Efficient Spectral-Aware Power Supply Noise Analysis for Low-Power Design Verification

Yinuo Bai*, Xiaoyu Yang*, Yicheng Lu*, Dan Niu†, Cheng Zhuo‡, Zhou Jin* and Weifeng Liu*
*Super Scientific Software Laboratory, China University of Petroleum-Beijing, China
†School of Automation, Southeast University, China
‡College of Information Science and Electronic Engineering, Zhejiang University, China
Email: *{bai, yangxiaoyu}@student.cup.edu.cn, *luyicheng_cup@163.com, †danniu2013@seu.edu.cn,
‡czhuo@zju.edu.cn, *{jinzhou,weifeng.liu}@cup.edu.cn

***Abstract*—The relentless pursuit of energy-efficient electronic devices necessitates advanced methodologies for low-power design verification, with a particular focus on mitigating power supply noise. The challenges posed by shrinking voltage margins in low-power designs lead to a significant demand for rapid and accurate power supply noise simulation and verification techniques. Too large supply noise inevitably results in the raise of supply level, thereby hurting the lower power design target. Spectral methods have demonstrated as a great alternative to produce a sparse sub-matrix with spectral-similarity property as the preconditioner to efficiently reduce the iteration number and solve the linear system for supply noise verification. However, existing methods either suffer from high computational complexity or rely on approximations to reduce computational time. Therefore, a novel approach is needed to efficiently generate high-quality preconditioners. In this paper, we propose a two-stage spectral-aware algorithm to address these challenges. Our approach has three main highlights. Firstly, by introducing spectral-aware weights, we can better assess the priority of edges and construct high-quality spanning trees with the minimum relative condition number. Secondly, by leveraging eigenvalue transformation strategies, we can quickly and accurately recover off-tree edges that are spectrally critical, avoiding time-consuming iterative computations. Thirdly, we proposed a fast computation method to further decrease the computational complexity of the effective resistance. Compared with two SOTA methods, GRASS and feGRASS, our approach demonstrates higher accuracy and efficiency in preconditioner generation (37.3x and 2.13x speedup, respectively) as well as significant improvements in accelerating the linear solver for power supply noise analysis in power grid simulation and other Laplacian graphs (5.16x and 1.70x speedup, respectively).**

***Index Terms*—Low power design verification, power supply noise analysis, power grid simulation, spectral graph sparsification, iterative solver**

I. Introduction

Due to scaling supply voltage and increasing current density, the narrowing gap between operating and threshold voltages makes circuit performance more susceptible to supply voltage fluctuations. Power supply noise (PSN) refers to undesirable voltage variations in the power supply. Accurate analysis is crucial in integrated circuit design as it helps understand the impact of voltage fluctuations on timing, signal integrity, and system reliability. Power supply noise analysis guides decisions on power distribution network design, decoupling capacitor placement, and noise mitigation strategies for improved circuit performance and reliability [1]–[3].

To analyze power supply noise, a series of linear equations is established using the modified nodal analysis (MNA) method. With ongoing semiconductor scaling and increasing device integration, the size of the matrix has expanded to billions of entries, presenting significant computational challenges for efficient and accurate power supply noise analysis. To enhance the efficiency of iterative solvers used for solving large linear systems [4]–[6], preconditioners are often employed. The Preconditioned Conjugate Gradient (PCG) method is a widely used example [7]. It significantly reduces the time required to solve large-scale linear systems, improving simulation efficiency in the analysis process.

Recently, the spectral graph sparsification has undergone significant development [8], [9] and shown its great potential in accelerating power supply noise analysis. The spectral graph sparsification, aiming to produce an ultra-sparse subgraph while maintaining spectral similarity to the original graph, could be leveraged as an effective method to generate high-quality preconditioners due to the similarity between the Laplacian matrices of graphs and the coefficient matrices of resistance circuit equations. The GRASS algorithm, introduced in [10], leverages approximate dominant generalized eigenvectors to effectively identify and recover spectrally critical off-tree edges. However, it heavily relies on the frequent computation of highly dimensional matrices' pseudoinverse, which can pose computational challenges, especially for large-scale matrices. To address this issue, alternative approaches have been proposed in [11] and [12], e.g., feGRASS, which focuses on effective edge weight and spectral edge similarity and exhibits remarkable efficiency by generating comparable preconditioners in significantly less time compared to GRASS. However, excessive approximations introduced during the analysis of trace reduction resulted in slightly inferior performance of the preconditioner.

