SF-SGL: Solver-Free Spectral Graph Learning from Linear Measurements

Ying Zhang*, Zhiqiang Zhao* and Zhuo Feng
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ, USA
{yzhan232, zzhao76, zhuo.feng}@stevens.edu

***Abstract*—This work introduces a highly-scalable spectral graph densification framework (SGL) for learning resistor networks with linear measurements, such as node voltages and currents. We show that the proposed graph learning approach is equivalent to solving the classical graphical Lasso problems with Laplacian-like precision matrices. We prove that given $O(\log N)$ pairs of voltage and current measurements, it is possible to recover sparse N-node resistor networks that can well preserve the effective resistance distances on the original graph. In addition, the learned graphs also preserve the structural (spectral) properties of the original graph, which can potentially be leveraged in many circuit design and optimization tasks.**

To achieve more scalable performance, we also introduce a solver-free method (SF-SGL) that exploits multilevel spectral approximation of the graphs and allows for a scalable and flexible decomposition of the entire graph spectrum (to be learned) into multiple different eigenvalue clusters (frequency bands). Such a solver-free approach allows us to more efficiently identify the most spectrally-critical edges for reducing various ranges of spectral embedding distortions. Through extensive experiments for a variety of real-world test cases, we show that the proposed approach is highly scalable for learning sparse resistor networks without sacrificing solution quality. We also introduce a data-driven EDA algorithm for vectorless power/thermal integrity verifications to allow estimating worst-case voltage/temperature (gradient) distributions across the entire chip by leveraging a few voltage/temperature measurements.

***Index Terms*—spectral graph theory, graph Laplacian estimation, graphical Lasso, data-driven EDA, vectorless verification**

I. Introduction

Recent years have witnessed a surge of interest in machine learning on graphs [1], with the goal of encoding high-dimensional data associated with nodes, edges, or (sub)graphs into low-dimensional vector representations that well preserve the original graph structural (manifold) information. Graph learning techniques have shown promising results for various important applications such as vertex (data) classification [2], [3], link prediction (recommendation systems) [4], [5], community detection [6]–[8], drug discovery [9], [10], solving partial differential equations (PDEs) [11]–[13], and electronic design automation (EDA) [14]–[17].

Modern graph learning tasks (without a known input graph topology) typically involve the following two key tasks: (1) graph topology learning for converting high-dimensional node feature (attribute) data into a graph representation, and (2) graph embedding for converting graph-structured data (e.g. graph topology and node features) into low-dimensional vector representations to facilitate downstream machine learning or data mining tasks.

Although there exist abundant research studies on graph embedding techniques [1], [18], [19], it still remains challenging to learn a meaningful graph topology from a given data set. To this end, the well-known graphical Lasso method has been proposed as a sparse penalized maximum likelihood estimator for the concentration or precision matrix (inverse of covariance matrix) of a multivariate elliptical distribution [20]. The latest graph signal processing (GSP) based learning techniques also have been proposed to estimate sparse graph Laplacians, which has shown very promising results [21], [22]. For example, the graph topology learning problem is addressed by restricting the precision matrix (inverse of the sample covariance matrix) to be a graph Laplacian-like matrix and maximizing a posterior estimation of attractive Gaussian Markov Random Field (GMRF), while an ℓ_1-regularization term is leveraged to promote graph sparsity [21]; a graph Laplacian learning method is proposed by imposing additional spectral constraints [23]; a graph topology learning approach (GLAD) is introduced based on an Alternating Minimization (AM) algorithm [24], while [25] tries to learn a mapping from node data to the graph structure based on the idea of learning to optimise (L2O) [26], [27].

However, the state-of-the-art graph topology learning methods [21], [22], [24] do not scale to large data sets due to their high algorithm complexity. For example, recent graph topology learning methods based on Laplacian matrix estimation require solving convex optimization problems, which have a time complexity of at least $O(N^2)$ per iteration for N data points and thus can only be applied to rather small data sets with a few hundreds of data points [21], [22], [24]; the state-of-the-art deep learning based approach (GLAD) can only handle a few thousands of nodes on a high-performance GPU [24]. Consequently, existing graph topology learning methods can not be efficiently applied in electronic design automation (EDA) tasks considering the sheer size of modern integrated circuit systems that typically involve millions or billions of elements.

To the best of our knowledge, this paper introduces the first scalable spectral method (SF-SGL) for learning resistor networks from linear voltage and current measurements based

*These authors contributed equally to this work

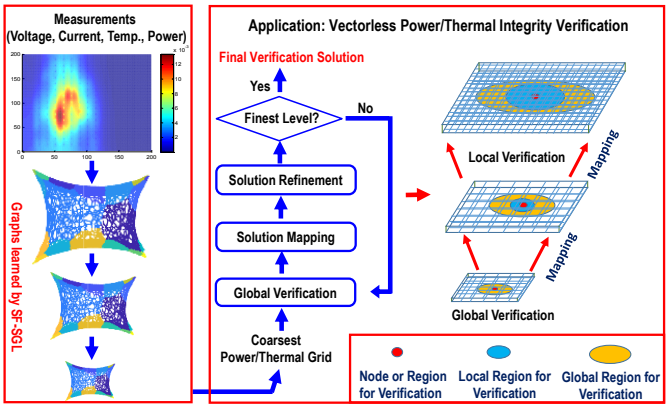


Figure 1. Spectral graph learning for data-driven vectorless verification.

on our recent SGL framework [28]. An indispensable step in the original SGL framework requires to compute Laplacian eigenvalues/eigenvectors for estimating spectral embedding distortions [28], which can be potentially accelerated by leveraging the latest theoretical results in spectral graph theory. However, the state-of-the-art spectral algorithms strongly rely on fast Laplacian solvers that are usually difficult to implement in practice and inherently-challenging to accelerate on parallel processors. For example, the effective-resistance based spectral sparsification method [29] requires multiple Laplacian matrix solutions for computing each edge's leverage (sampling) score, while the latest spectral-perturbation based algorithm [30] leverages a Laplacian solver for estimating each edge's spectral importance. The proposed algorithm (SF-SGL) is based on a scalable multilevel spectral graph densification framework for estimating attractive Gaussian Markov Random Fields (GMRFs). SF-SGL can efficiently solve the graphical Lasso problem [20] with a Laplacian-like precision matrix by iteratively including the most influential edges to dramatically reduce spectral embedding distortions. A unique property of the learned graphs is that the spectral embedding or effective-resistance distances on the constructed graph will encode the similarities between the original input data points (node voltage measurements). Our method allows each iteration to be completed in $O(N \log N)$ time, whereas existing state-of-the-art methods [21], [22] require at least $O(N^2)$ time for each iteration. Our analysis for sample complexity shows that by leveraging the proposed spectral algorithms it is possible to accurately estimate a sparse resistor network with only $O(\log N)$ voltage (and current) measurements (vectors).

The proposed solver-free approach (SF-SGL) will also bring new opportunities for developing brand new physics-informed, data-driven EDA algorithms and applications, as shown in Figure 1. In this work, by exploiting SF-SGL we introduce a data-driven EDA algorithm for vectorless power/thermal integrity verifications to allow estimating worst-case voltage/temperature (gradient) distributions across the entire chip by leveraging only a few voltage/temperature measurements that can be potentially obtained from on-chip voltage/temperature sensors [31]–[33].

The main contribution of this work has been summarized as follows:

1) This work introduces a spectral graph densification framework (SGL) for learning resistor networks with linear measurements. We prove that given $O(\log N)$ pairs of voltage and current measurements, it is possible to recover sparse N-node resistor networks that can well preserve the effective resistance distances on the original graph.
2) To achieve more scalable performance, a solver-free spectral graph learning framework (SF-SGL) is proposed that utilizes the multilevel spectral graph densification framework for constructing the learned graph. Compared to the previous SGL method which requires the Laplacian solver for estimating spectral embedding distortions, the proposed SF-SGL allows us to more efficiently identify the critical edges for constructing the learned graphs.
3) Our extensive experimental results show that the proposed method can produce a hierarchy of high-quality learned graphs in nearly-linear time for a variety of real-world, large-scale graphs and circuit networks. When compared with prior state-of-the-art spectral methods, such as SGL [28], the proposed SF-SGL can construct the learned graph in a much faster way with better graph quality.
4) The proposed method has been validated with the application of the data-driven EDA algorithm for vectorless power grid and thermal integrity verification for estimating worst-case voltage/temperature distributions of the entire chip, showing reliable verification accuracy and efficiency.

The rest of this paper is organized as follows: Section II introduces the background of graph learning techniques. Section III introduces the theoretical foundation of the original single-level spectral graph learning framework (SGL), which also includes the sample and algorithm complexity analysis. Section IV extends the SGL framework by introducing a more scalable solver-free multilevel graph learning approach (SF-SGL). Section V demonstrates extensive experimental results for learning a variety of real-world, large-scale graph problems, as well as data-driven vectorless integrity verification tasks, which is followed by the conclusion of this work in Section VI.

II. Background

A. Introduction to Graph Topology Learning

Given M observations on N data entities in a data matrix $X = [x_1, ..., x_M] \in \mathbb{R}^{N \times M}$, each column vector of X can be considered as a signal on a graph. For example, the MNIST data set [34], which includes $70,000$ images of handwritten digits with each image having 784 pixels, will result in a feature matrix $X \in \mathbb{R}^{N \times M}$ with $N = 70,000$ and $M = 784$. The recent GSP-based graph learning methods [21], [22], [35]–[37] estimate graph Laplacians from X for achieving the following two desired characteristics:

1) **Smoothness of graph signals.** The graph signals corresponding to the real-world data should be sufficiently smooth on the learned graph structure: the signal values will only change gradually across connected neighboring nodes.

2) **Sparsity of estimated Laplacians.** Graph sparsity is another critical consideration in graph learning. One of the most important motivations of learning a graph is to use it for downstream computing tasks. Therefore, more desired graph topology learning algorithms should allow better capturing and understanding the global structure (manifold) of the data set, while producing sufficiently sparse graphs that can be easily stored and efficiently manipulated in the downstream algorithms, such as circuit simulations/optimizations, network partitioning, dimensionality reduction, data visualization, etc.

B. Existing Methods for Graph Topology Learning

Problem formulation. Consider a random vector $x \sim N(0, \Sigma)$ with probability density function:

$$f(x) = \frac{\exp\left(-\frac{1}{2}x^\top \Sigma^{-1} x\right)}{(2\pi)^{N/2}\det(\Sigma)^{(1/2)}} \propto \det(\Theta)^{1/2}\exp\left(-\frac{1}{2}x^\top \Theta x\right), \tag{1}$$

where $\Sigma = \mathbb{E}[xx^\top] \succ 0$ denotes the covariance matrix, and $\Theta = \Sigma^{-1}$ denotes the precision matrix (inverse covariance matrix). Prior graph topology learning methods aim at estimating sparse precision matrix Θ from potentially high-dimensional input data, which fall into the following two categories:

(A) The graphical Lasso method aims at estimating a sparse precision matrix Θ using convex optimization to maximize the log-likelihood of $f(x)$ [20]:

$$\max_{\Theta} : \log\det(\Theta) - Tr(\Theta S) - \beta\|\Theta\|_1, \tag{2}$$

where Θ denotes a non-negative definite precision matrix, S denotes a sample covariance matrix, and β denotes a regularization parameter. The first two terms together can be interpreted as the log-likelihood under a GMRF. $\|\bullet\|_1$ denotes the entry-wise ℓ_1 norm, so $\beta\|\Theta\|_1$ becomes the sparsity promoting regularization term. This model learns the graph structure by maximizing the penalized log-likelihood. When the sample covariance matrix S is obtained from M i.i.d (independent and identically distributed) samples $X = [x_1, ..., x_M]$, where $X \sim N(0, S)$ has an N-dimensional Gaussian distribution with zero mean, each element in the precision matrix $\Theta_{i,j}$ encodes the conditional dependence between variables X_i and X_j. For example, $\Theta_{i,j} = 0$ implies that variables X_i and X_j are conditionally independent, given the rest.

