Chapter 18: Machine Learning for Analog Layout

Steven M. Burns, Hao Chen, Tonmoy Dhar, Ramesh Harjani, Jiang Hu, Nibedita Karmokar, Kishor Kunal, Yaguang Li, Yishuang Lin, Mingjie Liu, Meghna Madhusudan, Parijat Mukherjee, David Z. Pan, Jitesh Poojary, S. Ramprasath, Sachin S. Sapatnekar, Arvind K. Sharma, Wenbin Xu, Soner Yaldiz, Keren Zhu

Abstract The performance of analog circuits is critically dependent on layout parasitics, but layout has traditionally been a manual and time-consuming task. Recent advances in ML have enabled new capabilities to facilitate fast automated placement and routing. This chapter presents an overview of these techniques, including geometric constraint generation and constrained placement and routing. A variety of ML techniques are used in various steps of analog placement and routing, including graph neural networks, random forest methods, support vector machines, graph attention networks, generative adversarial networks, reinforcement learning, and variational autoencoders. This chapter shows how these general ML algorithms are specifically customized to the requirements of optimized analog layout.

1 Introduction

Design automation researchers have addressed the problem of analog layout synthesis for several decades [13, 15, 21, 25, 31, 49, 50, 58, 61, 70]. These techniques were

Steven M. Burns, Parijat Mukerjee, and Soner Yaldiz
Intel Labs, Hillsboro, OR, USA.

Tonmoy Dhar, Ramesh Harjani, Nibedita Karmokar, Kishor Kunal, Meghna Madhusudan, Jitesh Poojary, S. Ramprasath, Sachin S. Sapatnekar, Arvind K. Sharma
University of Minnesota, Minneapolis, MN, USA.

Jiang Hu, Yaguang Li, Yishuang Lin, Wenbin Xu
Texas A&M University, College Station, TX, USA.

Hao Chen, Mingjie Liu, David Z. Pan, Keren Zhu
The University of Texas at Austin, Austin, TX, USA.

Corresponding authors: David Z. Pan (dpan@utexas.edu) and Sachin S. Sapatnekar (sachin@umn.edu)
This work was supported in part by the DARPA IDEA program (SPAWAR contracts N660011824048 and N669911824049) and NSF CCF-1704758.

largely based on a toolbox of traditional algorithms such as numerical analysis, graph theory, and optimization techniques. However, these methods were generally unable to compete with a skilled designer in the quality of the solution that was produced. The problem has enjoyed a recent renaissance due to increasing interest in analog design as applications place a greater focus on interactions with the analog real world, and application drivers such as wireless systems and AI hardware make native analog design more attractive.

Several new approaches have been proposed in the recent past to tackle the problem of analog layout synthesis. Frameworks such as BAG [8, 14] have focused on procedural design with significant designer input, while approaches such as FASoC [3] leverage digital standard cells and digital layout methodologies for designing "digital analog" circuits. Recent techniques have considered the use of artificial neural networks for the layout of specific topologies [26] or for knowledge migration [27].

A new class of methods in MAGICAL [9, 10, 77] and ALIGN [1, 18, 39] has looked at approaches that could be used with much more limited human intervention, including no-human-in-the-loop automation. These frameworks are facilitated by the advent of machine learning (ML), which has substantially changed the algorithmic landscape. Today, with the emergence of artificial intelligence techniques that can compete with human skill, it is now realistic to think about automated analog layout that delivers a solution that is comparable in quality to manual design.

A typical analog layout generation flow consists of the following steps:

- *Circuit hierarchy specification* takes an input netlist and, through either design annotation or automated recognition techniques, determines the hierarchical blocks in the circuit.
- *Constraint specification* provides guidelines for layout, determining how the blocks in the circuit hierarchy must be arranged. These constraints include (a) *geometric constraints* for symmetrical placement and routing about specified axes, common-centroid or interdigitated layouts [33], and (b) *electrical constraints* that place limits on allowable interconnect RC parasitics during layout while ensuring that performance constraints are met; typically, this involves tradeoffs and complex relationships, i.e., allowing increased RCs on some wires while ensuring lower RCs on others.
- *Cell generation* generates the layout of devices and passives at the lowest level of the hierarchy, parameterized so that cells may be built for a specified set of transistor widths and gate lengths; may have a variety of aspect ratios; may be built with/without body contacts; etc.
- *Placement* is typically performed hierarchically, with the locations of blocks being determined at each step while honoring constraints on symmetry and performance, specified in the constraint generation step.
- *Routing* connects the placed blocks at each hierarchical level, using appropriate wire widths to ensure that performance specifications are met.

When judiciously used, ML methods can be of great help in automating analog design. While some steps can be handled using conventional algorithmic techniques

(e.g., cell generation in FinFET technologies, where the limited degrees of freedom render the problem amenable to algorithmic layout generation), ML techniques can help mimic the wisdom of the human designer in other steps. This chapter provides an overview of ML methods for analog layout. A recent paper [2] presents a survey of ML methods in analog design in general, and this chapter complements the overview with an in-depth view of the use of ML in analog layout automation. Section 2 presents a variety of approaches for circuit annotation and geometric constraint specification. Next, Section 3 overviews various ML techniques for well generation, placement, and routing. Finally, the chapter concludes with a view of future directions for ML-assisted analog layout.

2 Geometric Constraint Generation

2.1 Problem Statement

Geometric constraints help analog designs to achieve high performance and high yield by making them resilient to PVT variations. Symmetry considerations are particularly important in specifying constraints to be applied during analog layout synthesis. Analog designs frequently use differential topologies to reject common-mode noise, and layout symmetry helps in reducing mismatch between such devices, which is liable to significantly degrade circuit performance.

The research in extracting geometric constraints automatically has been evolving rapidly in recent years, motivated by advances in ML techniques and the requirements of analog layout automation [83]. This section presents several ML approaches for geometric constraint generation. The approaches in Sections 2.2 and 2.3 extract constraints based on the results of circuit annotation using graph convolutional networks and array recognition methods (including approximate graph isomorphism), respectively. The approaches in Section 2.4 and Section 2.5 directly extract the symmetry constraints, Section 2.4 describes a system-level symmetry constraint extraction algorithm leveraging statistical techniques to measure the graph similarity. The circuits with high similarity in their graph structures are identified as symmetry constraints. Section 2.5 describes two symmetry constraint extraction techniques using graph neural network, with supervised learning and unsupervised learning.

2.2 Subcircuit Annotation Using Graph Convolution Networks

A first step in generating constraints is to find hierarchies within a circuit, identifying specific circuit blocks. If an experienced designer wishes to retain specified hierarchies, this step may be skipped. Otherwise, existing hierarchies may be discarded to find hierarchies that are more amenable to layout and constraint generation. In analog circuits, this is difficult: a large number of circuit variants exist even for a

single functionality, e.g., between textbooks [66] and research papers, there are well over 100 widely used operational transconductance amplifier (OTA) topologies of various types (e.g., telescopic, folded cascode, Miller-compensated). Truly generalizable analog automation requires the ability to recognize all variants – including those that have not even been designed to date. Traditional methods have used two approaches: (1) *library-based* [51, 53], matching a circuit to prespecified templates, and requiring an enumeration of possible topologies in an exhaustive database, or (2) *knowledge-based* [30, 71], embedding rules for recognizing circuits; however, the rules must come from an expert designer who may struggle to provide a list (many rules are intuitively ingrained rather than explicitly stated). Moreover, it is difficult to capture rules for all variants. In this section, we describe ML methods for subcircuit recognition, which can be used to derive circuit hierarchies and block-level constraints.

As in [59], a circuit netlist is represented by an undirected bipartite graph $G(V, E)$, where $V = V_e \cup V_n$. The subsets V_e and V_n correspond, respectively, to elements (transistors/passives) in the netlist, and the set of nets. The edge set E consists of edges between a vertex in V_e corresponding to an element to the vertices in V_n corresponding to nets connected to its terminals. Similar to [43], each edge connected to a transistor is assigned a three-bit binary label, $l_g l_s l_d$, where $l_g/l_s/l_d$ are set to 1 only if the edge from the transistor vertex connects to the net vertex through its gate/source/drain, respectively. The subcircuit recognition problem is mapped to one of approximate subgraph isomorphism, with approximation allowing for variations around a core structure for a circuit block.

In [38], graph convolutional networks (GCNs) are used for this purpose. General graphs have no unique embedding. A GCN performs convolutions that are independent of the embedding of the graph in the plane. Various types of GCNs have been proposed [16, 28, 29, 35, 80], but they all share a framework that requires three fundamental steps: (i) the application of localized convolutional filters on graphs, (ii) a graph coarsening procedure that groups together similar vertices and (iii) a graph pooling operation for graph reduction. GCN techniques primarily differ in the nature of the filter that is used for convolution.

