. As shown in Figure 3 (a), at each level of node reduction, our process essentially aggregates nodes into groups and creates a pseudo-node to represent each group. On the contrary, other node reduction methods do not create pseudo-nodes. As shown in Figure 3 (b), at each level of simplification, they simply sample the graph and output a subset of nodes from the node set of the original graph. In this process, no pseudo-nodes are created.

4.3. Spectral Graph Drawing

SPS is a practically efficient solution for spectrum-preserving graph sparsification and Laplacian eigenvalue computation. This enables us to tackle much bigger graphs previously impossible by creating spectrally-simplified graphs at various levels of detail for graph drawing and interaction. We present two different layouts to use in conjunction with SPS: the eigenvector-based (EIGEN) and t-SNE-based (t-SNE) layouts. The EIGEN layout lays out the graph vertices using certain eigenvectors of the related matrices (we use the two leading eigenvectors in this paper). The t-SNE layout employs t-SNE to create a 2D embedding based on the leading eigenvectors (we empirically use the first 50 dominant eigenvectors in this paper).

Layout generation. To generate a layout for visualizing a given graph, we propose the following steps as outlined below:

- **Step 1:** Apply SPS to simplify the graph G_0, yielding the sparsified graphs $G_1, G_2, \ldots, G_r$ and their associated sparse Laplacian matrix $\mathbf{L}_{G_i}, i \in \{0, 1, \ldots, r\}$ of size $|N_i|^2$. Note that G_0 is the original graph and G_r is its most simplified form.
- **Step 2:** Perform an eigenanalysis [56, 57] on $\mathbf{L}_{G_r}$ to obtain the first k' leading eigenvectors and their associated eigen-

Data Set	Nodes	Edges	SPS						FF_E					FF_N					DSS
			SES	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	
small data sets (DSS compatible)																			
FACEBOOK	4,039	88,234	0.30	1.19	0.11	0.16	0.05	0.15	0.27	0.20	0.13	0.11	0.10	0.30	0.31	0.11	0.16	0.10	802.8
AIRFOIL	4,253	12,289	0.07	2.94	0.24	0.19	0.11	0.27	0.15	0.11	0.05	0.03	0.02	0.11	0.13	0.05	0.05	0.07	947.4
ND3K	9,000	3,279,690	5.93	1.18	0.17	0.19	0.07	0.17	4.02	4.04	4.22	3.90	2.68	2.98	2.43	2.43	2.19	2.29	12,774
USPS10NN	9,298	136,762	0.92	1.33	0.15	0.17	0.07	0.15	0.89	1.07	0.65	0.35	0.35	0.36	0.23	0.20	0.17	0.17	11,448
MYCIELSKIAN14	12,287	3,695,512	6.42	1.23	0.11	0.17	0.07	0.19	3.36	3.19	3.11	2.90	2.90	3.35	3.15	3.33	2.81	2.69	27,181
APPU	14,000	1,839,104	11.60	1.33	0.14	0.21	0.06	0.14	2.86	2.30	2.79	1.96	2.69	5.00	6.32	7.55	5.66	4.78	34,236
big data sets (DSS incompatible)																			
VSP	21,996	1,221,028	5.36	1.30	0.17	0.20	0.08	0.15	8.85	5.57	5.40	3.90	2.72	2.38	2.14	2.09	2.09	1.94	
PROTEIN_DB	36,417	2,154,174	11.19	1.36	0.18	0.20	0.08	0.15	16.30	9.15	6.43	4.77	4.06	6.38	4.04	3.09	3.01	3.05	
MESH	40,000	79,600	0.64	1.31	0.15	0.21	0.08	0.16	6.43	3.10	1.86	1.09	0.49	4.47	2.18	0.90	0.62	0.26	
CFD	70,656	878,854	10.48	1.59	0.24	0.24	0.08	0.16	20.74	14.00	9.38	4.84	2.60	28.69	13.58	4.77	3.04	1.58	
DBLP	317,080	1,049,866	19.08	3.63	0.50	0.26	0.18	0.11	306.12	164.19	63.47	31.07	19.03	63.87	19.48	7.58	5.55	5.50	
ND	325,729	1,469,679	19.11	2.74	0.63	0.44	0.15	0.25	501.65	245.22	144.24	61.28	17.25	129.88	23.77	11.46	4.89	2.51	
IL2010	451,554	1,082,232	9.47	3.06	0.91	0.53	0.20	0.20	855.67	388.49	202.90	97.24	51.84	773.92	300.86	130.06	60.46	25.30	

Table 1: Timing results (in seconds) for the data sets experimented. The data sets are ordered according to the number of nodes in the original graphs, and split into two groups (small and big data sets). The five levels of simplification under SPS is for the MNR step.

values (we set $k' = 50$). Each of these eigenvectors is $|N_r|$-dimensional and every graph node has a k'-dimensional representation.

- **Step 3:** Identify the largest eigengap, i.e., the largest difference of two neighboring eigenvalues, among the first k' eigenvalues to determine the desired number of clusters k. Perform spectral clustering using k-means to obtain cluster labels for the k different clusters.
- **Step 4:** Either use the two leading eigenvectors as 2D positions of the nodes (for the EIGEN layout), or perform dimensionality reduction, which maps the graph's node positions from k'D to 2D using t-SNE (for the t-SNE layout).
- **Step 5:** Map the cluster labels, eigenvectors, and t-SNE results from G_r to G_{r-1}, and repeat this iteratively until the mapping from G_1 to G_0 is obtained.

After these steps, we hold all the data needed (2D coordinates, cluster labels, Laplacian matrix) to display the graph in 2D for the various levels of detail from G_0 to G_r. Nodes are colored to show their cluster memberships where neighboring clusters shown in the layout use different colors. To draw the graph at a given level of detail i, we position the nodes of G_i according to the selected layout and draw a straight line for each edge present in $\mathbf{L}_{G_i}$. Note that our SPS algorithm is independent of the choices of graph layout. Although our layout algorithm is not interactive, the timing results in Table 1 show that the SPS algorithm allows efficient layout generation for large graphs.

Graph interaction. For graph interaction, we allow users to change the graph layout, the level of detail, and turn on or off edge bundling. Edge bundling is computed in real time as we avoid its pre-computation for every graph level by implementing FFTEB, the state-of-the-art edge bundling technique using the fast Fourier transform (FFT) [58].

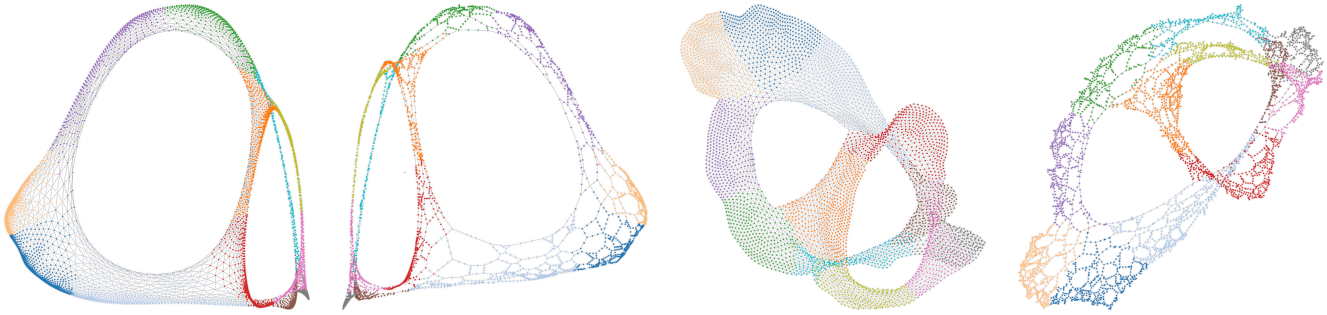
5. Results and Discussion

5.1. Data Sets and Methods

We experimented our approach with the graph data sets from different application domains as listed in Table 1. Among them, FACEBOOK and DBLP are from the social network domain, recording a friend network (FACEBOOK) and co-authorship relations (DBLP). AIRFOIL is a mesh graph from finite element analysis, ND3K is a graph generated from a 3D mesh problem, and MESH is a 200×200 mesh graph with uniform edge weights. USPS10NN is a k-NN network for handwritten digit recognition. MYCIELSKIAN14 represents a triangle-free graph with the chromatic number of 14. APPU and VSP are random graphs, representing the app benchmark from NASA Ames Research Center and a graph with a star-like structure. CFD is from computational fluid dynamics application representing a symmetric pressure matrix. ND is a web graph of the webpages of Notre Dame. IL2010 is a geographic network of the census blocks of Illinois. PROTEIN_DB is the protein databank of an enzyme found in HIV.