(B) The GSP-based Laplacian estimation methods have been recently introduced for more efficiently solving the following convex problem [22], [38]:

$$\max_{\Theta} : F(\Theta) = \log\det(\Theta) - \frac{1}{M}Tr(X^\top \Theta X) - \beta\|\Theta\|_1, \tag{3}$$

where $\Theta = L + \frac{1}{\sigma^2}I$, L represents the set of valid Laplacian matrices, where $Tr(\bullet)$ denotes the matrix trace, I denotes the identity matrix, and $\sigma^2 > 0$ represents the prior feature variance. The three terms in (3) correspond to the terms $\log\det(\Theta)$, $Tr(\Theta S)$ and $\beta\|\Theta\|_1$ in (2), respectively. When every column vector in the data matrix X is regarded as a graph signal vector, there is a close connection between the formulation in (3) and the original graphical Lasso problem [20]. Since $\Theta = L + \frac{1}{\sigma^2}I$ are symmetric and positive definite (PSD) matrices (or M matrices) with non-positive off-diagonal entries, this formulation will lead to the estimation of attractive GMRFs [22], [39]. When X is non-Gaussian, the formulation in (3) can be regarded as a Laplacian estimation method by minimizing the Bregman divergence between positive definite matrices induced by the function $\Theta \mapsto -\log\det(\Theta)$ [39].

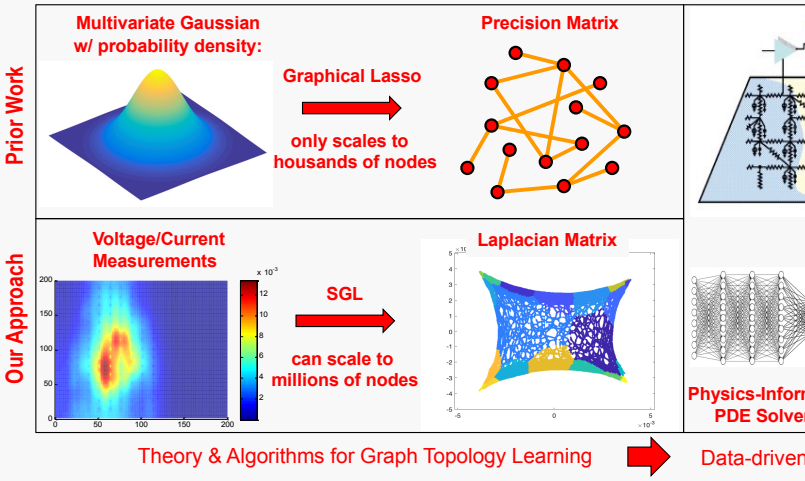


Figure 2. Overview of the proposed framework for graph topology learning.

III. SGL: A Spectral Learning Approach

Consider M linear measurements of N-dimensional voltage and current vectors stored in data matrices $X \in \mathbb{R}^{N\times M}$ and $Y \in \mathbb{R}^{N\times M}$, where the i-th column vector $X(:,i)$ is a voltage response (graph signal) vector corresponding to the i-th input current vector $Y(:,i)$. This work introduces a spectral graph learning method (SGL) for Laplacian matrix estimation by exploiting the linear voltage (X) and current (Y) measurements [28], as shown in Figure 2.

To quantify the smoothness of a graph signal vector x over a undirected graph $G = (V, E, w)$, the following Laplacian quadratic form can been adopted:

$$x^\top Lx = \sum_{(s,t)\in E} w_{s,t}(x(s) - x(t))^2, \tag{4}$$

where $L = D - W$ denotes the graph Laplacian matrix, $w_{s,t} = W(s,t)$ denotes the weight for edge (s,t), while D and W denote the degree and the weighted adjacency matrices, respectively. The smoothness of a set of signals X over graph G can be computed using the following matrix trace [36]:

$$Q(X, L) = Tr(X^\top LX). \tag{5}$$

A. Gradient Estimation via Perturbation Analysis

Express the graph Laplacian matrix as follows

$$L = \sum_{(s,t)\in E} w_{s,t}e_{s,t}e_{s,t}^\top, \tag{6}$$

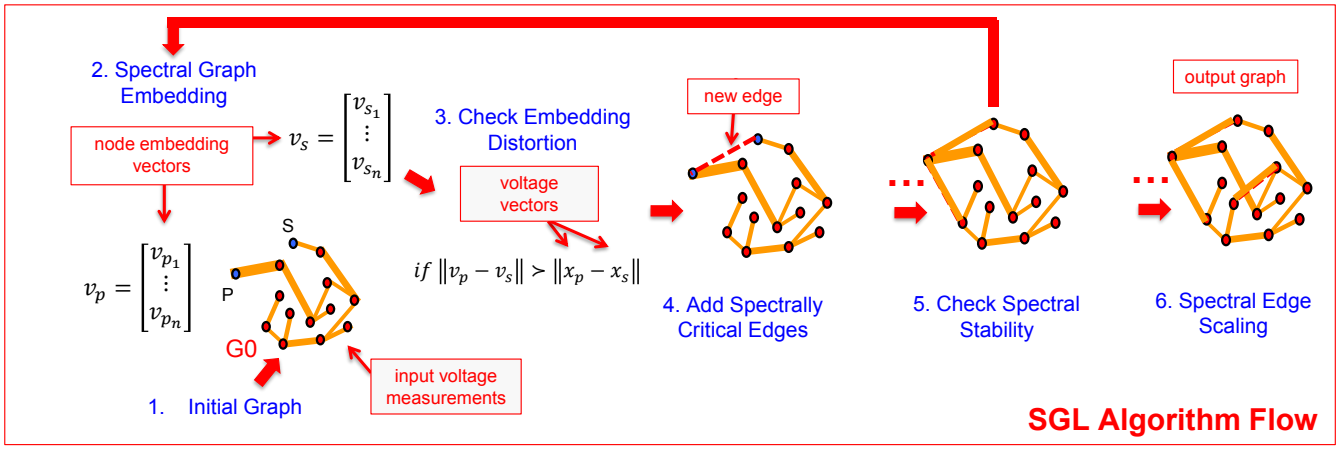


Figure 3. SGL for spectral graph topology learning from voltage/current measurements. (v_p and v_s are node embedding vectors for node p and s, x_p and x_s are voltage vectors for node p and s.)

where $e_s \in \mathbb{R}^N$ denotes the standard basis vector with all zero entries except for the s-th entry being 1, and $e_{s,t} = e_s - e_t$. By substituting (6) into (3), we have:

$$F = \sum_{i=1}^{N} \log(\lambda_i + \frac{1}{\sigma^2}) - \frac{1}{M}\left(\frac{Tr(X^\top X)}{\sigma^2} + \sum_{(s,t)\in E} w_{s,t}\|X^\top e_{s,t}\|_2^2\right) - 4\beta \sum_{(s,t)\in E} w_{s,t}, \tag{7}$$

where λ_i are non-decreasing eigenvalues of L for $i = 1, \cdots, N$. Given the eigenvalue λ_i and the corresponding eigenvector u_i, it satisfies:

$$Lu_i = \lambda_i u_i. \tag{8}$$

Taking the partial derivative of (7) with respect to $w_{s,t}$ leads to:

$$\frac{\partial F}{\partial w_{s,t}} = \sum_{i=1}^{N} \frac{1}{\lambda_i + 1/\sigma^2}\frac{\partial \lambda_i}{\partial w_{s,t}} - \frac{1}{M}\|X^\top e_{s,t}\|_2^2 - 4\beta. \tag{9}$$

The last two terms in (9) both imply constraints on graph sparsity: adding more edges will lead to a greater trace $Tr(X^\top \Theta X)$. Consequently, we can safely choose $\beta = 0$ in the rest of this work, without impacting the ranking of candidate edges and thus the final solution quality of SGL.

Theorem III.1. *The spectral perturbation $\delta\lambda_i$ due to adding a candidate edge (s,t) into the latest graph will be:*

$$\delta\lambda_i = \delta w_{s,t}\left(u_i^\top e_{s,t}\right)^2. \tag{10}$$

Proof. Consider the following spectral perturbation analysis:

$$(L + \delta L)(u_i + \delta u_i) = (\lambda_i + \delta\lambda_i)(u_i + \delta u_i), \tag{11}$$

where a small perturbation δL due to including a new edge (s,t) is applied to L, which results in perturbed eigenvalues and eigenvectors $\lambda_i + \delta\lambda_i$ and $u_i + \delta u_i$ for $i = 1, ..., N$, respectively. By only keeping the first-order terms, we have:

$$L\delta u_i + \delta L u_i = \lambda_i \delta u_i + \delta\lambda_i u_i. \tag{12}$$

Express δu_i in terms of the original eigenvectors u_i for $i = 1, ..., N$ as:

$$\delta u_i = \sum_{i=j}^{N} \alpha_j u_j. \tag{13}$$

By substituting (13) into (12), we have:

$$L\sum_{j=1}^{N} \alpha_j u_j + \delta L u_i = \lambda_i \sum_{j=1}^{N} \alpha_j u_j + \delta\lambda_i u_i. \tag{14}$$

After multiplying $u_i^\top$ to both sides of (14) we have:

$$\delta\lambda_i = \delta w_{s,t}\left(u_i^\top e_{s,t}\right)^2. \tag{15}$$

□

Subsequently, we construct the following eigensubspace matrix for spectral graph embedding using the first $r-1$ nontrivial Laplacian eigenvectors

$$U_r = \left[\frac{u_2}{\sqrt{\lambda_2 + 1/\sigma^2}}, ..., \frac{u_r}{\sqrt{\lambda_r + 1/\sigma^2}}\right]. \tag{16}$$

Theorem III.1 allows each edge's **spectral sensitivity** $s_{s,t} = \frac{\partial F}{\partial w_{s,t}}$ in (9) to be written as:

$$\begin{aligned} s_{s,t} &= \sum_{i=1}^{N} \frac{1}{\lambda_i + 1/\sigma^2}\frac{\partial \lambda_i}{\partial w_{s,t}} - \frac{1}{M}\|X^\top e_{s,t}\|_2^2 \\ &= \|U_N^\top e_{s,t}\|_2^2 - \frac{1}{M}\|X^\top e_{s,t}\|_2^2 \\ &\approx \|U_r^\top e_{s,t}\|_2^2 - \frac{1}{M}\|X^\top e_{s,t}\|_2^2 \\ &= z_{s,t}^{emb} - \frac{1}{M} z_{s,t}^{data}, where \quad r \ll N. \end{aligned} \tag{17}$$

In the above expressions, $z_{s,t}^{emb} = \|U_r^\top e_{s,t}\|_2^2$ and $z_{s,t}^{data} = \|X^\top e_{s,t}\|_2^2$ denote the ℓ_2 distances in the spectral embedding space as well as the data (voltage measurement) vector space, respectively. The partial derivative term $s_{s,t}$ in (17) can be leveraged for solving the optimization task in (3) using gradient-based methods, such as the general stagewise algorithm for group-structured learning [40].