A major class of GCNs is based on spectral methods [16, 35], which are independent of graph embedding and have been found to be extremely effective and therefore form the basis of the GCN used in [38]. A spectral representation in the Fourier space of a graph is enabled through the graph Laplacian representation. The Laplacian $L \in R^{n \times n}$ of an unweighted graph $G(V, E)$ with n vertices is often defined as $D - A$ where $A \in R^{n \times n}$ is the adjacency matrix of the graph and $D \in R^{n \times n}$ is a diagonal matrix whose diagonal entry corresponds to the degrees of all vertices, i.e., the row sums of the adjacency matrix. The normalized Laplacian representation is

$$L = I - D^{-1/2} A D^{-1/2} \tag{1}$$

The matrix L is symmetric, real, and positive definite: it has real nonnegative eigenvalues that are interpreted as the frequencies of the graph.

The approach proposed by [16] creates spectral filters around a vertex, where each filter functions within a region of radius K of (i.e., up to K edges away from) the vertex. A convolution operator on the graph is defined as

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x \tag{2}$$

where the normalized graph Laplacian is eigendecomposed as $L = U\Lambda U^T$, and g_θ is a filtering operator that acts on an input signal x to produce an output signal y. In this case, the signal corresponds to a region of the graph around a specific vertex.

Next, $g_\theta(L)$ is parameterized as a polynomial Chebyshev expansion, which truncates the filter expansion to order of $K - 1$. Given the K top eigenvalues of L (computed inexpensively using the Lanczos algorithm), for a graph of bounded degree where the number of edges is $O(n)$, this polynomial can be evaluated using K multiplications by a sparse L with a cost of $O(Kn) \ll O(n^2)$.

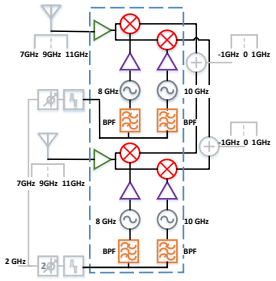
The GCN topology has two convolutional layers (performing the operations described above) and two pooling layers, which then feed a fully connected (fc) layer, whose outputs provide the classification results. Pooling combines similar vertices in a graph using the greedy Graclus heuristic, built on top of the Metis algorithm [34] for multilevel clustering [20]. The final layer is a fully connected layer of size 512 along with `softmax` function for classification.

Each vertex is associated with 18 features: 12 that annotate the element type; 5 that denote the type of net (input, output, bias, supply, ground); and 1 that describes the label for edges incident on a transistor vertex.

Rather than placing the full burden of recognition on the GCN, a set of simple postprocessing heuristics are used to complete the annotation. *Postprocessing I* involves graph-based heuristics which ensures consistent classification of nodes in the same channel-connected component (CCC) [67]. *Postprocessing II* uses circuit-specific knowledge: e.g., low-noise amplifiers (LNAs) and mixers can be structurally similar, but an LNA has an antenna input, while a mixer has an oscillating input. Such information can be designer-specified or inferred from the testbench in the input netlist.

Testcase 1 is a filter with an OTA and switched capacitors, and contains 32 devices and 25 nets. The telescopic OTA subcircuit used in this circuit is not seen by the training set. Using the GCN alone an accuracy of 56/57 is achieved in identifying OTA and bias circuit nodes. The misclassified vertices belong to the OTA interconnect ports, and all nodes (100%) are correctly classified after Postprocessing I.

Testcase 2 consists of a phased array system [54], illustrated in Fig. 1(a), containing a mixer (red), LNA (green), BPF (orange), oscillator (gray), VCO buffer (BUF) and inverter-based amplifier (INV) (violet) sub-blocks. The graph for the input netlist has 902 vertices (522 devices + 380 nets). The GCN based classification identifies nodes belonging to LNA, mixer, and oscillator and passes these results through postprocessing. After Postprocessing I, the BPF is identified as a combination of an oscillator with two input transistors. INV and BUF primitives are identified and a separate hierarchy is created for them which boosts the accuracy to 87.3%. During Postprocessing II, which uses an antenna label at LNA input and oscillating input for



(a)

Actual Class

Predicted class	LNA	MIXER	OSC	BPF	BUF	INV
LNA	78	0	0	0	0	0
MIXER	0	120	0	0	0	0
OSC	0	0	132	0	0	0
BPF	0	0	0	136	0	0
BUF	0	0	0	0	32	0
INV	0	0	0	0	0	24

(b)

Fig. 1: (a) Phased array system [54] and (b) results of GCN after postprocessing, showing the correctness of vertex classification.

mixer, all nodes are identified correctly. At this point, the classification result after post-processing is shown in Fig. 1(b): all 522 devices (100%) are classified correctly.

Once the circuit functionality is determined, graph-based approaches are used to identify primitives (e.g., differential pairs, current mirrors), including annotations for symmetry constraints at the primitive level and at upper levels of hierarchy. The annotation scheme is fast (a few minutes for the phased array on a desktop machine), and is dominated by the runtime of the GCN.

2.3 Array-based Methods for Subcircuit Annotation

The technique above requires the curation of a training set for each type of circuit structure. While this is feasible for standard circuit blocks, an alternative procedure can guide layout by recognizing repeated structures in a circuit and lay them out using array-based methods. Such methods are useful in building structures such as flash ADCs, binary-weighted DACs, R-2R DACs, and equalizers, where the same structure is repeated. For exact replicas, it is possible to use graph-based methods to recognize regularity, but analog designers often witness scenarios where circuit blocks are *nearly* identical and require symmetric/regular layout.

The work in [40] presents an ML method for recognizing approximate matches by error-tolerant matching. The circuit is represented by a bipartite graph as defined earlier. The approach is based on the concept of *graph edit distance* (GED) [82], a measure of similarity between two graphs G_1 and G_2. Given a set of graph edit operations (insertion, deletion, vertex/edge relabeling), the GED is a metric of the number of edit operations required to translate G_1 to G_2.

Let graphs G_1 and G_2 represent, respectively, the CS-LNA and the CG-LNA, as shown in Fig. 2, with element vertices at left and net vertices at right. To transform G_1 to G_2, the GED = 4: two edges in G_1 are deleted: (capacitor element, ground net)

and (transistor element (source label), V_{IN} net); two are added: (capacitor element, V_{IN} net) and (transistor element (source label), ground net).

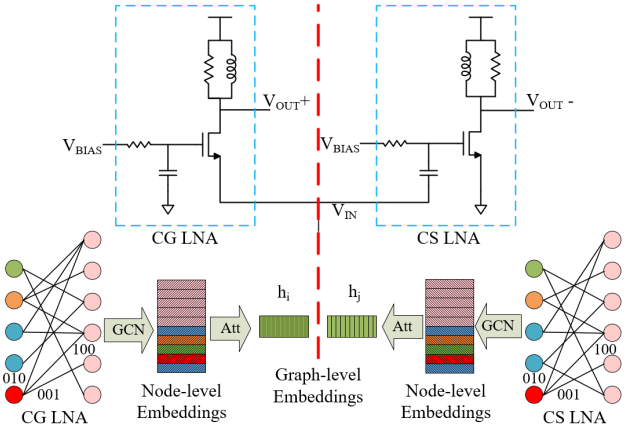


Fig. 2: Example showing graph embedding for common gate low noise amplifier (CG LNA) and common source LNA (CS LNA) in noise cancellation LNA.

This work uses a neural network that transforms the original NP-hard problem to a learning problem [4] for computing graph similarity. The method works in four steps. In the first step, each node in the graph is converted to a node-level embedding vector. The second step uses these embedding vectors to create a graph-level embedding of dimension d. The lower half of Fig. 2 illustrates these two steps for the graphs for the CG-LNA and CS-LNA. For each subblock in the circuit, these steps need to be carried out once, and the graph embeddings are stored for matching any two pairs of subblocks in later stages. The computational complexity of these two steps is linear in the number of nodes in the graph.

The last two steps are shown in Fig. 3. The third step feeds the graph-level embeddings from the second step for two candidate graphs to a trained neural tensor network that generates a similarity matrix between the graphs. The fourth step then processes this matrix using a fully connected neural network to yield a single score. This matching method for two subblocks uses the previously stored graph embeddings instead of the full subblock graphs. The complexity of these two steps is quadratic in d, where d is bounded by a small constant in practice, the procedure is computationally inexpensive as compared to an exact GED computational complexity which is exponential in the number of nodes of the graphs involved.

The optimized model is a three-layer GCN with 128 input channels (the number of channels is halved in each layer), with 8 slices in the neural tensor network (NTN), and a fully connected network with one hidden layer after the NTN. The trained net uses $d_1 = 64, d_2 = 32, d_3 = d = 16$.