To compare different graph sparsification methods, we evaluated the results of four methods: SPS (ours), *deterministic spectral sparsification* (DSS) [12], and two variants of a traversal-based sampling method named *forest fire* (FF) [59]. DSS picks edges with the largest effective resistances. Note that Eades et al. [12] also introduced a second variant of spectral sparsification, *stochastic spectral sparsification* (SSS). However, DSS has been shown to perform better than SSS. Hence, we only use DSS in our comparison, where the pseudoinverse is computed using OpenIMAJ [60]. As a probabilistic version of *snow-ball sampling* (SBS) [61], FF randomly selects a seed node with incident edges and adjacent nodes getting "burned" away recursively with a probability. In this work, we continue FF sampling until a desired number of edges (FF_E) or nodes (FF_N) are reached.

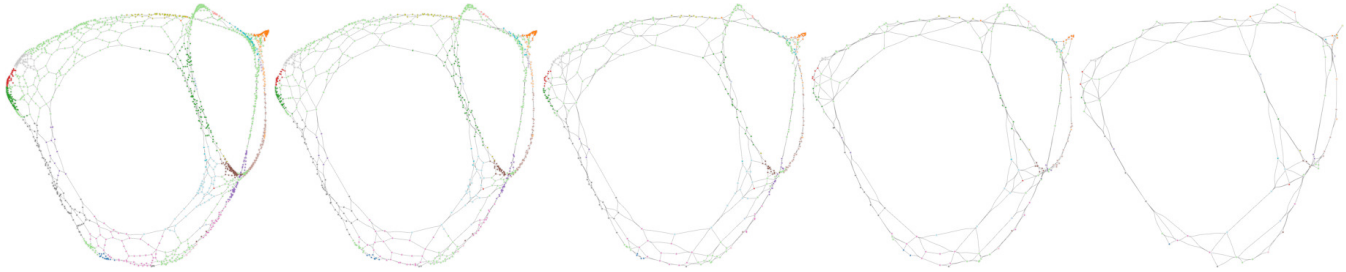
Besides DSS, the only other implementation publicly available is provided by Spielman, which is based on the effective-resistance sampling approach [14] and has been recently available for download [62]. However, such an implementation needs to set up input parameters carefully for each individual input graph and thus does not allow effective control of spectral approximation levels, such as the spectral similarity. In other words, it is impossible to control the approximation quality or sparsity of the sparsified graph using a common set of input parameters. In contrast, our SPS allows precise control of spectral similarity or graph sparsity, thereby enabling effective trade-offs between approximation quality and graph complexity. Our latest extensive experiments carried out on a series of public-domain graphs show that it is almost impossible to compare the sparsified graphs obtained by using our SPS method and Spielman's approach due to the above reasons.



(a) original (b) after SES (c) original (d) after SES

EIGEN FM^3

Fig. 4: Graph drawings of the AIRFOIL data set using the EIGEN ((a) and (b)) and FM^3 ((c) and (d)) layouts. The drawings show the original graph ((a) and (c)) and the reduced graph after SES ((b) and (b)).



(a) level 1 (b) level 2 (c) level 3 (d) level 4 (e) level 5

Fig. 5: Graph drawings of the AIRFOIL data set using the EIGEN layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification using the SPS algorithm.

For multilevel graph drawing, we compare MNR against METIS [63], a fast and high-quality multilevel scheme for graph partitioning. The version of METIS provided by Karypis and Kumar [63] is used, where we set the number of clusters METIS should produce to the number of nodes of the equivalent level of MNR. We merge a cluster i into a new node i' and add an edge between two new nodes i' and j' if there exists an edge from any node in cluster i to any node in cluster j. In order to make fair comparisons, we only use the graph after SES as input for METIS.

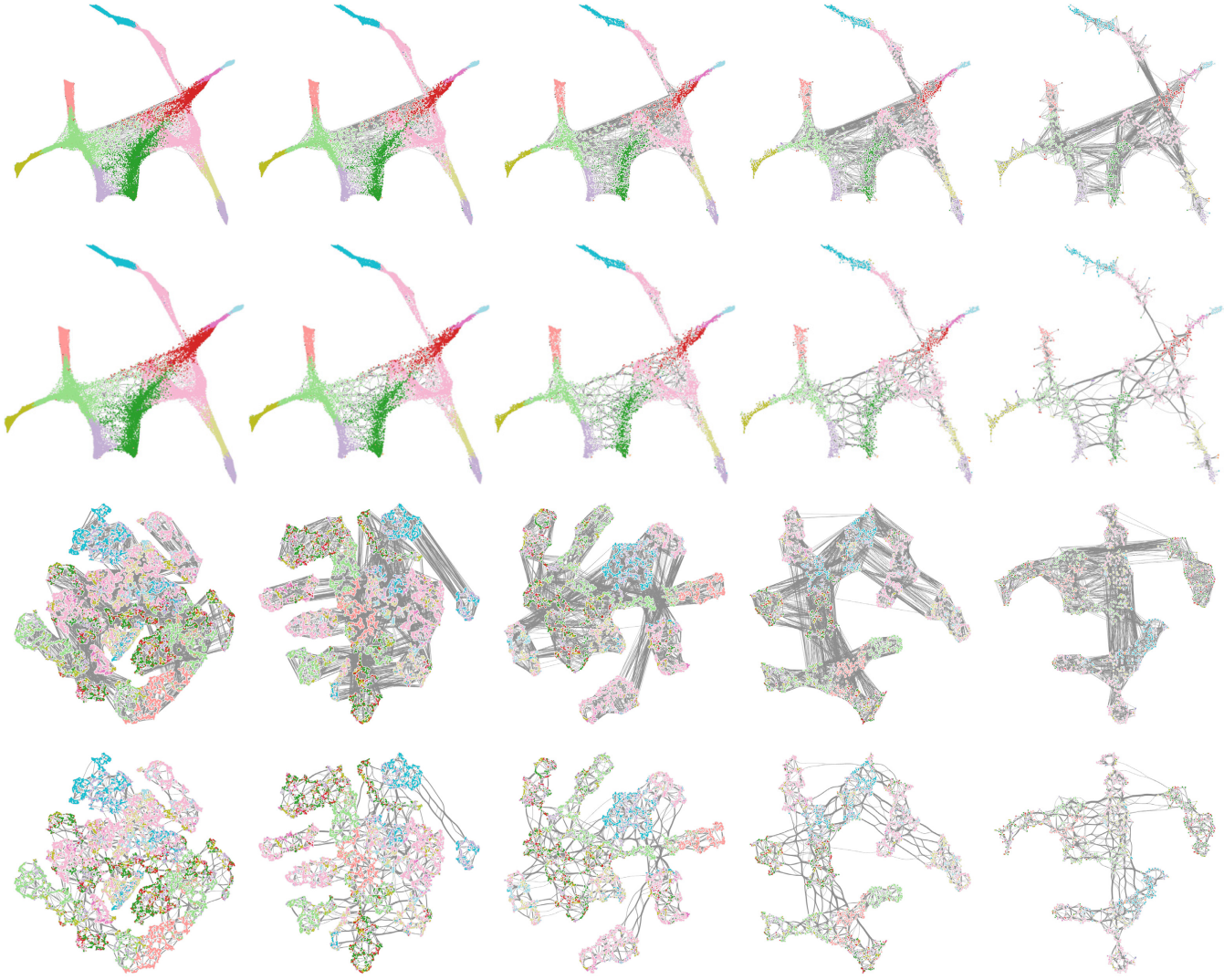
The graph data sets experimented are split into two groups: small data sets ($< 15,000$ nodes) and big data sets ($> 15,000$ nodes). This is due to the fact that DSS is not able to handle the big data sets on the machines we used. Given a data set, after edge sparsification, we produced five levels of node reduction for SPS and used the resulting numbers of edges and nodes as the targets to obtain the sparsification results for DSS (small data sets only) and the two variants of FF.

