A Customized Graph Neural Network Model for Guiding Analog IC Placement

Yaguang Li[1], Yishuang Lin[2], Meghna Madhusudan[3], Arvind Sharma[3], Wenbin Xu[1], Sachin S. Sapatnekar[3], Ramesh Harjani[3], Jiang Hu[1,2]
[1]Dept. of Electrical and Computer Engineering, Texas A&M University
[2]Dept. of Computer Science and Engineering, Texas A&M University
[3]Dept. of Electrical and Computer Engineering, University of Minnesota
{liyg, lionlin, wbxu, jianghu}@tamu.edu; {madhu028, aksharma, sachin, harjani}@umn.edu

ABSTRACT

Analog IC placement is typically a manual process that requires strong experience and trial-and-error iterations as it produces a large impact to circuit performance in a complicated manner. Although automatic analog placement has been studied for decades, existing methods are inadequate for achieving performance comparable with manual designs. In this work, a customized graph neural network model is developed for predicting the impact of placement on circuit performance. Knowledge obtained by such a model can be transferred among different topologies of the same circuit type. Simulation results show that the proposed model is superior to a recent CNN-based work in terms of both accuracy and knowledge transfer. It also outperforms a plug-in use of graph attention network. The proposed model is further applied in analog IC placement and achieves performance similar to manual designs.

1 INTRODUCTION

Analog IC performance highly depends on RC parasitics that are largely decided by circuit layout. The dependence can be very complicated as multiple competing performance characteristics are affected by numerous RC elements (wire segments) in an intertwined manner. As a result, analog IC layout design is still by and large a manual art. Obtaining a good analog circuit layout often requires multiple design iterations by experienced designers.

Automatic analog IC placement has been studied for decades [1–14]. One focus of analog placement is the enforcement of certain geometric constraints [3, 5, 7], such as symmetry and common centroid. Although such constraints help improve tolerance to variations, they do not directly address nominal performance. In [1], performance specifications are transformed to geometric constraints, such as the maximum length for a wire segment. However, the complicated dependence of performance on layout is overly simplified in the transformation and therefore the obtained geometric constraints are either unnecessarily tight or insufficient for performance guarantee. Performance is directly optimized in analog placement of [2]. However, its performance models are linear approximation and thus cannot handle many nonlinear behaviors of analog circuits. Some geometric constraints that are directly associated with certain performance characteristics are considered in [6, 8], but they are difficult to be extended to general performance metrics. Design knowledge reuse is achieved by retaining legacy design patterns [10] without directly considering performance. Till nowadays, adoption of automatic analog placement in industry has been rare as its promise on performance is still quite weak.

Recently, various machine learning techniques have been explored for analog circuit synthesis [15–20]. In [15], GNN (Graph Neural Network) is applied to produce layout templates for passive elements in RF circuits. A GAN (Generative Adversarial Network)-based well generation technique is proposed for analog circuit designs [16]. Variational auto-encoder is employed in [17] to learn from manual layout and provide routing guidance. The work of [18] makes use of GCN (Graph Convolutional Network) to recognize analog sub-circuits. A CNN (Convolutional Neural Network)-based analog circuit performance model is introduced in [19]. Analog transistor sizing is automated by GCN and reinforcement learning [20].

In this work, we attempt to make progress toward performance driven analog IC placement. A machine learning model, called PEA (Pooling with Edge Attention) network, is developed for predicting circuit performance from a placement solution. Instead of plug-in use of existing machine learning techniques, PEA network is a customized graph neural network that incorporates GAT (Graph Attention Network) [21] and DiffPool [22] as key ingredients. Knowledge obtained by a PEA network can be transferred among different topologies of the same circuit type. Simulation results show that PEA significantly outperforms the recent CNN (Convolutional Neural Network)-based model [19]. It is applied to guide analog placement and achieves performance similar to manual designs. The contributions of this work include the following.

- A GNN (Graph Neural Network) model is developed for predicting analog circuit performance for a given placement solution.
- This is a remarkably customized model instead of plug-in use of existing machine learning techniques.
- The proposed model allows knowledge transfer among different topologies of a type of analog circuits.
- The proposed model significantly outperforms the latest previous work [19] on prediction accuracy, knowledge transfer and training time. It is also more accurate than a plug-in use of GAT, which is one of the most influential GNN techniques.
- The model is applied in a performance driven analog placement and achieves performance similar to manual designs, but is orders of magnitude faster.

The rest of this paper is organized as follows. Background of GNN is briefly introduced in Section 2. The proposed machine learning model for circuit performance prediction is described in Section 3. Section 4 is focused on analog placement guided by

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415624

machine learning. Experimental results are shown in Section 5. Conclusions are provided in Section 6.

2 BACKGROUND ON GRAPH NEURAL NETWORKS

Due to the prevalent use of graph model in many computing applications, people strive to extend the success of CNN into graph domain. Many Graph Neural Network (GNN) techniques have been developed and a survey is provided in [23].

A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ consists of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. The number of nodes is denoted by $n = |\mathcal{V}|$. The edges can be represented by an adjacency matrix $A \in \{0,1\}^{n \times n}$. Each node $v_i \in V$ is associated with a vector of features $(x_1, x_2, ..., x_d)$. The features for all nodes form a matrix $X \in \mathbb{R}^{n \times d}$. A trained GNN takes X as input and decides the class of the entire graph or the class of every node in the graph. Next, we introduce two important techniques, attention-based graph convolution and graph pooling, both of which are employed in our customized GNN.

2.1 Attention-Based Graph Convolution

Graph convolution is an extension to the spatial convolution in CNNs. Its key concept is **aggregation**. That is, a node $v_i \in \mathcal{V}$ collects feature information of its neighboring nodes. Before aggregation, usually a transformation is performed as XW, where $W \in \mathbb{R}^{d \times d}$ is a trainable weight matrix. Then, the framework of aggregation is $\Phi_A XW$, where $\Phi_A \in \mathbb{R}^{n \times n}$ is a matrix depending on A or edge connections. The form of Φ_A varies among different GNN techniques. A **node embedding** is obtained as

$$Z^{(1)} = \sigma(\Phi_A XW)$$

where $\sigma(\cdot)$ is an activation function, such as sigmoid function. Usually, this procedure is repeated with multiple iterations and generates multiple layers of embedding $Z^{(0)} = X^{(1)} = X, Z^{(1)} = X^{(2)}, Z^{(2)} = X^{(3)}$.... A generic graph convolution operation is described as

$$Z^{(l)} = \sigma(\Phi_A^{(l)} X^{(l)} W^{(l)}) \tag{1}$$

where l is layer index. Please note $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$, which implies that feature dimension d can change from layer to layer according to different configurations.