Despite the quick advancements in spectral-based methods for fast power supply noise analysis, existing works are either highly intensive to ensure the effectiveness of preconditioners or introduce extra approximations to reduce the generation time. These two challenges remain the main obstacles for the applications of spectral-based solvers. In this paper, to simultaneously address both challenges, we propose a novel two-stage spectral graph sparsification method involving the construction of the

Maximum Spectral-Aware Weight Spanning Tree (MSWST) and the recovery of spectrally critical off-tree edges based on eigenvalue transformation. The proposed method achieves an average optimization of 37.30x and 2.13x in computation time, as well as 1.12x and 1.25x in the relative condition number compared with GRASS and feGRASS, respectively. Our major contributions are summarized as follows:

1) We propose the spectral-aware weight, which reweights edges based on the volume and distance of their endpoints. The MSWST aims to achieve a lower relative condition number.
2) We propose an eigenvalue transformation-based strategy for recovering off-tree edges. By converting the generalized eigenvalues into their reciprocals, we observe that analyzing the reciprocals uncovers a more concise, precise, and illuminating quantitative relationship between edge recovery and reduction in relative condition number.
3) We introduce a novel, rapid computation method of effective resistances. This method exploits an empirical connection between the quadratic forms, in the context, of the Laplacian matrix and its pseudoinverse. This connection will assist us in avoiding frequent calculations of high-dimensional matrices' pseudoinverse.

II. Background

A. Laplacian Matrix and Linear Circuit Equation

For a connected, weighted, undirected graph $G = (V, E, w)$, where V and E represent the sets of vertices and edges, respectively, the Laplacian matrix L_G is constructed as:

$$L_G = \sum_{(i,j)\in E} w_{i,j} g_{i,j} g_{i,j}^T \tag{1}$$

Here, $w_{i,j}$ is the weight of the edge connecting vertices i and j, and $g_{i,j} = g_i - g_j$ with g_i being the i-th column of the identity matrix.

The Laplacian matrices are similar to the coefficient matrices of the resistance circuit equations. The latter can be obtained by excluding a specific row and column from the corresponding Laplacian matrices. These linear circuit equations can be memory-efficiently solved on a large scale using iterative solvers like PCG [4], [13].

B. Preconditioner for the Iterative Solver

The quality of preconditioners heavily impacts the fast convergence of the PCG method. A commonly used metric to quantify preconditioner quality is the relative condition number between the preconditioner and the original matrix.

Given arbitrary matrices A and B, their relative condition number can be defined as follows:

$$\kappa(A, B) = \frac{\lambda_{\max}(A, B)}{\lambda_{\min}(A, B)} \tag{2}$$

Here, $\lambda(A, B)$ denotes the generalized eigenvalue between A and B. The corresponding generalized eigenvector $u \in \mathbf{R}^{n\times 1}$ bridges A and B through the following equation:

$$Au = \lambda Bu \tag{3}$$

The iteration complexity of PCG can be described by the following expression, where A is the original matrix, B is the preconditioner, and ϵ represents the desired accuracy of the solution [7]:

$$O(\kappa(A, B)^{\frac{1}{2}} \log \left[\frac{1}{\epsilon}\right]) \tag{4}$$

This expression signifies that the number of iterations required for convergence at a specific accuracy grows with the square root of the relative condition number.

Therefore, one of the most crucial objectives in achieving rapid convergence with a given level of accuracy is to generate a preconditioner with a smaller condition number $\kappa(A, B)$.

C. The Spanning Tree to Accelerate Iterative Solver

Recently, spectral graph sparsification has emerged as a powerful approach for generating preconditioners. Among subgraphs, spanning trees, particularly the least stretch spanning tree ($LSST$), have shown remarkable performance.

Let's consider a spanning tree $T = (V, E', w')$ of a graph G with $|E'| = |V| - 1$. L_G and L_T represent the Laplacian matrices of the original graph and its spanning tree, respectively. The stretch of the shortest path S connecting two randomly chosen elements from V is defined as follows:

$$\text{Stretch}(S) = w_s \sum_{e_i \in S} \frac{1}{w_{e_i}} \tag{5}$$

where w_{e_i} and w_s respectively stand for the weight of the sole edge and total edges in the path. The average stretch $AveStretch(T)$ of all such paths S_i in T can be defined as:

$$AveStretch(T) = \frac{1}{|E|} \sum_{i=1}^{C(|V|,2)} Stretch(S_i) \tag{6}$$

where $C(m, n)$ represents the combination number. It has been proven that [10]:

$$TotStretch(T) = |E| AveStretch(T) = Tr(L_T^+ L_G) \tag{7}$$

where $\text{Tr}(L_T^+ L_G)$ denotes the trace of $L_T^+ L_G$. The $LSST$ is the spanning tree that minimizes $TotStretch(T)$ among all spanning trees.