5.2. *Sparsification Timings*

Table 1 reports the timing results in seconds for graph sparsification. For SPS, FF_E, and FF_N, we show the computation time to achieve five different levels of sparsification. As the MNR step of SPS is an iterative algorithm, the results only show the time it takes from level i (finer) to level $i+1$ (coarser), while the entries for either FF_E or FF_N always show the total computation time starting from the original graph. For DSS, only a single computation time is reported for each data set, as the algorithm computes the effective resistance for every edge and then uses a desired number of edges with the highest resistance values as the result. All the reported timing results were collected from runs on Lenovo NeXtScale nx360 M5 Servers with dual 12 core Intel Xeon CPU E5-2680 v3 @ 2.50GHz Haswell processors and 256GB RAM.

Small data sets. The upper part of Table 1 shows that DSS cannot keep up with the speed of the other algorithms. Even for the smallest data sets (FACEBOOK and AIRFOIL), it already takes more than 10 minutes to compute the effective resistance value for the entire graph. In contrast, SPS and the two FF methods, always complete the computation under 20 seconds, with most of the cases below 10 seconds. When comparing SPS against FF_E and FF_N, we can see that either FF_E or FF_N outperforms SPS for all the data sets, due to the time spent by SPS on SES. However, the performance gap decreases with increasing graph size.

Big data sets. The lower part of Table 1 shows the timing results for the bigger data sets. Besides the spectral sparsification, SPS stays consistent with its low computation time. On the contrary, the computation time for FF_E and FF_N drastically increases along with the input graph's size. The first three data sets (VSP, PROTEIN_DB, and MESH), still show a similar timing performance for all three methods, due to the time spent by SPS on the SES step. After that, starting with CFD, the difference in computation time between SPS and the FF methods increases drastically to more than 10 folds (DBLP and ND), and about 70 folds (IL2010) at the finest level. At the coarsest level, however, the difference between SPS and FF_E vanishes



(a) level 1 (b) level 2 (c) level 3 (d) level 4 (e) level 5

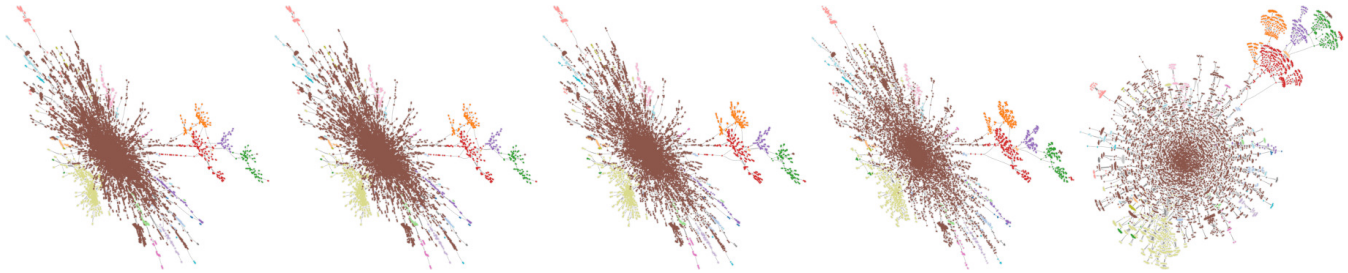
Fig. 6: Graph drawings of the CFD data set using the t-SNE layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification. Top two rows: MNR. Bottom two rows: METIS. For either method, the upper or lower row shows the drawing without or with edge bundling.

for DBLP and ND, and decreases to about five folds for IL2010. At this sparse level, FF_N outperforms any method except for the IL2010 data set. This demonstrates the competitive advantage of our SPS method in terms of computational scalability.

5.3. Graph Visualization

For graph drawing, we used the following methods: (1) EIGEN (refer to Section 4.3), (2) t-SNE (refer to Section 4.3), and (3) fast multipole multilevel method (FM^3) [64]. Note that we leveraged MATLAB for computing EIGEN and t-SNE, and OGDF [65] for computing FM^3. We chose FM^3, a force-directed layout for large graphs, because it has an efficient time complexity of $O(|N|\log|N|+|E|)$ and was recently applied to graph drawing with spectral sparification [12]. We did not draw the original graphs, but only their sparsified or sampled versions, as it is often not possible to draw the full-size graph due to the computational costs of EIGEN and t-SNE for large graphs. To circumvent the problem of not drawing the original big graph, we used the proxy quality metric [46] to evaluate the quality of the graph's proxy drawing. Our work demonstrates the capability of drawing graphs with spectral sparsification on data sets much larger than recently attempted by Eades et al. [12]. We implemented FFTEB to reduce visual clutter. Based on the spectral clustering result, we colored the nodes in different clusters with different colors. To allow easier visual comparison, for the FF and DSS sampling results, we kept the coloring based on the SPS clusters and used black for all the nodes that do not exist in the SPS results at the same sparsification level.

Edge sparsification. Figure 4 shows the AIRFOIL data set before and after SES. Ignoring the flip that occurred, we can see that in (a) and (b), the graph structure remains the same using the EIGEN layout, with (b) showing fewer edges. The drawings in (c) and (d) reveal the same using the FM^3 layout. This indicates that SES can successfully keep edges relevant for the graph structure while removing non-essential edges. Additionally, the spectral clusters are also well preserved in the drawing.



(a) level 1 (b) level 2 (c) level 3 (d) level 4 (e) level 5

Fig. 7: Graph drawings of the ND data set using the FM^3 layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification using the SPS algorithm.