The form of Φ_A is elaborated through the recently popular Graph Attention Network (GAT) [21]. The **attention** coefficient α_{ij} from node v_j to v_i is defined by

$$\begin{aligned} \alpha_{ij} &= \text{softmax}_{row}(\tau_{ij}) = \frac{e^{\tau_{ij}}}{\sum_{k \in \mathcal{N}_i} e^{\tau_{ik}}} \\ \tau_{ij} &= \text{LeakyReLU}(a^{(l)} \cdot [(W^{(l)^{\mathrm{T}}} X_i^{(l)}) || (W^{(l)^{\mathrm{T}}} X_j^{(l)})]) \end{aligned} \tag{2}$$

where $a^{(l)} \in \mathbb{R}^{2d_{l+1}}$ is a trainable weight vector, $X_i^{(l)}$ is a vector corresponding to node v_i, $\mathcal{N}_i$ is the neighborhood of node v_i, $\cdot$ is vector inner product operation, T means vector transposition and $||$ is vector concatenation operation. Please note the softmax here is row-wise as the k in the denominator enumerates columns for row i. LeakyReLU is a nonlinear function defined by

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ cx & x < 0 \end{cases} \tag{3}$$

where $c \in [0, 1)$ is a parameter. The attention-based **graph convolution** is described by

$$Z^{(l)} = \sigma(\alpha X^{(l)} W^{(l)}) \tag{4}$$

where $\alpha \in \mathbb{R}^{n \times n}$ is a matrix with $\alpha_{ij}, i, j = 1, 2, ..., n$ as its entries.

2.2 Graph Pooling

This is a migration of the pooling operation in CNN into graph domain and a well-known approach is called DiffPool [22]. The pooling here is similar to graph clustering and a cluster forms a new node for the next iteration/layer. At iteration l, the number of nodes is changed from n_l to n_{l+1}, where $n_{l+1} < n_l$ and $n_0 = n$. The clustering is realized through an assignment matrix $S^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$, where each row corresponds to a node at layer l and each column indicates a cluster (new node) for layer $l+1$. Therefore, the entry in row i and column j of $S^{(l)}$ is the probability of assigning node $v_i^{(l)}$ into cluster $v_j^{(l+1)}$, i.e., this is soft clustering. The assignment matrix of layer l is defined as

$$S^{(l)} = \text{softmax}_{row}[\sigma(\tilde{D}^{(l)^{-\frac{1}{2}}} \tilde{A}^{(l)} \tilde{D}^{(l)^{-\frac{1}{2}}} X^{(l)} W_{pool}^{(l)})] \tag{5}$$

where $W_{pool}^{(l)} \in \mathbb{R}^{d_l \times n_{l+1}}$ is a trainable weight matrix. In addition, $\tilde{A}^{(l)} = A^{(l)} + \mathbb{I}$, where $\mathbb{I}$ indicates identity matrix, and $\tilde{D}^{(l)} \in \mathbb{R}^{n_l \times n_l}$ is a diagonal matrix, where $\tilde{D}_{ii}^{(l)} = \sum_{j=1}^{n_l} \tilde{A}_{ij}^{(l)}$. Each element on the diagonal of $\tilde{D}^{(l)}$ represents the indegree of corresponding node. The row-wise softmax operation for all entries in the matrix guarantees that each node in layer l is assigned to clusters in layer $l+1$ with probabilities sum up to one.

Pooling operation is to aggregate embedding $Z^{(l)}$ into next layer by

$$X^{(l+1)} = S^{(l)^{\mathrm{T}}} Z^{(l)} \tag{6}$$

and then perform soft clustering by

$$A^{(l+1)} = S^{(l)^{\mathrm{T}}} A^{(l)} S^{(l)} \tag{7}$$

The pooling operation is often applied along with graph convolution at each layer.

3 PEA: A CUSTOMIZED GNN MODEL

We describe a customized GNN (Graph Neural Network) model, called **PEA**, which is the acronym for Pooling with Edge Attention. It takes an analog placement solution as input and predicts if its post-routing performance is satisfactory. Examples of performance characteristics include the gain and phase margin of an OTA (Operational Transconductance Amplifier). The performance labels in training data are obtained from post-layout circuit simulation results.

GNN versus CNN? In [19], a model of similar purpose employs CNN (Convolutional Neural Network) and takes placement images as its features. Arguably, GNN is superior in capturing the netlist topology, which is a graph. Moreover, GNN is more efficient in feature encoding. For instance, the shape of a transistor can be represented by two real numbers (width and height) in GNN while it requires an array of pixels for CNN. The spatial features can be easily embraced in GNN by taking the location coordinates as features. These observations motivate us to take a GNN approach.

3.1 Circuit Graph and Features

A netlist of circuit could be naturally encoded into a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, in which devices and IO pins are the graph nodes $\mathcal{V}$ and the connections between devices are the graph edges $\mathcal{E}$. Fig. 2 shows an example of encoding a netlist of 5T OTA (5-transistor Operational Transconductance Amplifier) into a circuit graph.

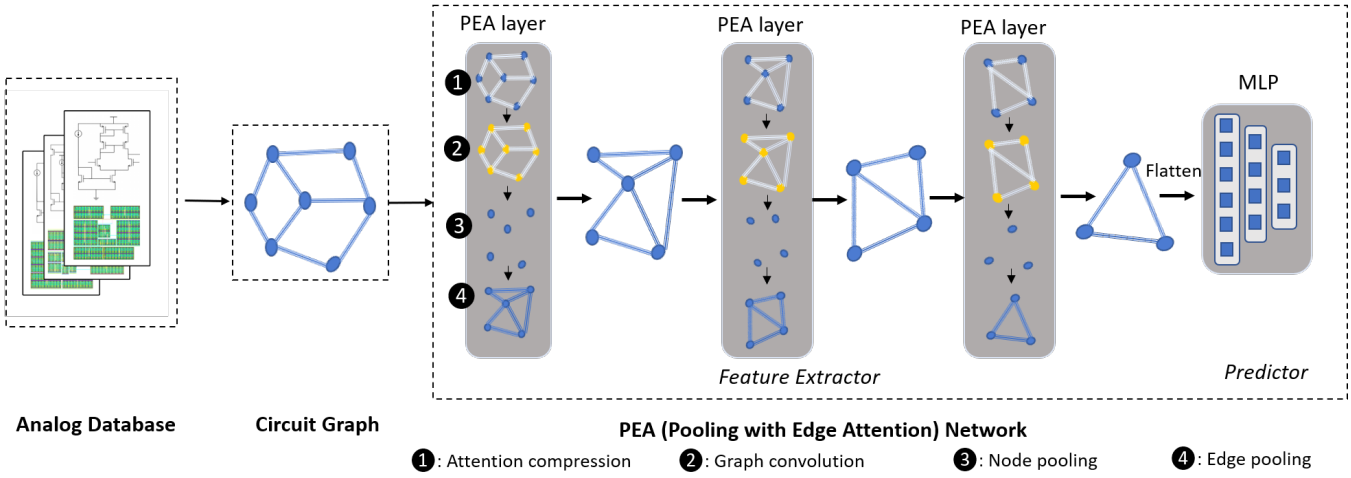


Figure 1: Overview of PEA network. MLP: multi-layer perceptron.

