An Efficient Placement Speedup Technique Based on Graph Signal Processing

Yiting Liu, Hai Zhou, Jia Wang, *Senior Member, IEEE*, Fan Yang, *Member, IEEE*, Xuan Zeng, *Senior Member, IEEE*, and Li Shang, *Member, IEEE*

***Abstract*—Placement is a critical task with high computation complexity in VLSI physical design. Modern analytical placers formulate the placement objective as a nonlinear optimization task, which suffers a long iteration time. To accelerate and enhance the placement process, recent studies have turned to deep learning-based approaches, particularly leveraging graph convolution networks (GCNs). However, learning-based placers require time- and data-consuming model training due to the complexity of circuit placement that involves large-scale cells and design-specific graph statistics. This article proposes GiFt, a parameter-free initialization technique for accelerating placement, rooted in graph signal processing. GiFt excels at capturing multiresolution smooth signals of circuit graphs to generate optimized initial placement solutions without the need for time-consuming model training, and meanwhile significantly reduces the number of iterations required by analytical placers. Moreover, we present GiFtPlus, an enhanced version of GiFt, which is more efficient in handling large-scale circuit placement and can accommodate location constraints. Experimental results on public benchmarks show that GiFt and GiFtPlus significantly improve placement efficiency, while achieving competitive or superior performance compared to state-of-the-art placers. In particular, the recently proposed GPU-accelerated analytical placer DREAMPlace uses up to 50% more total runtime than GiFtPlus-DREAMPlace.**

***Index Terms*—Graph convolution, graph filter, graph signal processing (GSP), physical design, placement initialization, placement.**

I. Introduction

PLACEMENT is a critical and complex task in VLSI physical designs. It determines the locations of connected cells to minimize total wirelength while adhering to constraints (e.g., density constraints). Despite continuous efforts to accelerate and improve the placement process, it remains challenging due to the growing complexity of modern designs. The state-of-the-art analytical placers [1], [2] model the placement problem as an electrostatic system and iteratively optimize it, which involves long iteration time. Recently proposed DREAMPlace [3] leverages GPU acceleration to enhance efficiency. However, long iterations remain a challenge to analytical placers.

To expedite the placement process, deep learning-based methods have drawn significant attention. Recently proposed graph convolution network (GCN)-based methods represent circuit netlist as graph and produce initial chip placement through efficient model inference. For instance, GraphPlanner [4] utilizes a variational GCN-based model to generate initial placement solutions, streamlining the placement procedure for improved efficiency. Researchers [5], [6] apply GCNs and their variants to encode the connectivity information for generating node embeddings. These embeddings are subsequently employed in reinforcement learning calculations to yield superior placement results.

However, existing empirical GCN-based approaches face two key limitations. First, these approaches require time-consuming parameter learning with high computation and memory costs. Specifically, GCN-based deep models equipped with a large number of learnable parameters demand a significant amount of training data and considerable training time. As the scale of modern designs continues to increase, the computational complexity and memory demands of GCN-based deep models increase accordingly. Second, it is challenging to guarantee the generalization of GCN-based models. The statistical characteristics of graph structures across various designs can vary significantly, and the existence of predetermined fixed cells (e.g., fixed IOs) exacerbates the issue. Consequently, even well-trained GCNs may struggle to generalize to other unseen designs, limiting their practical applicability. Therefore, an interesting question arises: *How to minimize the required optimization iterations of analytical solvers and yet avoid the high training costs of GCN-based methods?*

This article presents GiFt, an efficient parameter-free initialization approach for accelerating chip placement. GiFt can be seamlessly integrated with modern analytical placers to construct an ultrafast placement flow GiFt-Placer, which significantly reduces the numerous optimization iterations of placers without the need for time-consuming model training. In essence, GiFt is theoretically rooted in graph signal processing (GSP) and we emphasize the crucial role of *smoothness*, a

Received 19 September 2024; revised 8 December 2024 and 18 February 2025; accepted 23 March 2025. Date of publication 1 April 2025; date of current version 22 September 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB4405101, and in part by the National Natural Science Foundation of China under Grant 62090025 and Grant 92373207. This article was recommended by Associate Editor B. Yu. *(Corresponding author: Li Shang.)*

Yiting Liu and Li Shang are with the School of Computer Science, Fudan University, Shanghai 200433, China (e-mail: yitingliu20@fudan.edu.cn; lishang@fudan.edu.cn).

Hai Zhou is with the Department of ECE, Northwestern University, Evanston, IL 60208 USA (e-mail: haizhou@northwestern.edu).

Jia Wang is with the Department of ECE, Illinois Institute of Technology, Chicago, IL 60616 USA (e-mail: jwang@ece.iit.edu).

Fan Yang and Xuan Zeng are with the School of Microelectronics, Fudan University, Shanghai 200433, China (e-mail: yangfan@fudan.edu.cn; xzeng@fudan.edu.cn).

Digital Object Identifier 10.1109/TCAD.2025.3556968

key concept in GSP, in enhancing chip placement. Specifically, GiFt functions as a multifrequency *Graph Filter* with low computation complexity to promote both local and global signal smoothness on circuit graphs. This feature facilitates the generation of optimized initial placement solutions, which drives the subsequent placers to generate high-quality placement results with highly reduced iteration time. Moreover, we prove that both the classic eigenvector-based placers [7], [8] and recently emerged GCN-based placers [4], [5] are special cases of GiFt-Placer. Our work further points out that these approaches introduce redundant computations, which are unnecessary for high-quality chip placement. Moreover, we propose GiFtPlus, an enhanced version designed to more efficiently address large-scale circuit placement problems with location constraints. Experimental results on public benchmarks show that GiFt-Placer and GiFtPlus-Placer significantly improve placement efficiency, while achieving competitive or superior performance compared to state-of-the-art placers. In particular, compared to DREAMPlace, the recently proposed GPU-accelerated analytical placer, GiFtPlus improves runtime by over 50%.

The key contributions are listed as follows.

1) We propose a new multifrequency graph filter-based placement initialization approach called GiFt, which can comprehensively capture the graph structure and efficiently generate optimized cell locations.
2) GiFt offers high efficiency through sparse matrix multiplication and does not require a time-consuming model training process due to its parameter-free nature.
3) By integrating graph sparsification techniques and Dirichlet boundary conditions, we introduce GiFtPlus, an enhanced version of GiFt that more efficiently produces optimized cell locations for large-scale chip placement with location constraints.
4) GiFt and GiFtPlus can be seamlessly integrated with modern analytical placers to construct GiFt-Placer and GiFtPlus-Placer placement flow, which effectively reduces the number of iterations required for analytical placers and significantly improves placement efficiency, even surpassing the performance of analytical placers running on GPU versions.
5) By examining classic eigenvector-based placers and recent GCN-based placers from the perspective of GSP, we prove that they can be considered as special cases of GiFt-Placer, but they introduce unnecessary computation costs to chip placement.

The remainder of this article is organized as follows. Section II describes preliminaries for the rest of this article. Section III introduces the details of GiFt. Section IV revisits the eigenvector-based placer and GCN-based placer from GSP perspective, demonstrating that they are special case of GiFt. Section V presents the enhanced version GiFtPlus. Section VI presents the experimental results. Section VII discusses the theoretical foundations of this work. We conclude the article in Section VIII.

II. Preliminaries

This section introduces the fundamental knowledge of GSP and global placement.

A. Graph Signal Processing

Given a weighted undirected graph $G = (V, E)$ with a set of nodes V and edges E, it can be represented as an adjacency matrix $A = \{w_{i,j}\} \in \mathbb{R}^{N\times N}$ with $w_{i,j} > 0$ if node v_i and v_j are connected by edges, and $w_{i,j} = 0$ otherwise. The graph Laplacian matrix is defined as $L = D - A$, where $D = \text{diag}(d_1, d_2, \ldots, d_n) \in \mathbb{R}^{N\times N}$ represents the degree matrix of A with $d_i = \sum_{v_j\in V} w_{i,j}$. The normalized graph Laplacian matrix is defined as $\tilde{L} = D^{-(1/2)}LD^{-(1/2)}$.

The graph signal g is defined as a mapping $g : V \to \mathbb{R}$ and it can be expressed as an n-dimensional vector where each element g_i can encapsulate diverse types of information associated with node v_i [9]. GSP is a field dedicated to the analysis and manipulation of these signals on graphs, intending to extract meaningful insights or accomplish various tasks.

Smoothness: Smoothness in GSP refers to the rate at which signal values change across the nodes of a graph. A smooth signal shows minimal variation, meaning nodes that are directly connected exhibit similar signal values. In contrast, a less smooth signal displays significant differences between connected nodes. Quantitatively, smoothness can be measured using the graph Laplacian quadratic form, which provides a mathematical basis for evaluating how signal values differ across a graph.

The graph gradient of signal g at node v_i is defined as $\nabla_i g := [\{(\partial f/\partial e)|_i\}_{e\in E}] = [\sqrt{w_{i,j}}(g_i - g_j)]$. The smoothness of the graph signal can be quantified using the graph Laplacian quadratic form, as calculated by the following equation:

$$S(g) = \frac{1}{2}\sum_{v_i\in V} ||\nabla_i g||_2^2 = \sum_{(v_i,v_j)\in E} w_{i,j}\left(g_j - g_i\right)^2. \tag{1}$$

Graph Fourier Transform (GFT): The GFT is a typical method for transforming a graph signal from the spatial domain (where the signal values are associated with the nodes in a graph) to the spectral domain (where the signal is represented in terms of its frequency components over the graph). This transformation is critical for tasks, such as denoising, as it allows for direct manipulation of the frequency components, facilitating more efficient signal processing operations within the spectral domain.

Since the graph Laplacian matrix L is real and symmetric, it can be decomposed into $L = U\Lambda U^T$, where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ denotes eigenvalues with $\lambda_1 \le \lambda_2, \ldots, \le \lambda_n$ and $U = [u_1, \ldots, u_n]$ is the corresponding eigenvectors. Using eigenvector matrix U as the GFT basis, we call $\hat{g} = U^T g$ as GFT that transforms the graph signal g from the spatial domain to the spectral domain. And its inverse transform is defined as $g = U\hat{g}$.