The method is applied on an FIR Equalizer (Fig. 4(a)) circuit with 10 taps, each containing a differential pair, a current mirror DAC, and an XOR gate. All

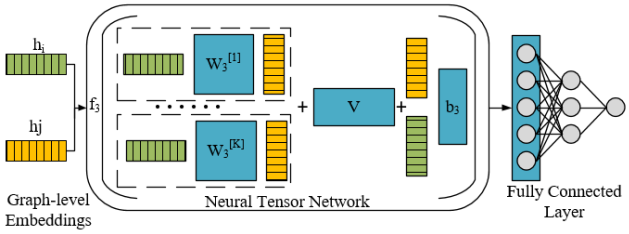
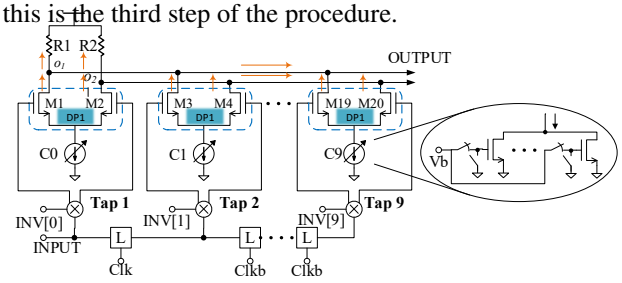
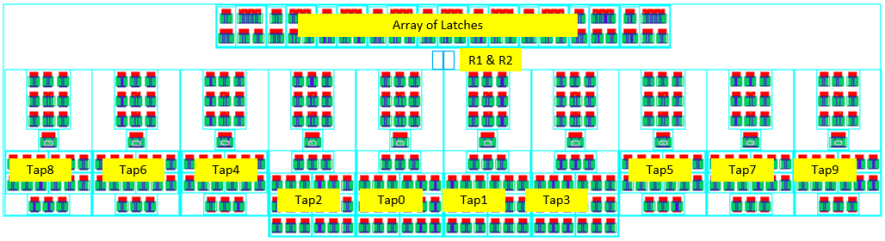


Fig. 3: GED prediction based on graph embeddings [4]. Here, h_i and h_j are the graph embeddings for two similarity candidates, f is an activation function, $W_3^{(1\cdots K)}$ is the weight tensor, V is a weight vector, and b_3 is a bias vector. The subscript "3" reflects the fact that this is the third step of the procedure.



(a)



(b)

Fig. 4: (a) Schematic and (b) layout of an FIR equalizer [32].

blocks in each tap share a common symmetry axis for matching. The first four taps use a 7-bit current mirror DAC, and the remaining taps have 5-bit current mirror DAC. To achieve better matching, the first four taps are placed in the center and the remaining taps are placed around these four, sharing a common symmetry axis. The layout of equalizer, shown in Fig. 4(b), meets all these requirements. This design demonstrates the detection and use of multiple lines of symmetry in a hierarchical way within primitives, within each tap, and globally at the block level.

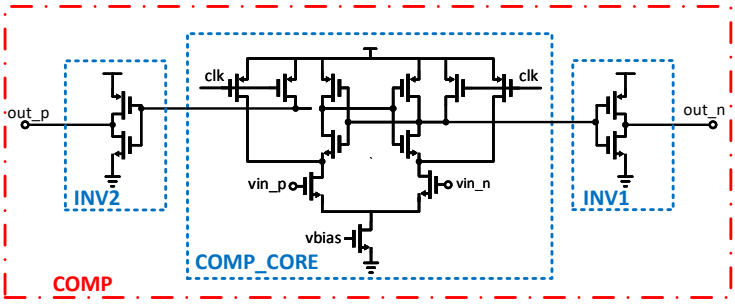


Fig. 5: Hierarchical circuit example [45].

2.4 System Symmetry Constraint with Graph Similarity

Many studies for symmetry constraint detection focus on generating constraints for building block level circuits such as differential amplifiers and comparators. On the other hand, methods for generating system symmetry constraints have rarely been studied. There still exists a gap in automatic constraint detection and constraint management for system level analog designs. Previous methods of constraint detection have difficulties in scalability and expressiveness when directly migrating to analog circuit systems.

The system symmetry constraint detection problem for analog circuits can be described as follows. The hierarchical circuit netlist N is given as input. The circuit hierarchy is abstracted into a tree T, with each node representing a subcircuit. For each subcircuit $v \in T$, its children $G = \{g | g \subseteq v\}$ are subcircuits referenced in v. Symmetry constraint pair (g_i, g_j) represents that critical matching is needed between the subcircuits, where $g_i, g_j \in G$. The system symmetry constraint detection problem is to generate constraint pairs for every subcircuit $v \in T$.

A simplified example of system symmetry constraints is shown in Fig. 5. The circuit COMP consists of 3 subcircuits: COMP_CORE, INV1 and INV2. The inverters in COMP need to be matched with symmetry constraint pair (INV1, INV2). System constraints do not consider transistor device symmetry in subcircuits, such as COMP_CORE. If COMP_CORE further contains other subcircuits, the system constraints between these subcircuits should also be considered.

The hierarchies of the netlist is abstracted into a tree representation. Each subcircuit instantiation is abstracted with a node in the tree. The root of the hierarchy tree is the entire analog circuit system. The leaf cell nodes are device level instances of transistors, resistors, capacitors, and diodes. The primal subcircuits are labeled as analog or digital in the netlist to improve constraint quality. Fig. 6 shows an example of the extracted hierarchy tree of the corresponding circuit in Fig. 5.

Constraint candidates are classified as valid or invalid during symmetry detection. Symmetry constraints are detected if the circuit graphs are similar. To improve constraint quality and reduce false alarms, the neighboring circuit topologies are extracted on the entire circuit graph. The sizes of extracted subgraphs are determined

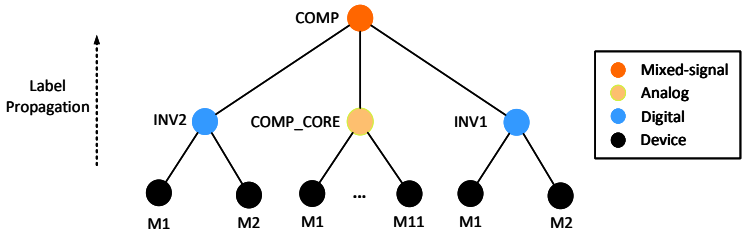


Fig. 6: Hierarchy tree of the circuit in Fig. 5 [45].

by graph centrality. The similarities of the extracted subgraphs are measured with a scalable graph similarity metric using spectral graph analysis.

The subgraphs of subcircuits with neighboring circuit topologies are extracted to improve constraint quality and reduce false alarms. Only comparing the subcircuits is not enough to fully characterize symmetry constraints. The extracted neighboring circuit size is critical to the quality of symmetry constraint detection. Extracting small subgraphs would not include enough information. On the other hand, extracted large subgraphs of closely connected subcircuits would fully include both subcircuits and all their neighboring circuit topologies. In this case, the subgraphs would be detected as similar and create unnecessary constrained symmetry. To resolve such issues, graph centrality is used to determine the extracted subgraph radius. In graph theory and network analysis, indicators of centrality assign numbers or rankings to nodes within a graph corresponding to their network position. The subcircuit graph centers and radius are calculated using graph centrality, such as the Jordan Center [52], Eigenvector Centrality [52], or PageRank Center [63]. The extracted subgraph radius is then defined as half of the shortest path distance between the centers of two graphs.

S^{3}DET [45] uses the two-sample Kolmogorov-Smirnov (K-S) test [7] to measure the similarity between the eigenvalue distributions of two graphs. The K-S test is a non-parametric statistical test used to compare a sample with a reference probability distribution. The extended two-sample K-S test is used to test whether the underlying probability distributions differ between the two samples. The K-S statistic of two empirical cumulative distribution function $F_{1,n}(x)$ and $F_{2,m}(x)$ with sample size n and m is defined as:

$$D_n = \sup_x |F_{1,n}(x) - F_{2,m}(x)|. \tag{3}$$

It quantifies the difference between the two distributions. In statistics, the p-value is the probability of obtaining results at least as extreme as the observed results of a statistical hypothesis test, assuming that the null hypothesis is correct. The p-value from the K-S test measures how likely these samples comes from the same distribution. A small p-value concludes that the two samples are from different distributions, while a large p-value infers that the distributions match. A scalable graph similarity algorithm is applied using graph spectral analysis. Since the eigenvalue distribution of the graph *Laplacian* is closely linked with the graph structure, the K-S statistic

of the eigenvalue distribution can be used as a graph similarity metric, with the following test:

1. The eigenvalues of the two graph *Laplacian* matrices are calculated and sorted.
2. The two-sample K-S test is conducted to test whether the underlying distributions differ for the two sets of eigenvalues.
3. The resulting p-value of the K-S test is used as the graph similarity score.
4. The two graphs are identified as similar if the similarity score is larger than a preset tolerance, tol.