The graph is represented in the form of an adjacency matrix $A \in [0,1]^{n \times n}$, a node feature matrix $X \in \mathbb{R}^{n \times d}$ and an edge feature matrix $E \in \mathbb{R}^{n \times n \times p}$, where $n = |\mathcal{V}|$ is the number of nodes, d is the number of features for each node and p is the number of features for each edge. Please note that each entry in an adjacency matrix is conventionally 0 or 1. In PEA, soft clustering is performed and the adjacency between two nodes can be a probability.

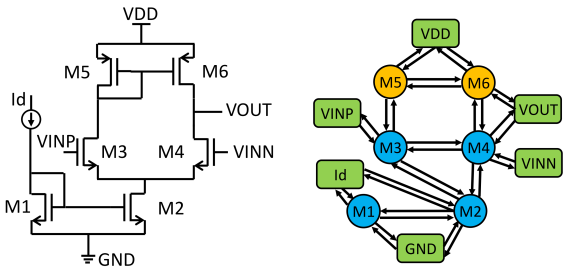


Figure 2: 5-transistor OTA and its graph encoding.

In the node feature matrix, $X_i \in \mathbb{R}^d, i = 1, 2, ..., n$, represents the feature vector of the i-th node. The d features include the following.

- Device type: PMOS, NMOS, capacitor, current source, GND, etc;
- Functional module where the device belongs to, such as bias current mirror, differential pair and active load;
- Device dimension;
- Device location.

In the edge feature matrix, $E_{ij} \in \mathbb{R}^p, i, j = 1, 2, ..., n$, represents the features of the edge from node j to node i. A node (device) often has multiple pins. We use pin_i^h and pin_j^u to indicate the pin of device $v_i \in \mathcal{V}$ and the pin of device $v_j \in \mathcal{V}$ that are connected. The p features include the following.

- Horizontal and vertical distance between pin_i^h and pin_j^u;
- Metal layer of pin_i^h and pin_j^u;
- Length of pin_i^h and pin_j^u;
- Type of pin_i^h and pin_j^u, such as transistor source, drain, gate, etc.

Intuitively, as circuit performance is affected by both node features (transistor's size/dimension, etc.) and edge features (distance between two transistors' pins, etc.), we apply both of them into our PEA network's attention mechanism. In this way, informations of neighbouring nodes are aggregated according to their node features and connection relationships.

3.2 PEA (Pooling with Edge Attention) Network

The proposed PEA network is composed by two stages: feature extractor and predictor as shown in Figure 1. The extractor consists of multiple PEA layers, each of which includes graph convolution and graph pooling. The predictor is a fully-connected neural network, a.k.a. multi-layer perceptron (MLP). Since MLP is relatively well-known, the description will be focused on the extractor.

The key ingredient of PEA network is the integration between edge feature/attention and graph pooling. The pooling, which is briefly introduced in Section 2.2, is to comprehend circuit hierarchy. The edge feature/attention is to capture connections among devices. Edge feature/attention [24] is not simple extension from those of nodes. Moreover, it has not been integrated with graph pooling [22], which is restricted to node features. A main contribution of PEA network is to enable pooling for edge feature/attention. A PEA network is composed by four phases:

(1) Edge-aware attention construction and compression.
(2) Graph convolution.
(3) Node pooling.
(4) Edge pooling.

Our customization is mainly at phases 1 and 4, while phases 2 and 3 are briefly described here for completeness. An overview of one PEA layer is summarized in Algorithm 1.

3.2.1 Edge-Aware Attention Construction and Compression. Edge attention is proposed in [24] by expanding attention matrix from 2D (see Section 2.1) to 3D so that $\alpha^{(l)} \in \mathbb{R}^{n_l \times n_l \times p_l}$, where the 3rd dimension corresponding to p_l is for edge features and called *channel*. While this approach has its merit, it is quite expensive in terms of both runtime and memory use. In order to overcome this

Algorithm 1 Pooling with Edge Attention (PEA) Layer

Inputs: $A^{(l)} \in \mathbb{R}^{n_l \times n_l}$, $X^{(l)} \in \mathbb{R}^{n_l \times d_l}$, $E^{(l)} \in \mathbb{R}^{n_l \times n_l \times p_l}$, n_{l+1}, d_{l+1}, p_{l+1}

Outputs: $A^{(l+1)} \in \mathbb{R}^{n_{l+1} \times n_{l+1}}$, $X^{(l+1)} \in \mathbb{R}^{n_{l+1} \times d_{l+1}}$, $E^{(l+1)} \in \mathbb{R}^{n_{l+1} \times n_{l+1} \times p_{l+1}}$

1: Initialize $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$, $W^{(l)}_{pool} \in \mathbb{R}^{d_l \times n_{l+1}}$, $W^{(l)}_{edge} \in \mathbb{R}^{p_l \times p_{l+1}}$, $a^{(l)} \in \mathbb{R}^{2d_{l+1}}$, $b^{(l)} \in \mathbb{R}^{p_l}$

2: Edge-aware attention: calculating attention coefficients $\hat{\alpha}^{(l)}_{ijk}$=LeakyReLU($a^{(l)} \cdot [W^{(l)\mathrm{T}} X^{(l)}_i || W^{(l)\mathrm{T}} X^{(l)}_j]) E^{(l)}_{ijk}$

3: Row-wise normalization $\tilde{\alpha}^{(l)}_{ijk} = \mathrm{softmax}_{row}(\hat{\alpha}^{(l)}_{ijk})$

4: Col-wise normalization $\alpha^{(l)}_{ijk} = \sum_{m=1}^{n_l} \frac{\tilde{\alpha}^{(l)}_{imk} \tilde{\alpha}^{(l)}_{jmk}}{\sum_{u=1}^{n_l} \tilde{\alpha}^{(l)}_{umk}}$

5: Graph convolution: node embedding $Z^{(l)} = \sigma(g(\alpha^{(l)}; b^{(l)}) X^{(l)} W^{(l)}) \in \mathbb{R}^{n_l \times d_{l+1}}$

6: $\tilde{A}^{(l)} = A^{(l)} + \mathbb{I}$

7: $\tilde{D}^{(l)}_{ii} = \sum_{j=1}^{n_l} \tilde{A}^{(l)}_{ij}$, $\tilde{D}^{(l)} \in \mathbb{R}^{n_l \times n_l}$