Graph Filter: A graph filter is a tool used in GSP to selectively emphasize or suppress certain frequency components of a signal on a graph, similar to how traditional filters work on time-based signals (like audio). Graph filters exploit the structure of the graph to enhance or reduce specific patterns within the signal, thus optimizing the signal processing tasks according to specific requirements.

In the spectral domain, the graph filter can be used to selectively filter some unwanted frequencies present in the

graph signal, which is formally defined as:

$$\mathcal{H} = U\text{diag}(h(\lambda_1), h(\lambda_2), \ldots, h(\lambda_n))U^T \tag{2}$$

where $h(\cdot)$ is a filter function applied to frequencies.

B. Global Placement

A placement instance can be formulated as a graph $G = (V, E)$ with a set of objects V (e.g., IOs, macros, and standard cells) and edges E. The main objective of placement $f(g)$ [see (3)] is to find a solution $g \in \mathbb{R}^{N\times 2}$ with minimized total wirelength $S(g)$ subject to density constraints (i.e., the density $\rho_b(g)$ does not exceed a predetermined density ρ_t), where N is the number of objects.

$$\min f(g) = S(g) \qquad \text{s.t.} \quad \rho_b(g) \le \rho_t. \tag{3}$$

The total wirelength $S(g)$ can be estimated as the weighted sum of squared distances between connected objects:

$$S(g) = \sum_{(v_i,v_j)\in E} w_{i,j}\Big((x_i - x_j)^2 + (y_i - y_j)^2\Big). \tag{4}$$

Considering one net with M pins, we set the weight $w = (2/M)$.

The placement area can be evenly partitioned into a series of grids (bins) [1]. The density of each grid b is computed using the formula in (5):

$$\rho_b(g) = \sum_{v\in V} l_x(v, b)l_y(v, b) \tag{5}$$

where $l_x(v, b)$ and $l_y(v, b)$ quantify the horizontal and vertical overlaps between the grid b and the object v.

III. Method

In this section, we first explore the placement process from the perspective of GSP and highlight the importance of smoothness for chip placement. Then we present GiFt, the parameter-free placement initialization approach, and demonstrate its efficacy via theoretical analysis. Finally, we prove that both the classic eigenvector-based placer [7], [8] and recently emerged GCN-based placer [4], [5] are special cases of the proposed approach, while both of them introduce extra computations and complexity.

A. Enhancing Placement From GSP Perspective

As demonstrated in (1) and (4), the optimization objective in placement, which aims to minimize quadratic wirelength, aligns with the goal of enhancing graph signal smoothness from the perspective of GSP. Inspired by this finding, we reframe the placement problem as an effort to improve the smoothness of graph signals on given circuit graphs. Next, we explain how we can improve the smoothness of graph signals across the graph.

The normalized form of the smoothness measurement [as depicted in (1)] can be calculated using the Rayleigh quotient, as shown below:

$$R(g) = \frac{S(g)}{||g||_2^2} = \frac{\sum_{(v_i,v_j)\in E} w_{i,j}\big(g_j - g_i\big)^2}{\sum_{v_i\in V} g_i^2} = \frac{g^T L g}{g^T g}. \tag{6}$$

The smaller value of $R(g)$ indicates higher smoothness of graph. Leveraging the GFT, we can transform (6) as follows:

$$R(g) = \frac{g^T L g}{g^T g} = \frac{g^T U \Lambda U^T g}{g^T U U^T g} = \frac{\hat{g}^T \Lambda \hat{g}}{\hat{g}^T \hat{g}} = \frac{\sum_{v_i\in V} \lambda_i \hat{g_i}^2}{\sum_{v_i\in V} \hat{g_i}^2}. \tag{7}$$

It is evident from (7) that $R(g_i) = \lambda_i$, where g_i denotes the graph signal defined on node v_i, and λ_i denotes ith eigenvalue of the graph Laplacian. This demonstrates that graph signals associated with lower eigenvalues (i.e., lower frequencies) exhibit a higher degree of smoothness. Consequently, it becomes essential to filter out undesired frequencies, particularly the high-frequency components in graph signals, to enhance the smoothness of these signals, thereby obtaining optimized placement solutions.

Building on this insight, we present our placement framework through the lens of GSP.

Given the input graph signals $g \in \mathbb{R}^{N\times 2}$, which can represent the initial cell locations, potentially containing unwanted noises, we apply a graph filter $\mathcal{H}$ to g to filter out undesired frequencies and obtain the filtered signals $g' \in \mathbb{R}^{N\times 2}$, which represents the cell locations processed by the graph filter. The framework can be expressed as follows:

$$g' = \mathcal{H}g = U\text{diag}(h(\lambda_1), h(\lambda_2), \ldots, h(\lambda_n))U^T g \tag{8}$$

where $h(\cdot)$ is a filter function defined on eigenvalues. This process can be interpreted from a GSP perspective: Given a graph signal g, it is initially transformed from the spatial domain to the spectral domain via the GFT. Subsequently, unwanted frequencies within the signal are removed using the filter $h(\cdot)$, and finally, the filtered signal is transformed back into the spatial domain through the inverse GFT.

Since the smoothness of graph signals is closely linked to the minimization of placement wirelength, a straightforward approach is to develop an ideal low-pass filter that directly eliminates all high-frequency signals. This can be formulated as follows:

$$h(\lambda_i) = \begin{cases} 1, & \text{if } \lambda_i < \lambda_t \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

where λ_t denotes the cut-off frequency. Then the filtered signals g' are given by

$$g' = \mathcal{H}g = U\text{diag}\left(\overbrace{1, \ldots, 1}^{t}, \overbrace{0, \ldots, 0}^{n-t}\right)U^T g = U_t U_t^T g. \tag{10}$$

Although this approach guarantees the global smoothness of graph signals, it requires eigendecomposition to obtain the eigenvectors corresponding to the first k lowest eigenvalues, which is a computationally expensive task for large-scale circuits. In addition, it is essential to note that globally smooth signals overlook valuable local information, which can be characterized as globally high-frequency yet locally smooth [10]. This oversight may lead to an over-smoothing issue, where neighboring locations become too similar, resulting in high overlap in local areas and violations of density constraints (see Section III-C for more details).

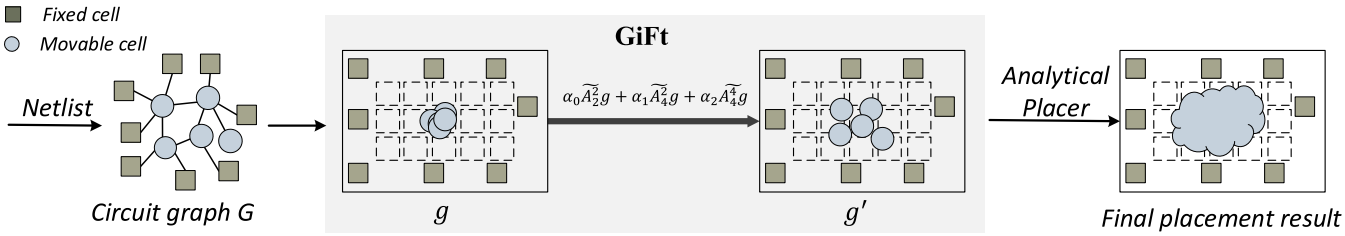

Fig. 1. Workflow of GiFt-equipped placement process.

To this end, we need to thoughtfully devise a new graph filter that is computationally efficient while considering multiresolution smooth signals, thereby yielding optimized cell locations g' that have minimized total wirelength (representing smoothness over the graph structure) and reduced overlap (i.e., avoid over-smoothing).

B. GiFt: Efficient Placement Initialization Technique

This section introduces GiFt, a GSP-based placement initialization approach for chip placement acceleration. GiFt functions as an efficient graph filter, utilizing multifrequency graph signals for comprehensive graph structural analysis and the generation of optimized cell locations. It can be seamlessly integrated with analytical placers to produce high-quality placement solutions while significantly reducing placement time. The GiFt-equipped placement process is depicted in Fig. 1.

1) Algorithm of GiFt: This section presents the theoretical underpinnings of GiFt.

From the perspective of GSP, the normalized adjacency matrix $\tilde{A}$ of the given circuit graph corresponds to the filter function $h(\lambda) = 1 - \lambda$. The theoretical proof is as follows.

Theorem 1: The normalized adjacency matrix $\tilde{A} = D^{-(1/2)}AD^{-(1/2)}$ is a graph filter corresponding to the filter function $h(\lambda) = 1 - \lambda$.

Proof: As the eigendecomposition of the normalized graph Laplacian is given by $\tilde{L} = U\Lambda U^T = U\text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)U^T$, $\tilde{L}$ corresponds to the filter function $h(\lambda) = \lambda$.

Since we have $L = D - A$, then

$$\tilde{L} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}} = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}. \quad (11)$$

Let u_i denote the eigenvector corresponding to the eigenvalue λ_i, and $Lu_i = \lambda_i u_i$, we have

$$Lu_i = \left(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)u_i = \lambda_i u_i. \quad (12)$$

It can be transformed to

$$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}u_i = (1 - \lambda_i)u_i \quad (13)$$

which indicates that $\tilde{A}$ is a graph filter corresponding to the filter function $h(\lambda) = 1 - \lambda$, that is

$$\tilde{A} = U\Lambda_A U^T = U\text{diag}(1 - \lambda_1, 1 - \lambda_2, \ldots, 1 - \lambda_n)U^T. \quad (14)$$

■

Since the eigenvalues of the normalized Laplacian fall within the interval of $[0, 2]$, $\tilde{A}$ acts as a band-stop filter that attenuates intermediate-frequency components, which is not effective for denoising graph signals.