B. Spectral Densification for Graph Topology Learning

Spectrally-critical edges. We define the spectral embedding distortion $\eta_{s,t}$ of an edge (s,t) to be:

$$\eta_{s,t} = M\frac{z_{s,t}^{emb}}{z_{s,t}^{data}}. \tag{18}$$

We also call the candidate edge (p,q) that has a relatively large spectral sensitivity ($s_{p,q}$) or embedding distortion $\eta_{p,q} = Mz_{p,q}^{emb}/z_{p,q}^{data}$ a spectrally-critical edge.

The proposed SGL iterations. Let's consider the $(k+1)$-th SGL iteration for finding a few most spectrally-critical candidate edges with the largest spectral sensitivities (or embedding distortions) to be added into the latest graph $G_k = (V_k, E_k, w_k)$. (17) implies that including such edges into the latest graph G_k will more effectively improve the objective function and significantly mitigate the spectral embedding distortions.

Spectral graph sparsification (prior work). Prior research shows that for every undirected graph there exists a sparsified graph with $O(\frac{N\log N}{\epsilon^2})$ edges that can be obtained by sampling each edge (p,q) with a probability (leverage score) p_e proportional to its effective resistance [29]:

$$p_e \propto \frac{R_{p,q}^{\mathit{eff}}}{R_{p,q}} = w_{p,q}R_{p,q}^{\mathit{eff}}, \tag{19}$$

where $R_{p,q} = 1/w_{p,q}$ represents the original resistance and $R_{p,q}^{\mathit{eff}}$ is the edge effective resistance. In addition, the following inequality will be valid [29]:

$$\forall x \in \mathbb{R}^N \quad (1-\epsilon)x^\top Lx \le x^\top \tilde{L}x \le (1+\epsilon)x^\top Lx, \tag{20}$$

where L and $\tilde{L}$ are the original and sparsified graph Laplacian matrices, respectively.

Spectral graph densification. For any candidate edge selected based on its spectral sensitivity during an SGL iteration, by setting its weight as:

$$w_{p,q} \propto \frac{1}{z_{p,q}^{data}}, \tag{21}$$

the spectral embedding distortion can be estimated as follows

$$\eta_{p,q} = M\frac{z_{p,q}^{emb}}{z_{p,q}^{data}} \propto w_{p,q}R_{p,q}^{\mathit{eff}}, \tag{22}$$

which becomes the leverage score for spectral graph sparsification [29]. Consequently, as opposed to spectral sparsification, SGL can be regarded as a spectral graph densification procedure that aims to identify and include $O(N\log N)$ spectrally-critical edges with large spectral sensitivities (embedding distortions).

Algorithm convergence. The optimal solution of (3) can be achieved when the maximum edge sensitivity (s_{max}) in (17) becomes zero or equivalently when the maximum spectral embedding distortion (η_{max}) in (18) becomes one. Moreover, upon the convergence of SGL iterations, the spectral embedding (effective-resistance) distances on the learned graphs will encode the ℓ_2 distances between the original data points (voltage measurements), which can be exploited in many important tasks, such as VLSI CAD, manifold learning, dimensionality reduction, and data visualization [41]–[44].

C. Sample Complexity of the SGL Algorithm

The sample complexity of SGL can be obtained by analyzing the required number of voltage vectors (measurements) for accurate graph learning. We assume $\sigma^2 \to +\infty$. For the ground-truth graph G_*, we define its edge weight matrix W_* to be a diagonal matrix with $W_*(i,i) = w_i$, and its injection matrix as:

$$B_*(i,p) = \begin{cases} 1 & \text{if } p \text{ is i-th edge's head} \\ -1 & \text{if } p \text{ is i-th edge's tail} \\ 0 & \text{otherwise .} \end{cases} \tag{23}$$

Consequently, the Laplacian matrix of G_* can be written as

$$L_* = B_*^\top W_* B_*. \tag{24}$$

Therefore, the effective resistance $R_*^{\mathit{eff}}(s,t)$ between nodes s and t can be expressed as:

$$R_*^{\mathit{eff}}(s,t) = e_{s,t}^\top L_*^+ e_{s,t} = \|W_*^{\frac{1}{2}} B_* L_*^+ e_{s,t}\|_2^2, \tag{25}$$

where L_*^+ represents the Moore–Penrose pseudoinverse of L_*. According to the Johnson-Lindenstrauss Lemma, the effective-resistance distance for every pair of nodes satisfies [29]:

$$(1-\epsilon)R_*^{\mathit{eff}}(s,t) \le \|X^\top e_{s,t}\|_2^2 \le (1+\epsilon)R_*^{\mathit{eff}}(s,t), \tag{26}$$

where the data (voltage measurement) matrix $X \in \mathbb{R}^{N\times M}$ is created by going through the following steps:

1) Let C be a random $\pm\frac{1}{\sqrt{M}}$ matrix of dimension $M \times |E|$, where $|E|$ denotes the number of edges and $M = 24\log\frac{N}{\epsilon^2}$ denotes the number of voltage measurements;
2) Construct $Y = CW_*^{\frac{1}{2}}B_*$, with the i-th row vector denoted by $y_i^\top$;
3) Solve $L_* x_i = y_i$ for all rows in C ($1 \le i \le M$), and construct X matrix using x_i as its i-th column vector.

Obviously, (26) implies that given $M \ge O(\log N/\epsilon^2)$ voltage vectors (measurements) obtained through the above procedure, $(1\pm\epsilon)$-approximate effective resistances can be computed by

$$\tilde{R}_*^{\mathit{eff}}(s,t) = \|X^\top e_{s,t}\|_2^2 \tag{27}$$

for any pair of nodes (s,t) in the original graph G_*. Consider the following close connection between effective resistances and spectral graph properties (such as the first few Laplacian eigenvalues/eigenvectors):

$$R_{s,t}^{\mathit{eff}} = \|U_N^\top e_{s,t}\|_2^2, \text{ where } U_N = \left[\frac{u_2}{\sqrt{\lambda_2}}, ..., \frac{u_N}{\sqrt{\lambda_N}}\right]. \tag{28}$$

Therefore, using $O(\log N)$ voltage measurements would be sufficient for SGL to learn an N-node graph for well preserving the original graph effective-resistance distances.

D. Key Steps in the SGL Algorithm

To achieve good runtime and memory efficiency in graph learning tasks that may involve a large number of nodes, the proposed SGL algorithm strives to iteratively identify and include the most influential (spectrally-critical) edges into the latest graph until no such edges can be found, through the following key steps, as shown in Figure 3:

Step 1: Initial graph construction. (17) implies that by iteratively identifying and adding the most influential edges (with the largest sensitivities) into the latest graph, the graph spectral embedding (or effective-resistance) distance will encode the ℓ_2 distances between the original data (voltage measurement) vectors. The ideal pool of candidate edges should include all possible edge connections, e.g. forming a complete graph with $O(N^2)$ edges for N data samples. The proposed graph learning algorithm can be better understood through the following steps: a) each initial edge weight of the complete graph is set to be a very small value (close to zero); b) the edge weight with the highest spectral sensitivity will be updated with a greater value; c) repeat step b) until no edge can be updated (no positive edge sensitivity exists). The above iterative scheme is equivalent to starting with no edge connectivity. However, considering all edge connections will require the quadratic complexity, which may lead to rather poor runtime scalability. To achieve a better runtime scalability, k-nearest-neighbor (kNN) graph [45] can be leveraged as a sparsified complete graph. Note that when the effective-resistance distances encode the Euclidean distances between data samples, the proposed graph learning iterations will converge. This implies that when the learned graph has a tree-like structure, the effective-resistance distances will approximately match the shortest path distances and thus encode the Euclidean distances between data samples. Therefore, Euclidean distance becomes a natural choice for kNN graph construction.

However, choosing an optimal k value (the number of nearest neighbors) for constructing kNN graphs can still be challenging for general graph learning tasks: choosing a too large k allows well approximating the global structure of the manifold for the original data points (voltage measurements), but will result in a rather dense graph; choosing a too small k may lead to many small isolated graphs, which may slow down the iterations. Since circuit networks are typically very sparse (e.g. 2D or 3D meshes) in nature, the voltage or current measurements (vectors) usually lie near low-dimensional manifolds, which allows choosing a proper k for our graph learning tasks. To achieve a good trade-off between complexity and quality, the initial graph will be set up through the following steps: **(1)** Construct a connected kNN graph with a relatively small k value (e.g. $5 \leq k \leq 10$), which will suffice for approximating the global manifold corresponding to the original measurement data; **(2)** Sparsify the kNN graph by extracting a maximum spanning tree (MST) that can serve as a reasonably good initial graph in practice. Later, SGL will incrementally improve the graph by iteratively adding the most influential off-tree edges from the initial kNN graph until convergence. As shown in Figure 18, using different k values will not significantly impact the final objective function value after going through the proposed graph learning iterations.

Step 2: Spectral graph embedding. Spectral graph embedding leverages the first few Laplacian eigenvectors for mapping nodes onto low-dimensional space [42]. The eigenvalue decomposition of Laplacian matrix is usually the computationally demanding in spectral graph embedding, especially for large graphs. To achieve good scalability, it is possible to exploit recent fast Laplacian eigensolvers with nearly-linear time complexity [46].

Step 3: Influential edge identification. Given the first few Laplacian eigenvalues and eigenvectors, SGL will efficiently identify the most influential off-tree edges by looking at each candidate edge's sensitivity score defined in (17). Specifically, each off-tree edge that belongs to the kNN graph will be sorted according to its edge sensitivity. Only a few most influential edges with the largest sensitivities will be included into the latest graph. Note that when $r \ll N$, the following inequality holds for any edge (s,t):

$$\|U_r^\top e_{s,t}\|_2^2 = z_{s,t}^{emb} < \|U_N^\top e_{s,t}\|_2^2 \leq R^{\mathit{eff}}(s,t), \qquad (29)$$

which implies that the edge sensitivities ($s_{s,t}$) approximately computed using the first r eigenvectors will always be lower than the actual values. It is obvious that using more eigenvectors for spectral embedding will lead to more accurate estimation of edge sensitivities. For typical circuit networks, edge sensitivities computed using only a small number (e.g. $r < 10$) of eigenvectors will be sufficiently accurate for ranking the off-tree edges.

Step 4: Convergence checking. In this work, we examine the maximum edge sensitivities computed by (17) for checking the convergence of SGL iterations. If there exists no off-tree edge that has a sensitivity greater than a given threshold ($s_{max} \geq tol$), the SGL iterations will be terminated. We note that choosing different tolerance (tol) levels will result in graphs with different densities. For example, choosing a smaller threshold will result in more edges to be included so that the final spectral embedding distances on the learned graph can more accurately encode the distances between the original data points (voltage measurements).