8: Assignment matrix $S^{(l)} = \mathrm{softmax}_{row}[\sigma(\tilde{D}^{(l)^{-\frac{1}{2}}} \tilde{A}^{(l)} \tilde{D}^{(l)^{-\frac{1}{2}}} X^{(l)} W^{(l)}_{pool})] \in \mathbb{R}^{n_l \times n_{l+1}}$

9: Node pooling: generating new adjacency matrix $A^{(l+1)} = S^{(l)\mathrm{T}} A^{(l)} S^{(l)} \in \mathbb{R}^{n_{l+1} \times n_{l+1}}$

10: Node pooling: generating new node feature matrix $X^{(l+1)} = S^{(l)\mathrm{T}} Z^{(l)} \in \mathbb{R}^{n_{l+1} \times d_{l+1}}$

11: Edge pooling: edge-feature-encoded attention $Q^{(l)}_{ijk} = \sum_{m=1}^{p_l} \alpha^{(l)}_{ijm} {W_{edge}}^{(l)}_{mk}$, $Q^{(l)} \in \mathbb{R}^{n_l \times n_l \times p_{l+1}}$

12: Edge pooling: generating new edge feature matrix $E^{(l+1)} = ||_{k=1}^{p_{l+1}} (S^{(l)\mathrm{T}} Q^{(l)}_{\cdot\cdot k} S^{(l)}) \in \mathbb{R}^{n_{l+1} \times n_{l+1} \times p_{l+1}}$

drawback, we suggest a new edge-aware attention model, where the raw attention is defined as

$$\hat{\alpha}^{(l)}_{ijk} = f^{(l)}(X^{(l)}_i, X^{(l)}_j, E^{(l)}_{ijk}) = \tau_{ij} E^{(l)}_{ijk} \tag{8}$$

and τ_{ij} is given by Equation (2). This is step 2 of Algorithm 1. Then, attention matrix is obtained through bidirectional normalization (BN) as

$$\alpha^{(l)} = BN(\hat{\alpha}^{(l)}) = \begin{cases} \tilde{\alpha}^{(l)}_{ijk} = \mathrm{softmax}_{row}(\hat{\alpha}^{(l)}_{ijk}) \\ \alpha^{(l)}_{ijk} = \sum_{m=1}^{n_l} \frac{\tilde{\alpha}^{(l)}_{imk} \tilde{\alpha}^{(l)}_{jmk}}{\sum_{u=1}^{n_l} \tilde{\alpha}^{(l)}_{umk}} \end{cases} \tag{9}$$

The normalization corresponds to steps 3 and 4 in Algorithm 1. It avoids computing overflow from multiplication and guarantees that in each channel k, the sum in each row and each column of $\alpha^{(l)}$ is 1.

We further compress the 3D attention matrix into 2D by a compression operator $g : \mathbb{R}^{p_l} \to \mathbb{R}$ defined as

$$e_{ij} = g(\alpha^{(l)}_{ij}; b^{(l)}) = \sum_{k=1}^{p_l} \alpha^{(l)}_{ijk} b^{(l)}_k \tag{10}$$

where $b^{(l)} \in \mathbb{R}^{p_l}$ is a trainable weight vector.

3.2.2 Graph Convolution. After the attention compression, graph convolution is perform in the same way as conventional approach (Section 2.1) except that the attention is replaced by the compressed one $g(\alpha^{(l)}; b^{(l)})$ so that

$$Z^{(l)} = \sigma(g(\alpha^{(l)}; b^{(l)}) X^{(l)} W^{(l)}) \tag{11}$$

Step 5 of Algorithm 1 covers this convolution operation as well as the compression by Equation (10).

3.2.3 Node Pooling. This phase is the same as DiffPool [22], which is summarized by steps 6-10 in Algorithm 1. First, the assignment matrix $S^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ is obtained according to Equation (5). Then, a new node feature matrix $X^{(l+1)}$ and a new adjacency matrix $A^{(l+1)}$ are computed by Equations (6) and (7), respectively.

3.2.4 Edge Pooling. Edge pooling is a new technique that has not been seen in previous work, to the best of our knowledge. We developed an edge pooling technique, which can be decomposed to two sub-steps: channel pooling and node-space pooling. Please note the node-space here is for edge features and hence the node-space pooling for edge features is different from node pooling described by Equation (6).

The channel pooling is designed to be $h : \mathbb{R}^{p_l} \to \mathbb{R}^{p_{l+1}}$

$$\begin{aligned} Q^{(l)}_{ij} &= h(\alpha^{(l)}_{ij}; W^{(l)}_{edge}) \\ Q^{(l)}_{ijk} &= \sum_{m=1}^{p_l} \alpha^{(l)}_{ijm} {W_{edge}}^{(l)}_{mk} \end{aligned} \tag{12}$$

where $Q^{(l)} \in \mathbb{R}^{n_l \times n_l \times p_{l+1}}$ is edge-feature-encoded attention and $W^{(l)}_{edge} \in \mathbb{R}^{p_l \times p_{l+1}}$ is a trainable weight matrix. This transformation (step 11 in Algorithm 1) changes the channel dimension from p_l for attention $\alpha^{(l)}$ to p_{l+1} for $Q^{(l)}$. Since attention α incorporates edge feature information in Equation (8), so does Q.

Based on the edge-feature-encoded attention $Q^{(l)}$, the node-space pooling for edge features is designed to be $t : \mathbb{R}^{n_{l+1} \times n_l} \times \mathbb{R}^{n_l \times n_l \times p_{l+1}} \times \mathbb{R}^{n_l \times n_{l+1}} \to \mathbb{R}^{n_{l+1} \times n_{l+1} \times p_{l+1}}$:

$$\begin{aligned} E^{(l+1)} &= t(S^{(l)\mathrm{T}}, Q^{(l)}, S^{(l)}) \\ &= ||_{k=1}^{p_{l+1}} (S^{(l)\mathrm{T}} Q^{(l)}_{\cdot\cdot k} S^{(l)}) \end{aligned} \tag{13}$$

where $E^{(l+1)} \in \mathbb{R}^{n_{l+1} \times n_{l+1} \times p_{l+1}}$ is the edge feature matrix after the complete pooling. In the pooling (step 12 of Algorithm 1), "$\cdot\cdot k$" is a slicing operation defined by

$$(Q^{(l)}_{\cdot\cdot k})_{ij} = Q^{(l)}_{ijk} \quad i, j \in 1, 2, ..., n_l \tag{14}$$

where $Q^{(l)}_{\cdot\cdot k}$ is a 2D matrix for channel k, and all channels are concatenated by $|| : \mathbb{R}^{n_{l+1} \times n_{l+1}} \to \mathbb{R}^{n_{l+1} \times n_{l+1} \times p_{l+1}}$, which is defined as

$$\begin{aligned} U &= ||_{k=1}^{p_{l+1}} V_k \\ U_{ijk} &= (V_k)_{ij} \end{aligned} \tag{15}$$

where $V_k \in \mathbb{R}^{n_{l+1} \times n_{l+1}}, k \in 1, 2, ..., p_{l+1}$. The edge pooling here is designed to be consistent with the node pooling by Equation (6). The edge pooling process is illustrated in Figure 3.