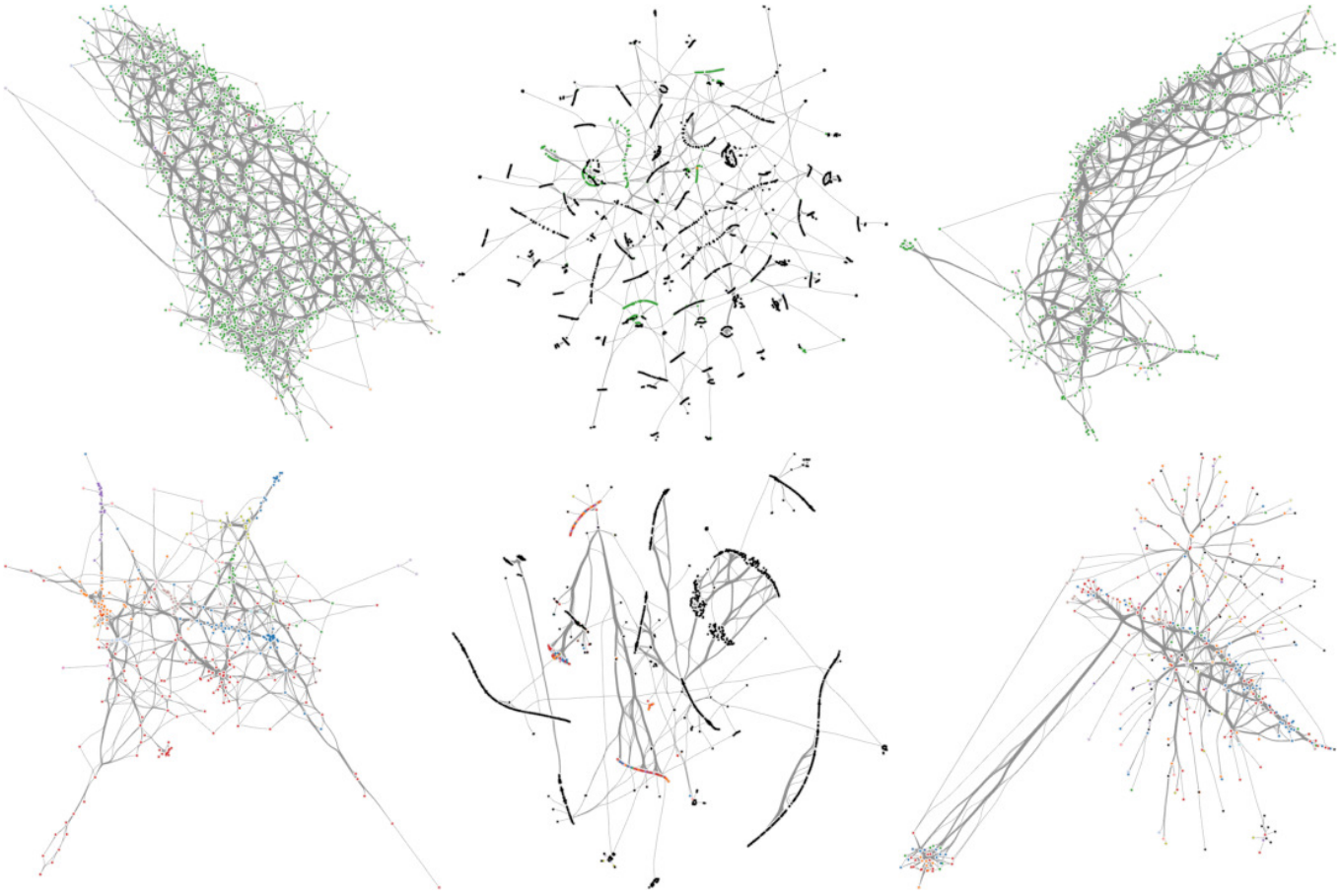
Comparison across simplification levels. In Figure 5, we compare the five levels of sparsification using the AIRFOIL data set using the EIGEN layout. We can see that although the number of nodes halves at each level of simplification, the overall graph structure remains the same. Even the coarsest level (Figure 5 (e)) shows the two big circle-like structures as the most distinguishable features of this data set. Similarly, the first two rows of Figure 6 show the graph drawings for the five sparsification levels of the CFD data set using the t-SNE layout. Again we can see that the structure from the layout at the fifth level is preserved through the multilevel eigensolver. For this much bigger and denser data set, we do not observe an almost bone-like structure like for the AIRFOIL one at the coarsest level. However, we can still witness how the number of nodes in each cluster reduces successively between the neighboring levels without losing the inter-cluster connectivity. Figure 7 shows the drawing of the ND data set at its five sparsification levels using the FM^3 layout. This even bigger graph does not show much difference at the first four levels as the numbers of nodes at these four levels remain pretty high. At the last level (Figure 7 (e)), however, we can see a drastic skew in the layout. Although this represents a strong change in the layout, the graph features, especially the clusters, still remain easily distinguishable. These three examples show how well the multilevel eigensolver allows using the layout from a coarser level and map it back to the original one without changing the overall graph structure.

Comparison across sparsification methods. Figure 8 shows a comparison of the three sparsification methods for the ND3K (top row) and FACEBOOK (bottom row) data sets. We use the t-SNE layout and the third level of sparsification. In Figure 8 (a) and (c), we can see that the two spectrum-based methods do a better job at preserving the underlying graph structure compared to the FF$_E$ result shown in Figure 8 (b). The drawing of the FF$_E$ method seems rather random and contains a large number of small node clusters (shown in black) that do not exist in the SPS result. It is worth noting that the two spectrum-based methods mostly agree on the chosen nodes, while the FF$_E$ method contains many nodes that do not exist in the SPS variant. In the second row of Figure 8, we can see that the FF$_E$ method needs more nodes than the other two methods for the FACEBOOK data set to achieve the desired number of edges. This shows that spectrum-based methods are better suited to give an overview of the most important nodes of the graph than the FF$_E$ sampling.

In Figure 9, we show similar comparisons for the PROTEIN_DB and IL2010 data sets using the FM^3 layout. PROTEIN_DB shows the finest level of sparsification while IL2010 shows the coarsest level. For the PROTEIN_DB data set, we can see that the layout produced by FF$_E$ mixes the clusters together, resulting in a confusing structure. The layout produced by SPS shows a much smoother and nicer cluster separation and a more revealing overall structure. When it comes to the IL2010 data set, FF$_E$ results in a tree-like graph, while SPS shows a more dispersed structure that looks similar to a flipped version of the underlying geographical map of the state of Illinois. Capturing and representing features like geographical and geometric structures underlines the advantages of SPS over random sampling methods.

Figure 10 shows the USPS10NN and MESH data sets using the EIGEN (top row) and t-SNE (bottom row) layouts. USPS10NN uses the finest level of sparsification while MESH uses the coarsest level. For the USPS10NN data set, FF$_E$ finds one cluster instead of multiple ones like the SPS method. Thus the resulting drawing for the FF$_E$ sample is very dense and cluttered into one corner (EIGEN) or a hairball (t-SNE) instead of more evenly distributed like the drawing of SPS. For the MESH data set, the drawing of the FF$_E$ sample again shows tree-like and hairball-like structures for the EIGEN and t-SNE layouts, respectively. The drawing of the SPS sample, on the other hand, highlights the grid-like structure of the underlying mesh in either layout. This shows that based on spectral analysis, SPS can reveal the underlying structures well at both the finest and coarsest levels.

Comparison of MNR and METIS. In Figure 11, we show a comparison of the MNR and METIS methods. For both drawings, we use the FM^3 layout and keep the cluster labels from SPS for easier comparison. Besides the different cluster ordering, there is no significant visual difference. Nevertheless, we point out that unlike METIS, MNR preserves the spectrum of the graph and does not require layout recomputation as we move from the coarsest level to the finest level. This can be seen in Figure 6, where we show the t-SNE layout for the five sparsification levels for MNR and METIS along with edge bundling disabled and enabled. We can see that the graph structure in the drawing is fairly consistent across the five levels with MNR, which is certainly not the case with METIS. Since the t-SNE layout is based on the leading eigenvectors resulting from SPS,



(a) SPS (b) FF_E (c) DSS

Fig. 8: Graph drawings of the different sampling methods for the ND3K (top row) and FACEBOOK (bottom row) data set using the t-SNE layout. All drawings use the third level of sparsification.

we can claim that MNR better preserves the spectrum of the underlying graph. Furthermore, the MNR results show many shorter edges, indicating a well-translated structure from the reduced graph into the drawing.

For edge bundling, clearly, it helps to reduce visual clutter, especially for the five levels with METIS where edges are longer. However, edge bundling introduces ambiguities at the endpoints of thicker bundles.

Visual Quality of Spectral Graph. We point out that spectral drawing of a graph may not necessarily lead to good visual quality. The general idea behind spectral graph drawing is to translate the spectral properties of the graph to the visualization. Prior works on graph drawing using spectral information [34, 35, 36, 38, 37] do not necessarily generate visually pleasing or aesthetic layouts either. Our observations are that spectral methods are good for drawing grid- or mesh-like graphs, but could be bad for other graphs. In those cases, the nodes in the spectral layout could overlap with each other (due to the great similarity of their spectral properties) or form a linearization pattern.