Moreover, it is important to note that the generalized eigenvalues and eigenvectors between L_G and L_T are equivalent to ordinary those of $L_T^+ L_G$. Additionally, it can be observed that $\lambda_{\min}(L_G, L_P)$ is always greater than 1. Based on these facts, we can derive the following inequality:

$$\begin{aligned} \kappa(L_G, L_T) &\le \lambda_{\max}(L_G, L_T) \\ &< \sum_{i}^{n} \lambda_i(L_G, L_T) = \text{Tr}(L_T^+ L_G) \end{aligned} \tag{8}$$

Eq. (7) and (8) explain why the $LSST$ was chosen as a preconditioner to achieve preliminary success. The lower the $TotStretch$, the lower the upper boundary of κ. The $LSST$ holds the minimum $TotStretch$.

Furthermore, in subsequent experiments, more complex subgraphs formed by recovering a few off-tree edges in spanning trees have shown even better performance. These subgraphs

can further reduce κ, leading to a more significant drop in the number of iterations. The construction of spanning trees and the recovery of off-tree edges are identified as the two fundamental steps in spectral graph sparsification.

Many previous approaches have attempted to efficiently construct the preconditioner with a minimum κ in the steps mentioned above. However, there is still room for improvement. Limited consideration of factors such as volume and distance, which can significantly affect the condition number, is one deficiency. The other one is the loss of effectiveness when loads of approximations are made to avoid computationally intensive calculations [11], [14], such as the computation of the pseudo-inverse and quadratic form of large-scale matrices.

III. Proposed Preconditioner Generator for power supply noise analysis

In this section, we propose a new spectral sparsification method to overcome the limitations of existing approaches and provide a more effective method for preconditioner construction in large-scale linear systems, which includes the use of a spectral-aware weight to produce a better spanning tree and an eigenvalue-transformation-based strategy to find and recover the most spectral-critical off-tree edges. Additionally, we further reduce the time complexity of effective resistance computation based on the Rayleigh-Ritz theorem [15].

A. Spectral-Aware Weight to Produce Spanning Trees

The construction of $LSST$ primarily emphasizes the influence of its elements and structure on $Tr(L_T^+ L_G)$. This suggests the potential existence of superior spanning tree candidates that consider more precise boundaries. Accordingly, we propose a novel method to construct such trees.

To achieve this objective, we perform normalization on the vector u and substitute A and B in Eq. (3) with L_G and L_T respectively. By left-multiplying both sides of Eq. (3) with u^T, we obtain the following equation:

$$u^T L_G u = \lambda u^T L_T u \tag{9}$$

$$\lambda = \frac{u^T L_G u}{u^T L_T u} \tag{10}$$

Consequently, for $\lambda_{max}(L_G, L_T)$, based on Rayleigh-Ritz theorem [15], we have the following equality:

$$\lambda_{max} = \max \frac{u^T L_G u}{u^T L_T u} \geq \max \frac{g_i^T L_G g_i}{g_i^T L_T g_i} \tag{11}$$

In this context, the term $g_i^T L_G g_i$ represents the sum of edge weights connecting vertex i in T, which defines the vertex volume. For convenience, we define the maximum volume mismatch (MVM) as $\max \frac{g_i^T L_G g_i}{g_i^T L_T g_i}$, indicating the largest disparity in volumes between G and T. Minimizing MVM proves to be a more effective approach in reducing $\kappa(L_G, L_T)$, as illustrated in Eq. (11).

Thus, to minimize $\kappa(L_G, L_T)$, we prioritize edges connected to vertices with higher volume within G. Moreover, it is advantageous to integrate the fundamental principle of $LSST$ construction, namely the centralized structure, into the construction of the new spanning tree.

Building upon the preceding analysis, we introduce a novel type of edge weight:

$$W_{spec} = w_{i,j} \times \frac{\max(deg(i), deg(j))}{log(dist(i,r) + dist(j,r) + 1)} \tag{12}$$

In this definition, $w_{i,j}$ represents the weight of the edge, $deg(i)$ and $deg(j)$ denote the degrees of its endpoints, and $dist(i,r)$ and $dist(j,r)$ indicate the distances from endpoints to the root vertex r. Typically, the vertex with the highest volume is designated as the root vertex. This index is referred to as the edge's spectral-aware weight.

The spectral-aware weight of an edge is determined by its weight, the degrees of its endpoints, and the distances from its endpoints to the root vertex. This definition ensures that edges connecting higher-degree vertices or those closer to the root vertex are given higher weight. Therefore, when attempting to construct a MSWST, these edges are more likely to be retained within the spanning tree. The algorithm is illustrated in Algorithm 1.

Algorithm 1 Construction of the Maximum Spectral-Aware Weight Spanning Tree (MSWST)

Input: Undirected, connected, weighted graph $G = (V, E, w)$.
Output: The MSWST T

1: Find the vertex r with the largest volume in V.
2: Run BFS search from r to obtain the no-weighted distance between it and other points.
3: Compute the spectral-aware weight of all edges in E according to (12), and obtain graph $G' = (V, E, w_{spec})$.
4: Run Kruskal algorithm on G' to obtain the MSWST T.