2.5 Symmetry Constraint with Graph Neural Networks

Recent advances in graph neural networks (GNNs) show great potential toward a more accurate and efficient analog layout constraint annotation. As a circuit netlist can naturally be modeled as a graph, GNNs learn the interconnect structures by mining graph information. In [22], a graph learning framework with path-based features to mimic electric potential in circuit analysis is presented. Leveraging a probability-based filtering technique, the false positive rates can be reduced.

The method is based on a supervised inductive graph-learning-based methodology for device-level AMS symmetry constraint extraction. Fig. 7 shows the overall flow, which consists of three main stages: pre-processing, GraphSage-based detection model, and post-processing.

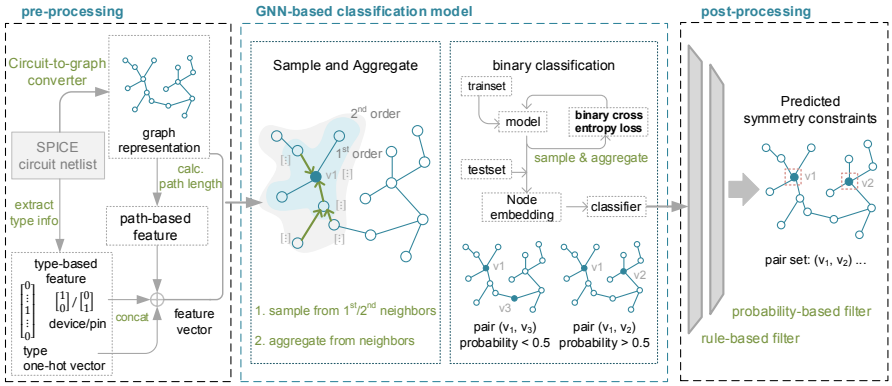


Fig. 7: Workflow of the GNN-based framework in [22].

The pre-processing stage takes raw SPICE analog netlists as input and constructs graph representations for each circuit. In this stage, feature vectors are extracted from the type information of the netlists and the structure of the graph. In the detection model stage, symmetry constraint detection problem is mapped into a binary classification problem. GraphSage [28], a general inductive approach that

generates node embeddings for graph data, is adapted to measure the similarity between a pair of nodes. The model is modified and trained in a supervised manner. The modified model can produce a set of potential pairs with symmetry constraints. All predicted pairs will go through a rule-based filter and a probability-based filter in the post-processing stage to eliminate most of the false positive pairs. After filtering, our framework produces the symmetry constraints detected.

The device type information is encoded as part of the node feature. To distinguish device nodes from pin nodes, a two-dimensional vector to indicate whether a node is a device or a pin is used, where $[0, 1]$, $[1, 0]$ stand for a device and a pin, respectively. Then, the device types (i.e., capacitor, resistor, diode, NMOS, PMOS, IO) and pin types (i.e., source, drain, gate, substrate, passive, cathode of a diode, and anode of a diode) are translated into two one-hot vectors. The representation includes power/ground nets and a power node and a ground node as auxiliary nodes. These type-related vectors make up the first two parts of node features.

A novel path-based feature is also proposed inspired by the electric potential in circuit analysis. The feature is not to simulate the circuit, but to characterize the "global position" of each node in the graph by VSS/Ground-sourced path lengths. The neighbor structures of the two nodes are not the same, but the electric potential values of their corresponding pins are the same according to DC analysis. The path-based feature intended to capture the intuitive relevance between electric potential and path from ground node by developing a path-based feature.

Aggregator functions are crucial to the sampling and aggregation process. The mean aggregator concatenates the current node representation h_v^{k-1} and the average of aggregated neighbor node representations. $\mathcal{N}(v)$ stands for the sampled neighbor set of node v and W is a learnable parameter matrix. $\sigma(\cdot)$ denotes an activation function that introduces nonlinearity to our model. ReLU is taken as the activation function $\sigma(\cdot)$ in aggregation process. Equation (4) summarizes the mean aggregator,

$$h_v^k = \sigma\left(W \cdot \{h_v^{k-1} \oplus MEAN(h_u^{k-1}, \forall u \in \mathcal{N}(v))\}\right). \tag{4}$$

To train the aggregator functions mentioned before, we apply binary cross entropy loss, a classic loss function of binary classification, which facilitates high accuracy in our applications. The loss function is declared as:

$$loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(prob_i) + (1 - y_i) \cdot \log(1 - prob_i), \tag{5}$$

where N denotes the number of node pairs, y_i and $prob_i$ are the ground truth label and predicted label of the i_{th} pair respectively.

In [12], an unsupervised inductive graph-learning-based methodology for both system-level and device-level AMS symmetry constraint extraction is proposed. Fig. 8 shows the overall computation flow. With the unsupervised learning technique, the proposed framework learns a strategy for extracting latent information of matching circuit structures, thus amenable to general AMS designs.

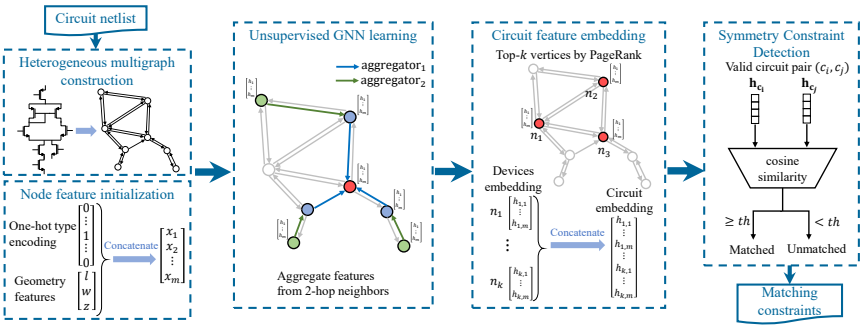


Fig. 8: Computation flow of the GNN-based framework in [12].

In the framework, a heterogeneous multigraph (i.e., multigraph with various edge types) for circuit netlist representation is first constructed. The proposed heterogeneous multigraph model consists of four different types of edges, representing the connections between different ports of a devices. Fig. 9 illustrates an example of the heterogeneous multigraph representation. In the figure, the four devices m_0, m_1, m_2, and C_L are mapped to four graph vertices. Consider the connection from the drain of m_1 to the drain of m_2, an edge $e_1 = (m_1, m_2, p_{drain})$ is added. Other edges are added similarly. Note that parallel edges occur if some nets connect to multiple terminals of a transistor.

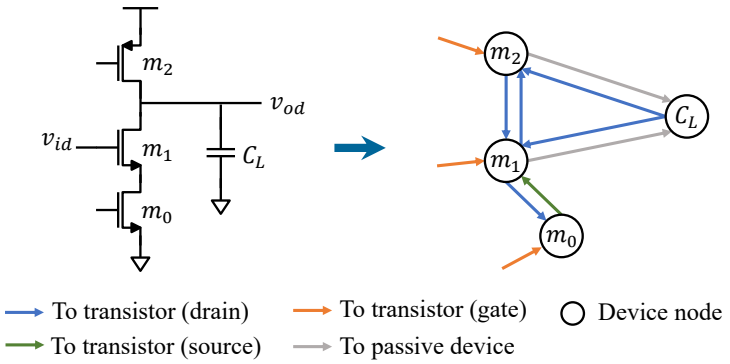


Fig. 9: Example of the heterogeneous multigraph model in [12].

After constructing the heterogeneous multigraph, the initial feature vector for each vertex in the graph is determined. As modern analog devices are much more sophisticated, in order to precisely describe a device's physical structure, dozens of design parameters are needed. However, using all the design parameters might cause overfitting in the learning model and restrict its ability to identify nonidentical

matching structures. Therefore, the used features and their dimension are summarized in Table 1. The first 15 dimensions form a one-hot vector that represents the devices type (e.g., NMOS, PMOS, MOM capacitor). Besides, [12] also integrates physical details of circuit sizing as features, and therefore lowers the false alarms significantly.

Feature	Length	Description
Device type	15	The one-hot device type encoding.
Geometry	2	The length and width of the device
Layer	1	The number of metal layers.

Table 1: Initial vertex features in the heterogeneous multigraph of the input circuit.