6. Quantitative Comparison

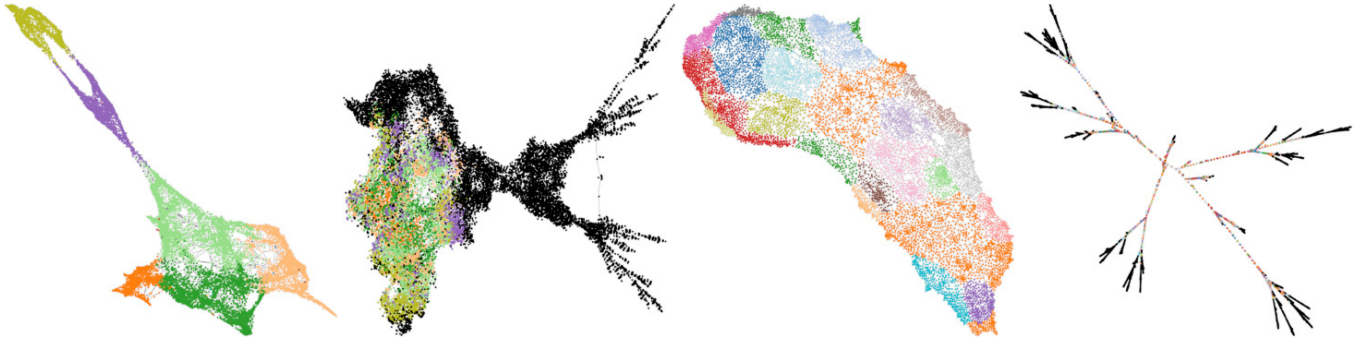
6.1. Quality Metrics

To evaluate the visual quality of graph samples, Nguyen et al. [46] introduced the proxy quality metric, which compares the drawing of a graph sample to the underlying graph in order to express the faithfulness of the drawing. This metric compares the similarity of each node in the drawing to the node in the underlying graph using one-to-one correspondence. The SPS algorithm, however, does not preserve such a correspondence due to the introduction of pseudo-nodes in MNR (refer to Section 4.2). Therefore, we use the proxy quality metric to compare the samples after SES but before MNR. We employ four other statistical metrics to quantify the sampling quality of MNR.

The proxy quality metric obtains a shape graph from the sampled graph drawing and then compares it to the original graph. Formally

$$Q_{\mu,\phi}(G,S(G)) = \mu(G,\phi(S(G))), \tag{14}$$

where μ is a comparison function that compares the two graphs and returns a real number, ϕ is a shape graph function, and $S(G)$ is a sample of the original graph G. Examples of shape graphs include the α-*shape* [66], *k-nearest neighbor graph* (k-NN graph), *Gabriel graph*, *relative neighborhood graph* (RNG),



(a) SPS (b) FF_E (c) SPS (d) FF_E

Fig. 9: Graph drawings of the different sampling methods using the FM^3 layout. (a) and (b) are for the PROTEIN_DB data set at the finest level of sparsification. (c) and (d) are for the IL2010 data set at the coarsest level of sparsification.

and *Euclidean minimum spanning tree* (EMST). The similarity between two graphs of the same vertex set can be measured efficiently using the *mean Jaccard similarity* (MJS). In this paper, we used the Gabriel graph as the shape graph function ϕ and the MJS as the comparison function μ. The MJS between $S(G)$ and G is defined as

$$\mathrm{MJS}(S(G),G) = \frac{1}{|N|}\sum_{v\in N}\frac{|N_{S(G)}(v)\cap N_G(v)|}{|N_{S(G)}(v)\cup N_G(v)|}, \qquad (15)$$

where $N_{S(G)}(v)$ and $N_G(v)$ are the neighborhoods of node v in $S(G)$ and G, respectively. For simplicity, we define Q(SPS), Q(DSS), and $Q(FF_E)$ for the MJS between the original graph and its sample with SPS, DSS, and FF_E, respectively.

To evaluate the quality of the MNR samples, we use four of the five metrics used by Hong et al. [43]:

- *degree correlation assortativity* (DCA) which describes how well similar nodes are connected to each other [67];
- *closeness centrality* (CCe) which sums the lengths of the shortest paths from each node to all other nodes [68];
- *clustering coefficient* (CCo) which measures how well nodes cluster together within the graph [69];
- *average neighborhood degree* (AND) which averages the degrees of neighboring nodes for each node [70].

We do not use the fifth metric, largest connected component (LCC), as SPS and FF always yield a graph with a single connected component. We compare the metric on a given sample and the original graph using the Kolmogorov-Smirnov (KS) test. The KS-test computes the difference between two probability distributions and describes it as a result between 0 (same) and 1 (completely dissimilar).

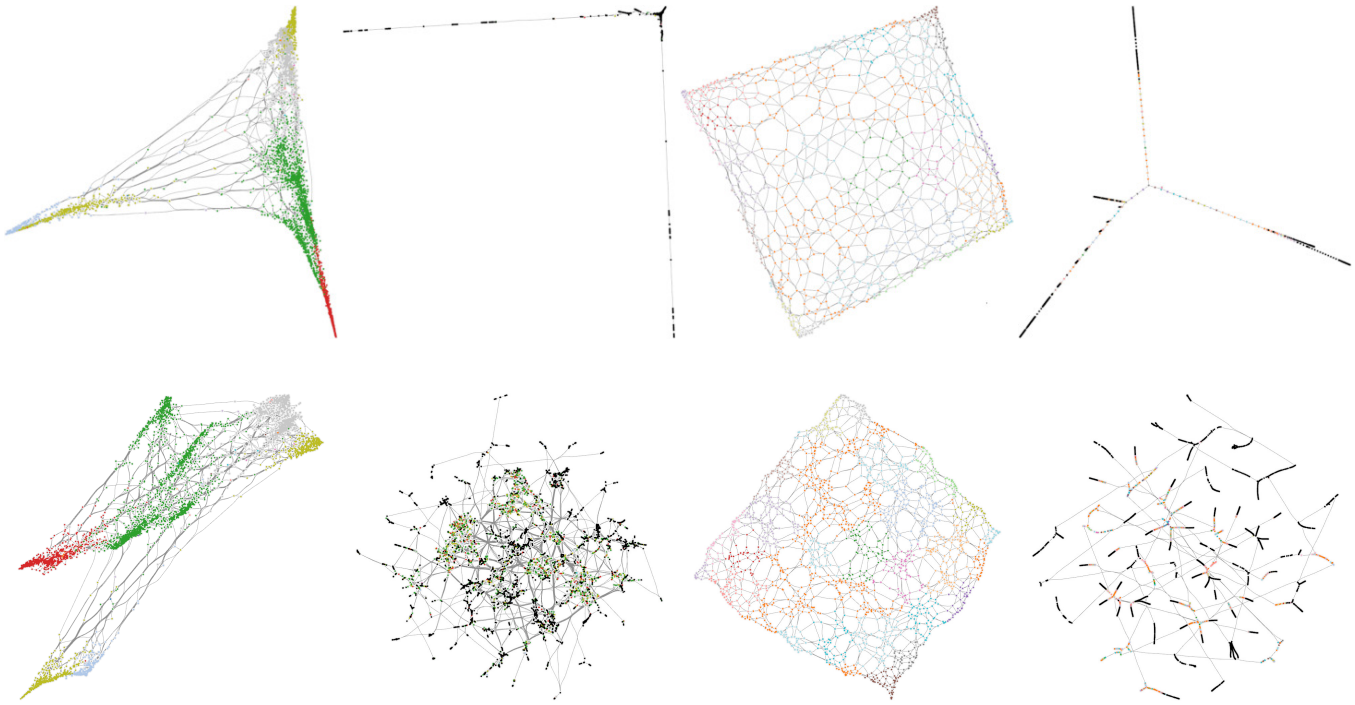
6.2. Comparison Results

Proxy quality metric. In Table 2, we report the averaged MJS ratios Q(SPS)/Q(DSS) and Q(SPS)/$Q(FF_E)$ for the comparison between SPS and DSS, as well as SPS and FF_E respectively. Note that we only use FF_E here, as we only compare the results of SES, an edge-based sparsification technique. We use the t-SNE, EIGEN, and FM^3 layouts for this comparison. The ratio values above 1.0 favor SPS over the comparing method. We can see that SPS generally achieves a better quality than DSS and FF_E, with the exception of the AIRFOIL and USPS10NN data sets when compared to FF_E. This is mainly because for these two data sets, FF_E sampling vastly outperforms SPS sampling with the t-SNE and FM^3 layouts. Further worth mentioning are the high values of Q(SPS)/Q(DSS) for ND3K and Q(SPS)/$Q(FF_E)$ for MESH. These are due to the fact that SPS sampling vastly outperforms the sampling being compared across all three layouts. With these results, we conclude that the SES step of SPS preserves the structure of the original graph better than DSS and FF_E.