Step 5: Spectral edge scaling. To further improve the spectral approximation quality, we choose to globally scale up edge weights of the learned graph obtained via SGL. Given the ground truth Laplacian matrix L_*, the effective resistance between nodes s and t can be represented as $R_{s,t}^{\mathit{eff}} = e_{s,t}^T L_*^+ e_{s,t}$. If we consider the graph as a resistor network with each conductance value corresponding to each edge weight, $R_{s,t}^{\mathit{eff}}$ can be regarded as the power dissipation of the resistor network when a unit current is flowing into node s and out from node t. By relaxing the vector $\mathbf{e}_{s,t}$ with a random vector y_i that is orthogonal to the all-one vector, it can be shown that matching the power dissipations between the original graph and the learned graph will immediately lead to improved spectral approximation of the learned graph. Given the normalized current vectors $Y = (y_1, \cdots, y_M)$, corresponding measurement (voltage) vectors $X = (x_1, \cdots, x_M)$ and $\tilde{X} = (\tilde{x}_1, \cdots, \tilde{x}_M)$ can be computed for the original graph and the learned graph, respectively, where $x_i = L_*^+ y_i$ and $\tilde{x}_i = L^+ y_i$. To better match the structural properties of the original graph, we propose to globally scale up the edge weights in learned graph:

$$w_{s,t} = w_{s,t} * \alpha' \qquad (30)$$

with the following scaling factor α':

$$\alpha' = \frac{1}{M} \sum_{i=1}^{M} \frac{y_i^\top L^+ y_i}{y_i^\top L_*^+ y_i} = \frac{1}{M} \sum_{i=1}^{M} \frac{y_i^\top \tilde{x}}{y_i^\top x_i}. \qquad (31)$$

E. Algorithm Flow and Complexity of SGL

The detailed SGL algorithm flow has been shown in Algorithm 1. All the aforementioned steps in SGL can be accomplished in nearly-linear time by leveraging recent high-performance algorithms for kNN graph construction [45], spectral graph embedding for influential edge identification [41], [46], and fast Laplacian solver for edge scaling [47], [48]. Consequently, each SGL iteration can be accomplished in nearly-linear time, whereas the state-of-the-art methods require at least $O(N^2)$ time [22].

Algorithm 1 The SGL Algorithm Flow.

Input: The voltage measurement matrix $X \in \mathbb{R}^{N\times M}$, input current measurement matrix $Y \in \mathbb{R}^{N\times M}$, k for initial kNN graph construction, r for constructing the projection matrix in (16), the maximum edge sensitivity tolerance (tol), and the edge sampling ratio ($0 < \beta \leq 1$). **Output:** The learned graph $G = (V, E, w)$.

1: Construct a kNN graph $G_o = (V, E_o, w_o)$ based on X.
2: Extract an MST subgraph T from G_o.
3: Assign $G = T = (V, E, w)$ as the initial graph.
4: **while** $s_{max} \geq tol$ **do**
5: Compute the projection matrix U_r with (16) for the latest graph G.
6: Sort off-tree edges $(s,t) \in E_o \setminus E$ according to their sensitivities computed by $s_{s,t} = \frac{\partial F}{\partial w_{s,t}}$ using (17).
7: Include an off-tree edge (s,t) into G if its $s_{s,t} > tol$ and it has been ranked among the top $\lceil N\beta \rceil$ edges.
8: Record the maximum edge sensitivity s_{max}.
9: **end while**
10: Do spectral edge scaling using X and Y following (30);
11: Return the learned graph G.

IV. SF-SGL: Solver-Free Spectral Graph Learning

Overview of SF-SGL. In this work, we extend SGL by introducing a solver-free, multilevel spectral graph learning scheme (SF-SGL). The key procedures in SF-SGL are described as follows: (1) given the measurement vectors, such as voltage and current matrices X and Y, a hierarchy of coarse level kNN graphs will be constructed by exploiting a scalable spectral graph coarsening framework [46], as shown in Figure 4; (2) the SGL algorithm will be leveraged for graph learning starting from the coarsest level: an MST is first extracted, which is followed by a procedure for identifying spectrally-critical off-tree edges (that belong to the coarsest kNN graph) with large embedding distortions that can be efficiently computed by exploiting a solver-free local spectral graph embedding scheme; (3) after SGL converges at the coarsest level, the learned graph will be progressively mapped to the finer levels, as show in Figure 5, where additional candidate edges (that belong to the coarsened kNN graphs) will be identified and included into the latest graphs with updated local embedding vectors, as shown in Figure 6. Once the learned graph is obtained, the proposed framework and learned graph can be efficiently applied in different tasks. One immediate application is to facilitate the vectorless power grid and thermal integrity verification, as shown in Figure 1.

In the rest of this paper, given an initial kNN graph $G_0 = G$, a series of coarsened kNN graphs $G_1, G_2, ..., G_{l_f}$ will be generated via the spectral graph coarsening [41], where G_{l_f} denotes the coarsest kNN graph. The multilevel learned graphs are denoted by $P_0, P_1, ..., P_{l_f}$. For the sake of simplicity, all the symbols used in this paper are summarized in Table I.

Table I
SUMMARY OF SYMBOLS IN THE SF-SGL FRAMEWORK
($l = 0, 1, ..., l_f, i = 1, ..., n_l$).

symbols	description	symbols	description
Θ	precision matrix	$\mathcal{C}$	sample covariance matrix
H_l	coarsening operator	X_l	feature matrix at level l
$z_{p,q}^{data}$	data distance	$z_{p,q}^{emb}$	embedding distance
$s_{p,q}$	spectral sensitivity	$\eta_{p,q}$	distortion
$G_l = (V_l, E_l)$	an undirected graph at level l	$P_l = (V_l, E_l)$	the learned graph
$\omega_l(p,q)$	weight of edge (p,q) for G_l	$\tilde{\omega}_l(p,q)$	weight of edge (p,q) for P_l
E_l	edge set of G_l	$\tilde{E}_l$	edge set of P_l
$m_l = \lvert E_l \rvert$	number of edges in G_l	$\tilde{m}_l = \lvert \tilde{E}_l \rvert$	number of edges in P_l
V_l	node set at level l	$n_l = \lvert V_l \rvert$	number of nodes
L_{G_l}	Laplacian of graph G_l	L_{P_l}	Laplacian of graph P_l
$u_l^{(i)}$ and $\tilde{u}_l^{(i)}$	eigenvectors of L_{G_l} and L_{P_l}	$\lambda_l^{(i)}$ and $\tilde{\lambda}_l^{(i)}$	eigenvalues of L_{G_l} and L_{P_l}

A. Spectral Graph Coarsening

Node aggregation sets for graph coarsening. As shown in Figure 4, coarsening the graph from G_{l-1} to G_l requires to cluster the node set V_{l-1} of G_{l-1} into n_l different node aggregation sets $S_{l-1}^{(i)}$ with $i = 1, ..., n_l$. The subgraph $G_{l-1}[S_{l-1}^{(i)}]$ induced by each node aggregation set $S_{l-1}^{(i)}$ will be a strongly-connected component in graph G_{l-1}, which will be aggregated into a single node in the coarser graph G_l. Consequently, once the node aggregation sets are determined, a node-mapping matrix H_l can be uniquely constructed accordingly, which allows creating G_l as follows:

$$L_{G_l} := H_l^{\mp} L_{G_{l-1}} H_l^{+}, \quad x_l := H_l x_{l-1}, \tag{32}$$

where $H_l \in \mathbb{R}^{n_l \times n_{l-1}}$, $x_l \in \mathbb{R}^{n_l \times 1}$, $l = 1, ..., l_f$, and $H_l^{\top}, H_l^{+}, H_l^{\mp}$ denote the transpose, pseudoinverse, and transposed pseudoinverse of H_l, respectively. H_l and H_l^{+} can be constructed as follows [49]:

$$H_l(i,p) = \begin{cases} \frac{1}{\lvert S_{l-1}^{(i)} \rvert} & \text{if p} \in S_{l-1}^{(i)} \\ 0 & \text{otherwise .} \end{cases}, \; H_l^{+}(p,i) = \begin{cases} 1 & \text{if p} \in S_{l-1}^{(i)} \\ 0 & \text{otherwise.} \end{cases} \tag{33}$$

Spectral coarsening via local embedding. The key to spectral coarsening in SF-SGL is to determine the node aggregation sets. To this end, a low-pass graph filtering scheme has been adopted for local spectral embedding, which can be achieved in linear time [41]. Let $b_l^{(k)} \in \mathbb{R}^{n_l \times 1}$ with $k = 1, \cdots, K$ be the initial random vectors which are orthogonal to the all-one vector. After applying a few steps of Gaussian-Seidel relaxations for solving the linear system of equations $L_{G_l} b_l^{(k)} = 0$ with $k = 1, \cdots, K$ [50], we can associate each node in the graph G_l with a K-dimensional vector using the embedding matrix $B_l = [b_l^{(1)}, ..., b_l^{(K)}]$. Since each $b_l^{(k)}$ can be considered as the linear combinations of the first few Laplacian eigenvectors after smoothing (low-pass filtering), the embedding matrix B_l

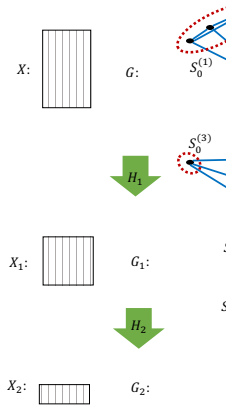


Figure 4. Graph coarsening by local embedding.

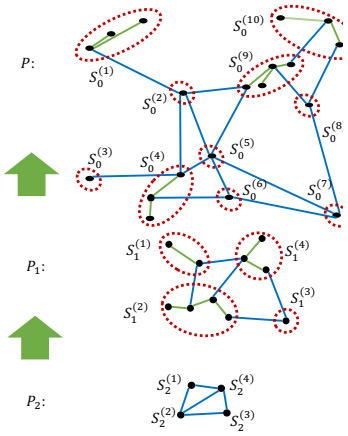


Figure 5. Graph learning from bottom up.

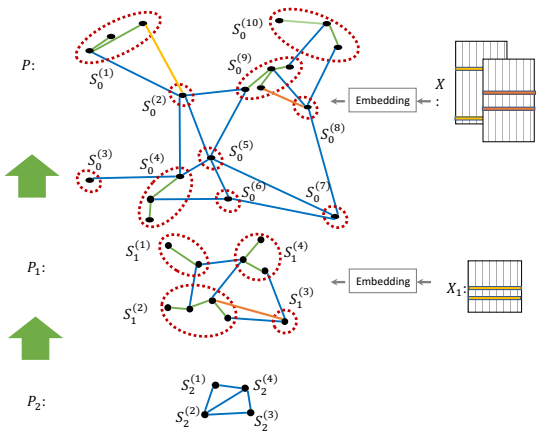


Figure 6. Influential off-tree edge identification.

can well approximate the eigensubspace matrix U_r in (16). Consequently, the nodes that are close to each other in the spectral embedding space can be clustered into the same aggregation set. After finding all node aggregation sets, the node mapping matrix H_l that uniquely determines a coarse-level graph can be formed. Subsequently, the node feature matrix at level l can also be obtained by $X_l = H_l X_{l-1}$, as shown in Figure 4. To better encode the feature information on the coarsened graph, we propose the following scheme for updating edge weight between neighbouring nodes s and t based on their feature distance:

$$\omega_l(s,t) = 1/\|X_l^\top e_{s,t}\|_2^2. \tag{34}$$

Bounding approximation errors. To ensure the solution quality of the proposed graph learning framework, it is important to estimate and bound the errors due to spectral graph coarsening. Graph G_l is considered as a good approximation of its finer version G_{l-1} if the following holds [49]:

$$\begin{aligned} \gamma_1 \lambda_{l-1}^{(i)} \le \lambda_l^{(i)} \le \gamma_2 \frac{(1+\epsilon)^2}{1-\tau\epsilon^2} \lambda_{l-1}^{(i)} \\ \frac{1}{\sigma_{l-1}} \|u_{l-1}^{(i)}\|_{L_{G_{l-1}}} \le \|u_l^{(i)}\|_{L_{G_l}} \le \sigma_{l-1} \|u_{l-1}^{(i)}\|_{L_{G_{l-1}}}, \end{aligned} \tag{35}$$

where $l = 1, ..., l_f$, $\lambda_l^{(1)}, .., \lambda_l^{(K)}$ denote the K non-decreasing eigenvalues of L_{G_l}, $\tau = \lambda_{l-1}^{(K)}/\lambda_{l-1}^{(2)}$, $\epsilon = (\sigma_{l-1}^2 - 1)/(\sigma_{l-1}^2 + 1)$ with $\sigma_{l-1} \le (\frac{1+\sqrt{\tau}}{1-\sqrt{\tau}})^{\frac{1}{2}}$, and $\gamma_1(\gamma_2)$ denotes the smallest (largest) eigenvalue of $(H_l H_l^\top)^{-1}$.