3.2.5 Predictor. Suppose there are L PEA layers. The outputs of the last layer including adjacency matrix $A^{(L)}$, node feature matrix $X^{(L)}$ and edge feature matrix $E^{(L)}$ are all flattened and then concatenated into a 1D vector, which is fed to an MLP (Multi-Layer Perceptrons). The MLP output is the classification/regression of analog performance defined by users.

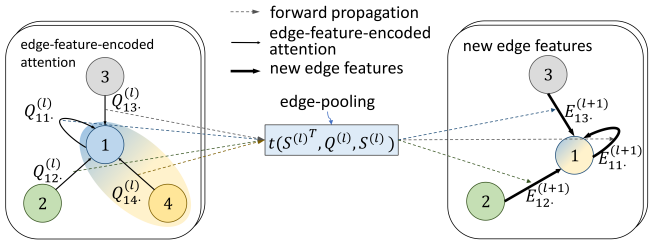


Figure 3: Edge pooling.

3.3 Knowledge Transfer among Different Topologies

An analog circuit of the same functionality can be realized in different designs or topologies. For instance, OTA can have different topologies such as Cascode OTA and Current Mirror OTA. Usually, the graph sizes and structures are different among different topologies. The knowledge obtained by training a PEA network on one topology can be transferred to a different topology with a fine tuning by a small amount of additional training data from the target topology.

According to Algorithm 1, PEA layers of a trained PEA network are specified by $W^{(l)}$, $W^{(l)}_{pool}$, $W^{(l)}_{edge}$, $a^{(l)}$ and $b^{(l)}$, $l = 1, 2, ..., L$, if there are L PEA layers. The sizes of these matrices and vectors are decided by the number of features and the number of nodes starting from layer 1, $n^{(l)}, l = 1, 2, ..., L$. Therefore, their sizes are independent of data sample size $n = n^{(0)}$ as long as $n \geqslant n^{(1)}$. In other words, the same model can be applied with different topologies with $n \geqslant n^{(1)}$. The pooling in the first PEA layer can transform a graph of many different sizes to $n^{(1)}$. This is another reason why we incorporate graph pooling in our customization.

4 PERFORMANCE DRIVEN PLACEMENT GUIDED BY PEA

Like many existing approaches [5, 7], the simulated annealing framework is adopted for the performance driven analog IC placement. The cost function to be minimized is

$$\alpha \cdot A + \beta \cdot W + \gamma \cdot Q \tag{16}$$

where A is normalized total area, W is normalized total wirelength estimated according to HPWL (Half Perimeter Wirelength), Q is a performance cost estimated by machine learning model, and α, β and γ are weighting factors that sum to 1. Besides minimizing the cost function, our placement enforces geometric constraints, such as symmetry and common centroid, like in [7].

The performance of an analog circuit is usually evaluated by multiple metrics. For instance, important performance metrics for OTA circuits include gain, bandwidth and phase margin. Suppose there are m metrics $y_1, y_2, ..., y_m$, and each $y_i, i = 1, 2, ..., m$ has a design specification or threshold ϕ_i, which is usually defined by users. The PEA network can be applied to classify if $y_i \geqslant \phi_i, i = 1, 2, ..., m$, i.e., if design specifications are satisfied. The overall performance cost can be defined as

$$Q_I = \sum_{i=1}^{m} w_i \cdot P(y_i < \phi_i) \tag{17}$$

where $P(y_i < \phi_i)$ is the probability of violating design specification and can be obtained by the softmax output at PEA network. The weighting factors $w_i, i = 1, 2, ..., m$ are decided by users and satisfy $\sum_{i=1}^{m} w_i = 1$.

Alternatively, the performance cost can be defined by

$$Q_{II} = P(\sum_{i=1}^{m} w_i \cdot \min(\frac{y_i}{\phi_i}, 1) < T) \tag{18}$$

where T is the specification of overall performance and can be obtained according to legacy designs. The classification on whether or not $\sum_{i=1}^{m} w_i \cdot \min(\frac{y_i}{\phi_i}, 1) < T$ can be obtained through PEA network. Cost Q_{II} relies on an additional threshold T compared to Q_I. However, it requires only one output from PEA while Q_I needs m outputs from PEA.

5 EXPERIMENT

5.1 Experiment Setup

The experiments are conducted on a Linux machine using Xeon (R) E5-2680 V2 processor with 2.8GHz frequency and 256G memory.

5.1.1 Model and Placement Implementation. The machine learning models are implemented in Python. Our analog IC placer, which is described in Section 4, and router are programmed in C++. Table 1 summarizes the configuration of the PEA network used in the experiment, which contains 4 PEA layers and 5 MLP layers. Our work is mainly compared with the recent CNN-based analog performance model [19]. The source code of [19] is obtained and modified to accommodate our testcases in the experiment.

	Layer Configuration			#trainable parameters
Feature Extractor	#nodes	#node features	#edge features	
PEA layers	12	15	24	1191
	12	15	24	1191
	6	10	12	728
	6	10	12	424
Predictor	#neurons			
MLP layers		32		16928
		16		528
		8		136
		4		36
		1		5
Total				21167

Table 1: PEA network configuration.

5.1.2 Testcase and Training Data. The testcases are OTA (Operational Transconductance Amplifier) designs of three different topologies: 5T OTA, Cascode OTA and Current Mirror OTA. These designs employ the ASAP 7nm process technology [25]. 8108, 7758 and 9858 placement solutions are generated for the 5T, Cascode and Current Mirror OTAs, respectively. Every placement solution is routed and the performance of each layout is evaluated by gain, bandwidth, unity gain frequency and phase margin, which are obtained through parasitic extraction and SPICE simulation. For each topology, 80% and 20% of the data samples are used for training and testing, respectively. None of the testing data can be seen during training. On average, each data sample feature for PEA network