Sampling quality metrics. Figure 12 shows the KS-test results between the original graph and a given sample for the four different metrics (lower KS-test values are better). In the charts, we can see that SPS (either SPS_{SES} or SPS_{ORI}) generally outperforms the FF_E sampling methods, but there is no clear winner between SPS and DSS. METIS behaves very similar to SPS_{SES} in terms of DCA and CCe, while it performs better in terms of CCo and worse in terms of AND. While DCA remains mostly stable among all methods and sample sizes, the other metrics show interesting trends. CCe yields worse results for SPS_{SES} than SPS_{ORI}. This means that the shortest path lengths after MNR are closer to the ones of the original graph than to those after SES.

For the sake of argument, consider the average of the shortest path lengths for each node to all other nodes. If we compare the distribution of those average shortest path lengths (1) between the sampled graph and the original graph and (2) between the sampled graph and the graph after SES, then the difference between the sampled graph and the graph after SES will be smaller. This is because SES takes a graph as input and produces another graph that is similar to a spanning tree of the original graph. Evaluating the average of the shortest paths (for each node) in a spanning tree will be quite different from using the original graph.

Now if we consider the distribution of the average of shortest paths in a graph after applying the MNR procedure. As shown in Figure 3 (a), MNR reduces the graph through node aggregation: a pseudo-node at level $i+1$ represents multiple nodes at level i. Any edge between two nodes which are both represented by the same pseudo-node is removed. The pseudo-node



(a) SPS (b) FF_E (c) SPS (d) FF_E

Fig. 10: Graph drawings of the different sampling methods using the EIGEN (top row) and t-SNE (bottom row) layouts. (a) and (b) are for the USPS10NN data set at the finest level of sparsification. (c) and (d) are for the MESH data set at the coarsest level of sparsification.

Data Set	Edges after SES	Q(SPS)/Q(FF_E)	Q(SPS)/Q(DSS)
FACEBOOK	7,872	1.03	2.84
AIRFOIL	4,934	0.72	3.19
ND3K	16,745	1.73	50.33
USPS10NN	12,900	0.89	2.96
MYCIELSKIAN14	14,060	1.34	*
APPU	16,366	2.34	3.65

Data Set	Edges after SES	Q(SPS)/Q(FF_E)
VSP	42,752	2.30
PROTEIN_DB	70,491	2.49
MESH	45,261	66.71
CFD	106,879	1.64
DBLP	358,226	6.25
ND	388,436	2.85
IL2010	504,465	7.60

Table 2: Averaged results over three layout algorithms of the quantitative comparison using MJS. Left table: small graphs. Right table: big graphs. The columns show the number of edges after SES and the quality ratios (higher is better). The * denotes that DSS does not achieve a MJS within the machine precision, i.e., it is very close to zero.

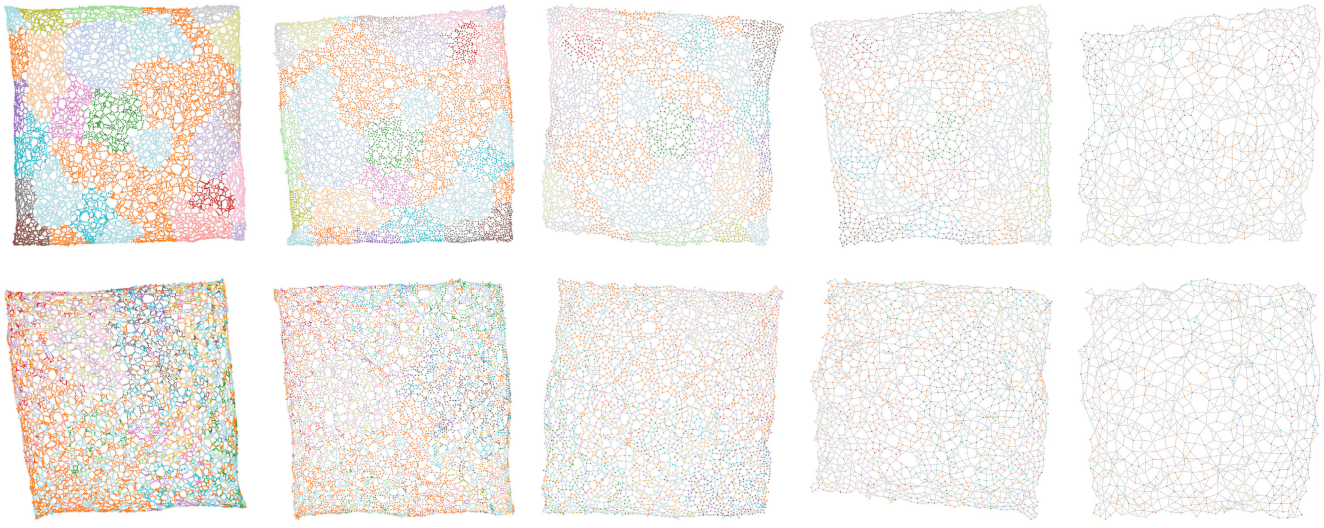
becomes incident to any edge that connects two nodes of which only one of them is represented by the pseudo-node. As we aggregate nodes together and take the edges from all their original nodes, the graph at a coarser level is less similar to a spanning tree of the original graph. The impact of this is the opposite of what SES has on a graph. Therefore, comparing closeness centrality after MNR with respect to the original graph shows more similarity than that after SES.

For CCo, we see that typically after the third level of MNR, the SPS_{SES} and SPS_{ORI} lines cross. The reason for this is analog to what is discussed previously. The difference is that we consider between-cluster and within-cluster edges in the graph. Since MNR is applied after SES, it uses a spanning tree as input. Therefore at the finer levels, it is more like a spanning tree and less like the original graph, while at the coarser level, MNR produces a graph that is less like a spanning tree. The worse score for METIS in terms of CCo is due to the number of edges. Over all data sets and all simplification levels, METIS produces an average of 13% (20-70% for denser graphs, e.g., ND and VSP, and less than 10% for sparser graphs, e.g., AIRFOIL, IL2010) more edges compared to MNR. As the input graph for this comparison is the graph after SES, i.e., a very sparse graph, the denser output can translate into a different CCo.

For AND, we can see that SPS_{SES} and SPS_{ORI} trend toward similar values the more iterations of SPS we run. This is because, with a more reduced graph, there are only the important nodes and their neighborhood relationships left to represent the original (sparsified) graph. Interestingly, METIS has an AND value more similar to the graph after SES than MNR. This shows that our MNR removes edges more aggressively to preserve spectral properties.