Furthermore, it is important to note that we employ different operations for distance and volume. This distinction arises from their influences on different boundaries of κ. The volume has a crucial influence on minimizing MVM, which is a more precise boundary. Therefore, it is essential and justified to utilize a linear operation with a higher derivative for it.

B. Eigenvalue Transformation to Recover Edges

After constructing the MSWST, the key point is to select a small number of spectral-critical off-tree edges and recover them in the spanning tree. This step aims to generate the final output subgraph, denoted as P, with further decreased $\kappa(L_G, L_P)$.

However, accurately determining the impact of removing or adding edges on the variation of $\kappa(L_G, L_T)$ or $\lambda_{max}(L_G, L_P)$ poses a significant challenge. In this subsection, we present a novel method utilizing eigenvalue transformation to elucidate the relationship between edge recovery and $\kappa(L_G, L_P)$.

By referencing Eq. (3) and dividing both sides by λ, we can derive the following equation:

$$Bu = \frac{1}{\lambda} Au \tag{13}$$

By replacing A and B with L_G and L_P, respectively, the equation could reveal the relationship between $\lambda(L_G, L_P)$ and $\lambda(L_P, L_G)$:

$$\lambda(L_G, L_P) = \frac{1}{\lambda(L_P, L_G)} \tag{14}$$

With this new understanding, we can review Eq. (2) as follows:

$$\kappa(L_G, L_P) = \frac{\frac{1}{\lambda_{min}(L_P, L_G)}}{\frac{1}{\lambda_{max}(L_P, L_G)}} \tag{15}$$

$$= \frac{\lambda_{max}(L_P, L_G)}{\lambda_{min}(L_P, L_G)} \tag{16}$$

$$\leq \frac{1}{\lambda_{min}(L_P, L_G)} \tag{17}$$

Eq. (15) to (17) establish a new upper bound for $\kappa(L_G, L_P)$. Therefore, through eigenvalue transformation, we can analyze the impact on $\kappa(L_G, L_P)$ from relatively computationally tractable $\lambda(L_P, L_G)$ instead of $\lambda(L_G, L_P)$. Meanwhile, $\lambda_{min}(L_P, L_G)$ obeys a similar inequality:

$$\lambda_{min}(L_P, L_G) < \sum_{i=1}^{n} \lambda_i(L_P, L_G) = Tr(L_G^+ L_P) \tag{18}$$

Thus, if the impact of edge recovery on the $Tr(L_G^+ L_P)$ can be determined, we can indirectly infer its influence on $\lambda_{min}(L_P, L_G)$. Assuming that P becomes P' after recovering an edge $w_{i,j} g_{i,j} g_{i,j}^T$, we have:

$$Tr(L_G^+ L_{P'}) = Tr(L_G^+ (L_P + w_{i,j} g_{i,j} g_{i,j}^T)) \tag{19}$$

$$= Tr(L_G^+ L_P) + w_{i,j} Tr(L_G^+ g_{i,j} g_{i,j}^T) \tag{20}$$

Due to $Tr(AB) = Tr(BA)$:

$$Tr(L_G^+ L_{P'}) = Tr(L_G^+ L_P) + w_{i,j} Tr(g_{i,j}^T L_G^+ g_{i,j}) \tag{21}$$

$$= Tr(L_G^+ L_P) + w_{i,j} g_{i,j}^T L_G^+ g_{i,j} \tag{22}$$

Since L_G^+ is a semi-positive definite matrix and the edge weight $w_{i,j}$ is positive, the term $w_{i,j} g_{i,j}^T L_G^+ g_{i,j}$ is positive. Therefore, recovering a single edge can increase $Tr(L_G^+ L_P)$, leading to an increase in $\lambda_{min}(L_P, L_G)$ and ultimately a decrease in $\kappa(L_G, L_P)$, as stated in Eq. (17).

Algorithm 2 Preconditioner Generator for Iterative Solvers

Input: Connected, weighted, undirected graph $G = (V, E, w)$.
Output: Ultra-sparse subgraph P, which preserve the spectral characteristics of the original graph G.

1: Run algorithm 1 to get MSWST T.
2: Compute $w_{i,j} g_{i,j}^T L_G g_{i,j}$ of all off-tree edges in E.
3: Sort all off-tree edges in descending order according to $w_{i,j} g_{i,j}^T L_G g_{i,j}$.
4: Recover a small number of off-tree edges in T to obtain ultra-sparse subgraph P.

Therefore, we can compute the value of $w_{i,j} g_{i,j}^T L_G^+ g_{i,j}$ for each edge and recover a limited number of edges in descending order. The recovery allows us to generate the subgraph with a lower $\kappa(L_G, L_P)$. The whole algorithm is illustrated in Algorithm 2.