	Circuit	Accuracy			Precision			Recall			FPR			AUROC		
		PEA	GAT	CNN	PEA	GAT	CNN	PEA	GAT	CNN	PEA	GAT	CNN	PEA	GAT	CNN
$\hat{Q}_I$	Cascode	84.4%	83.8%	78.7%	75.6%	73.4%	56.8%	51.2%	49.5%	27.0%	5.1%	5.3%	4.9%	0.863	0.858	0.706
	CM	89.3%	84.7%	77.5%	83.0%	78.3%	54.6%	71.5%	52.9%	24.8%	4.7%	4.7%	4.9%	0.933	0.897	0.802
	5T	93.7%	92.6%	85.1%	85.8%	86.4%	80.1%	89.7%	83.5%	53.5%	5.0%	4.4%	4.7%	0.982	0.971	0.880
	Avg.	89.1%	87.0%	80.4%	81.5%	79.4%	63.8%	70.8%	62.0%	35.1%	5.0%	4.8%	4.8%	0.926	0.909	0.796
$\hat{Q}_{II}$	Cascode	86.6%	85.2%	81.8%	82.1%	78.3%	73.7%	59.3%	55.9%	41.0%	4.3%	5.1%	4.8%	0.896	0.879	0.748
	CM	88.2%	87.5%	80.0%	81.8%	80.6%	68.7%	66.9%	64.4%	40.0%	4.8%	5.0%	4.9%	0.950	0.932	0.853
	5T	94.5%	92.8%	89.8%	86.5%	85.1%	83.9%	93.3%	87.3%	74.9%	5.1%	5.3%	5.0%	0.981	0.973	0.890
	Avg.	89.8%	88.5%	83.9%	83.5%	81.3%	75.4%	73.2%	69.2%	52.0%	4.7%	5.2%	4.9%	0.942	0.928	0.830

Table 2: Self-sustained learning results from our PEA model, GAT and CNN [19]. CM indicates Current-Mirror. Results of $\hat{Q}_I$ are the average among classifications of Gain, BW, UFG and PM. Results of $\hat{Q}_{II}$ are from binary classification version of Equation (18).

contains 5178 floating point numbers. Same as [19], the size of each input image to the CNN model is $64 \times 64 \times 5$. Placement images of one data sample for the CNN model [19] uses about 4× data or more than input features for our PEA network.

5.2 Results on Analog Performance Prediction

The classification performance by a machine learning model is evaluated by the following metrics based on TP (True Positive), TN (True Negative), FP (False Positive) and FN (False Negative).

- **Recall**, a.k.a. **TPR** (True Positive Rate): $\frac{TP}{TP+FN}$.
- **FPR** (False Positive Rate): $\frac{FP}{FP+TN}$.
- **Accuracy**: $\frac{TP+TN}{TP+TN+FP+FN}$.
- **Precision**: $\frac{TP}{TP+FP}$.

There is a tradeoff between TPR (Recall) and FPR by varying certain threshold in the classification. The TPR-FPR tradeoff curve is usually called **ROC** (Receiver Operating Characteristic) curve. The area under ROC curve (**AUROC**) is a metric for assessing the overall performance of entire tradeoff. AUROC is equal to 1 for a perfect model and equals 0.5 for random guesses.

5.2.1 Results on Self-Sustained Learning. The *self-sustained learning* here means the model is trained and applied on the same topology and 80% of the total data is employed for the training. The testing data are the remaining 20% of the entire data. Our proposed PEA network is compared with a plug-in use of GAT [21], which is one of the most popular graph neural networks, and CNN [19], which is a recent work on analog performance prediction for placement solutions. The model evaluation is performed for two different classification formulations.

(1) $\hat{Q}_I$: Four performance metrics, Gain, UGF (Unity Gain Frequency), BW (Bandwidth) and PM (Phase Margin), are classified separately by four differently trained models. The results are the average from these four metrics.
(2) $\hat{Q}_{II}$: This is to classify if $\frac{1}{4}\sum_{i=1}^{4}\min(\frac{y_i}{\phi_i}, 1) < T$, where y_i corresponds to Gain, UGF, BW and PM, ϕ_i are their specifications and T is a threshold. This is the binary classification version of Equation (18).

The main results of self-sustained learning are shown in Table 2. Our PEA network is superior to the previous work [19] on all of accuracy, precision and AUROC. On average, it improves AUROC by 16% and 13% on $\hat{Q}_I$ and $\hat{Q}_{II}$, respectively, compared to [19]. For the similar FPR, our PEA network achieves much better recall (TPR) than [19]. Such results indicate that it is important to consider graph structure as in PEA and only considering layout images like [19] is insufficient. Our PEA network also outperforms the plug-in use of GAT in most of the metrics except a few cases where the results are similar. This is an evidence for confirming the effectiveness of our customization on GNN techniques.

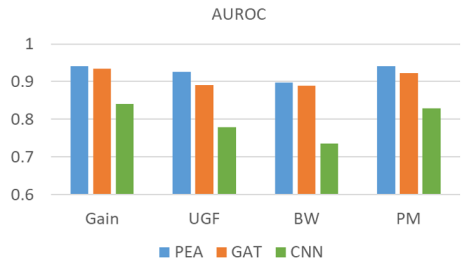


Figure 4: AUROC results on classifying the four performance metrics, averaged among the three topologies.

The separated AUROC results for the four different performance metrics are plotted in Figure 4. Our PEA network significantly outperforms CNN [19] on every performance metric. BW (bandwidth) is the most difficult to classify among the four as its dependence on placement is quite complex. The advantage of our PEA network versus the previous work [19] is the largest on this difficult case.

5.2.2 Results on Transfer Learning. In Table 3, we compare the results between learning with transfer and without transfer for the classification formulation of $\hat{Q}_{II}$. Transfer means knowledge obtained from training in one topology, called source topology, can be reused and helpful in another topology, called target topology. Columns 2 and 3 list the source and target topologies for the transfer. The "Transfer" results are obtained by a major training with 80% of data on S (source) and minor fine tuning with 10% of data on T (target), and predicting on T. Please note that the data amount for all these three circuit topologies are similar. In "No Trnsf", the training from 80% of S topology data is skipped. Therefore, the comparison would show if the knowledge learned from S is carried to T.