With the constructed multigraph and the initial features, the framework then iteratively aggregates the features of neighbor vertices to recognize the localized interconnection and peripheral structures of each vertex. The feature aggregating function to aggregate the features of K-hop neighbors is shown as follows.

$$h_v^{(k)} = GRU\big(h_v^{(k-1)}, \sum_{u \in \mathcal{N}_{in}(v)} W_{e_{uv}} h_u^{(k-1)}\big), \tag{6}$$

where $h_v^{(k)}$ is the feature vector of vertex v at the k^{th} layers of the GNNs, $GRU(\cdot,\cdot)$ denote a gated recurrent unit [17], $\mathcal{N}_{in}(v)$ is the in-neighbors of v, and $W_{e_{uv}}$ represents the linear transformation matrix with respect to edge e_{uv}. In [12], K is set to 2, and the output dimension of each neural network is set to 18. The GNN model is then trained with an unsupervised loss function

$$\begin{aligned} \mathcal{L}_{tot} &= \sum_{v \in V} \mathcal{L}(z_v), \\ \mathcal{L}(z_v) &= -\sum_{u \in \mathcal{N}_{in}(v)} \log(\sigma(z_u^\intercal z_v)) - \sum_{i=1}^{B} \mathbb{E}_{\tilde{u} \sim Neg(v)} \log(1 - \sigma(z_{\tilde{u}}^\intercal z_v)), \end{aligned} \tag{7}$$

where $z_v = h_v^{(K)}$ is the final feature representation of a vertex v, $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function, $\mathbb{E}$ denotes the expected value, $Neg(v)$ is the negative sampling distribution with respect to v, and B denotes the total number of negative samples. Minimizing the overall loss $\mathcal{L}_{tot}$, the GNN models learns the strategy to improve feature similarity between each vertex and its neighbors while enlarging its discrepancy with the negative samples, and thus implicitly integrates the localized structure information into each vertex.

With the trained feature vectors of each device, the subcircuit feature representation is then determined by concatenating the top-M representative vertices. The PageRank algorithm [63] is utilized to select the top-M representative nodes. The PageRank score of each vertex v can be computed as follows.

$$PR(v) = \frac{1-\gamma}{|V|} + \gamma \cdot \sum_{u \in \mathcal{N}_{in}(v)} \frac{PR(u)}{|\mathcal{N}_{out}(v)|}, \tag{8}$$

where V is the vertex set, γ is the damping factor, $\mathcal{N}_{in}(v)$ is the set of in-neighbors (i.e., $\forall u \in V$ such that there exists an direct edge from u to v) of v, and $\mathcal{N}_{out}(v)$ is the set of out-neighbors (i.e., $\forall u \in V$ such that there exists an direct edge from v to u) of v.

Finally, symmetry constraints are generated by comparing the cosine similarity of the trained features for the subcircuits and devices, where the cosine similarity λ_{sim} of two trained features z_i and z_j is defined as

$$\lambda_{sim} = \frac{z_i \cdot z_j}{\|z_i\| \|z_j\|}. \tag{9}$$

Given a similarity threshold λ_{th}, if the cosine similarity of the features of two devices (resp. subcircuits) is larger than λ_{th}, then a device-level (resp. system-level) symmetry constraint between the two objects will be added.

3 Constrained Placement and Routing

3.1 Placement Quality Prediction

During iterative placement and routing, the ability of a candidate layout to meet performance specifications depends on interconnect RC parasitics. This is illustrated in Table 2: post-layout parasitics can cause as much as 22% loss of unity gain frequency from the schematic (pre-layout) values for this testcase.

Characteristic	Schematic	Layout	Change
DC Gain (dB)	39.30	37.25	–5%
Bandwidth (MHz)	10.64	10.47	–2%
Unity Gain Frequency (MHz)	440	383	–22%

Table 2: Schematic and post-layout performance of an OTA.

We overview a set of compact ML-based models for a cost function component that rapidly predicts the quality of a candidate layout. This can be used to reward (or penalize) the optimization cost function when the layout meets (or fails to meet) performance specifications.

3.1.1 Applying Standard ML Models

In [42], the performance of a circuit is evaluated by a set of performance functions $z_1, z_2, ...$ for an analog block. For each z_i, a *satisfaction function* is defined as

$$\psi_i(z_i) = \begin{cases} 1 & z_i \geq \theta_i \\ \frac{z_i}{\theta_i} & z_i < \theta_i \end{cases} \tag{10}$$

where θ_i is the design specification for z_i. For K performance functions, if w_i represents the weight factor of function i s.t. $\sum_{i=1}^{K} w_i = 1$, the performance Figure of Merit (FOM) is defined as

$$\Phi = \sum_{i=1}^{K} w_i \cdot \psi_i(z_i) \tag{11}$$

If all performance specifications are satisfied, $\Phi = 1$. To estimate the probability that a specification is satisfied, a Probability of Demerit (POD) is defined as:

$$\Delta = \sum_{i=1}^{K} w_i \cdot P(z_i < \theta_i) \tag{12}$$

where P indicates the probability of violating specifications. ML models are built for Δ, and it is incorporated as an additional component of the placement cost function.

In [42], the feature space corresponds to the set of lengths of all interconnects in the circuit. All wires are assumed to have the same width and thickness, and therefore, the RCs of these wires depend on their lengths. Three ML models are evaluated: neural network (NN) [37], random forest (RF) [6] and support vector machine (SVM) [68]. Training is performed using a pre-routing estimate based on the star model. The accuracy of these models on different circuit characteristics of different OTA designs is shown in Fig. 10. It is seen that NN outperforms both RF and SVM on every case, while SVM is better than RF on gain, bandwidth, and phase margin.

3.1.2 Stratified Sampling with SVM/MLP Models

In [19], a framework was proposed for extracting these relationships by building ML models for each performance constraint, extracting both linear and nonlinear correlations among all the sensitive parasitics over a multidimensional search space of RC parasitics. It consists of the following steps:
(1) Feature space pruning: This step reduces the dimension of the feature space of RC parasitics by (a) identifying variables that the performance constraints are insensitive to, and (b) by range reduction, which determines upper bounds on the parasitics.
(2) Sparse sampling and linear SVM classification: Next, a sparse sample set in the updated feature space is generated using stratified sampling based on the Latin hypercube sampling method. These samples are labeled for each performance constraint

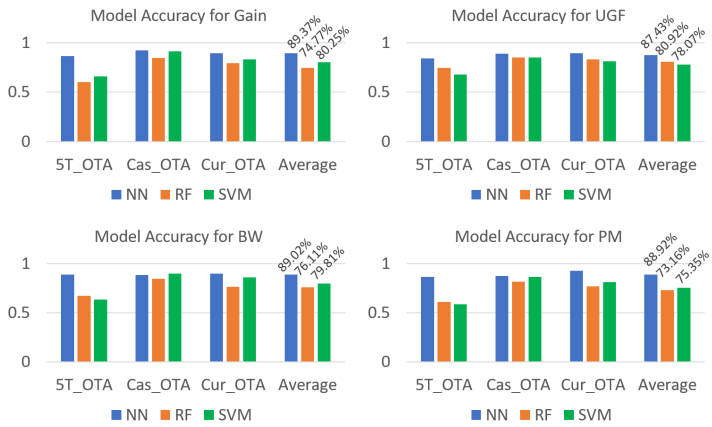


Fig. 10: ML model accuracy on gain, unity gain frequency (UGF), bandwidth (BW) and phase margin (PM).

associated with performance parameter p_k. For each p_k, a support vector machine (SVM) model is built to extract correlations among the features. If the classification error falls below a user-specified threshold, ϵ_0, the SVM provides a good model for p_k.
(3) Dense sampling and classification: If the error exceeds ϵ_0, a larger number of samples is generated to drive higher accuracy, labeling each sample. Next, an SVM is built with the denser samples: if its error is within a user-specified threshold, ϵ_1, the SVM is a good model; else, a multilevel perceptron (MLP) model is built.

Performance specifications	5T OTA		Telescopic OTA		Two-stage OTA	
	With framework	Without framework	With framework	Without framework	With framework	Without framework
Gain (dB)	20.57 ✓	19.09 ✓	42.13 ✓	38.12 ✗	26.57 ✓	24.38 ✗
BW (MHz)	103.26 ✓	126.20 ✓	5.49 ✓	7.64 ✓	46.84 ✓	41.22 ✓
UGF (GHz)	1.17 ✓	1.14 ✓	0.70 ✓	0.61 ✗	1.00 ✓	0.92 ✗
PM (^{o})	110.33 ✓	116.77 ✓	133.41 ✓	106.50 ✓	94.43 ✓	82.05 ✓
CMRR (dB)	52.08 ✓	52.92 ✓	69.15 ✓	62.14 ✗	32.71 ✓	38.27 ✓
PSRR (dB)	21.39 ✓	18.47 ✗	42.45 ✓	53.52 ✓	26.94 ✓	24.37 ✗
SR (V/μS)	156.62 ✓	156.63 ✓	414.24 ✓	424.23 ✓	408.19 ✓	386.07 ✓
ICMR (V)	0.60-0.75 ✓	0.60-0.75 ✓	0.55-0.85 ✓	0.55-0.85 ✓	0.60-0.75 ✓	0.60-0.75 ✓

Table 3: Post-layout performance of the OTA testcases.