C. Further Reduce the Computing Complexity

Computing L_G^+ accurately remains a challenge. To address this issue, we utilize the Rayleigh-Ritz theorem [15]. In this subsection, we carefully add the same slight positive number to each diagonal element in the Laplacian matrix, which transforms it into a full-rank matrix.

Theorem I: With $g_{i,j}$ defined in Eq. (1), the quadratic forms of L_G and L_G^+ are related as follows:

$$(g_{i,j}^T L_G g_{i,j})(g_{i,j}^T L_G^+ g_{i,j}) \approx 4 \tag{23}$$

Proof: Based on the Rayleigh-Ritz theorem, we have:

$$\begin{aligned} \lambda_{\min}(L_G) < \frac{g_{i,j}^T L_G g_{i,j}}{g_{i,j}^T g_{i,j}} < \lambda_{\max}(L_G) \\ \lambda_{\min}(L_G^+) < \frac{g_{i,j}^T L_G^+ g_{i,j}}{g_{i,j}^T g_{i,j}} < \lambda_{\max}(L_G^+) \end{aligned} \tag{24}$$

It is worth noting that:

$$\begin{aligned} \lambda_{\max}(L_G)\lambda_{\min}(L_G^+) = 1 \\ \lambda_{\min}(L_G)\lambda_{\max}(L_G^+) = 1 \end{aligned} \tag{25}$$

$$g_{i,j}^T g_{i,j} = 2 \tag{26}$$

Therefore, we have the following inequality:

$$4\frac{\lambda_{\min}(L_G)}{\lambda_{\max}(L_G)} < (g_{i,j}^T L_G g_{i,j})(g_{i,j}^T L_G^+ g_{i,j}) < 4\frac{\lambda_{\max}(L_G^+)}{\lambda_{\min}(L_G)} \tag{27}$$

In fact, for a nonsingular graph, the spectral interval of its Laplacian matrix is very small, which leads to the approximation in Eq.(23). That ends the proof.

This theorem establishes the inverse proportionality between two quadratic forms under certain conditions. This property aids in computing our algorithm without L_G^+ by solely calculating $g_{i,j}^T L_G g_{i,j}$ for each edge and recovering them in ascending order.

Another unresolved and challenging problem is the reduction of time overheads associated with the quadratic form in the context. The following theorem addresses this problem:

Theorem II: With $g_{i,j}$ defined in Eq. (1), the quadratic form can be computed in $O(1)$ time.

Proof:

$$g_{i,j}^T L_G g_{i,j} = g_{i,j}^T ((L_G)_i - (L_G)_j) \tag{28}$$

where $(L_G)_i$ represents the i-th column of L_G,

$$g_{i,j}^T L_G g_{i,j} = L_G(i,i) + L_G(j,j) - L_G(i,j) - L_G(j,i) \tag{29}$$

Considering that L_G is a symmetric matrix,

$$g_{i,j}^T L_G g_{i,j} = L_G(i,i) + L_G(j,j) - 2L_G(i,j) \tag{30}$$

Equation (30) shows that the computation involves only three elements in the matrix and a fundamental operation among them, which concludes the proof.

IV. EXPERIMENTAL RESULTS

A. Experiment Setup

We implement our method using the C++ programming language and run it on an AMD EPYC 7702 CPU with 2.0 GHz and 512 GB RAM. We also select a diverse range of benchmark datasets, as illustrated in TABLE I, including six power grid matrices downloaded from the IBM power grid circuits (ibmpg3-ibmpg8) [16] and 14 large-scale connected graphs downloaded from the SuiteSparse Matrix Collection [17]. Such datasets represent a significant computational challenge and are commonly encountered in many applications, including circuit simulation and network analysis.

TABLE I: Test Cases and Their Detailed Information

Index	Case	Node	Edge	Index	Case	Node	Edge
1	ibmpg3	8.5×10^5	1.4×10^6	11	NACA0015	1.7×10^7	5.0×10^7
2	ibmpg4	9.5×10^5	1.6×10^6	12	delaunay_n19	5.2×10^5	1.6×10^6
3	ibmpg5	1.1×10^6	1.6×10^6	13	delaunay_n20	1.0×10^6	3.1×10^6
4	ibmpg6	1.7×10^6	2.5×10^6	14	delaunay_n21	2.1×10^6	6.3×10^6
5	ibmpg7	1.5×10^6	2.4×10^6	15	delaunay_n22	4.2×10^6	1.3×10^7
6	ibmpg8	1.5×10^6	2.4×10^6	16	delaunay_n23	8.4×10^6	2.5×10^7
7	333SP	3.7×10^6	1.1×10^7	17	delaunay_n24	1.7×10^7	5.0×10^7
8	M6	3.5×10^6	1.1×10^7	18	com-Youtube	1.1×10^6	3.0×10^6
9	AS365	3.8×10^6	1.1×10^7	19	com-DBLP	3.2×10^5	1.0×10^6
10	NRL	4.2×10^6	1.2×10^7	20	venturiLevel3	4.0×10^6	8.0×10^6

We conduct the comparison among the proposed algorithm and two state-of-the-art spectral graph sparsification algorithms, GRASS [10] and feGRASS [11] in various dimensions, ranging from time overheads, the relative condition number to solver's performance.