To address this problem, we introduce two enhancements to $\tilde{A}$ to boost its denoising ability. First, we add self-loops to $\tilde{A}$ (called augmented $\tilde{A}$) to shrink high-frequency components (e.g., large eigenvalues) [11], thereby removing high-frequency noises. As depicted in Fig. 2(a), by adding self-loops as follows $A = A + \sigma I$ [where $\sigma = 0, 1, 2, 3$ in Fig. 2(a)], the large eigenvalues become smaller, leading the augmented $\tilde{A}$ to perform like a low-pass filter. It is important to note that if σ is too large, most eigenvalues will approach zero, rendering the graph filter less effective in noise removal. Second, we generalize the augmented $\tilde{A}$ to $\tilde{A}^k$. This modification transforms its corresponding filter function from $h(\lambda) = 1 - \lambda$ to $h_k(\lambda) = (1 - \lambda)^k$, where k controls the strength of the graph filter. Fig. 2(b) and (c) show the filter strength associated with $\tilde{A}$ and augmented $\tilde{A}$ for various values of k. As k increases, the low-pass filtering effect of the augmented $\tilde{A}$ becomes more pronounced. The underlying dataset for Fig. 2 is mgc_edit_dist_1 benchmark in the ISPD2014 benchmark suite [12]. To this end, by adjusting the values of σ and k, we can effectively regulate the extent to which high-frequency signals are filtered out, ultimately achieving smooth signals across multiple resolutions.

Building upon this analysis, we propose GiFt, functioning as multifrequency graph filters, to generate optimized cell locations as follows:

$$g' = GiFt(g) = \alpha_0\tilde{A}_2^2 g + \alpha_1\tilde{A}_4^2 g + \alpha_2\tilde{A}_4^4 g. \quad (15)$$

Here, $\tilde{A}_1^2$, $\tilde{A}_2^2$, and $\tilde{A}_4^3$ all function as low-pass filters with varying degrees of filtering strength, which are formulated as follows:

$$\begin{aligned}\tilde{A}_2^2 &= \left((D + 2I)^{-\frac{1}{2}}(A + 2I)(D + 2I)^{-\frac{1}{2}}\right)^2 \\ \tilde{A}_4^2 &= \left((D + 4I)^{-\frac{1}{2}}(A + 4I)(D + 4I)^{-\frac{1}{2}}\right)^2 \\ \tilde{A}_4^4 &= \left((D + 4I)^{-\frac{1}{2}}(A + 4I)(D + 4I)^{-\frac{1}{2}}\right)^4. \end{aligned} \quad (16)$$

To differentiate, we refer to $\tilde{A}_2^2$ as a *high-pass filter* that allows some relatively high-frequency signals to pass through to capture local information. $\tilde{A}_4^2$ corresponds to a *medium-pass filter*, and $\tilde{A}_4^4$ represents a *low-pass filter* designed to exclusively preserve low-frequency signals, thereby enhancing global smoothness. α_0, α_1 and α_2 are weight coefficients that determine the proportion of globally smooth signals and locally smooth signals. g symbolizes the input signals, representing the initial cell locations. g' denotes denoised graph signals, i.e., cell locations produced by GiFt.

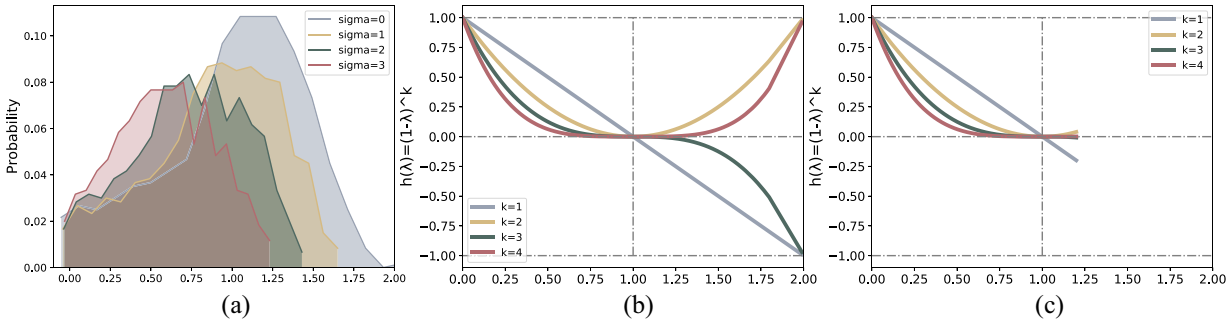

Fig. 2. Eigenvalue distributions of normalized Laplacians with different graph filters: (a) under different self-loops. (b) with $\tilde{A}$ filter at various k, (c) with augmented $\tilde{A}$ filter at various k.

Algorithm 1 GiFt-Placer Algorithm

Input: A circuit netlist modeled by an undirected weighted graph $G = (V, E)$
Output: Legalized placement result,
1: Set initial locations g of cells $v \in V$. Locations of movable cells are placed at the center of the placement region and $1^T g^{(i)=0}$. Fixed cells are set at their predetermined locations.
2: Compute optimized cell locations g' using GiFt: $g' = \alpha_0 \tilde{A}_2^2 g + \alpha_1 \tilde{A}_2^4 g + \alpha_2 \tilde{A}_4^4 g$
3: Input g' to subsequent placer to complete placement task
4: **return** The legalized placement result

The *input signal* g includes the movable cell locations g_m and the fixed cell locations g_f, i.e., $g = \{g_m, g_f\}$. g_m can be regarded as a random vector, represented as a linear combination of the eigenvalues of the graph Laplacian. We utilize multifrequency graph filters to effectively eliminate unwanted frequencies from the random graph signal g_m. To this end, we introduce an *initial location strategy*. g_m is configured as a set of 2-D random vectors orthogonal to the all-one vector, ensuring that $1^T g^{(i)} = 0$. The fixed cell locations g_f are placed at their predetermined locations.

2) Workflow of GiFt-Equipped Placement Process: Since density constraints are not rigorously enforced in (15), the predicted solutions g' can undergo further refinement through the subsequent placer. By integrating GiFt with the analytical placer, we develop the ultrafast placement flow GiFt-Placer.

The algorithm of GiFt-Placer is summarized in Algorithm 1. Given a circuit netlist, we use clique net model [13] to convert it into a weighted graph G. The graph is then processed by GiFt to generate optimized cell locations g' (lines 1 and 2). g' is then fed into the analytical placer, serving as the starting point for placement optimization, to complete the placement task with substantially reduced iteration time (lines 3 and 4).

C. Strength and Computation of GiFt

The primary strengths of GiFt can be summarized into three key aspects.

1) *Denoising Ability:* GiFt effectively generates denoised graph signals (i.e., optimized cell locations) based on the graph topology.
2) *Control of Signal Filtering Strength:* GiFt can integrate multifrequency graph signals to achieve smoothness across multiresolutions of the graph.
3) *Computational Efficiency:* GiFt maintains low computational complexity through sparse matrix multiplication.

This section provides further evidence to demonstrate these capabilities of GiFt.

1) Denoising Ability: As theoretically proven in Section III-B, given random initial cell locations, GiFt thoroughly explores the circuit graph topology to generate denoised graph signals (i.e., optimized cell locations). This section provides numerical examples to illustrate its denoising ability.

We start with randomly placing 20 cells within an $[0, 1] \times [0, 1]$ square and constructing a weighted graph based on the physical distances between cells, with closer cells receiving higher edge weights. The graph structure is shown in Fig. 3. Following this, we apply the initial signal strategy to distribute cells across the chip region, which can be considered a noisy signal. We use Morton code [14] to map the 2-D coordinates into one dimension values while preserving the locality of the data points, and we color the cells according to their values. As shown in Fig. 3(a), highly connected cells display different colors, indicating the inconsistent distribution of noisy signals across the graph structure. Fig. 3(b) presents the denoised signal using GiFt, where highly connected cells appear in similar color, demonstrating that they are closely distributed in 2-d coordinates. This effectively demonstrates GiFt's capability to denoise graph data.

2) Control of Signal Filtering Strength: The strength of the proposed graph filter $\tilde{A}_\sigma^k$ is controlled by the parameters σ and k. As described in Section III-B, larger values of σ and k result in a stronger filter for high-frequency signals. However, large values also lead to over-smoothing issue, which violates the density constraints. Therefore, selecting appropriate values for σ and k is crucial to balancing the filter's strength and avoiding over-smoothing.

Fig. 4 visualizes the raw and filtered cell locations produced by GiFt with different combinations of σ and k. Highly connected cells are indicated by the same color. As σ and k increase, GiFt yields smoother embeddings, resulting in more tightly packed clusters structure, particularly for highly connected cells. However, this clustering violates density constraints. Moreover, a high value of k also increases computational complexity, as discussed in Section III-C3. To effectively explore multiresolution graph topology information and balance signal smoothness with computational complexity, we use a combination of filters, $\alpha_0 \tilde{A}_2^2 + \alpha_1 \tilde{A}_2^4 + \alpha_2 \tilde{A}_4^4$, to

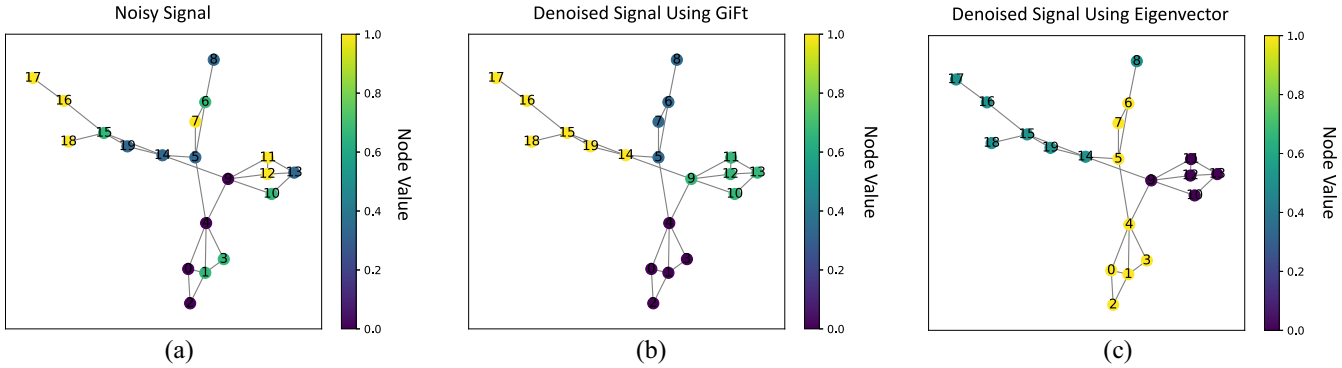

Fig. 3. Denoising example. (a) Noisy signal. (b) Denoised signal using GiFt. (c) Denoised signal using eigenvectors.