The constraint framework is applied to guide the placement of the above OTA and VCO circuits within the placement engine in ALIGN [1, 18, 39], adding a penalty for violating a performance constraint. The performances of the three OTA layouts (Fig. 11), extracted from post-layout analyses, are summarized in Table 3. All three of the automatically generated layouts maintain the required design constraints (✓),

but layouts generated without the framework fail one or more constraints (✗). For the VCO schematic in Fig. 12(a), Fig. 12(b) shows the circuit layout, and it meets all specifications.

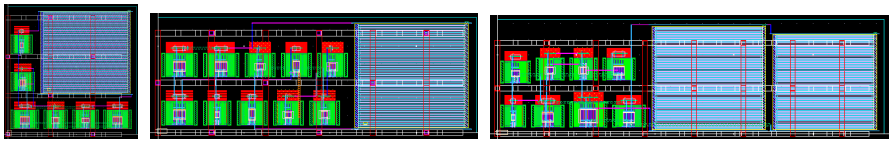
3.1.3 Performance Prediction with Convolutional Neural Networks

In [47], a 3D CNN model is proposed to predict the placement quality of OTA circuit layouts. The complex and intricate nature of analog circuit behaviors make extracting performance relevant features from placement extremely important. The performance impact of a device placement lies in both the placement location and circuit topology. As an example, the mismatch of differential input pairs has a larger impact towards offset compared with the load. Thus, to ensure a good and generalized model, extracted features must be both easily extendable to different circuit topologies and able to encode effective placement information.

To leverage the success of convolutional neural networks in computer vision tasks, we represent intermediate layout placement results into 2D images. Instead of compacting the entire circuit placement into a single image, we separate devices into different images based on the circuit topology. For OTA circuits, we propose to divide the circuit into subcircuits based on functionality as shown in Fig. 13.

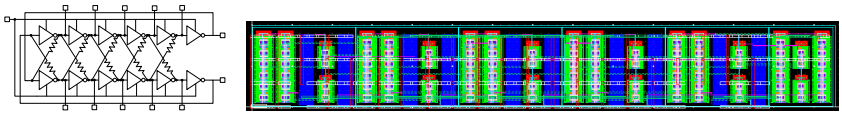
The devices are abstracted into rectangles and scaled according to the placement results into a image. The net routing demand is the aggregated pin boundary box for each net. In all our experiment, the image size is selected to be 64*64. Device types are encoded as different intensities in the image. Figure 14 show a OTA layout with the corresponding extracted placement feature images.

Convolutional neural networks have been primarily applied on 2D images as a class of deep models for feature construction. Conventional 2D CNNs extract features from local neighborhoods on feature maps in the previous layer. Formally, given the pixel value at position (x, y) in the jth feature map in the ith layer, the convolutional



(a) (b) (c)

Fig. 11: Automated layouts of the (a) 5T OTA (9.63μm × 9.60μm), (b) telescopic OTA (6.85μm × 18.65μm), and (c) two-stage OTA (7.42μm × 24.49μm) with constraints generated by the framework. [Not drawn to scale]



(a) (b)

Fig. 12: VCO (a) schematic, (b) layout (10.11μm × 72.78μm) using the method.

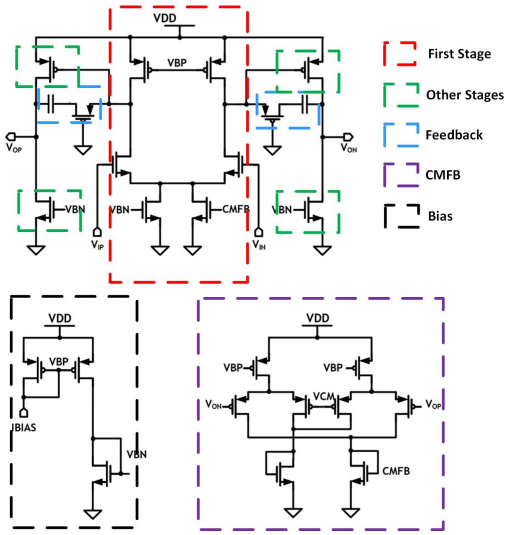


Fig. 13: Subcircuits of OTA [47].

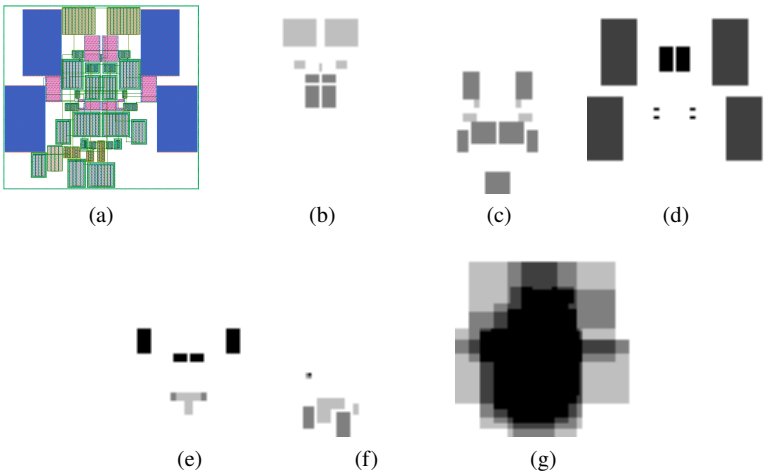


Fig. 14: (a) is the OTA layout. (b)-(g) are the extracted image features of the layout with first stage, other stage, feedback, CMFB, bias, and net routing demand, respectively [47].

layer output v_{ij}^{xy} is given by

$$v_{ij}^{xy} = \sigma\Big(\sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1} w_{ijm}^{pq} v_{(i-1)m}^{(x+p)(y+q)} + b_{ij}\Big), \tag{13}$$

where $\sigma(\cdot)$ is the activation function, b_{ij} is the bias for feature map, m indexes over the set of feature maps in this layer, and w_{ijm}^{pq} is the value of the weight kernel at the position (p, q) connected to the kth feature map. The output feature is thus the activation output of a weighted sum over all the kernel maps with the previous layer images.

3D convolution layers were first proposed to incorporate both spatial and temporal information for action recognition in videos. In contrast to 2D CNNs where the convolution kernel is a 2D map, 3D convolution is achieved by convolving a 3D kernel to the cube formed by stacking multiple contiguous images together:

$$v_{ij}^{xyz} = \sigma\left(\sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)} + b_{ij}\right), \tag{14}$$

with r being the value across the third dimension. Images captured across time from videos were stacked to form a 3D input tensor for action recognition.

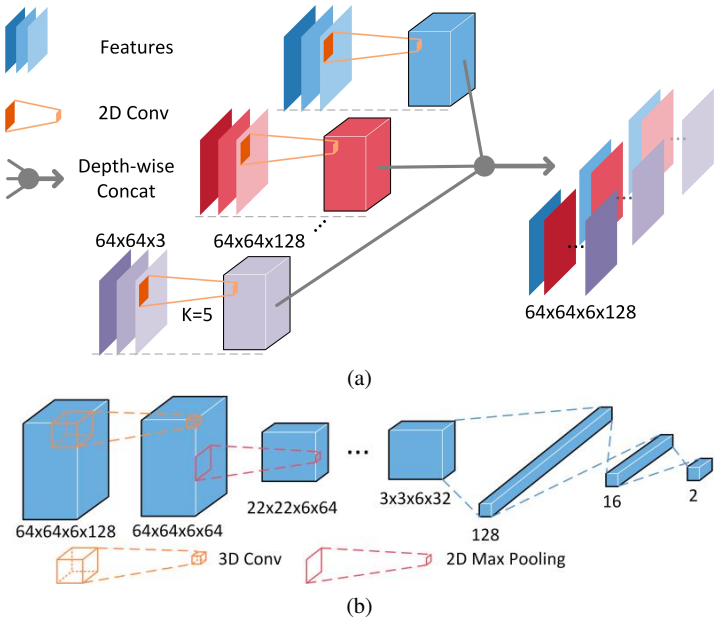


Fig. 15: Neural network architecture. (a) Initial separate 2D CNN. (b) 3D CNN classifier [47].

The use of 3D CNNs to effectively capture the relative location information between the different placement subcircuits. Fig. 15 shows the overall model of the 3D CNN network for placement quality prediction. Each extracted placement feature image is augmented into feature sets with coordinate channels. Initial features are then extracted separately for each feature sets with 2D convolutional layers. The

outputs are then stacked to form 3D tensors. The 3D tensors are fed to the 3D CNN for placement quality prediction.

3.1.4 PEA: Pooling with Edge Attention using a GAT

In a graph attention network (GAT) [69] on a graph $G(V, E)$, 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 $X \in \mathbb{R}^{n\times d}$, and the set E is represented by an adjacency matrix $A \in \{0, 1\}^{n\times n}$. A trained GNN takes X as input and decides the class of an entire graph or the class of every node in a graph.