B. Time Consumption to Construct Sparsifier

To ensure a fair comparison, we specify that the output subgraphs generated by all three algorithms must satisfy identical conditions, specifically, the recovery of $0.05|V|$ off-tree edges during the construction of the preconditioner. TABLE II presents the efficiency and effectiveness of GRASS and feGRASS in comparison to our proposed method for producing the preconditioner, indicated by Ratio1 and Ratio2, respectively. As demonstrated in TABLE II, our proposed method exhibits a significant improvement in speed compared with GRASS, achieving an average (maximum) speedup of 37.3x (83.27x) and even surpassing feGRASS with an average (maximum) speedup of 2.13x (2.55x). This improved performance is attributed to our approach of analyzing trace reduction instead of estimating the dominant generalized eigenvalue and eigenvector in GRASS, as well as leveraging eigenvalue transformation to convert the objective from $Tr(L_T^+L_G)$ in feGRASS to $Tr(L_G^+L_T)$.

C. Relative Condition Number for Preconditioner

The minimum generalized eigenvalue is typically approximately equal to 1 and its computation is extremely time-consuming. Consequently, we compute the maximum eigenvalue and consider it as κ instead. It is also worth noting that in the experiment, the singularity of the Laplacian matrix is the insurmountable challenge for the computation of generalized eigenvalues. To address this issue without significantly disturbing the results, We add a small, uniform value to all diagonal elements, thus forcing the Laplacian matrix to be full-rank.

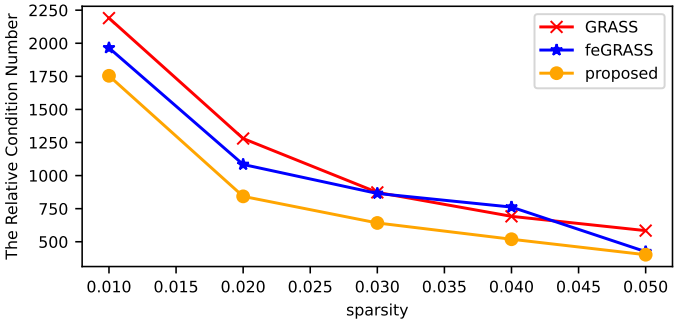

Fig. 1: Tendency between the sparsity and relative condition number of the generated sparse subgraph, for test case $delaunay\_n19$.

In TABLE II, we could observe that the proposed algorithm outperforms feGRASS for all cases. The Ratio2 can reach 1.25x on average, and the maximum one can reach 1.65x, even achieving excellent results beyond GRASS in half of cases. It is evident that eigenvalue transformation significantly reduces computational complexity while ensuring the effectiveness of the results.

Furthermore, Taking the example of $delaunay\_n19$, as illustrated in Fig.1, we adjust the number of recovered off-tree edges and track the trend of the relative condition number of different preconditioners. Our proposed method consistently outperforms the other algorithms across all indicators. Notably, our method exhibits a steeper decline near the starting point. These indicate the effectiveness and practicality of eigenvalue transformation to identify spectrally-critical off-tree edges.

D. Iterative Solver Performance

We take the sparsifier produced from the proposed method, GRASS, and feGRASS, respectively, as the preconditioner of the iterative solver and compare the total iteration number and computational time. PCG is chosen as the iterative solver for the test experiment. The preparatory items for the experiment are as follows. Firstly, it is essential to perform a Cholmod factorization algorithm on the sparse subgraphs output by each algorithm prior to running PCG [18]. Secondly, since the specific form of right-hand side vector b does not impact the solution time, a randomly generated b is used for testing. Finally, a termination criterion is set for the PCG runtime: $||L_G x - b|| < 10^{-3}||b||$. It is worth noting that we embedded the spectral sparsification algorithm and Cholmod factorization into the PCG, and the time in the table actually includes the spectral sparsification time, Cholmod factorization time, and solver's runtime.