B. Graph Topology Learning via A Bottom-up Approach

A spectrum decomposition perspective. As shown in [51], the spectrally-coarsened graphs will carry different (spectrum) bandwidths of the original graph G. For example, the coarsest graph Laplacian $L_{G_{l_f}}$ will only preserve the key spectral properties of G_0, such as the first few Laplacian eigenvalues and eigenvectors, whereas the Laplacians of the increasingly finer graphs will match the lower to moderate eigenvalues of L_{G_0}. Consequently, the coarsened graphs can be considered as a cascade of low-pass graph filters with gradually decreasing bandwidths: the finest graph retains the highest bandwidth, whereas the coarsest graph retains the lowest bandwidth.

Mapping the learned graphs to finer levels. SF-SGL strives to progressively form a series of increasingly finer graphs such that the spectral embedding distortions can be significantly mitigated. As shown in Figure 5, to map the learned graph P_l to a finer level, two different types of edges will be formed for constructing the graph P_{l-1} at the $(l-1)$-th level: (1) any node in P_l corresponds to a node aggregation set in P_{l-1}, in which an MST will be extracted to form the **inner-cluster edges**; (2) any edge in P_l corresponds to at least one edge connecting between the two node aggregation sets in P_{l-1}, while only the edge with the largest weight will be kept as the **inter-cluster edge** for constructing P_{l-1}.

Spectrally-critical edges at coarse levels. As shown in Figure 6, in order to identify and include spectrally-critical off-subgraph edges with large spectral embedding distortions, the aforementioned local spectral embedding will be applied for graph P_{l-1} to generate its embedding matrix $B'_{l-1} = [b_{l-1}^{(1)}, \cdots, b_{l-1}^{(K)}]$. The spectral embedding distortion of the coarse level graph P_{l-1} can be computed as

$$\eta_{s,t} = z_{s,t}^{emb}/z_{s,t}^{data}, \tag{36}$$

where $z_{s,t}^{emb} = \|B_{l-1}^{'\top} e_{s,t}\|_2^2$ and $z_{s,t}^{data} = \|X_{l-1}^\top e_{s,t}\|_2^2$. A larger embedding distortion of an off-subgraph edge indicates that including this edge into the learned graph will more precisely encode the distances between the data points. Note that the edges identified by SF-SGL on the coarsest graph will tend to connect the most distant nodes for drastically mitigating long-range spectral embedding distortions, thereby impacting only the first few Laplacian eigenvalues (low-frequency band); on the other hand, the edges identified on the finest graph will tend to only connect local nodes for mitigating short-range embedding distortions, thereby impacting relatively large eigenvalues (high-frequency band).

C. Algorithm Flow and Complexity of SF-SGL

The proposed multilevel graph learning framework (SF-SGL) allows a flexible decomposition of the entire graph

spectrum (to be learned) into multiple different ei clusters, such that the most spectrally-critical edges the key to mitigating various ranges of spectral er distortions can be more efficiently identified. Alg shows the algorithm flow for the proposed SF-SGL fra All the aforementioned steps in SF-SGL can be acco in nearly-linear time by leveraging recent high per algorithms for kNN graph construction [45], solver-f tral graph coarsening and embedding, as well as fast l solver. Consequently, SF-SGL can be accomplished i linear time.

Algorithm 2 The SF-SGL algorithm flow.

Input: Voltage and current measurement matrices: X, $Y \in \mathbb{R}^{N\times M}$, k for initial kNN graph construction;
Output: Learned graph P;

$P_l = \emptyset$ for $l = 0, ..., l_f$;
Construct an initial kNN graph G as G_0 based on X;
Perform spectral graph coarsening to get a series of coarsened kNN graphs $G_1, ..., G_{l_f}$ as well as the coarsening operators $H_1, ..., H_{l_f}$;
Let $X_0 = X$ and calculate the feature matrices $X_1, ..., X_{l_f}$ by $X_l = HX_{l-1}$ for $l = 1, ..., l_f$;
Set $l = l_f$ and learn a sparse graph P_l through SGL iterations;
while $l \geq 1$ **do**
  for each node $i \in V_l$ **do**
    Find the induced subgraph $G_{l-1}[S^{(i)}_{l-1}]$ w.r.t. the aggregation set $S^{(i)}_{l-1}$;
    Extract the MST of the induced subgraph $G_{l-1}[S^{(i)}_{l-1}]$ and add its edges into P_{l-1} ;
  end for
  for each edge $(s,t) \in \tilde{E}_l$ **do**
    Add the edge with maximum weight between sets $S^{(s)}_{l-1}$ and $S^{(t)}_{l-1}$ into graph P_{l-1} ;
  end for
  Compute $\eta_{s,t}$ for each candidate edge $(s,t) \in E_{l-1} \setminus \tilde{E}_{l-1}$;
  Add the candidate edges with large $\eta_{s,t}$ values into P_{l-1};
  $l = l - 1$
end while
Scale up the edge weights in P_0 and return the learned graph P.

V. Experimental Results

The proposed SGL and SF-SGL algorithms have been implemented in Matlab and C++. The test cases in this paper have been selected from a great variety of matrices that have been used in circuit simulation and finite element analysis problems. Note that the prior state-of-the-art graph learning algorithms [21] can take excessively long time even for very small graphs. For example, learning the test case "airfoil" graph with $4,253$ nodes using the CGL algorithm [21] can not be completed after 12 hours, whereas the proposed SGL algorithm only takes about 4 seconds to achieve a high-quality solution; learning the "yaleShieldBig" graph with $1,059$ nodes using CGL takes $1,235$ seconds, whereas SGL only takes about 2 seconds (over $600\times$ speedups) for achieving similar solution (spectrum preservation) quality. Therefore, in this work we will only compare with the graph construction method based on the standard kNN algorithm. All of our experiments have been conducted using a single CPU core of a computing platform running 64-bit RHEW 7.2 with a 2.67GHz 12-core CPU and 50 GB memory.

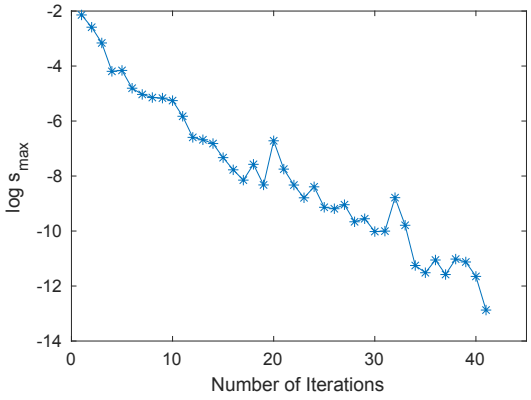


Figure 7. The decreasing maximum sensitivities ("2D mesh" graph).

A. Experimental Setup

To create the linear voltage and current measurements, the following procedure has been applied: (1) we first randomly generate M current source vectors with each element sampled from a standard normal distribution; (2) each current vector will be normalized and orthogonalized to the all-one vector; (3) M voltage vector measurements will be collected by solving the original graph Laplacian matrix with the M current vectors as the (right-hand-side) input vectors; (4) the voltage and current vectors will be stored in data matrices $X = [x_1, ..., x_M]$ and $Y = [y_1, ..., y_M] \in \mathbb{R}^{N\times M}$, respectively, which will be used as the input data of the proposed SGL and SF-SGL algorithms. By default, $M = 50$ is used for generating the voltage and current measurements. By default, we choose $k = 5$ for constructing the kNN graph for all test cases in SGL, and set $r = 5$ for constructing the spectral embedding matrix in (16). The edge sampling ratio $\beta = 10^{-3}$ has been used. The SGL iterations will be terminated if $s_{max} < tol = 10^{-12}$. When approximately computing the objective function value (3), the first 50 nonzero Laplacian eigenvalues are used.

To visualize the structure of each graph, the spectral graph drawing technique has been adopted [52]: to generate the 2D graph layouts, each entry of the first two nontrivial Laplacian eigenvectors (u_2, u_3) will be used as the x or y coordinate for embedding each node (data point), respectively. We also assign the nodes with the same color if they belong to the same node cluster determined by spectral graph clustering [46].

B. Comprehensive Results for Graph Learning with SGL

Algorithm convergence. As shown in Figure 7, for the "2D mesh" graph ($|V| = 10,000$, $|E| = 20,000$) learning task, SGL requires about 40 iterations to converge to $s_{max} \leq 10^{-12}$ when starting from an initial MST of a 5NN graph.

Comparison with kNN graph. As shown in Figure 8, for the "fe_4elt2" graph ($|V| = 11,143$, $|E| = 32,818$) learning task, SGL converges in about 90 iterations when starting from an initial MST of a 5NN graph. For the 5NN graph, we do the same spectral edge scaling. As shown in Figures 8 and 9, the SGL-learned graph achieves a more optimal objective function value and a much better spectral approximation than the 5NN graph. As observed, the SGL-learned graph has a

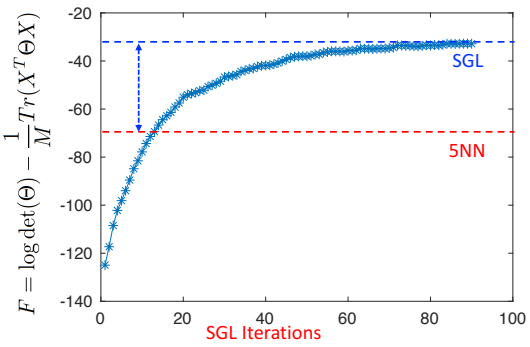


Figure 8. The objective function values ("fe_4elt2" graph).

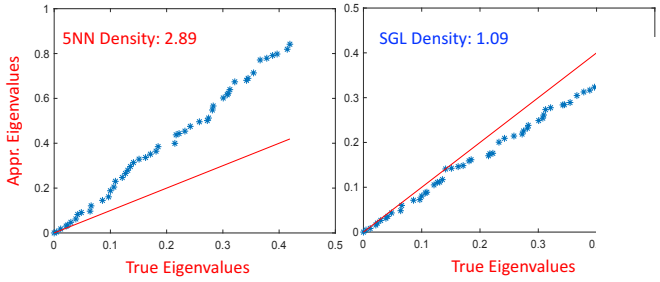


Figure 9. The comparison with a 5NN graph ("fe_4elt2" graph)

Learning circuit networks. As shown in Figures 10, 12 for the "airfoil" ($|V| = 4,253$, $|E| = 12,289$), the ' ($|V| = 10,240$, $|E| = 30,380$), and the "G2_circuit" $150,102$, $|E| = 288,286$) graphs, SGL can consistentl sparse graphs which are slightly denser than spannin while preserving the key graph spectral properties. In 13, we observe highly correlated results when comparing the effective resistances computed on the original graphs with the graphs learned by SGL.

Learning reduced networks. As shown in Figure 14, by randomly choosing a small portion of node voltage measurements (without using any currents measurements in Y matrix) for graph learning, SGL can learn spectrally-similar graphs of much smaller sizes: when 20% and 10% node voltage measurements are used for graph learning, $5\times$ and $10\times$ smaller resistor networks can be constructed, respectively, while preserving the key spectral (structural) properties of the original graph.