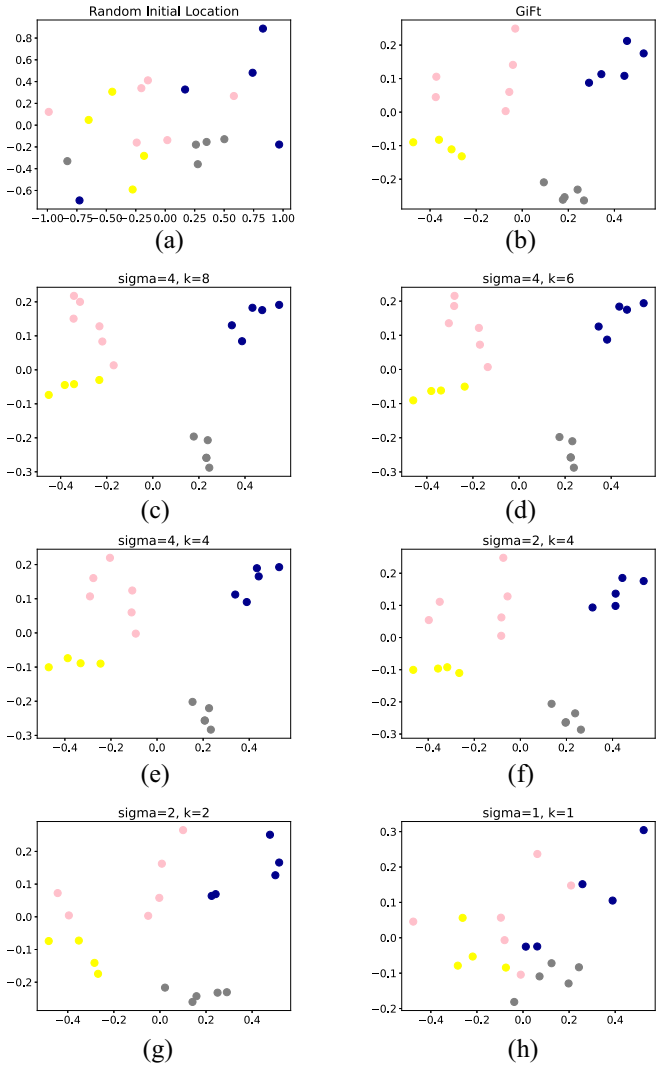

Fig. 4. Placement results with different filters. (a) Random initial locations. (b) Placement results predicted by GiFt. (c) Placement results with $\tilde{A}_4^8$. (d) Placement results with $\tilde{A}_4^6$. (e) Placement results with $\tilde{A}_4^4$. (f) Placement results with $\tilde{A}_2^4$. (g) Placement results with $\tilde{A}_2^2$. (h) Placement results with $\tilde{A}_1^1$.

generate optimized cell locations. The efficacy of this approach is supported by experimental results in Section VI.

3) Computation Efficiency: The computation of GiFt involves the operation $\tilde{A}_\sigma^k g$, where $\tilde{A}_\sigma^k$ is a multifrequency graph filter and g represents the initial cell locations, sized $N \times 2$ with N being the number of cells. For a sparse graph, $\tilde{A}_\sigma^k = ((D+\sigma I)^{-(1/2)}(A+\sigma I)(D+\sigma I)^{-(1/2)})^k$ is the multiplication of sparse matrices. Let e denote the number of nonzero entries in $\tilde{A}_\sigma$, with $e \ll N^2$. Therefore, computing $\tilde{A}_\sigma^k g$ involves repeatedly left-multiplying g by $\tilde{A}_\sigma$ for k iterations, resulting in a computational complexity of $O(2ek)$.

We conclude the power of GiFt for chip placement as follows.

1) *Reduced Iteration Count:* As GiFt can capture comprehensive graph structures from globally smooth signals to locally smooth ones, it can produce optimized cell locations with reduced overlap. By integrating the predicted locations with analytical placers, the placement process benefits from significantly reduced iterations, as the locations predicted by GiFt provide clear and valuable guidance for the subsequent optimization process.
2) *Low Computation Complexity:* GiFt excels in efficiency through sparse matrix multiplication. Its calculation only involves the normalization of the adjacency matrix, eliminating the need for the time-consuming parameter learning process with back-propagation.

IV. Revisit Existing Placers From GSP Perspective

This section analyzes the classic eigenvector-based placer and recently emerged GCN-based placer from the GSP perspective. It is proved that both of them are special cases of the proposed approach with different graph filters.

A. Eigenvector-Based Placer

The classic eigenvector-based placer [7] assumes that all cells are movable. They use the eigenvectors corresponding to the second and third smallest eigenvalues as cell locations. From the GSP perspective, it is an ideal low-pass filter that only saves the lowest frequencies. The corresponding filter function can be defined as

$$h(\lambda_i) = \begin{cases} 1, & \text{if } 1 < i \leq 3 \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

As eigenvectors (graph signals in the spectrum domain) are directly employed to represent cell locations, there is no need to transform them into the spatial domain using inverse GFT

U. Therefore, the filtered signal can be calculated by

$$g' = \text{diag}(0, 1, 1, 0, \ldots, 0)U = [u_2, u_3]. \tag{18}$$

Although this method achieves signal smoothness, it incurs high computational costs when applied to large-scale circuits due to the necessity of eigendecomposition. Moreover, as demonstrated in Section III-A, confining graph signals to merely the lowest-frequency components leads to the loss of valuable information present in higher-frequency components, yielding suboptimal solutions. Fig. 3(c) illustrates the denoising results using eigenvectors. We can see that eigenvectors exhibit over-smoothing issue that gathering cell 5–7 to cell 0–4, demonstrating its ignorance for valuable information in higher-frequency components.

B. GCN-Based Placer

To enhance the placement process, many recent studies have recruited GCNs [4], [5] for help. These studies apply GCNs to encode connectivity information and generate node embeddings for subsequent calculations.

The GCN model operates in three main steps. First, a renormalization trick is applied to the adjacency matrix A by adding an self-loop to each vertex, resulting in a new adjacency matrix $A_1 = A + I$. The degree matrix is also adjusted to $D_1 = D + I$ and then symmetrically normalized to form $\hat{A}_1 = D_1^{-(1/2)} A_1 D_1^{-(1/2)}$. Then, the propagation rule for each layer is defined as follows:

$$H^{(t+1)} = \mathcal{F}\left(\hat{A}_1 H^{(t)} W^{(t)}\right) \tag{19}$$

where $H^{(t)}$ represents the vertex feature matrix at the tth layer, with $H^{(0)} = g$ denoting the initial feature matrix, and $W^{(t)}$ is the trainable weight matrix for the layer. The activation function $\mathcal{F}$ is typically a nonlinear function like ReLU, defined as $RELU(\cdot) = \max(0, \cdot)$. The graph convolution process in each layer involves multiplying the feature matrix $H^{(t)}$ from the left by renormalized adjacency matrix $\hat{A}_1$, with the output then being projected through $W^{(t)}$. This setup stacks two layers, and the output features are apssed through a softmax function to generate the prediction matrix:

$$Z = \text{softmax}\left(\hat{A}_1 ReLU(\hat{A}_1 H^{(0)} W^{(0)}) W^{(1)}\right). \tag{20}$$

The model is trained using a loss function on labeled samples.

In essence, GCN constructs graph filtering in each layer with the filter $\tilde{A}_1$ and the signal matrix $H^{(t)}$. By performing all the graph convolutions in (20), the filter of GCN is $\tilde{A}_1^2$, which can be viewed as a special case of GiFt.

However, GCNs introduce the learnable parameter W, which requires a time-consuming training process via back-propagation. This training overhead is redundant for efficient chip placement. The underlying reason is that the statistical characteristics of graph structures can vary significantly across different circuit designs, and the presence of fixed cells (e.g.,, fixed IOs) exacerbates this issue. Consequently, it is challenging to train a set of fixed parameters that would be effective across a range of diverse designs. On the other hand, by carefully designing the graph filter, it can efficiently remove high-frequency noises and produce optimized cell locations based on multiresolution smooth signals. As demonstrated in Section VI, our proposed approach without the need for model training, achieves competitive or superior performance compared to GCNs, providing further evidence in support of our assertion.

Algorithm 2 GiFtPlus-Placer Algorithm

Input: A circuit netlist modeled by an undirected weighted graph $G = (V, E)$
Output: Legalized placement result.
1: Use feGRASS to perform graph sparsification and obtain the sparse graph G'.
2: Set all fixed cells at their predetermined locations and conduct the convex quadratic wirelength optimization to obtain the movable cell locations g^1.
3: Set cells with location constraints to zero and use GiFt to solve subproblem L^0, yielding the solution g^0.
4: Calculate the optimized cell locations meeting location constraints $g' = g^0 + g^1$
5: Location refinement based on graph filters to obtain the refined version g''
6: Input g'' into subsequent placer to complete placement task
7: **return** The legalized placement result

V. GiFtPlus: Enhanced Version for Large-Scale Chip Placement With Location Constraints

This section introduces GiFtPlus, an enhanced version of GiFt, designed to 1) handle large-scale circuit placement more efficiently and 2) handle location constraints.