In each layer, GAT computation consists of two steps: *weighting*, which computes XW, where $W \in \mathbb{R}^{d\times d}$ is a trainable weight matrix, and *aggregation*, where each node $v_i \in \mathcal{V}$ collects feature information of its neighboring nodes. A generic graph convolution operation in layer l is described as

$$Z^{(l)} = \sigma(\Phi_A^{(l)} X^{(l)} W^{(l)}) \tag{15}$$

where $\sigma(\cdot)$ is an activation function (e.g., sigmoid), and the form of Φ_A is elaborated in [69] and Chapter 4. The **attention** coefficient α_{ij} from node v_j to v_i is given by

$$\begin{aligned} \alpha_{ij} &= \text{softmax}_{row}(\tau_{ij}) = \frac{e^{\tau_{ij}}}{\sum_{k\in\mathcal{N}_i} e^{\tau_{ik}}} \\ \tau_{ij} &= \text{LeakyReLU}(a^{(l)} \cdot [(W^{(l)^{\mathrm{T}}} X_i^{(l)}) || (W^{(l)^{\mathrm{T}}} X_j^{(l)})]) \end{aligned} \tag{16}$$

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. Finally, the **pooling** operation is based on DiffPool [81].

The circuit netlist is 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 $\mathcal{E}$. 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: (1) Device type: PMOS, NMOS, capacitor, current source, GND, etc; (2) Functional module where the device belongs to, such as bias current mirror, differential pair, and active load; (3) Device dimension; (4) Device location.

In [41], a new edge attention network, PEA, is introduced. The network, shown in Fig. 16, is composed of two stages: feature extractor and predictor. The extractor consists of multiple PEA layers, each of which includes graph convolution and graph pooling. The predictor is an MLP. The key ingredient of PEA network is the integration between edge feature/attention and graph pooling. PEA is composed of four phases: (1) Edge-aware attention construction and compression; (2) Graph convolution; (3) Node pooling; (4) Edge pooling. PEA customizes a standard GAT primarily in phases 1 and 4.

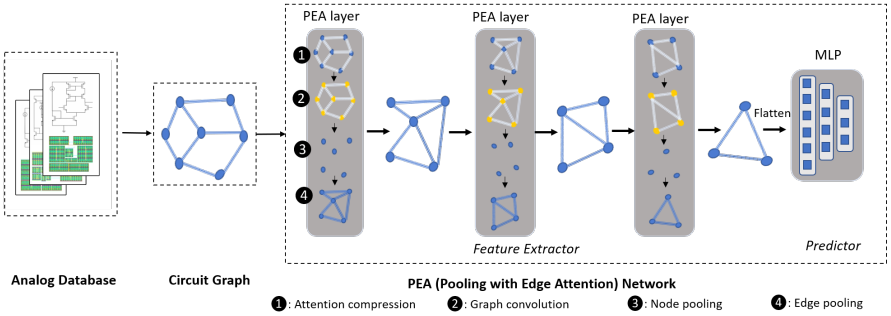


Fig. 16: Overview of the PEA network.

Edge attention was previously proposed in [23] by expanding the attention matrix from 2D to 3D using channels so that $\alpha^{(l)} \in \mathbb{R}^{n_l \times n_l \times p_l}$, but this approach is expensive in runtime and memory use. The edge-aware attention model in PEA overcomes this by defining the raw attention as

$$\hat{\alpha}_{ijk}^{(l)} = f^{(l)}(X_i^{(l)}, X_j^{(l)}, E_{ijk}^{(l)}) = \tau_{ij} E_{ijk}^{(l)} \tag{17}$$

where $E_{ijk}^{(l)}$ corresponds to the edge featue and τ_{ij} is given by Equation (16). The attention matrix is obtained through bidirectional normalization (BN) as

$$\alpha^{(l)} = BN(\hat{\alpha}^{(l)}) = \begin{cases} \tilde{\alpha}_{ijk}^{(l)} = \text{softmax}_{row}(\hat{\alpha}_{ijk}^{(l)}) \\ \alpha_{ijk}^{(l)} = \sum_{m=1}^{n_l} \frac{\tilde{\alpha}_{imk}^{(l)} \tilde{\alpha}_{jmk}^{(l)}}{\sum_{u=1}^{n_l} \tilde{\alpha}_{umk}^{(l)}} \end{cases} \tag{18}$$

where n_l is the number of nodes at layer l. This normalization 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.

After the attention compression, graph convolution is performed in the same way as the conventional approach, except that the attention is replaced by the compressed version, $g(\alpha^{(l)}; b^{(l)})$, and this is followed by node-pooling. This is succeeded by *edge pooling*, an original contribution of [41], consisting of 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 conventional node pooling.

The channel pooling operation $h : \mathbb{R}^{p_l} \rightarrow \mathbb{R}^{p_{l+1}}$ is performed as follows:

$$\begin{aligned} Q_{ij}^{(l)} &= h(\alpha_{ij}^{(l)}; W_{edge}^{(l)}) \\ Q_{ijk}^{(l)} &= \sum_{m=1}^{p_l} \alpha_{ijm}^{(l)} {W_{edge}}_{mk}^{(l)} \end{aligned} \tag{19}$$

where $Q^{(l)} \in \mathbb{R}^{n_l \times n_l \times p_{l+1}}$ is edge-feature-encoded attention and $W_{edge}^{(l)} \in \mathbb{R}^{p_l \times p_{l+1}}$ is a trainable weight matrix. This transformation 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 (17), so does Q.

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

$$\begin{aligned} E^{(l+1)} &= t(S^{(l)\mathrm{T}}, Q^{(l)}, S^{(l)}) \\ &= ||_{k=1}^{p_{l+1}} (S^{(l)\mathrm{T}} Q_{\cdot\cdot k}^{(l)} S^{(l)}) \end{aligned} \tag{20}$$

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, "$\cdot\cdot k$" is a slicing operation defined by

$$(Q_{\cdot\cdot k}^{(l)})_{ij} = Q_{ijk}^{(l)} \quad i, j \in 1, 2, ..., n_l \tag{21}$$

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

$$U = ||_{k=1}^{p_{l+1}} V_k, \quad U_{ijk} = (V_k)_{ij} \tag{22}$$

where $V_k \in \mathbb{R}^{n_{l+1} \times n_{l+1}}$, $k \in 1, 2, ..., p_{l+1}$. Edge pooling is illustrated in Fig. 17.

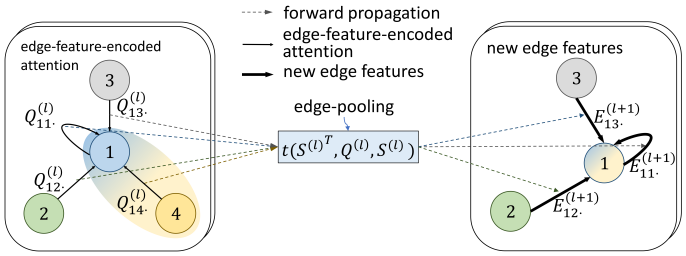


Fig. 17: Edge pooling in the PEA network.

A PEA network with L layers can be applied to determine whether performance constraints $y_i \geqslant \phi_i, i = 1, 2, ..., m$ are satisfied. The overall performance cost can be defined as

$$Q_I = \sum_{i=1}^{m} w_i \cdot P(y_i < \phi_i) \tag{23}$$

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

$$Q_{II} = P\left(\sum_{i=1}^{m} w_i \cdot \min(\frac{y_i}{\phi_i}, 1) < \mathcal{T}\right) \tag{24}$$

where $\mathcal{T}$ is the specification of overall performance and can be obtained according to legacy designs. The classification on whether $\sum_{i=1}^{m} w_i \cdot \min(\frac{y_i}{\phi_i}, 1) < \mathcal{T}$ can be obtained through PEA network. Cost Q_{II} relies on an additional threshold $\mathcal{T}$ compared to Q_I. However, it requires only one output from PEA while Q_I needs m outputs from PEA.

	Schematic	Manual	Conventional Automatic	CNN		PEA		
				SS Q_{II}	Transfer Q_{II}	SS Q_{II}	Transfer Q_{II}	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).

A sample result for a cascode OTA is shown in Table 4. Here, *self-sustained learning* 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. 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. The three PEA-guided results are significantly closer to manual layout than both the previous work [49] and placement guided by the CNN-based model [47].

3.2 Analog Placement

The placement stage determines the locations of each module and device. Analog placement constraints are imposed to the placer to mitigate the layout effects on circuit electrical behaviors. A typical analog placer optimizes area and wirelength.