The results in Table 3 indicate that the transfer with PEA network almost always improves classification quality compared with no transfer. This confirms that our PEA network can generally achieves knowledge transfer. The only minor exception is that FPR and

Model	Circuit S	Circuit T	Accuracy Transfer	Accuracy No Trnsf	Precision Transfer	Precision No Trnsf	Recall Transfer	Recall No Trnsf	FPR Transfer	FPR No Trnsf	AUROC Transfer	AUROC No Trnsf
PEA	Cascode	CM	79.5%	76.8%	69.5%	58.1%	28.8%	18.6%	4.1%	4.4%	0.872	0.840
	CM	Cascode	83.5%	80.6%	77.2%	70.3%	47.9%	38.4%	4.7%	5.4%	0.852	0.805
	CM	5T	92.2%	92.2%	83.5%	85.4%	87.1%	84.0%	6.0%	5.0%	0.950	0.941
	Avg.		85.1%	83.2%	76.7%	71.2%	54.6%	47.0%	4.9%	4.9%	0.891	0.862
CNN	Cascode	CM	77.4%	75.9%	60.1%	52.3%	22.8%	16.6%	4.9%	4.9%	0.822	0.797
	CM	Cascode	73.8%	79.5%	37.4%	69.5%	8.9%	28.1%	5.0%	4.5%	0.613	0.740
	CM	5T	76.8%	74.6%	62.1%	52.5%	25.8%	15.1%	5.5%	4.7%	0.856	0.849
	Avg.		76.0%	76.6%	53.2%	58.1%	19.1%	19.9%	5.1%	4.6%	0.764	0.795

Table 3: Transfer learning results from source circuit S to target circuit T. Transfer: training with 80% data from S and 10% data from T, and predicting on T. No Trnsf: training with 10% data from T and predicting on T. CM: Current Mirror.

precision are slightly degraded for the transfer from Current Mirror OTA to 5T OTA. For the CNN-based work [19], the transfer from Current Mirror OTA to Cascode OTA causes significant degradation on all metrics.

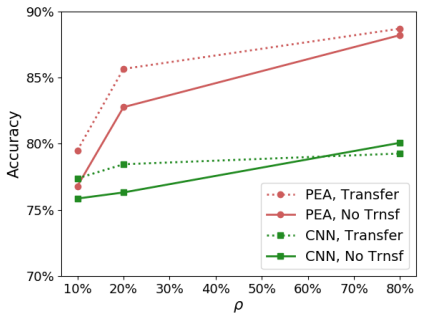


Figure 5: Effect of transfer from Cascode OTA to Current Mirror OTA. ρ of Current Mirror OTA data is used in training for all methods.

In Figure 5, we show the effect of ρ, which is the ratio of target topology (Current Mirror) data used in training for both "Transfer" from Cascode to Current Mirror and "No Trnsf" at Current Mirror OTA. When ρ is low, the effect of transfer is evident for both our PEA network and the previous CNN approach [19]. The classification accuracy increases with ρ. When ρ is large, the effect of transfer diminishes.

In Figure 6, ROC curves for classifying $\hat{Q}_{II}$ on Current Mirror OTA are plotted for different methods. It shows that self-sustained learning is better and transfer learning still has room for improvement. Almost all PEA solutions dominate the CNN-based previous work [19].

5.2.3 Model Training Time. In self-sustained learning, training one PEA network takes about 728 seconds while training one CNN model [19] costs about 7263 seconds. Thus, our training is approximately 10× faster than the previous work [19].

5.3 Results on Analog Placement

The results of performance driven placement guided by machine learning model are compared with manual layout and a conventional automatic method [7], which does not include performance in its objectives. Five variants of performance driven placement based on Section 4 are tested. They are guided by combinations of PEA

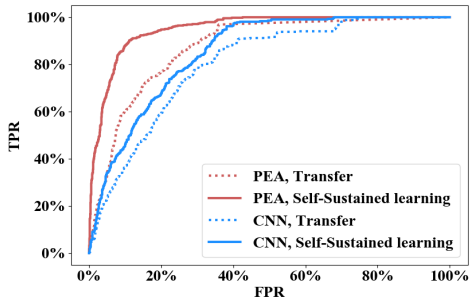


Figure 6: ROC curves for classifying $\hat{Q}_{II}$ on Current Mirror OTA. Knowledge is transferred from Cascode OTA in the transfer learning.

vs. CNN [19], SS (Self-sustained learning) vs. transfer learning, and performance cost Q_I defined by Equation (17) and Q_{II} defined by Equation (18). To capture the overall circuit performance, a Figure of Merit (FOM) is defined as

$$FOM = \sum_{i=1}^{4} w_i \cdot \min(\frac{y_i}{\phi_i}, 1) \tag{19}$$

where w_i denotes weights, y_i corresponds to Gain, UFG, BW and PM, and ϕ_i indicates their specifications. The value of FOM is in $[0, 1]$ and ideally 1. This definition is consistent with the performance cost defined in Equations (17) and (18).

The results from the three circuits are shown in Tables 4, 5 and 6. For both the Cascode OTA and 5T OTA, all the three PEA-guided results are significantly closer to manual layout than both the previous work [7] and placement guided by the CNN-based model [19]. For Current Mirror OTA, the best automatic results are from all of our models and CNN self-sustained learning. The overall advantage of placement guided by PEA versus the previous work [19] is evident. Examples of layout generated by manual design and PEA network are demonstrated in Figure 7. The symmetry constraints for 3 pairs of transistors are enforced in both the manual layout and the placement guided by PEA. There are 4 stand alone transistors without symmetry constraints. PEA guides the placement of these 4 transistors for improving performance while the manual design places them symmetrically.

Total placement and routing runtime estimation is provided in Table 7. Approximately, the automatic layout, where placement is

	Schematic	Manual	Conventional Automatic	CNN SS Q_{II}	CNN Transfer Q_{II}	PEA SS Q_{II}	PEA Transfer Q_{II}	PEA Transfer Q_I
Gain (dB)	37.0	33.0	23.7	27.7	30.1	32.2	32.5	33.1
UGF (MHz)	1522.9	1167.0	947.6	1003.0	617.1	1072.0	948.9	1042.0
BW (MHz)	21.8	26.8	56.0	33.8	17.5	26.9	22.4	24.8
PM (degree)	82.1	80.7	108.5	113.7	104.7	90.8	93.0	85.5
FOM	1.00	0.85	0.71	0.75	0.66	0.82	0.80	0.83
Area (μm^2)	-	26.5	24.1	40.4	37.1	34.0	34.4	32.4

Table 4: Results of Cascode OTA. SS: Self-Sustained Learning.

	Schematic	Manual	Conventional Automatic	CNN SS Q_{II}	CNN Transfer Q_{II}	PEA SS Q_{II}	PEA Transfer Q_{II}	PEA Transfer Q_I
Gain (dB)	32.6	32.5	33.1	33.0	28.7	32.7	33.1	33.0
UGF (MHz)	531.0	530.0	451.0	484.0	424.3	502.9	495.1	481.6
BW (MHz)	12.5	12.3	10.2	11.0	15.8	11.9	11.2	11.0
PM (degree)	82.8	80.1	78.5	78.0	80.3	76.9	77.4	77.6
FOM	1.00	0.99	0.90	0.93	0.85	0.95	0.94	0.93
Area (μm^2)	-	15.8	14.5	17.1	25.9	17.6	25.9	21.5

Table 5: Results of Current Mirror OTA.