A. Solution Overview

The overall algorithmic flow of GiFtPlus is depicted in Algorithm 2. Given a circuit netlist, it is represented as a weighted undirected graph using clique net model [15]. GiFtPlus starts with conducting graph sparsification using feGRASS [16] to remove less critical edges and construct an ultrasparse graph (see Section V-B). This step aims to improve the efficiency of GiFtPlus in large-scale circuit designs. Next, GiFtPlus decomposes the placement problem with location constraints into two subproblems based on Dirichlet boundary conditions. This allows each problem to be solved efficiently and jointly produce optimized cell locations while adhering to the location constraints (see Section V-C). The third step involves location refinement based on graph filter to further enhance the quality of cell locations (see Section V-D). The results produced by GiFtPlus can serve as initial locations that efficiently drive downstream placers to achieve final legalized placement solution with high placement quality.

B. Efficiency Improvement Based on Graph Sparsification

The computational complexity of GiFt is $O(2ek)$ (cf. Section III-C3), where k represents the number of matrix multiplications and e denotes the number of nonzero entries in the adjacency matrix. Since k is user-adjustable and typically less than 4, e dominates the computational complexity. For large-scale circuits with a high number of edges, applying graph sparsification techniques to remove less spectrally critical edges can significantly improve the efficiency of GiFt.

To this end, we utilize feGRASS [16], a fast and effective graph sparsification tool, to construct an ultrasparse graph that retains spectral properties similar to the original graph. This approach enhances the scalability and efficiency of GiFt. More details about feGRASS can be found in this article [16].

C. Placement With Location Constraints

Location constraints refer to those cells with predetermined locations, such as IOs. These fixed cells can be regarded as the vertex boundary [17], which act as immovable reference points that define the placement constraints for the movable cells. These constraints are imposed by application-specific requirements and often lack statistical regularity. GiFt is difficult to precisely account for the cells with fixed locations. This section introduces a highly efficient method using Dirichlet boundary conditions [18], which can be integrated in GiFtPlus to accurately handle location constraints.

Theorem 2: A placement problem with location constraints can be decomposed into a convex quadratic optimization problem with location constraints and an NP-hard combinatorial optimization problem with homogeneous Dirichlet boundary conditions [18].

Theorem 2 demonstrates that the placement problem defined as

$$\begin{aligned} \min L(g) = & \sum_{(v_i,v_j)\in E} w_{i,j}\Big((x_i - x_j)^2 + (y_i - y_j)^2\Big) \\ \text{s.t.} \quad & \rho(g) \le \rho_t \end{aligned} \tag{21}$$

can be decomposed to the following two subproblems:

$$\min L(g) = \min L^1\Big(g^1\Big) + \Big(\min L^0(g^0) \quad \text{s.t.} \quad \rho(g^0) \le \rho_t\Big) \tag{22}$$

where

$$\begin{aligned} L^1\Big(g^1\Big) = & \sum_{(v_i,v_j)\in E} w_{i,j}\Big((x_i^1 - x_j^1)^2 + (y_i^1 - y_j^1)^2\Big) \\ L^0\Big(g^0\Big) = & \sum_{(v_i,v_j)\in E} w_{i,j}\Big((x_i^0 - x_j^0)^2 + (y_i^0 - y_j^0)^2\Big) \\ \text{s.t.} \quad & \rho\Big(g^0\Big) \le \rho_t. \end{aligned} \tag{23}$$

Subproblem L^1: This subproblem focuses on minimizing the total wirelength while ensuring that all fixed cells remain at their predetermined locations. The optimization of L^1 can be divided into x- and y-components, both of which are convex quadratic optimization problems. These can be solved by setting the partial derivatives of $L_x^1(g^1)$ and $L_y^1(g^1)$ to zero [19].

Subproblem L^0: The goal of L^0 is to minimize the total wirelength with all fixed cells set to zero (i.e., satisfying Dirichlet boundary conditions) while reducing overlap, akin to a conventional placement problem without fixed cells. As it is an NP-hard combinatorial problem [20] and challenging to solve efficiently, previous work [18] proposes a GNN-based model for this task. However, as discussed in Section IV, GNN-based models require extensive time and data for model training, and it is challenging to guarantee their generalization. Since GiFt can efficiently captures graph structure information and produces optimized cell locations, we employ GiFt to handle this subproblem.

Placement Solution: The optimized placement solution that satisfies location constraints is obtained by summing the solution g_i^1 of L^1 and the solution g_i^0 of L^0, that is

$$g_i = g_i^0 + g_i^1 \quad i \in \widetilde{V}. \tag{24}$$

D. Location Refinement

We apply location refinement process based on the graph filter to further improve the quality of cell locations produced by GiFtPlus. Specifically, we smooth the cell locations g across the graph by optimizing the following objective:

$$\min\{||g - g'||_2^2 + \text{tr}\big(g'^T L g'\big)\} \tag{25}$$

where g denotes the cell locations determined in Section V-C, which contain both movable and predetermined fixed cell locations. g' denotes the refined cell locations, and L denotes the graph Laplacian matrix. The first term ensures that the refined cell locations g' agrees with the cell locations g, while the second term promotes the smoothness of the graph.

By setting the derivative of (25) to zero, we arrive at the following expression:

$$g' = (I + L)^{-1} g. \tag{26}$$

Since L associates with the filter function λ as proved in Theorem 1, $(I + L)^{-1}$ can be transformed into the spectral domain. This transformation yields the graph filter:

$$h(\lambda) = (1 + \lambda)^{-1}. \tag{27}$$

Then we approximate $h(\lambda)$ using its first-order Taylor expansion:

$$\hat{h}(\lambda) = 1 - \lambda. \tag{28}$$

By transforming $\hat{h}(\lambda)$ back into the spatial domain, we obtain:

$$\hat{h}(L) = I - L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = \tilde{A}. \tag{29}$$

To further enhance the filtering capability, we can add self-loops σ and a scaled coefficient k into (29) according to Section III-B1. Finally, the refined cell locations can be obtained by

$$g' = \tilde{A}_\sigma^k g \tag{30}$$

This article experimentally set both σ and k to 2.

VI. Experimental Results

This section evaluates the performance of GiFt and GiFtPlus using the public benchmark suite ISPD2005 [21], ISPD2006 [22], ISPD2014 [12] and TILOS MacroPlacement Testcases [23].

TABLE I
STATISTICS OF ISPD2005 PLACEMENT BENCHMARKS

Benchmark	#Objs	#Nets	#Movable	#Fixed
adaptec1	211K	221K	210904	543
adaptec2	255K	266K	254457	566
adaptec3	452K	466K	450927	723
adaptec4	496K	516K	494716	1329
bigblue1	278K	284K	277604	560
bigblue2	558K	577K	534782	23084
bigblue3	1097K	1123K	1095519	1293
bigblue4	2177K	2230K	2169183	8170

TABLE II
STATISTICS OF ISPD2006 PLACEMENT BENCHMARKS

Benchmark	#Objs	#Nets	#Movable	#Fixed
adaptec5	843K	868K	842482	646
newblue1	330K	339K	330137	337
newblue2	442K	465K	330239	1277
newblue3	494K	552K	482833	11178
newblue4	646K	637K	642717	3422
newblue5	1233K	1284K	1228177	4881
newblue6	1255K	1288K	1248150	6889
newblue7	2508K	2637K	2481372	26582

TABLE III
STATISTICS OF ISPD2014 PLACEMENT BENCHMARKS

Benchmark	#Objs	#Nets	#Movable	#Fixed
mgc_des_perf_1	112644	112878	112270	374
mgc_des_perf_2	112644	112878	112270	374
mgc_edit_dist_1	130661	133223	128087	2574
mgc_edit_dist_2	130661	133223	128087	2574
mgc_fft	32281	33307	29271	3010
mgc_matrix_mult	155325	158527	150523	4802
mgc_pci_bridge32_1	30675	30835	30314	361
mgc_pci_bridge32_2	30675	30835	30314	361

TABLE IV
STATISTICS OF TILOS MACROPLACEMENT TESTCASES

Benchmark	#Objs	#Nets	#Macros
ariane133	183K	197K	133
ariane136	186K	201K	136
Black_Parrot	250K	275K	196

A. Experiment Settings

GiFt and GiFtPlus are implemented in Python with PyTorch. The coefficients α_0, α_1 and α_2 in (15) are set to 0.1, 0.7 and 0.2, respectively. We integrate GiFt and GiFtPlus with two state-of-the-art placers, RePlAce [2] and GPU-accelerated DREAMPlace [3], to construct GiFt-RePlAce, GiFtPlus-RePlAce and GiFt-DREAMPlace, GiFtPlus-DREAMPlace placement flows. The integration involves using GiFt-generated cell locations as starting points for movable cells in both RePlAce and DREAMPlace. The density constraints follow the specifications provided in the benchmarks. Then these placers complete the placement process, resolving overlaps and producing legalized placement results.

To show the efficacy of GiFt and GiFtPlus, we evaluate HPWL and total runtime on ISPD2005 benchmark suite [21] (see Table I), ISPD2006 benchmark suite [22] (see Table II), ISPD2014 benchmark suite [12] (see Table III) and TILOS MacroPlacement testcases [23] (see Table IV). We compare GiFt-DREAMPlace and GiFtPlus-DREAMPlace against several other competitors, including recently proposed placement initialization work: Chen's work1 [24] and Chen's work2 [25]. To support the ideas proposed in Section IV, we also compare our proposed work with the eigenvector-based initial placement [7]-DREAMPlace (EI-DREAMPlace for short), an initial placement (using the eigenvectors corresponding to the second and third smallest eigenvalues) and placement flow, and GraphPlanner-DREAMPlace [4], a state-of-the-art integrated GCN-based floorplanning and placement flow. Additionally, we illustrate that GiFtPlus can generate informative clusters, meaning that highly connected cells remain intact before and after placement.