In TABLE II, it is evident that the preconditioners generated by the proposed method exhibit superior performance on iteration number, compared to those produced by feGRASS. Notably, in nearly half of the cases, our method outperforms

TABLE II: performance comparisons

Case	GRASS [10]				feGRASS [11]				Proposed											
	Preconditioner		Solver		Preconditioner		Solver		Preconditioner						Solver					
	Runtime(s)	κ	#iter	time(s)	Runtime(s)	κ	#iter	time(s)	Runtime(s)	Ratio1	Ratio2	κ	Ratio1	Ratio2	#iter	Ratio1	Ratio2	time(s)	Ratio1	Ratio2
ibmpg3	6.41	**68.32**	**23**	8.31	0.32	87.32	38	1.96	**0.17**	37.71x	1.88x	72.15	0.95x	1.21x	27	0.85x	1.41x	**0.58**	14.42x	3.40x
ibmpg4	7.20	70.16	**27**	12.04	0.39	52.20	52	3.35	**0.19**	37.89x	2.05x	**37.88**	1.85x	1.38x	37	0.73x	1.41x	**0.82**	14.75x	4.10x
ibmpg5	10.09	**76.92**	25	15.15	0.59	128.19	48	3.43	**0.24**	42.04x	2.46x	98.27	0.78x	1.30x	**21**	1.19x	2.29x	**0.89**	17.06x	3.86x
ibmpg6	15.00	**81.24**	**24**	16.42	0.96	203.26	47	2.63	**0.42**	35.71x	2.29x	151.78	0.54x	1.34x	32	0.75x	1.47x	**2.29**	7.16x	1.15x
ibmpg7	13.32	**58.76**	28	16.99	0.97	130.44	19	2.54	**0.38**	35.05x	2.55x	86.34	0.68x	1.51x	**17**	1.65x	1.12x	**1.21**	14.02x	2.10x
ibmpg8	13.20	**79.12**	32	19.27	0.59	152.65	18	3.10	**0.31**	42.58x	1.90x	114.58	0.69x	1.33x	**17**	1.88x	1.06x	**1.45**	13.27x	2.13x
333SP	62.52	**1156.34**	**168**	**69.12**	3.35	6089.40	412	135.64	**1.44**	43.42x	2.33x	4145.52	0.28x	1.47x	307	0.55x	1.34x	107.82	0.64x	1.26x
M6	72.72	**1169.63**	175	85.22	4.17	2683.89	215	97.88	**1.82**	39.96x	2.29x	1561.18	0.75x	1.72x	**162**	1.08x	1.33x	**83.17**	1.02x	1.18x
AS365	83.16	628.20	**145**	126.57	4.46	532.55	243	95.36	**1.97**	42.21x	2.26x	**503.34**	1.25x	1.06x	169	0.86x	1.44x	**74.92**	1.69x	1.27x
NLR	92.52	**1059.48**	129	148.93	5.02	3410.83	225	115.78	**2.17**	42.64x	2.31x	2807.35	0.38x	1.21x	**117**	1.10x	1.92x	**89.40**	1.67x	1.30x
NACA0015	17.40	510.33	**193**	20.53	0.95	513.74	279	17.28	**0.46**	37.83x	2.07x	**506.90**	1.01x	1.01x	201	0.96x	1.39x	**13.05**	1.57x	1.32x
delaunay_n19	4.37	584.96	193	7.72	0.30	425.50	152	4.66	**0.17**	25.71x	1.76x	**402.34**	1.45x	1.06x	**142**	1.36x	1.07x	**3.48**	2.22x	1.34x
delaunay_n20	12.48	585.31	134	14.55	0.65	434.80	142	9.47	**0.35**	35.66x	1.86x	**404.70**	1.45x	1.07x	**128**	1.05x	1.11x	**7.58**	1.92x	1.25x
delaunay_n21	23.28	574.37	173	24.83	1.39	453.65	154	18.49	**0.72**	32.33x	1.93x	**418.46**	1.37x	1.08x	**135**	1.28x	1.14x	**14.95**	1.66x	1.24x
delaunay_n22	53.28	580.14	167	53.72	3.01	458.26	129	39.93	**1.43**	37.26x	2.10x	**424.33**	1.37x	1.08x	**98**	1.70x	1.32x	**32.99**	1.63x	1.21x
delaunay_n23	110.52	591.98	174	104.81	6.46	473.06	137	80.04	**3.02**	36.60x	2.14x	**435.33**	1.36x	1.09x	**112**	1.55x	1.22x	**65.09**	1.61x	1.23x
delaunay_n24	22.92	623.47	165	197.15	14.19	467.83	143	152.16	**6.70**	3.42x	2.12x	**433.04**	1.44x	1.08x	**132**	1.25x	1.08x	**124.27**	1.59x	1.22x
com-Youtube	27.48	**542.56**	**123**	266.58	0.73	2047.63	182	218.76	**0.33**	83.27x	2.21x	1657.59	0.33x	1.24x	142	0.87x	1.28x	**204.59**	1.30x	1.07x
com-DBLP	5.14	689.70	169	8.79	0.23	684.75	132	4.00	**0.11**	46.73x	2.09x	**413.79**	1.67x	1.65x	**102**	1.66x	1.29x	**3.55**	2.47x	1.13x
venturiLevel3	5.60	896.46	165	47.98	1.49	385.95	149	35.76	**0.71**	7.89x	2.10x	**326.46**	2.75x	1.18x	**114**	1.45x	1.31x	**29.84**	1.61x	1.20x
Average	-	-	-	-	-	-	-	-	-	37.30x	2.13x	-	1.12x	1.25x	-	1.19x	1.35x	-	5.16x	1.70x