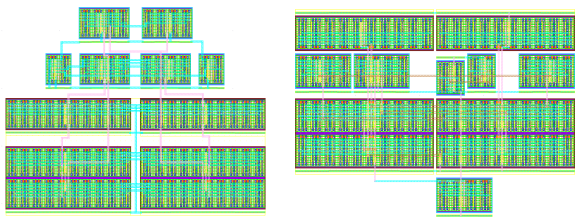
	Schematic	Manual	Conventional Automatic	CNN SS Q_{II}	CNN Transfer Q_{II}	PEA SS Q_{II}	PEA Transfer Q_{II}	PEA Transfer Q_I
Gain (dB)	32.4	32.4	26.5	32.0	24.7	31.4	31.8	32.8
UGF (MHz)	1105.0	870.4	656.9	671.1	667.9	799.5	771.0	744.9
BW (MHz)	26.5	21.0	34.2	16.8	38.6	21.2	19.7	16.9
PM (degree)	86.5	85.1	85.5	94.1	95.1	94.2	93.8	94.1
FOM	1.00	0.89	0.77	0.80	0.75	0.85	0.84	0.83
Area (μm^2)	-	18.7	16.4	26.1	22.0	18.0	22.0	26.1

Table 6: Results of 5T OTA.

	Manual (min)	Conven Auto [7] (s)	PEA (s)	CNN [19] (s)
Cascode	110	2.27	52.4	50.0
CM	160	1.38	42.7	45.9
5T	90	1.26	67.3	52.9
Avg.	120	1.63	54.1	49.6

Table 7: Total place and route runtime.

guided by machine learning (PEA or CNN), is about 133× faster than manual design.



(a) Manual design (b) Guided by PEA

Figure 7: Layout of Current Mirror OTA.

6 CONCLUSIONS AND FUTURE RESEARCH

This work proposes a customized GNN approach, called PEA network, for predicting analog circuit performance of a placement solution. It is superior to the recent CNN-based work on accuracy, knowledge transfer and training time. It also outperforms a plug-in use of GAT, which is one of the most influential GNN techniques. The post-layout circuit performance from PEA-guided placement is better than placement guided by CNN. Our placement achieves performance similar to manual design but is two orders of magnitude faster. In future research, we will further improve the knowledge transfer capability and explore transfer among different types of circuits beyond OTA.

ACKNOWLEDGEMENT

This work is supported by the DARPA ERI IDEA program. We thank Prof. Shuiwang Ji of Texas A&M University for helpful discussions.

REFERENCES

[1] U. Choudhury and A. Sangiovanni-Vincentelli, "Automatic generation of parasitic constraints for performance-constrained physical design of analog circuits," *IEEE TCAD*, vol. 12, no. 2, pp. 208–224, 1993.

[2] K. Lampaert, G. Gielen, and W. M. Sansen, "A performance-driven placement tool for analog integrated circuits," *IEEE JSSC*, vol. 30, no. 7, pp. 773–780, 1995.

[3] F. Balasa and K. Lampaert, "Symmetry within the sequence-pair representation in the context of placement for analog design," *IEEE TCAD*, vol. 19, no. 7, pp. 721–731, 2000.

[4] M. Strasser, M. Eick, H. Grab, U. Schlichtmann, and F. M. Johannes, "Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions," in *Proc. ICCAD*, 2008.

[5] P.-H. Lin, Y.-W. Chang, and S.-C. Lin, "Analog placement based on symmetry-island formulation," *IEEE TCAD*, vol. 28, no. 6, pp. 791–804, 2009.

[6] C.-W. Lin, J.-M. Lin, C.-P. Huang, and S.-J. Chang, "Performance-driven analog placement considering boundary constraint," in *Proc. DAC*, 2010, pp. 292–297.

[7] Q. Ma, L. Xiao, Y.-C. Tam, and E. F. Young, "Simultaneous handling of symmetry, common centroid, and general placement constraints," *IEEE TCAD*, vol. 30, no. 1, pp. 85–95, 2010.

[8] P.-H. Wu, M. P.-H. Lin, Y.-R. Chen, B.-S. Chou, T.-C. Chen, T.-Y. Ho, and B.-D. Liu, "Performance-driven analog placement considering monotonic current paths," in *Proc. ICCAD*, 2012, pp. 613–619.

[9] H. C. Ou, K. H. Tseng, J. Y. Liu, I. P. Wu, and Y. W. Chang, "Layout-dependent-effects-aware analytical analog placement," in *Proc. DAC*, 2015, pp. 1–6.

[10] P.-H. Wu, P. H. Lin, and T. Y. Ho, "Analog layout synthesis with knowledge mining," in *European Conference on Circuit Theory and Design (ECCTD)*, 2015, pp. 1–4.

[11] P. H. Lin, Y. W. Chang, and C. M. Hung, "Recent research development and new challenges in analog layout synthesis," in *Proc. ASPDAC*, 2016, pp. 617–622.

[12] B. Xu, S. Li, X. Xu, N. Sun, and D. Z. Pan, "Hierarchical and analytical placement techniques for high-performance analog circuits," in *Proc. ISPD*, 2017, pp. 55–62.

[13] B. Xu, S. Li, C.-W. Pui, D. Liu, L. Shen, Y. Lin, N. Sun, and D. Z. Pan, "Device layer-aware analytical placement for analog circuits," in *Proc. ISPD*, 2019, pp. 19–26.

[14] Z. Liu and L. Zhang, "A performance-constrained template-based layout retargeting algorithm for analog integrated circuits," in *Proc. ASPDAC*, 2010, pp. 293–298.

[15] G. Zhang, H. He, and D. Katabi, "Circuit-GNN: Graph neural networks for distributed circuit design," in *Proc. ICML*, 2019, pp. 7364–7373.

[16] B. Xu, Y. Lin, X. Tang, S. Li, L. Shen, N. Sun, and D. Z. Pan, "Wellgan: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout," in *Proc. DAC*, 2019, pp. 1–6.

[17] K. Zhu, M. Liu, Y. Lin, B. Xu, S. Li, X. Tang, N. Sun, and D. Z. Pan, "Genius route: A new analog routing paradigm using generative neural network guidance," in *Proc. ICCAD*, 2019, pp. 1–8.

[18] K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. Sharma, W. Xu, S. M. Burns, J. Hu, R. Harjani, and S. S. Sapatnekar, "GANA: Graph convolutional network based automated netlist annotation for analog circuits," in *Proc. DATE*, 2020.

[19] M. Liu, K. Zhu, J. Gu, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "Towards decrypting the art of analog layout, placement quality prediction via transfer learning," in *Proc. DATE*, 2020, pp. 1–6.

[20] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," *arXiv preprint arXiv:2005.00406*, 2020.

[21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[22] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in neural information processing systems*, 2018, pp. 4800–4810.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 1, no. 1, pp. 1–21, 2019.

[24] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *Proc. CVPR*, 2019, pp. 9211–9219.

[25] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.