RePlAce, GiFt-RePlAce, GiFtPlus-RePlAce, and Chen's works are performed on a workstation with Intel i7-7700 3.6 GHz CPU and 16 GB memory. All other experiments are conducted on a Linux server with 16-core Inter Xeon Gold 6226R @ 2.9 GHz and NVIDIA 2080 Ti GPU.

B. Placement Performance

1) Comparison to the State-of-the-Art Placers: Tables V and VI present the placement results using ISPD2014 benchmarks. The execution time of GiFt and GiFtPlus is less than 5 s on the CPU platform and less than 3 s on the GPU platform. From the result we can see that GiFtPlus-DREAMPlace outperforms both GiFt-DREAMPlace and DREAMPlace alone. Specifically, compared to GiFtPlus-DREAMPlace, GiFt-DREAMPlace and DREAMPlace alone result in 0.5% and 0.1% worse HPWL, require 8% and 44% more placement iterations, and incur 3% and 51% longer total runtime, respectively. Compared to GiFtPlus-RePlAce, GiFt-RePlAce and RePlAce alone result in 0.3% and 0.1% worse HPWL, require 10% and 48% more placement iterations, and incur 2% and 54% longer total runtime, respectively.

We further evaluate the proposed method using larger benchmarks, including ISPD2006 and TILOS MacroPlacement Testcases on CPU platform. As shown in Table VII, with similar HPWL (an average of 0.1% improvement by GiFtPlus), DREAMPlace uses 31% more runtime than GiFtPlus. As shown in Table VIII, with similar HPWL (an average of 0.2% improvement by GiFtPlus), RePlAce uses 87% more runtime than GiFtPlus. These results demonstrate the efficacy of the proposed approach in accelerating the placement process while delivering high-quality placement solutions.

Fig. 5(a) and (b) presents cell locations produced by GiFt and GiFtPlus on mgc_matrix_mult benchmark, respectively. To investigate the effect of GiFt and GiFtPlus, we compare the density curves during placement. As illustrated in Fig. 5(c), both GiFt and GiFtPlus can effectively drive the placer to bypass the time-consuming initialization process and complete placement with significantly fewer iterations. Notably, GiFtPlus outperforms GiFt as it more precisely consider location constraints. Owing to the high efficiency of GiFt and GiFtPlus, the total runtime of the placement process is substantially reduced.

2) Informative Cluster: We further investigate the structural information of the cell locations generated by GiFtPlus. Fig. 6(a1) and (b1) present the locations of movable cells predicted by GiFtPlus on mgc_matrix_mult benchmark and

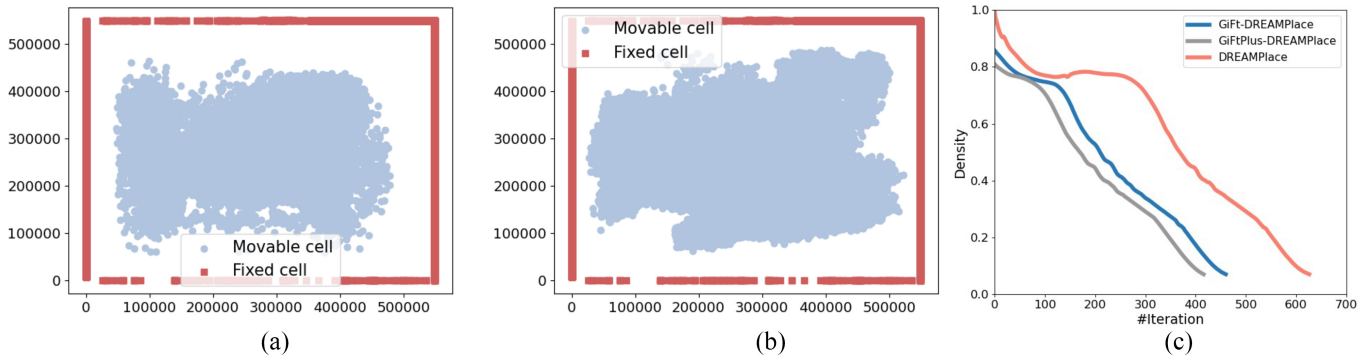

Fig. 5. (a) Cell locations produced by GiFt. (b) Cell locations produced by GiFtPlus. (c) Density curves during placement.

TABLE V
EXPERIMENTAL RESULTS ON ISPD2014 BENCHMARKS USING GPU PLATFORM. VALUES IN PARENTHESES REPRESENT THE TIME SPENT BY GIFT AND GIFTPLUS

	DREAMPlace			GiFt-DREAMPlace			GiFtPlus-DREAMPlace		
Benchmark	HPWL (10^6)	Iteration	Runtime (s)	HPWL (10^6)	Iteration	Runtime (s)	HPWL (10^6)	Iteration	Runtime (s)
mgc_des_perf_1	**5.54**	568	52	5.66	433	31 (0.54)	5.62	**418**	**30 (1.85)**
mgc_des_perf_2	**5.96**	579	50	6.23	447	35 (0.25)	6.01	**436**	**34 (1.70)**
mgc_edit_dist_1	**14.24**	614	64	14.25	438	47 (0.43)	14.25	**404**	**45 (1.91)**
mgc_edit_dist_2	13.98	658	67	13.97	445	49 (0.44)	**13.96**	**412**	**47 (1.93)**
mgc_fft	1.96	579	32	**1.95**	460	**23 (0.43)**	**1.95**	**455**	24 (1.94)
mgc_matrix_mult	10.43	628	62	**10.39**	496	43 (0.17)	**10.39**	**422**	**40 (1.45)**
mgc_pci_bridge32_1	1.13	742	33	1.01	529	22 (0.11)	**1.00**	**473**	**21 (1.43)**
mgc_pci_bridge32_2	1.02	623	32	**1.02**	502	19 (1.08)	**1.02**	**456**	**18 (2.35)**
Ratio	1.001	1.44	1.51	1.005	1.08	1.03	1.000	1.00	1.00

TABLE VI
EXPERIMENTAL RESULTS ON ISPD2014 BENCHMARKS USING CPU PLATFORM. VALUES IN PARENTHESES REPRESENT THE TIME SPENT BY GIFT AND GIFTPLUS

	RePlAce			GiFt-RePlAce			GiFtPlus-RePlAce		
Benchmark	HPWL (10^6)	Iteration	Runtime (s)	HPWL (10^6)	Iteration	Runtime (s)	HPWL (10^6)	Iteration	Runtime (s)
mgc_des_perf_1	**5.43**	492	78	5.45	351	52 (1.79)	5.44	**322**	**48 (3.1)**
mgc_des_perf_2	**5.56**	489	82	5.71	351	56 (1.77)	5.58	**339**	**55 (3.1)**
mgc_edit_dist_1	**13.92**	491	140	**13.92**	346	96 (3.93)	**13.92**	**315**	**93 (4.2)**
mgc_edit_dist_2	13.67	495	141	13.66	348	93 (3.71)	**13.65**	**308**	**92 (4.3)**
mgc_fft	1.93	398	52	**1.93**	325	**30 (3.75)**	**1.93**	**323**	32 (4.6)
mgc_matrix_mult	10.12	440	103	**10.07**	330	72 (1.82)	**10.07**	**301**	**71 (3.1)**
mgc_pci_bridge32_1	0.95	448	49	0.94	349	32 (1.45)	**0.93**	**301**	**30 (2.4)**
mgc_pci_bridge32_2	0.97	450	50	**0.96**	349	32 (1.05)	**0.96**	**299**	**29 (2.9)**
Ratio	1.001	1.48	1.54	1.003	1.10	1.02	1.000	1.00	1.00

TABLE VII
PLACEMENT HPWL (10^7) AND TOTAL RUNTIME (S) ON ISPD2006 BENCHMARKS

	DREAMPlace		GiFt		GiFtPlus	
Benchmark	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
adaptec5	3.00	229	3.02	168	2.99	173
newblue1	5.67	121	5.81	95	5.68	100
newblue2	18.24	150	18.68	110	18.43	119
newblue3	25.65	174	25.52	113	25.56	124
newblue4	21.58	178	21.59	109	21.47	121
newblue5	38.31	288	38.89	201	38.20	221
newblue6	44.00	315	44.29	207	44.01	222
newblue7	93.02	521	95.24	432	93.10	453
Ratio	1.001	1.31	1.014	0.93	1.000	1.00

TABLE VIII
PLACEMENT HPWL (10^9) AND TOTAL RUNTIME (S) RESULTS ON TILOS MACROPLACEMENT TESTCASES

	RePlAce		GiFt-RePlAce		GiFtPlus-RePlAce	
Benchmark	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
ariane133	9.67	248	9.7	162	9.69	171
ariane136	9.58	293	9.61	166	9.57	188
Black_Parrot	12.52	485	12.6	172	12.45	190
Ratio	1.002	1.87	1.006	0.91	1.000	1.00

mgc_pci_bridge32_1 benchmark, respectively. We apply K-Means clustering to the cells based on the locations produced by GiFtPlus, assigning the same color to cells within the same cluster. GiFt-driven placement results are shown in Fig. 6(a2) and (b2). Fig. 6(a3) and (b3) illustrate the placement results produced by DREAMPlace alone.