7. Conclusions and Future Work

We have presented SPS, an effective solution for spectrum-preserving sparsification of big graphs. The innovation of SPS is that for the first time, it combines spectral graph sparsification to achieve scalable visualization of large graphs while allowing for spectral clustering analysis at the same time. Our SPS algorithm includes two steps: spectral edge sparsification (SES) followed by multilevel node reduction (MNR). The SES algorithm is three to four orders of magnitude faster than the state-



(a) level 1 (b) level 2 (c) level 3 (d) level 4 (e) level 5

Fig. 11: Graph drawings of the MESH data set using the FM^3 layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification. Top row: MNR. Bottom row: METIS.

of-the-art DSS algorithm. The dramatic gain in speed performance enables us to handle edge sparsification and subsequent node reduction on big graphs with hundreds of thousands of nodes and millions of edges, which was previously impossible. Furthermore, using different graph drawing layouts (EIGEN/t-SNE, FM^3), we find that in general, SPS outperforms DSS and FF under a proxy quality metric (for the SES step) and other statistical properties of the graphs (for the MNR step). We demonstrate the effectiveness of our approach using results gathered from a number of graph data sets of varying sizes and characteristics. In the future, we will integrate advanced user interactions (such as focus+context visualization) and evaluate this graph visualization and exploration framework through a formal user study.

Acknowledgements

This research was supported in part by the U.S. National Science Foundation through grants IIS-1455886, CNS-1629914, DUE-1833129, CCF-1350206, CCF-1618364, and CCF-1909105, and the NVIDIA GPU Grant Program. The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] Teng, SH. Scalable algorithms for data and network analysis. Foundations and Trends in Theoretical Computer Science 2016;12(1-2):1–274.

[2] Spielman, DA, Teng, SH. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. SIAM Journal on Matrix Analysis and Application 2014;35(3):835–885.

[3] Christiano, P, Kelner, J, Madry, A, Spielman, DA, Teng, SH. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In: Proceedings of ACM Symposium on Theory of Computing. 2011, p. 273–282.

[4] Shuman, DI, Narang, SK, Frossard, P, Ortega, A, Vandergheynst, P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing 2013;30(3):83–98.

[5] Rohe, K, Chatterjee, S, Yu, B. Spectral clustering and the high-dimensional stochastic blockmodel. The Annals of Statistics 2011;39(4):1878–1915.

[6] Defferrard, M, Bresson, X, Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of Advances in Neural Information Processing Systems. 2016, p. 3844–3852.

[7] Hu, X, Lu, A, Wu, X. Spectrum-based network visualization for topology analysis. IEEE Computer Graphics and Applications 2013;33(1):58–68.

[8] Lee, JR, Gharan, SO, Trevisan, L. Multiway spectral partitioning and higher-order Cheeger inequalities. Journal of the ACM 2014;61(6):37:1–37:30.

[9] Kolev, P, Mehlhorn, K. A note on spectral clustering. arXiv:1509.09188; 2015.

[10] Peng, R, Sun, H, Zanetti, L. Partitioning well-clustered graphs: Spectral clustering works! Proceedings of Machine Learning Research 2015;40:1423–1455.

[11] Chen, WY, Song, Y, Bai, H, Lin, CJ, Chang, EY. Parallel spectral clustering in distributed systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 2011;33(3):568–586.

[12] Eades, P, Nguyen, QH, Hong, SH. Drawing big graphs using spectral sparsification. In: Proceedings of International Symposium on Graph Drawing. 2017, p. 272–286.

[13] Spielman, DA, Teng, SH. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of ACM Symposium on Theory of Computing. 2004, p. 81–90.

[14] Spielman, DA, Srivastava, N. Graph sparsification by effective resistances. SIAM Journal on Computing 2011;40(6):1913–1926.

[15] Spielman, DA. Algorithms, graph theory, and linear qquations in Laplacian matrices. In: Proceedings of International Congress of Mathematicians; vol. 4. 2010, p. 2698–2722.

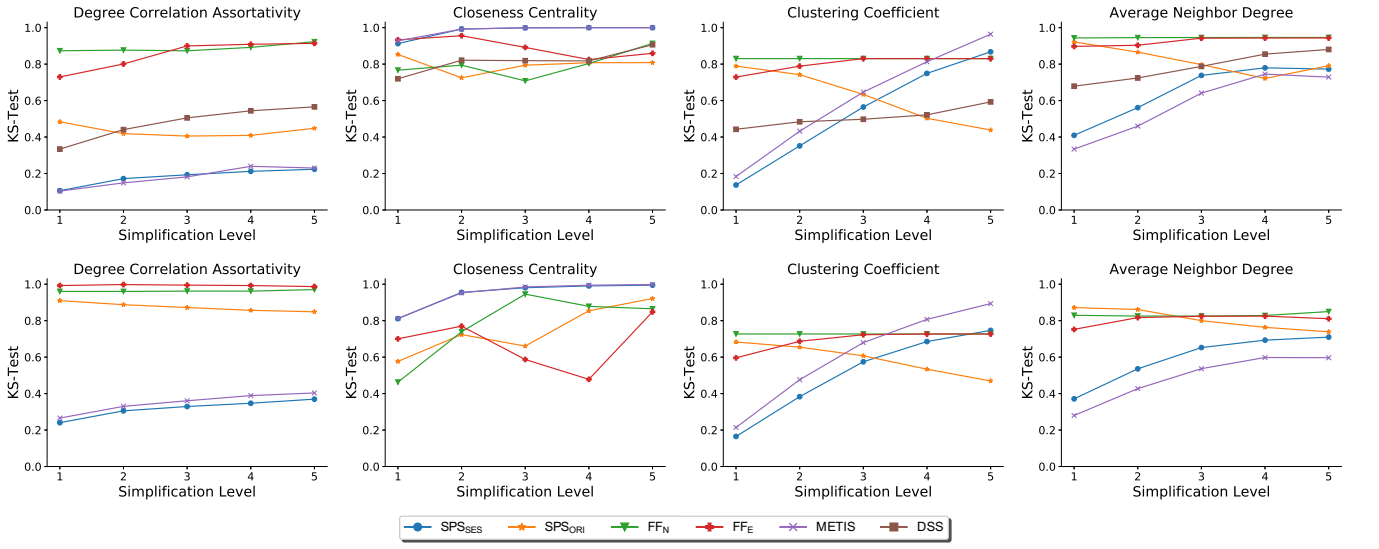
[16] Kolla, A, Makarychev, Y, Saberi, A, Teng, SH. Subgraph sparsification and nearly optimal ultrasparsifiers. In: Proceedings of ACM Symposium on Theory of Computing. 2010, p. 57–66.

[17] Koutis, I, Miller, G, Peng, R. Approaching optimality for solving SDD linear systems. In: Proceedings of IEEE Symposium on Foundations of Computer Science. 2010, p. 235–244.

[18] Fung, W, Hariharan, R, Harvey, N, Panigrahi, D. A general framework for graph sparsification. In: Proceedings of ACM Symposium on Theory of Computing. 2011, p. 71–80.

[19] Spielman, DA, Teng, SH. Spectral sparsification of graphs. SIAM Journal on Computing 2011;40(4):981–1025.

[20] Leskovec, J, Krevl, A. SNAP datasets: Stanford large network dataset collection. http://snap.stanford.edu/data; 2014.



(a) DCA (b) CCe (c) CCo (d) AND

Fig. 12: Averaged KS-test values (lower is better) between the original graph and the sample. Top row: small graphs. Bottom row: big graphs. SPS_{SES} denotes the comparison between MNR and the graph after SES. SPS_{ORI} denotes the comparison between MNR and the original graph.

[21] Davis, TA, Hu, Y. The University of Florida sparse matrix collection. ACM Transaction on Mathematical Software 2011;38(1):1:1–1:25.

[22] Muja, M, Lowe, DG. Scalable nearest neighbor algorithms for high dimensional data. IEEE Transactions on Pattern Analysis and Machine Intelligence 2014;36(11):2227–2240.

[23] Williams, C, Seeger, M. Using the Nyström method to speed up kernel machines. In: Proceedings of Advances in Neural Information Processing Systems. 2001, p. 682–688.

[24] Zhang, K, Tsang, IW, Kwok, JT. Improved Nyström low-rank approximation and error analysis. In: Proceedings of International Conference on Machine Learning. 2008, p. 1232–1239.

[25] Chen, X, Cai, D. Large scale spectral clustering with landmark-based representation. In: Proceedings of AAAI Conference on Artificial Intelligence. 2011, p. 313–318.