GRASS. The result is similar to that in the comparison of the relative condition number. The Ratio1 achieves an average value of 1.19x, with a maximum of 1.88x observed on $ibmpg8$. When compared with feGRASS, Ratio2 consistently exceeds 1, ranging from 1.06x to 2.29x. While GRASS is known for its rigorous mathematical foundation, which ensures its preconditioners' superior performance, it falls behind the latter two methods in terms of time consumption.

In terms of time consumption of solvers, The solver embedded with our proposed algorithm exhibits a significant acceleration compared to the other two methods. The average Ratio2 reaches 1.70x, while the average Ratio1 reaches 5.16x. From the experimental results, it is evident that when considering the time required for preconditioner generation throughout the entire simulation process, our algorithm achieves the best balance between time and effectiveness.

V. Conclusion

In this paper, we introduce a novel spectral graph sparsification method aimed at generating high-quality preconditioners for iterative solvers in power supply noise analysis. Our approach focuses on reducing time complexity and enhancing the effectiveness of preconditioner generation through the introduction of spectral-aware weights, eigenvalue transformations and fast computation method of effective resistance. Experimental results demonstrate the superior performance of our approach, with significant improvements observed in both computation time and the relative condition number. The potential applications of this approach extend to diverse research areas, including power grid simulation and beyond.

VI. Acknowledgement

Zhou Jin and Weifeng Liu are the corresponding authors of this paper. This work was supported in part by the NSFC (Grant No. 62034007, 62204265, 62374031) and National Key R&D Program of China (Grant No. 2022YFB4400400).

References

[1] S. Sreevidya, R. Holla, and R. Raghu, "Low power physical design and verification in 16nm finfet technology," *ICECA*, 2019.

[2] H. Xu and A. A. Abidi, "Analysis and design of regenerative comparators for low offset and noise," *TCAS-I*, 2019.

[3] X. Zha, H. Pei, D. Niu, X. Wu, and Z. Jin, "Deep learning enhanced time-step control in pseudo transient analysis for efficient nonlinear dc simulation," *ISEDA*, 2023.

[4] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994.

[5] T. W. W. L. Y. L. E. Y. J. Z. X. G. F. L. J. Z. Z. J. Xu Fu, Bingbin Zhang and W. Liu, "Pangulu: A scalable regular two-dimensional block-cyclic sparse direct solver on distributed heterogeneous systems," *SC*, 2023.

[6] T. Wang, W. Li, H. Pei, Y. Sun, Z. Jin, and W. Liu, "Accelerating sparse lu factorization with density-aware adaptive matrix multiplication for circuit simulation," *DAC*, 2023.

[7] O. Axelsson and G. Lindskog, "On the rate of convergence of the preconditioned conjugate gradient method," *Numer. Math.*, 1986.

[8] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SICOMP*, 2011.

[9] I. Koutis, G. L. Miller, and R. Peng, "Approaching optimality for solving sdd linear systems," *FOCS*, 2010.

[10] Z. Feng, "Grass: Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis," *DAC*, 2016.

[11] Z. Liu, W. Yu, and Z. Feng, "fegrass: fast and effective graph spectral sparsification for scalable power grid analysis," *TCAD*, 2021.

[12] Y. Zhang, Z. Zhao, and Z. Feng, "SF-GRASS: Solver-free graph spectral sparsification," *ICCAD*, 2020.

[13] Z. Jin, H. Pei, Y. Dong, X. Jin, X. Wu, W. W. Xing, and D. Niu, "Accelerating nonlinear dc circuit simulation with reinforcement learning," *DAC*, 2022.

[14] S. Kim, S.-Y. Lee, S. Park, K. R. Kim, and S. Kang, "A logic synthesis methodology for low-power ternary logic circuits," *IEEE Trans. Circuits Syst. I*, 2020.

[15] J. W. Daniel, "The rayleigh-ritz method for eigenvalues of self-adjoint operators," *Math. Comp.*, 1951.

[16] S. R. Nassif, *IBM Power Grid Benchmarks*, 2020. [Online]. Available: https://web.ece.ucsb.edu/lip/PGBenchmarks/ibmpgbench.html

[17] T. A. Davis, *SuiteSparse Matrix Collection*, 2010. [Online]. Available: https://sparse.tamu.edu/

[18] C. Li, C. An, Z. Gao, F. Yang, Y. Su, and X. Zeng, "Unleashing the power of graph spectral sparsification for power grid analysis via incomplete cholesky factorization," *TCAD*, 2023.