From Fig. 6(a1)–(a3), we can see that highly connected movable cells remain intact before and after the global placement. With similar placement quality (0.3% HPWL improvement by GiFtPlus), DREAMPlace uses 49% more iterations than GiFtPlus on this benchmark. Fig. 6(b1)–(b3) illustrate that the cell distributions produced by GiFtPlus-DREAMPlace closely match the distributions predicted by GiFtPlus, in contrast to those produced by DREAMPlace alone. In this benchmark, DREAMPlace exhibits performance that is 11% and 57% worse, respectively, in terms of HPWL

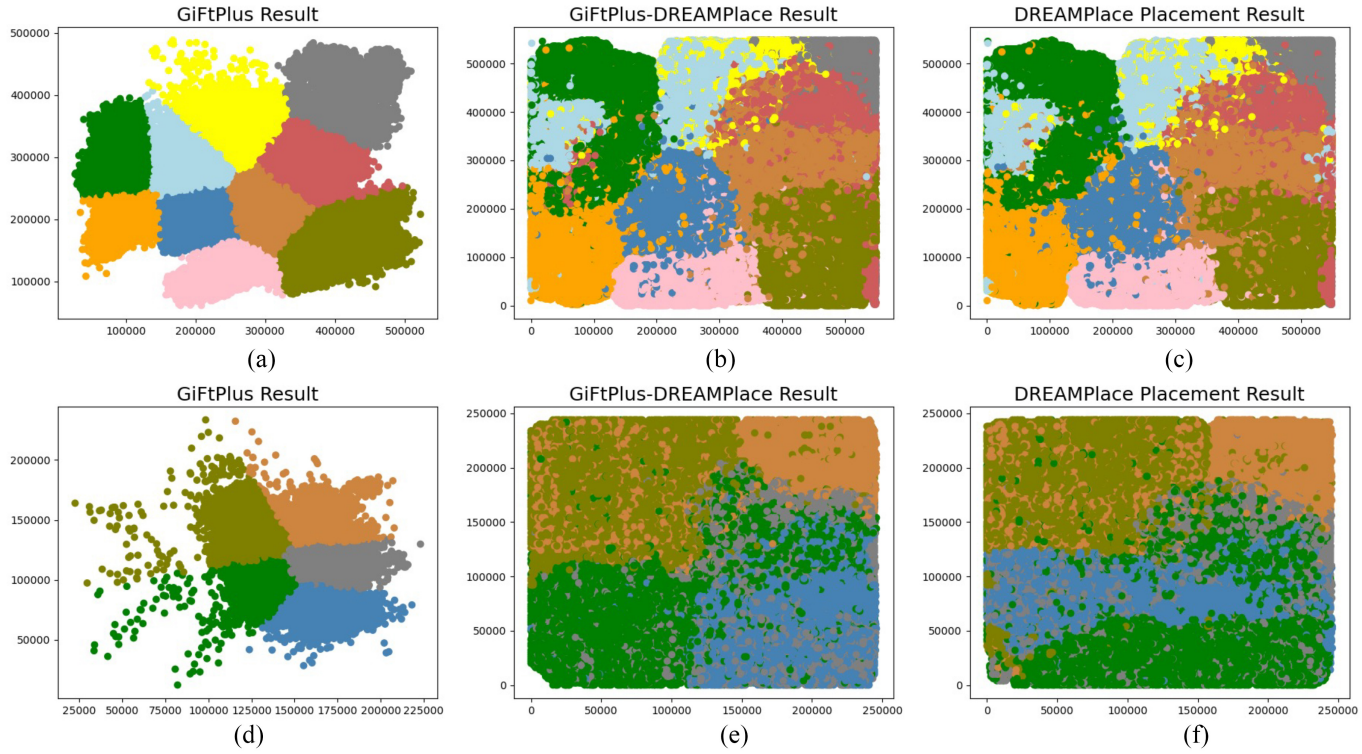

Fig. 6. Visualization of the effect of cluster information predicted by GiFtPlus. (a) The locations of movable cells predicted by GiFtPlus on mgc_matrix_mult benchmark. (b) GiFtPlus-DREAMPlace placement result on mgc_matrix_mult benchmark. (c) DREAMPlace placement result on mgc_matrix_mult benchmark. (d) The locations of movable cells predicted by GiFtPlus on mgc_pci_bridge32_1 benchmark. (e) GiFtPlus-DREAMPlace placement result on mgc_pci_bridge32_1 benchmark. (f) DREAMPlace placement result on mgc_pci_bridge32_1 benchmark.

and the number of placement iterations, when compared to GiFtPlus-DREAMPlace. These results demonstrate that GiFtPlus effectively forms informative clusters that guide downstream placers to produce high-quality placement results with significantly reduced iterations.

3) Comparison to Other Initial Placement Strategies: We compare GiFt and GiFtPlus with recently proposed placement initialization method: Chen's work1 (Projected Eigenvectors) [24] and Chen's work2 (Projected Eigenvectors+SSM) [25]. Since work1 and work2 do not use GPU acceleration, for a fair comparison, We run work1, work2, DREAMPlace, GiFt, and GiFtPlus on the CPU platform. Experimental results in Table IX show that work1 and work2 require 98% and 22x more placement runtime, respectively, compared to GiFtPlus.

Specifically, Chen's work1, Chen's work2 and our proposed method all aim to enhance placement initialization to benefit subsequent placers. These methods all formulate the placement initialization to a quadratic optimization problem. But we have different focuses and use different methodologies to solve it. Chen's work focus on improving the quality of placement results at the expense of increased runtime. They formulate the placement initialization problem as a quadratic-constrained quadratic programming (QCQP) problem with quadratic equality constraints, solving it using an iterative projected-subspace-descent algorithm, which suffers extremely large runtime overhead. On the other hand, our approach focuses on accelerating the placement process. Experimental results demonstrate that the proposed ultrafast placement initialization approach significantly speedup the placement process without compromising placement quality. We will further enhance GiFtPlus to improve its placement quality in future work.

To support the ideas proposed in Section III-C, we further compare GiFt and GiFtPlus with eigenvector-based initial placement and GNN-based initial placement. Table X presents the placement results with different initial placement-placement flows. The GPU version of DREAMPlace is used as the subsequent placer in this experiment. Compared with EI-DREAMPlace and GraphPlanner-DREAMPlace, GiFtPlus-DREAMPlace achieves 0.6% and 1% improvements in HPWL, 16% and 14% reductions in the number of iterations, and 61% and 3× reductions in total runtime. This result reinforces the findings presented in Section III-C and further proves the efficacy of GiFt and GiFtPlus in accelerating the placement process.

4) Runtime Breakdown: We analyze the runtime breakdown of GiFt-DREAMPlace and GiFtPlus-DREAMPlace on newblue7 benchmark, which is the largest design among ISPD2006 benchmarks.

The calculation of GiFt mainly involves multiple sparse matrix multiplications (see Section III-C). It accounts for 3% of the total runtime, and the remaining 97% is spent on placement.

For GiFtPlus, it includes several steps as described in Section V-A. Specifically, 1% of the total runtime is consumed by graph sparsification, 5% by solving two decomposed subproblems, 2% by the location refinement, and the remaining

TABLE IX
PLACEMENT HPWL (10^7) AND TOTAL RUNTIME (S) RESULTS ON ISPD2005 BENCHMARKS. "RANDOM" REPRESENTS THE RANDOM INITIALIZATION, WHICH IS USED AS THE DEFAULT INITIALIZATION APPROACH IN DREAMPLACE. ALL INITIALIZATION APPROACHES ARE INTEGRATED WITH DREAMPLACE

Benchmark	Random		Projected Eigenvectors [24]		Projected Eigenvectors+SSM [25]		GiFt		GiFtPlus	
	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
adaptec1	73.24	112	70.36	161	70.32	2778	73.13	84	72.81	88
adaptec2	82.51	201	81.68	253	81.21	2520	81.49	154	81.55	160
adaptec3	194.12	321	189.13	514	187.95	6205	193.67	246	193.62	260
adaptec4	174.43	402	171.73	530	171.62	5221	176.78	298	173.61	319
bigblue1	89.43	172	87.32	225	87.04	4820	89.53	128	89.93	130
bigblue2	136.88	392	132.49	503	131.37	5955	139.72	291	138.26	310
bigblue3	303.99	1189	298.47	1247	297.31	12715	312.50	860	302.83	878
bigblue4	743.75	1603	726.71	2920	724.78	34802	768.98	1190	744.99	1201
Ratio	1.003	1.31	0.980	1.90	0.977	22.42	1.021	0.97	1.000	1.00

TABLE X
EXPERIMENTAL RESULTS ON ISPD2014 BENCHMARKS. IN THE "EI-DREAMPLACE" COLUMN, THE VALUES IN PARENTHESES REPRESENT THE TIME TAKEN FOR EIGENDECOMPOSITION. IN THE "GRAPHPLANNER-DREAMPLACE" COLUMN, THE VALUES IN PARENTHESES REPRESENT MODEL TRAINING TIME. IN THE "GIFT-DREAMPLACE" COLUMN, THE VALUES IN PARENTHESES REPRESENT THE TIME SPENT BY GIFT. "GIFTPLUS-DREAMPLACE" COLUMN, THE VALUES IN PARENTHESES REPRESENT THE TIME SPENT BY GIFTPLUS

Benchmark	EI-DREAMPlace			GraphPlanner-DREAMPlace			GiFt-DREAMPlace			GiFtPlus-DREAMPlace		
	HPWL (10^6)	Iteration	Runtime (s)	HPWL (10^6)	Iteration	Runtime (s)	HPWL (10^6)	Iteration	Runtime (s)	HPWL (10^6)	Iteration	Runtime (s)
mgc_des_perf_1	5.65	481	57 (7)	5.93	454	33	5.66	433	31 (0.54)	5.62	**418**	**30 (1.85)**
mgc_des_perf_2	**5.99**	495	59 (7)	6.11	460	30	6.23	447	35 (0.25)	6.01	**436**	**34 (1.70)**
mgc_edit_dist_1	14.26	473	66 (4)	14.41	500	44	**14.25**	438	47 (0.43)	14.25	**404**	**45 (1.91)**
mgc_edit_dist_2	13.97	490	68 (4)	14.11	491	54	13.97	445	49 (0.44)	**13.96**	**412**	**47 (1.93)**
mgc_fft	1.96	498	52 (15)	2.02	462	22	**1.95**	460	23 (0.43)	**1.95**	**455**	24 (1.94)
mgc_matrix_mult	10.41	539	59 (2)	10.52	498	41	**10.39**	496	43 (0.17)	**10.39**	**422**	**40 (1.45)**
mgc_pci_bridge32_1	1.14	535	28 (2)	1.08	604	26	1.01	529	22 (0.11)	**1.00**	**473**	**21 (1.43)**
mgc_pci_bridge32_2	1.15	533	28 (2)	1.03	511	20	1.02	502	19 (0.10)	**1.02**	**456**	**18 (2.35)**
Ratio	1.006	1.16	1.61	1.022	1.14	3.27 (+0.16h)	1.005	1.08	1.03	1.000	1.00	1.00

92% is due to the placement. Note that two decomposed subproblems (L^0 and L^1) can be solved in parallel. Of the total runtime, 3% is spent on solving L^0 using GiFt, and 5% is spent on solving L^1 using convex quadratic optimization. Therefore, solving both subproblems together accounts for 5% of the total runtime.