Learning with noisy measurements. We show the results of the "2D mesh" graph learning with noisy voltage measurements. For each SGL graph learning task, each input voltage measurement (vector) $\tilde{x}$ will be computed by: $\tilde{x} = x + \zeta \|x\|_2 \epsilon$, where ϵ denotes a normalized Gaussian noise vector, and ζ denotes the noise level. As shown in Figure 15, the increasing noise levels will result in worse approximations of the original spectral properties. It is also observed that even with a very significant noise level of $\zeta = 0.5$, the graph learned by the proposed SGL algorithm can still preserve the first few

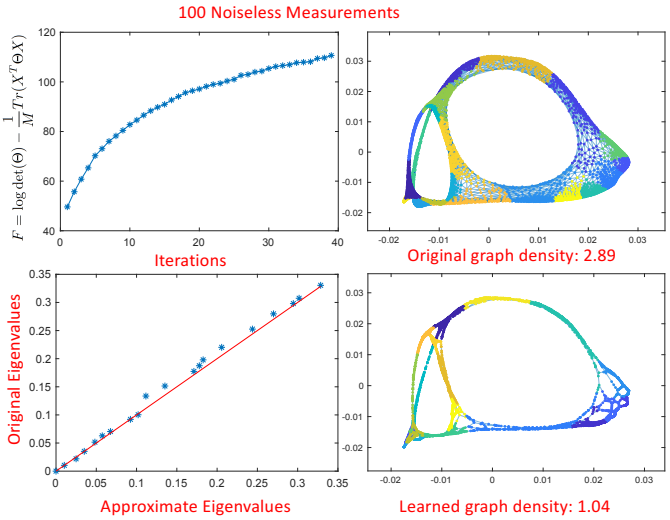


Figure 10. The results for learning the "airfoil" graph.

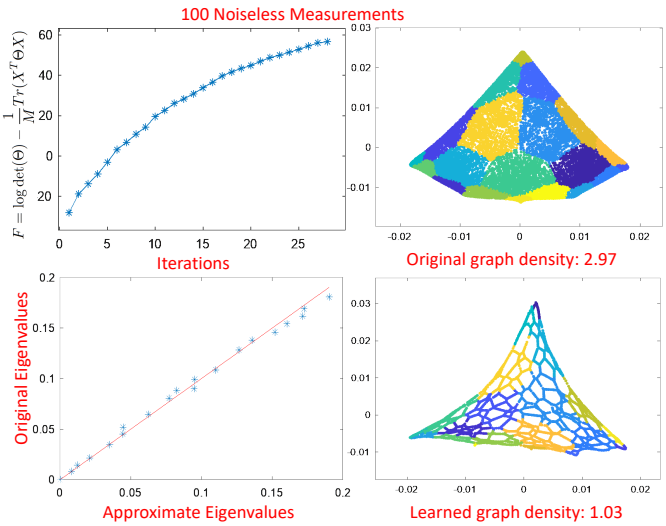
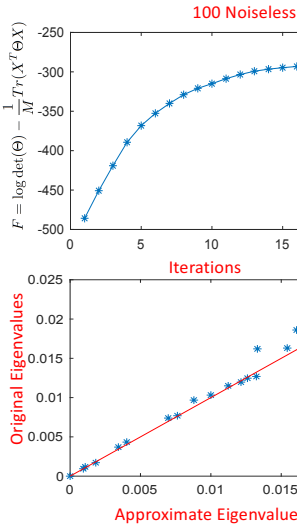


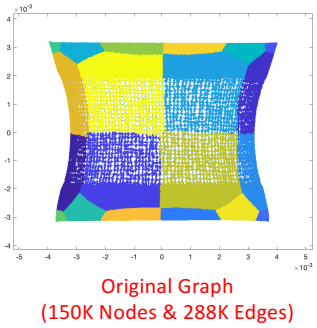
Figure 11. The results for learning the "crack" graph.

Laplacian eigenvalues that are key to the graph structural (global) properties.

Sample complexity and runtime scalability Figure 16 shows how the sample complexity (number of measurements) may impact the graph learning quality. As observed, with increasing number of samples (measurements), substantially improved approximation of the graph spectral properties can be achieved. In the last, we show the runtime scalability of the proposed SGL algorithm in Figure 17. The runtime includes the total time of Step 2 to Step 5 but does not include the time for Step 1. Note that modern kNN algorithms can achieve highly scalable runtime performance [45].

Ablation analysis of SGL. We provide comprehensive ablation analysis for the proposed SGL algorithm. In Figure 18, we show that different choices of the number of nearest neighbors for constructing kNN graphs will only slightly impact the final solution quality (objective function values) of SGL. Figure 19 shows the average relative errors of the first 50





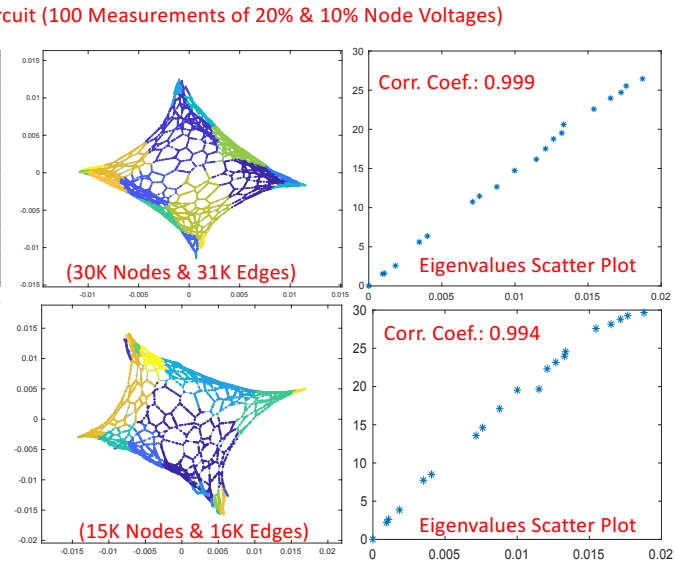


Figure 12. The results for learni

Figure 14. The reduced graphs learned by SGL ("G2_circuit").

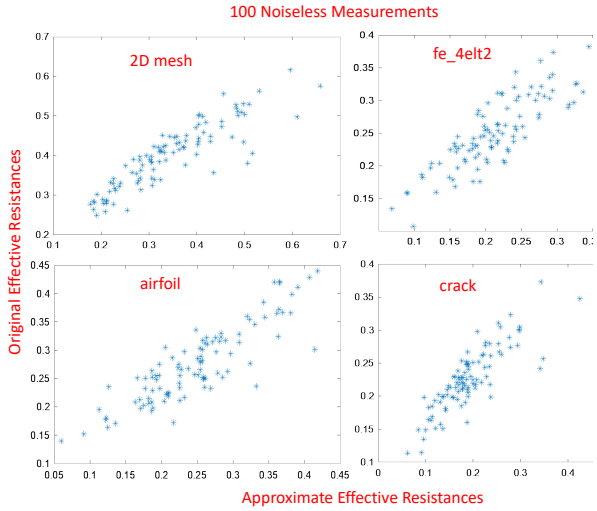
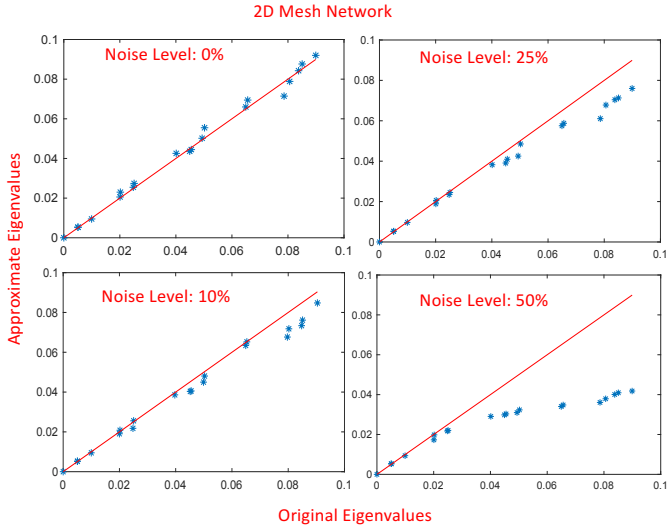


Figure 13. The effective resistances correlations (scatter plots).



eigenvalues between the learned graph and the origin with various number of the nearest neighbors k. For given the first 50 pairs of learned graph eigenvalue the original graph eigenvalues λ_i, the average relat can be computed as $\frac{1}{50}\sum_{i=1}^{50}\frac{|\tilde{\lambda}_i-\lambda_i|}{\lambda_i}$. It can be obse the learned graphs can well preserve the spectral p of the original graph with different k for initial kNN In Figure 20, it is observed that with a higher dim for spectral graph embedding, more accurate preser Laplacian eigenvalues in the learned networks can be a In Figure 21, we show that adding more edges in e iteration will substantially improve the overall graph runtime without compromising the solution quality.

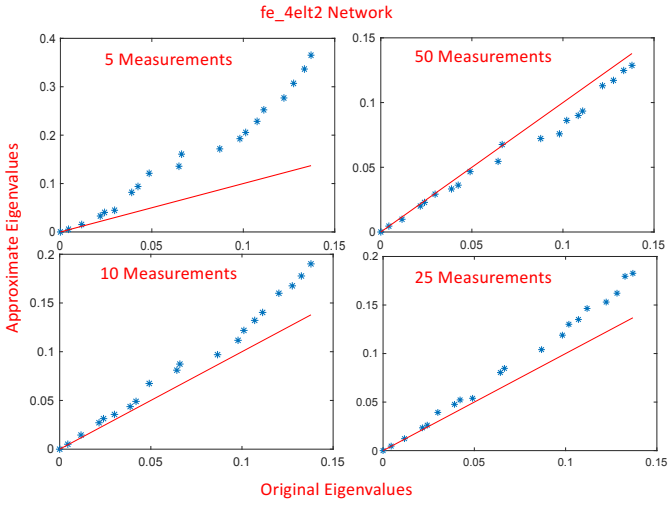
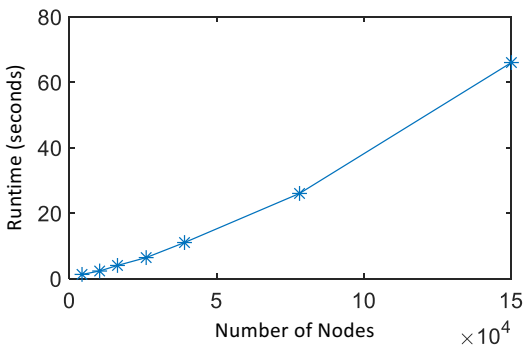


Figure 16. The effect of the number of measurements ("fe_4elt2" graph).

C. Comprehensive Results for Graph Learning with

Spectrum/resistance preservation. Figure 22 shows the learned graph comparison using the proposed SF-SGL and



crack" graph)

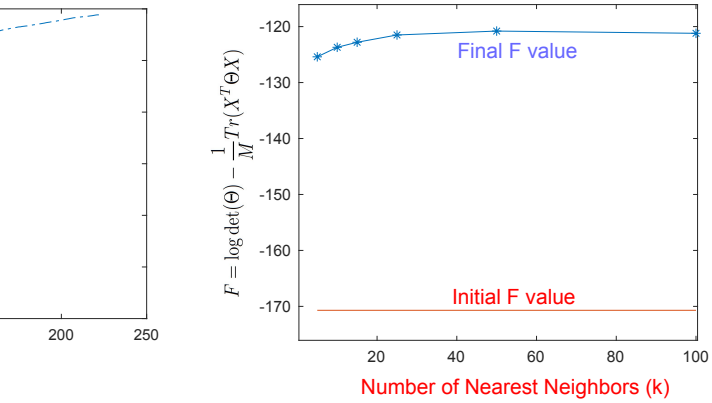
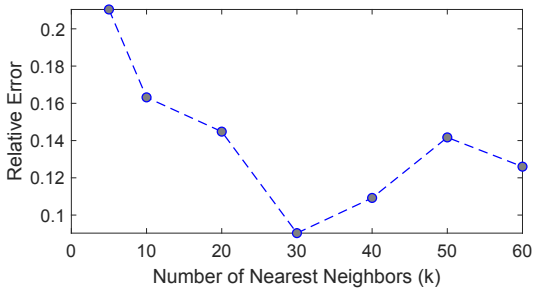


Figure 18. The final object function values achieved with various kNN graphs.