[26] Yan, D, Huang, L, Jordan, MI. Fast approximate spectral clustering. In: Proceedings of ACM SIGKDD Conference. 2009, p. 907–916.

[27] Liu, J, Wang, C, Danilevsky, M, Han, J. Large-scale spectral clustering on graphs. In: Proceedings of International Joint Conference on Artificial Intelligence. 2013, p. 1486–1492.

[28] Satuluri, V, Parthasarathy, S, Ruan, Y. Local graph sparsification for scalable clustering. In: Proceedings of ACM SIGMOD Conference. 2011, p. 721–732.

[29] Feng, Z. Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis. In: Proceedings of ACM/EDAC/IEEE Conference on Design Automation. 2016, p. 1–6.

[30] Feng, Z. Similarity-aware spectral sparsification by edge filtering. In: Proceedings of ACM/EDAC/IEEE Conference on Design Automation. 2018, p. 152:1–152:6.

[31] Zhao, Z, Wang, Y, Feng, Z. Nearly-linear time spectral graph reduction for scalable graph partitioning and data visualization. arXiv preprint arXiv:181208942 2018;.

[32] Zhao, Z, Wang, Y, Feng, Z. SAMG: Sparsified graph theoretic algebraic multigrid for solving large symmetric diagonally dominant (SDD) matrices. In: Proceedings of ACM/IEEE International Conference on Computer-Aided Design. 2017, p. 601–606.

[33] Hall, KM. An *r*-dimensional quadratic placement algorithm. Management Science 1970;17(3):219–229.

[34] Pisanski, T, Shawe-Taylor, J. Characterizing graph drawing with eigenvectors. Journal of Chemical Information and Computer Sciences 2000;40(3):567–571.

[35] Brandes, U, Willhalm, T. Visualizing bibliographic networks with a reshaped landscape metaphor. In: Proceedings of Eurographics - IEEE TCVG Symposium on Visualization. 2002, p. 159–164.

[36] Koren, Y, Carmel, L, Harel, D. ACE: A fast multiscale eigenvectors computation for drawing huge graphs. In: Proceedings of IEEE Symposium on Information Visualization. 2002, p. 137–144.

[37] Koren, Y. On spectral graph drawing. In: Proceedings of International Conference on Computing and Combinatorics. 2003, p. 496–508.

[38] Harel, D, Koren, Y. Graph drawing by high-dimensional embedding. In: Proceedings of International Symposium on Graph Drawing. 2002, p. 207–219.

[39] Harel, D, Koren, Y. Graph drawing by high-dimensional embedding. Journal of Graph Algorithms and Applications 2004;8(2):195–214.

[40] Koren, Y. Drawing graphs by eigenvectors: Theory and practice. Computers & Mathematics with Applications 2005;49(11-12):1867–1888.

[41] Hu, P, Lau, WC. A survey and taxonomy of graph sampling. arXiv:1308.5865; 2013.

[42] Zhang, F, Zhang, S, Wong, PC, Swan, JE, Jankun-Kelly, T. A visual and statistical benchmark for graph sampling methods. In: Proceedings of IEEE VIS Workshop on Exploring Graphs at Scale. 2015,.

[43] Hong, SH, Nguyen, Q, Meidiana, A, Li, J, Eades, P. BC tree-based proxy graphs for visualization of big graphs. In: Proceedings of IEEE Pacific Visualization Symposium. 2018, p. 11–20.

[44] Wu, Y, Cao, N, Archambault, D, Shen, Q, Qu, H, Cui, W. Evaluation of graph sampling: A visualization perspective. IEEE Transactions on Visualization and Computer Graphics 2017;23(1):401–410.

[45] Eades, P, Hong, SH, Nguyen, A, Klein, K. Shape-based quality metrics for large graph visualization. Journal of Graph Algorithms and Applications 2017;21(1):29–53.

[46] Nguyen, QH, Hong, SH, Eades, P, Meidiana, A. Proxy graph: Visual quality metrics of big graph sampling. IEEE Transactions on Visualization and Computer Graphics 2017;23(6):1600–1611.

[47] van der Maaten, LJP, Hinton, GE. Visualizing high-dimensional data using t-SNE. Journal of Machine Learning Research 2008;9:2579–2605.

[48] van der Maaten, LJP. Accelerating t-SNE using tree-based algorithms. Journal of Machine Learning Research 2014;15:1–21.

[49] Elkin, M, Emek, Y, Spielman, DA, Srivastava, N. Lower-stretch spanning trees. SIAM Journal on Computing 2008;38(2):608–628.

[50] Abraham, I, Neiman, O. Using petal-decompositions to build a low stretch spanning tree. In: Proceedings of ACM Symposium on Theory of Computing. 2012, p. 395–406.

[51] Spielman, DA, Woo, J. A note on preconditioning by low-stretch spanning trees. arXiv:0903.2816; 2009.

[52] Koutis, I, Miller, G, Tolliver, D. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. Computer Vision and Image Understanding 2011;115(12):1638–1646.

[53] Livne, O, Brandt, A. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. SIAM Journal on Scientific Computing

2012;34(4):B499–B522.
[54] Chen, J, Safro, I. Algebraic distance on graphs. SIAM Journal on Scientific Computing 2011;33(6):3468–3490.
[55] Briggs, WL, Van Emden, H, McCormick, SF. A Multigrid Tutorial. Second ed.; SIAM; 2000.
[56] Lehoucq, RB, Sorensen, DC, Yang, C. ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. Society for Industrial and Applied Mathematics; 1998.
[57] Stewart, GW. A Krylov-Schur algorithm for large eigenproblems. SIAM Journal on Matrix Analysis and Applications 2002;23(3):601–614.
[58] Lhuillier, A, Hurter, C, Telea, A. FFTEB: Edge bundling of huge graphs by the fast Fourier transform. In: Proceedings of IEEE Pacific Visualization Symposium. 2017, p. 190–199.
[59] Leskovec, J, Kleinberg, J, Faloutsos, C. Graphs over time: Densification laws, shrinking diamaters and possible explanations. In: Proceedings of ACM SIGKDD Conference. 2005, p. 177–187.
[60] Hare, J, Samangooei, S, Dupplaw, D. Open Intelligent Multimedia Analysis for Java (OpenIMAJ). `http://openimaj.org/`; 2019.
[61] Thomson, SK. Sampling. Second ed.; Wiley-Interscience; 2002.
[62] Spielman, D. A., . Laplacians.jl: Algorithms inspired by graph Laplacians: linear equation solvers, sparsification, clustering, optimization, etc. https://github.com/danspielman/Laplacians.jl; 2019.
[63] Karypis, G, Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 1999;20(1):359–392.
[64] Hachul, S, Jünger, M. Drawing large graphs with a potential-field-based multilevel algorithm. In: Proceedings of International Symposium on Graph Drawing. 2004, p. 285–295.
[65] Chimani, M, Klein, K, Beyer, S, Gronemann, M, Hedtke, I, Kurz, D, et al. Open Graph Drawing Framework (OGDF). `https://ogdf.uos.de/`; 2019.
[66] Edelsbrunner, H, Kirkpatrick, DG, Seidel, R. On the shape of a set of points in the plane. IEEE Transactions on Information Theory 1983;29(4):551–558.
[67] Newman, ME. Mixing patterns in networks. Physical Review E 2003;67(2):026126.
[68] Freeman, LC. Centrality in social networks conceptual clarification. Social Networks 1978;1(3):215–239.
[69] Saramäki, J, Kivelä, M, Onnela, JP, Kaski, K, Kertesz, J. Generalizations of the clustering coefficient to weighted complex networks. Physical Review E 2007;75(2):027105.
[70] Barrat, A, Barthelemy, M, Pastor-Satorras, R, Vespignani, A. The architecture of complex weighted networks. Proceedings of the National Academy of Sciences 2004;101(11):3747–3752.