VII. DISCUSSION

This section provides the theoretical foundations about why GiFt and GiFtPlus can drive the gradient descent-based method (e.g., DREAMPlace) to efficiently solve the placement problem.

The placement problem is a nonconvex nonlinear optimization problem. Gradient-descent methods, such as DREAMPlace and RePlAce require a large number of optimization iterations and are prone to falling into local optima. A critical factor influencing the solution quality of these gradient-descent methods is their dependency on the initial solution [26]. Our GSP-based method provides an optimized starting point for subsequent placers, that is, it allows gradient descent-based placers to begin the optimization process from a more advantageous position. As a result, the placement problem can be tackled more effectively, leading to high-quality solutions with a significantly reduced number of iterations.

Specifically, conventional analytical placers heuristically set the initial locations of movable cells at the center of the placement region with minimum wirelength yet high overlap, and spread cells out based on an iteratively increasing density penalty. Such an initialization process requires a long iteration time. Our GSP-based method streamlines this process. It can efficiently predict optimized cell locations, characterized by both locally and globally smooth graph signals. As demonstrated in this article, this methodology aligns with the objective of optimizing total wirelength while maintaining low overlap. The predicted cell locations serve as the starting point for subsequent placers, driving them to bypass the time-consuming initialization phase and efficiently converge to the high-quality solutions. By addressing the limitations of conventional methods and providing a robust starting point for optimization, the GSP-based approach demonstrates a clear advantage in solving complex, nonconvex optimization problems in chip placement.

VIII. CONCLUSION

This article presents the power of GSP in accelerating chip placement. By underscoring the significance of signal smoothness in addressing placement problems, we propose GiFt and its enhanced version GiFtPlus, highly efficient placement speedup techniques designed to leverage multiresolution smooth signals for the generation of high-quality initial placement solutions. By integrating GiFt and GiFtPlus with analytical placers, they can substantially reduce placement optimization iterations without the need for time-consuming model training. Experimental results demonstrate that GiFt-Placer and GiFtPlus-Placer consistently achieve competitive or superior performance compared to state-of-the-art placers. In particular, it significantly enhances placement efficiency and even outperforms analytical placers running on GPU versions.

REFERENCES

[1] J. Lu et al., "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, vol. 20, no. 2, pp. 1–34, 2015.

[2] C. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1717–1730, Sep. 2019.

[3] J. Gu, Z. Jiang, Y. Lin, and D. Z. Pan, "DREAMPlace 3.0: Multi-electrostatics based robust VLSI placement with region constraints," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design.*, 2020, pp. 1–9.

[4] Y. Liu et al., "Graphplanner: Floorplanning with graph neural network," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 2, pp. 1–24, 2022.

[5] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 1–12.

[6] A. Mirhoseini et al., "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[7] K. M. Hall, "An *r*-dimensional quadratic placement algorithm," *Manag. Sci.*, vol. 17, no. 3, pp. 219–229, 1970.

[8] J. Frankle and R. Karp, "Circuit placements and cost bounds by eigenvector decomposition," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design.*, 1986, pp. 414–417.

[9] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[10] J. Liu et al., "Personalized graph signal processing for collaborative filtering," in *Proc. ACM Web Conf.*, 2023, pp. 1264–1272.

[11] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871.

[12] V. Yutsis, I. S. Bustany, D. Chinnery, J. R. Shinnerl, and W.-H. Liu, "ISPD 2014 benchmarks with sub-45nm technology rules for detailed-routing-driven placement," in *Proc. Int. Symp. Phys. Design*, 2014, pp. 161–168.

[13] N. Viswanathan and C. C.-N. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 5, pp. 722–733, May 2005.

[14] G. M. Morton, *A Computer Oriented Geodetic Data Base; and A New Technique in File Sequencing*, IBM Ltd., Ottawa, ON, Canada, 1966.

[15] N. Viswanathan and C. C.-N. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 5, pp. 722–733, May 2005.

[16] Z. Liu, W. Yu, and Z. Feng, "FeGRASS: Fast and effective graph spectral sparsification for scalable power grid analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 3, pp. 681–694, Mar. 2022.

[17] F. Chung, "Random walks and local cuts in graphs," *Linear Algebra Appl.*, vol. 423, no. 1, pp. 22–32, 2007.

[18] Y. Liu, H. Zhou, J. Wang, F. Yang, X. Zeng, and L. Shang, "Hierarchical graph learning-based floorplanning with dirichlet boundary conditions," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 5, pp. 810–822, May 2024.

[19] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, 1st ed. Dordrecht, The Netherlands: Springer, 2011.

[20] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete graph problems," *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.

[21] G. J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. C. Yildiz, "The ISPD2005 placement contest and benchmark suite," in *Proc. Int. Symp. Phys. Design (ISPD)*, San Francisco, CA, USA, 2005, pp. 216–220.

[22] G.-J. Nam, "ISPD 2006 placement contest: Benchmark suite and results," in *Proc. ISPD* 2006, p. 167.

[23] "TILOS MacroPlacement repository."Accessed: May 14, 2024. [Online]. Available: https://github.com/TILOS-AI-Institute/MacroPlacement.

[24] P. Chen, C.-K. Cheng, A. Chern, C. Holtz, A. Li, and Y. Wang, "Placement initialization via a projected eigenvector algorithm: Late breaking results," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, 2022, pp. 1398–1399.

[25] P. Chen, C.-K. Cheng, A. Chern, C. Holtz, A. Li, and Y. Wang, "Placement initialization via sequential subspace optimization with sphere constraints," in *Proc. Int. Symp. Phys. Des.*, 2023, pp. 133–140.

[26] J. Lu et al., "ePlace-MS: Electrostatics-based placement for mixed-size circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 685–698, May 2015.

Yiting Liu received the B.S. degree from Northwest A&F University, Xianyang, China, in 2017, the M.S. degree from the Beijing Institute of Technology, Beijing, China, in 2020, and the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2024.

She is a Postdoctoral Scholar with the University of California at San Diego, San Diego, CA, USA. Her current research interests include electronic design automation, VLSI physical design, and machine learning.

Hai Zhou received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer sciences from the University of Texas at Austin, Austin, TX, USA, in 1999.

He is a Professor of Electrical And Computer Engineering with Northwestern University, Evanston, IL, USA. He has published more than 180 papers in flagship journals and conferences in these areas. His research interests include VLSI computer-aided design, formal methods, and hardware security.

Prof. Zhou was a recipient of a CAREER Award from the National Science Foundation.

Jia Wang (Senior Member, IEEE) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2002, and the Ph.D. degree in computer engineering from Northwestern University, Evanston, IL, USA, in 2008.

He is currently an Associate Professor of Electrical And Computer Engineering with Illinois Institute of Technology, Chicago, IL, USA. His current research interests include computer-aided design of very large scale integrated circuits and algorithm design.

Fan Yang (Member, IEEE) received the B.S. degree from Xi'an Jiaotong University, Xi'an, China, in 2003, and the Ph.D. degree from Fudan University, Shanghai, China, in 2008.

He is currently a Full Professor with the Microelectronics Department, Fudan University. His research interests include model order reduction, circuit simulation, high-level synthesis, acceleration of artificial neural networks, yield analysis, and design for manufacturability.

Xuan Zeng (Senior Member, IEEE) received the B.S. and Ph.D. degrees in electrical engineering from Fudan University, Shanghai, China, in 1991 and 1997, respectively.

She is currently a Full Professor with the Microelectronics Department, Fudan University, where she served as the Director of the State Key Laboratory of Application Specific Integrated Circuits and Systems from 2008 to 2012. She was a Visiting Professor with the Department of Electrical Engineering, Texas A&M University at College Station, College Station, TX, USA, and the Microelectronics Department, Technische Universiteit Delft, Delft, The Netherlands, in 2002 and 2003, respectively. Her current research interests include analog circuit modeling and synthesis, design for manufacturability, high-speed interconnect analysis and optimization, and circuit simulation.

Prof. Zeng received the Changjiang Distinguished Professor with the Ministry of Education Department of China in 2014, the Chinese National Science Funds for Distinguished Young Scientists in 2011, the First-Class of Natural Science Prize of Shanghai in 2012, the 10th For Women in Science Award in China in 2013, and the Shanghai Municipal Natural Science Peony Award in 2014. She received the Best Paper Award from the 8th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference 2017. She is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and the *ACM Transactions on Design Automation of Electronic Systems*.

Li Shang (Member, IEEE) received his B.S. and M.S. degrees from Tsinghua University, Beijing, China in 1997 and 1999, respectively, and the Ph.D. from Princeton University, Princeton, NJ, USA, in 2004.

He is a Professor with the School of Computer Science, Fudan University, Shanghai, China. He was the Deputy Director and the Chief Architect of Intel Labs China, Shanghai, China, and an Associate Professor with the University of Colorado Boulder, Boulder, CO, USA. His research interests include embedded systems, machine learning, and EDA & VLSI. He has published over 180 peer-reviewed papers, received multiple best paper awards and nominations, and has been cited over 9,000 times.

Prof. Shang was a recipient of the NSF Career Award.