Experiment Results ("airfoil" graph)

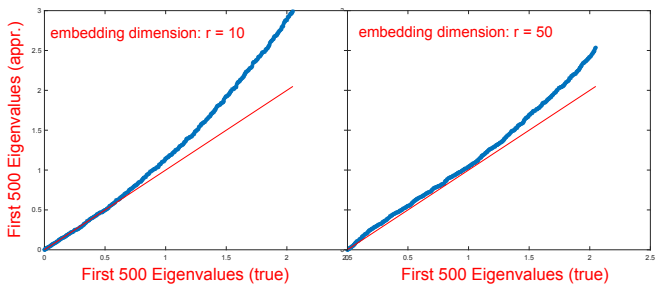


Figure 20. SGL results with different spectral embedding dimensions r.

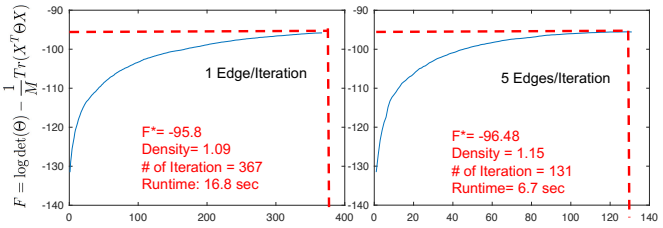


Figure 21. SGL iterations with different number of edges added per iteration.

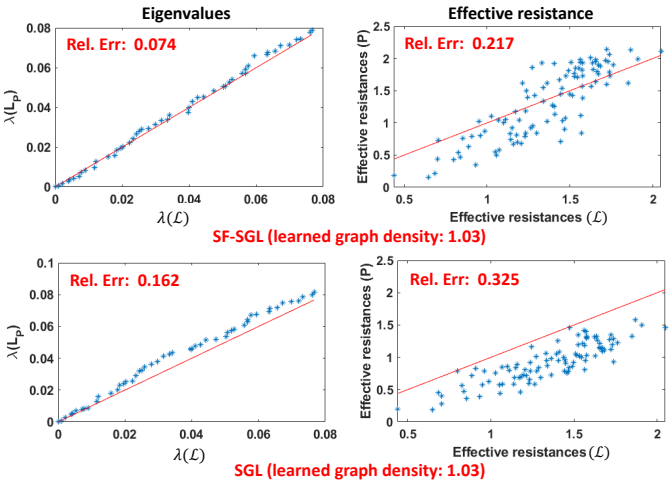


Figure 22. SF-SGL vs SGL on learned graphs ("fe_4elt").

SGL algorithms on graph "fe_4elt", where the first 50 eigenvalues are compared between the learned graphs and the original graph. Meanwhile, we randomly sample 100 pairs of the nodes to compute their effective resistances on the learned graphs and the original graph. **Rel. Err** represents the average relative errors given k pairs of the approximated solution $\tilde{a}$ and the true solution a, such that **Rel. Err** $= \frac{1}{k}\sum_{i=1}^{k}\frac{|\tilde{a}-a|}{a}$. Under this condition, k will be 50 for eigenvalue comparison and 100 for effective resistance comparison. It can be observed that the graph learned with SF-SGL can better preserve the spectral properties of the original graph.

Learning with noises. Figure 23 shows the results of the "airfoil" graph learning with noisy voltage measurements, where graph density of the learned graph is 1.04. We observe that the measurement noises will result in worse approximations of the original spectral graph properties. However, the learned graph can still well preserve the key spectral properties of the original graph even with 30% of noise level in the measurement data.

More detailed results. Table II shows more comprehensive results on learned graphs using the proposed SF-SGL and SGL algorithms, where τ_G and τ_P represent the graph densities ($|E|/|V|$) for the original graph and the learned graph, respectively; $Err(\lambda)$ denotes the average relative error of the first 50 eigenvalues between the original graph and the learned graph, and $Err(R)$ is the average relative error of the effective resistances measured on 100 pairs of randomly

Table II
COMPARISON OF SPECTRAL GRAPH LEARNING RESULTS BETWEEN SF-SGL AND SGL.

Test cases	$\|V\|$	$\|E\|$	τ_G	τ_P	SGL				SF-SGL			
					$Err(\lambda)$	$Err(R)$	T_P	T	$Err(\lambda)$	$Err(R)$	T_P	T
airfoil	$4.3E3$	$1.2E4$	2.89	1.04	0.228	0.296	$5.5s$	$7.5s$	**0.125**	**0.223**	$0.2s$ $(27X)$	$0.3s$ $(25X)$
fe_4elt	$1.1E4$	$3.3E4$	2.97	1.03	0.162	0.325	$10.6s$	$16.8s$	**0.074**	**0.217**	$0.5s$ $(21X)$	$0.7s$ $(24X)$
crack	$1.0E4$	$3.0E4$	2.97	1.04	**0.049**	0.271	$10.1s$	$15.8s$	0.176	**0.141**	$0.7s$ $(14X)$	$0.9s$ $(17X)$
fe_sphere	$1.6E4$	$4.9E4$	3.00	1.05	0.084	0.243	$19.5s$	$29.9s$	**0.078**	**0.163**	$1.3s$ $(15X)$	$1.8s$ $(16X)$
2Dmesh	$2.5E5$	$5.0E5$	2.00	1.02	0.136	0.263	$356.7s$	$2,464.7s$	**0.062**	**0.119**	$21.3s$ $(17X)$	$31.6s$ $(78X)$
3Dmesh	$2.5E5$	$7.4E5$	2.95	1.12	0.207	0.577	$1,054.7s$	$3,155.7s$	**0.125**	**0.260**	$36.8s$ $(28X)$	$47.8s$ $(66X)$
mat3	$4.0E5$	$1.2E6$	2.89	1.13	-	-	-	-	0.236	0.350	$67.7s$	$86.7s$

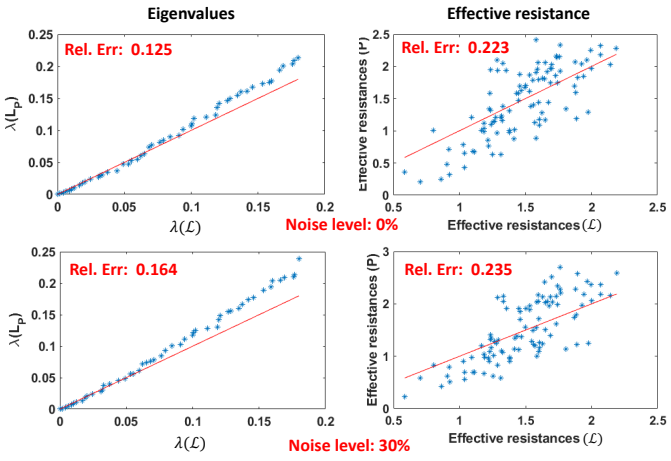


Figure 23. Learned graphs with noise (the "airfoil" graph).

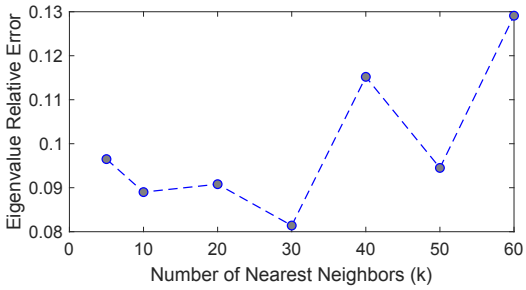


Figure 24. The average relative errors of the first 50 eigenvalues between the learned graph and the original graph with different k (the "fe_4elt" graph).

picked nodes, with better results highlighted in the table; T_P and T represent the graph learning time (without considering the initial kNN graph construction time) and total runtime of the corresponding graph learning algorithms, respectively. The SF-SGL runtime speedups over SGL are also included in the table. We observe that the graphs learned by SF-SGL can more accurately preserve the spectral properties of the original graph, while the runtime is much smaller than SGL. For example, SF-SGL successfully generate the learned graph for the "mat3" graph, while SGL failed due to the limited memory resources.

Learning with different kNN graphs. Given the original "fe_4elt" graph and the learned graph generated by SF-SGL with a graph density as 1.05, Figure 24 shows the average relative errors of the first 50 eigenvalues between learned

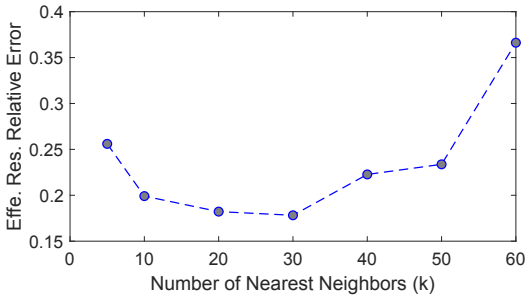


Figure 25. The average relative errors of the effective resistances (100 pairs) between the learned graph and the original graph with different k (the "fe_4elt" graph).

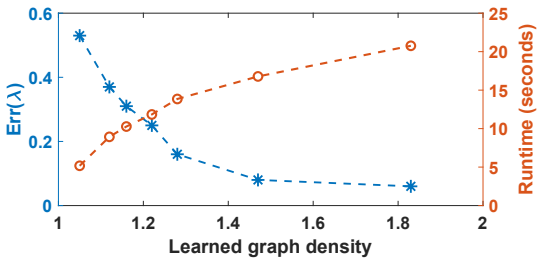


Figure 26. Learned graphs with various graph densities (the "mat1" graph).

graph and original graph with various number of the nearest neighbors k. While Figure 25 shows the average relative errors of the effective resistance values for 100 randomly picked node pairs on the learned graph and the original graph with various number of the nearest neighbors k. It can be observed that the learned graphs can well preserve the spectral properties of the original graph with various number of the nearest neighbors for initial kNN graphs.

Trade-off analysis. Figure 26 shows the tradeoffs between the eigenvalue average relative errors/runtime and graph densities learned by SF-SGL for the "mat1" graph. It can be shown that although higher graph density will result in higher runtime, it will achieve better spectrum preservation of the original graph.

Runtime scalability. Figure 27 shows the nearly-linear runtime scalability of the proposed SF-SGL algorithm even for very large graphs.

D. Data-Driven Vectorless Integrity Verification

The integrity of power distribution networks must be verified throughout the design process to ensure that the

Table III
APPLICATION OF SF-SGL IN VECTORLESS INTEGRITY VERIFICATION TASKS.

Test cases	$\|V\|$	$\|E\|$	Original graph				Learned graph					
			τ_G	$T(chol)$	$T(sol)$	$T(lp)$	τ_P	T	$T(chol)$	$T(sol)$	$T(lp)$	Err
ibm4	$9.5E5$	$1.6E6$	1.63	$11.86s$	$52.82s$	$171.60s$	1.02	$237.94s$	$1.19s$	$3.00s$	$162.57s$	0.01%
thu1	$5.0E6$	$8.2E6$	1.66	$134.91s$	$618.52s$	$1,469.92s$	1.03	$1,338.41s$	$7.70s$	$16.03s$	$1,025.85s$	0.01%
mat1	$1.0E5$	$2.8E5$	2.88	$3.75s$	$15.66s$	$14.07s$	1.09	$19.09s$	$0.25s$	$0.56s$	$12.62s$	0.84%
mat2	$2.0E5$	$5.8E5$	2.93	$16.60s$	$56.77s$	$23.82s$	1.07	$43.63s$	$0.59s$	$1.15s$	$23.07s$	0.99%
mat3	$4.0E5$	$1.2E6$	2.89	$27.29s$	$89.04s$	$69.21s$	1.07	$90.19s$	$1.21s$	$2.15s$	$79.90s$	0.45%
mat4	$9.0E5$	$2.6E6$	2.89	$169.40s$	$333.96s$	$531.38s$	1.04	$235.79s$	$3.28s$	$6.91s$	$196.98s$	0.10%
mat5	$1.6E6$	$4.6E6$	2.90	$1,031.42s$	$38,062.90s$	$1623.62s$	1.04	$511.16s$	$9.01s$	$9.99s$	$379.10s$	0.46%

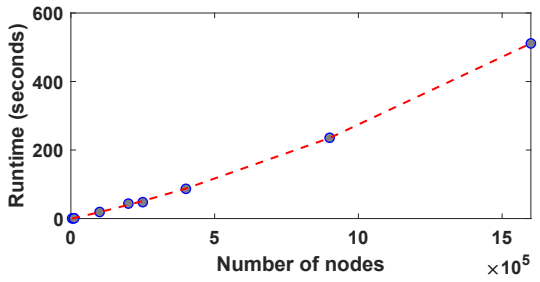


Figure 27. The runtime scalability of the SF-SGL algorithm.

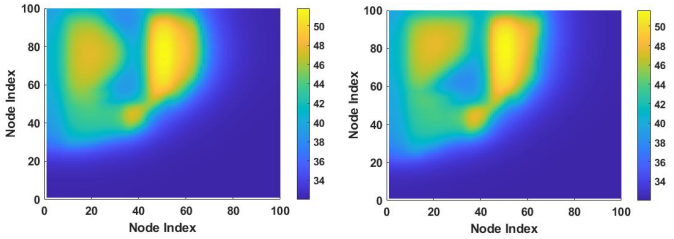


Figure 28. Worst-case temperature distributions obtained by vectorless verification with the original thermal grid (left) and the learned grid (right).

supply voltage fluctuations are within certain thresholds. To achieve the desired levels of chip reliability and functionality, compute-intensive full-chip thermal analysis and integrity verifications are indispensable, which typically involves estimating thermal profiles under a variety of workloads and power budgets. In this work, we introduce a data-driven vectorless power/thermal integrity verification framework: **(1)** given a collection of voltage/temperature measurements that can be potentially obtained from on-chip voltage/temperature sensors [32], [33], the proposed data-driven method will first construct a sparse power/thermal grid network leveraging the proposed graph topology learning approach; **(2)** next, vectorless power/thermal integrity verification framework will be exploited for estimating the worst-case voltage/temperature (gradient) distributions [53], [54].

Table III shows the single-level vectorless integrity verification results between learned graphs and original graphs, where "ibm4" and "thu1" are power grid test cases, "mat1" to "mat5" are 3D thermal grids; T, $T(chol)$, $T(sol)$ and $T(lp)$ represent the runtime for SF-SGL, Cholesky factorization, adjoint sensitivity calculation using matrix factors and the linear programming (LP) computation, respectively. $T(sol)$ and $T(lp)$ are runtime results computed by summing up the runtime for verifying 100 randomly chosen nodes. Err denotes the average relative solution error compared to the ones obtained with the original graph. Compared with the verification results on original graphs, the SF-SGL learned graphs have achieved the very similar results with much lower overall runtime.

Figure 28 shows the worst-case temperature distributions obtained by vectorless verification using the original thermal grid (left) and the learned grid (right) for the "mat1" graph. It can be observed that very similar worst-case thermal profiles can be obtained using the SF-SGL learned graph.

VI. CONCLUSIONS

This work proposes highly-scalable spectral algorithms for learning large resistor networks from linear voltage and current measurements. We show that the proposed graph learning approach is equivalent to solving the classical graphical Lasso problems with Laplacian-like precision matrices. A unique feature of the proposed methods is that the learned graphs will have the spectral embedding or effective-resistance distances to encode the similarities between the original input data points (voltage measurements). As an important extension, we also introduce a more scalable solver-free spectral graph learning (SF-SGL) algorithm. Such a scalable framework allows learning a hierarchy of spectrally-reduced and sparsified graphs in nearly-linear time, which can become key to accelerating many graph-based numerical computing tasks. The proposed spectral approach is simple to implement and inherently parallel friendly. The proposed spectral algorithms allow each iteration to be completed in $O(N \log N)$ time, whereas existing state-of-the-art methods require at least $O(N^2)$ time for each iteration. We also provide a sample complexity analysis to show that it is possible to accurately recover a resistor network with only $O(\log N)$ voltage measurements (vectors). Our extensive experimental results show that the proposed method can produce a hierarchy of high-quality learned graphs in nearly-linear time for a variety of real-world, large-scale graphs and circuit networks when compared with prior state-of-the-art spectral methods.

VII. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grants CCF-2021309, CCF-2011412, CCF-2212370, and CCF-2205572.

REFERENCES

[1] William L Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin.*, 2017.

[2] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *Int'l Conf. on Learning Representations (ICLR)*, 2017.

[3] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[4] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *Int'l Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 974–983, 2018.

[5] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[6] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[7] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[8] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[9] Prakash Chandra Rathi, R Frederick Ludlow, and Marcel L Verdonk. Practical high-quality electrostatic potential surfaces for drug discovery using a graph-convolutional deep neural network. *Journal of Medicinal Chemistry*, 2019.

[10] Jaechang Lim, Seongok Ryu, Kyubyong Park, Yo Joong Choe, Jiyeon Ham, and Woo Youn Kim. Predicting drug–target interaction using a novel graph neural network with 3d structure-embedded graph representation. *Journal of chemical information and modeling*, 59(9):3981–3988, 2019.

[11] Filipe de Avila Belbute-Peres, Thomas Economon, and Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, pages 2402–2411. PMLR, 2020.

[12] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33, 2020.

[13] Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. In *International Conference on Learning Representations*, 2020.

[14] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu. High Performance Graph Convolutional Networks with Applications in Testability Analysis. *Design Automation Conf. (DAC)*, 2019.

[15] Guo Zhang, Hao He, and Dina Katabi. Circuit-gnn: Graph neural networks for distributed circuit design. In *International Conference on Machine Learning*, pages 7364–7373. PMLR, 2019.

[16] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[17] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Rrepresentation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, 2017.

[19] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. GraphZoom: A Multi-level Spectral Approach for Accurate and Scalable Graph Embedding. In *International Conference on Learning Representations*, 2019.

[20] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[21] Hilmi E Egilmez, Eduardo Pavez, and Antonio Ortega. Graph learning from data under laplacian and structural constraints. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):825–841, 2017.

[22] Xiaowen Dong, Dorina Thanou, Michael Rabbat, and Pascal Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63, 2019.

[23] Sandeep Anil Kumar, Jiaxi Ying, et al. Structured graph learning via laplacian spectral constraints. *Advances in Neural Information Processing Systems 32 (Nips 2019)*, 2019.

[24] Harsh Shrivastava, Xinshi Chen, Binghong Chen, Guanghui Lan, Srinivas Aluru, Han Liu, and Le Song. Glad: Learning sparse graph recovery. In *International Conference on Learning Representations*, 2019.

[25] Xingyue Pu, Tianyue Cao, Xiaoyun Zhang, Xiaowen Dong, and Siheng Chen. Learning to learn graph topologies. *Advances in Neural Information Processing Systems*, 34, 2021.

[26] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[27] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, 2021.

[28] Zhuo Feng. SGL: Spectral Graph Learning from Measurements. *in IEEE/ACM Design Automation Conference (DAC)*, 2021.

[29] Daniel Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[30] Zhuo Feng. Grass: Graph spectral sparsification leveraging scalable spectral perturbation analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4944–4957, 2020.

[31] Ching-Che Chung and Cheng-Ruei Yang. An autocalibrated all-digital temperature sensor for on-chip thermal monitoring. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(2):105–109, 2011.

[32] Md Toufiq Hasan Anik, Mohammad Ebrahimabadi, Hamed Pirsiavash, Jean-Luc Danger, Sylvain Guilley, and Naghmeh Karimi. On-chip voltage and temperature digital sensor for security, reliability, and portability. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 506–509. IEEE, 2020.

[33] Chia-Yuan Ku and Tsung-Te Liu. A voltage-scalable low-power all-digital temperature sensor for on-chip thermal monitoring. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66(10):1658–1662, 2019.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23):6160–6173, 2016.

[36] Vassilis Kalofolias. How to learn a graph from smooth signals. In *Artificial Intelligence and Statistics*, pages 920–929, 2016.

[37] Vassilis Kalofolias and Nathanaël Perraudin. Large scale graph learning from smooth signals. *International Conference on Learning Representations (ICLR 2019)*, 2019.

[38] Brenden Lake and Joshua Tenenbaum. Discovering structure by learning sparse graphs. 2010.

[39] Martin Slawski and Matthias Hein. Estimation of positive definite m-matrices and structure learning for attractive gaussian markov random fields. *Linear Algebra and its Applications*, 473:145–179, 2015.

[40] Ryan J Tibshirani. A general framework for fast stagewise algorithms. *J. Mach. Learn. Res.*, 16(1):2543–2588, 2015.

[41] Zhiqiang Zhao and Zhuo Feng. Effective-resistance preserving spectral reduction of graphs. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 109:1–109:6, New York, NY, USA, 2019. ACM.

[42] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[43] CJ Carey. Graph construction for manifold discovery. 2017.

[44] Martin Imre, Jun Tao, Yongyu Wang, Zhiqiang Zhao, Zhuo Feng, and Chaoli Wang. Spectrum-preserving sparsification for visualization of big graphs. *Computers & Graphics*, 87:89–102, 2020.

[45] Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[46] Zhiqiang Zhao, Ying Zhang, and Zhuo Feng. Towards scalable spectral embedding and data visualization via spectral coarsening. *Proceedings of ACM International Conference on Web Search and Data Mining (WSDM)*, 2021.

[47] I. Koutis, G. Miller, and R. Peng. Approaching Optimality for Solving SDD Linear Systems. In *Proc. IEEE FOCS*, pages 235–244, 2010.

[48] Zhiqiang Zhao, Yongyu Wang, and Zhuo Feng. SAMG: Sparsified Graph Theoretic Algebraic Multigrid for Solving Large Symmetric Diagonally Dominant (SDD) Matrices. In *Proceedings of the 36th International Conference on Computer-Aided Design (ICCAD)*. ACM, 2017.

[49] Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

[50] O. Livne and A. Brandt. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

[51] Ying Zhang, Zhiqiang Zhao, and Zhuo Feng. Sf-grass: Solver-free graph spectral sparsification. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–8. IEEE, 2020.

[52] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

[53] Zhiqiang Zhao and Zhuo Feng. A spectral approach to scalable vectorless thermal integrity verification. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 412–417. IEEE, 2020.

[54] Zhiqiang Zhao and Zhuo Feng. A spectral graph sparsification approach to scalable vectorless power grid integrity verification. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.