3.2.1 Incorporating Extracted Constraints into Analog Placers

Conventionally, analog placement engines take manually labeled constraints as input. Recent fully-automated analog layout frameworks, such as MAGICAL [9, 77] and ALIGN [1, 18, 39], have moved towards replacing human-generated constraints

with automatic generated constraints. Geometric constraints such as symmetry are extracted from the netlist, as discussed in Section 2.

To handle performance constraints, the cost function for placement must be augmented using the techniques discussed in Section 3.1. The stochastic approach is a classical paradigm on solving AMS placement problems, where a placement solution is represented with an intermediate data structure, such as O-tree [65] and segment tree [5]. Then a randomized search scheme, such as a simulated annealing-based optimizer, perturbs the underlying data structure and searches for an optimal solution. Other solution algorithms used for placement include mixed integer linear programming (MILP) [75, 73] and nonlinear programming (NLP) [62, 74, 85], where the input constraints are coded as part of MILP problem and are automatically handled during optimization. However, the scalability of MILP-based placer is limited. NLP, on the other hand, relaxes the hard constraints into penalties in the objective function and is therefore, in general, more efficient in computation compared to MILP. When NLP is used, the final placement must be legalized after the termination of the nonlinear optimization step.

3.2.2 Well Generation and Well-Aware Placement

The well layer defines the doping area that acts as the bulks of MOSFETs. For example, a typical P-MOS device needs to be built on an N-well layer and needs contacts to supply the bulk voltage (usually VDD). In a typical digital design methodology, wells are pre-designed within standard cells so that well generation is not needed in layout automation flow. However, analog design methodologies often use customized device layouts, and wells are usually distinctly drawn in manual designs. Manual designs often share wells between transistors to reduce spacing and the number of contacts to optimize area and interconnection. Furthermore, well geometries also impact circuit performance through layout-dependent effects, such as the well proximity effect (WPE). Therefore, inserting wells is an additional task to analog layout flow compared to its digital counterpart.

In [76], a ML-guided well generation framework is proposed. The WellGAN framework generates wells following the guidance from a generative ML model, a generative adversarial network (GAN) [24]. Fig. 18 shows the overall flow for the WellGAN framework. In the training phase, an AMS circuit layout database is utilized to build a conditional-GAN model [55] for the inference. In the inference phase, the trained GAN model predicts the well region and guides the well generation. Fig. 19 gives an illustration of the WellGAN framework. It presents the placement features and wells as images. A generator network G produces the well region guidance from the placement features. A discriminator network D is also utilized to assist the training process.

Data Representation The WellGAN framework represents the layout with images. The oxide diffusion (OD) layers are selected to be the input patterns. The first channel of an image represents the OD layers within the wells, such as PMOS devices for N-Well generation task. The second channel extracts the OD layers outside the

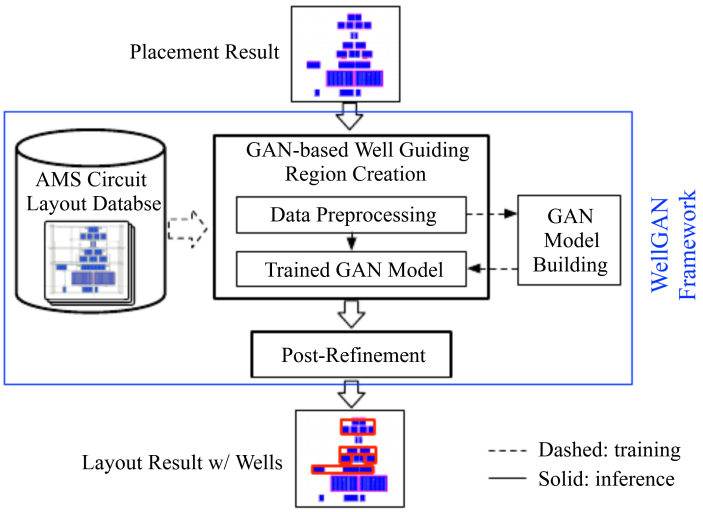


Fig. 18: The WellGAN framework [76].

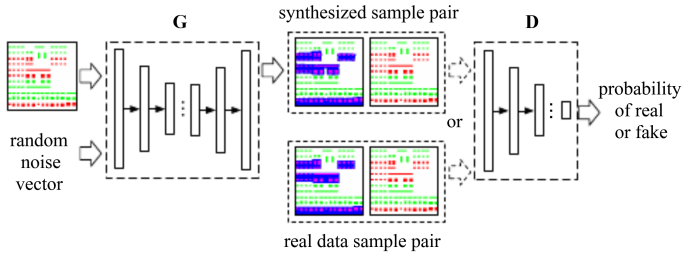


Fig. 19: An illustration of the WellGAN framework [76].

wells, such as PMOS devices. The third channel is for the targeting well guidance. The ground truth of well regions and the outputs of ML prediction are encoded in this channel.

Data Preprocessing The data pre-processing step extracts the OD layers and well shapes from the layouts. In the training phase, the manual-designed layouts are pre-processed into the three channel images containing OD and well layers. In the inference phase, the OD layers of the placement are extracted into two channels. Then clipping, zero-padding, and scaling are applied to transform the layouts into equally-sized image clips to facilitate the modeling.

Well Guidance Generation WellGAN uses a conditional GAN (CGAN) to predict the well regions. It takes the processed data and generate the well region guidance. CGAN simultaneously trains two models: a generative model G and a discriminative model D. The generator G observes input x, the input placement features represented as images, and a random noise vector z. It learns to generate the output

y, which includes the placement features and well images. On the other hand, the discriminator D sees y and x and learns to discriminate the "fake" generated y from the ground truth. The CGAN minimizes the loss function in the training process as shown in Equation (25).

$$\begin{aligned}\min_G \max_D \mathcal{L}_{CGAN} &= \mathbb{E}_{x,y\sim p_d(x,y)}[\log D(x,y)] \\ &+ \mathbb{E}_{x\sim p_d(x),z\sim p_z}[\log(1-D(x,G(x,z)))] \\ &+ \lambda_{L_1}\mathbb{E}_{x,y\sim p_d(x,y),z\sim p_z}[\|y-G(x,z)\|_1],\end{aligned} \tag{25}$$

where p_d and p_z denote the probability distributions for the learning targeting and random noise vector z.

The CGAN network is trained to learn the well regions from training data extracted from manual layouts.

Post-Refinement After obtaining the well region guidance in image form from the CGAN model, a post-refinement stage finally generates the well shapes. The image clips are first merged to reconstruct the whole layout. Then the image is transformed into a binary image by applying a threshold the pixel values. The polygons in the binary image is extracted and rectilinearized. The resulting rectangles are then legalized by mapping them to a coarse grid and resolving remaining spacing rule violations. After the post-refinement, the resulting well shapes are inserted into the layout.

Table 5 shows the element-wise difference distribution from the manual layouts. The smaller mean error and standard deviation demonstrates the WellGAN framework better mimics the manual layout expertise through ML guidance.

Metric	WellGAN	Baseline
Mean	5.67%	12.65%
Standard Deviation	3.58	10.25

Table 5: Statistics of Manhattan norm of element-wise difference for the test results [76].

In [84], the WellGAN framework is further extended to integrate with the analog placement engine. Fig. 20 shows the overall flow of the framework. Different from the WellGAN flow where the well generation is an independent step from the placement, in the ML-guided well-aware analog placement engine, placement and well generation are iteratively optimized so that the resulting placement solution is well-aware and encourages well sharing to achieve further optimized area.

The well-aware global placement is formulated as an NLP optimization problem, where area, wirelength and constraints are simultaneously considered in the objective function. The well guidance from the GAN model is transformed as a fence region objective and being optimized together with the other terms in the NLP problem. After an iteration of global placement, the current placement is updated and the GAN

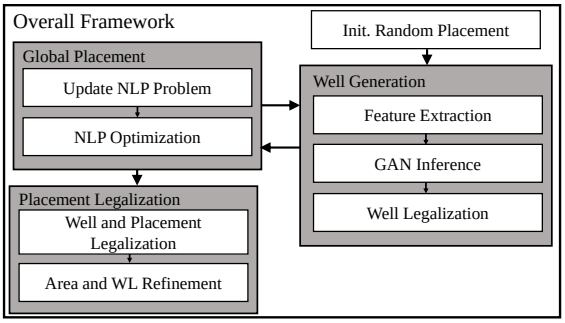


Fig. 20: The overall flow of the ML-guided well-aware analog placement [84].

model re-predicts the well guidance. This synergistic process seamlessly integrates the placement and ML-guided well generation subroutines.

After the global placement, the well shapes are generated and legalized. The wirelength and area are further optimized with a linear programming-based refinement subroutine. Table 6 shows the experimental results of the well-aware placement framework over individual well flow and the WellGAN flow. On the average of ratios, the work reduces area (HPWL) by 82% (26%) over "individual wells" and 74% (46%) over "WellGAN". It demonstrates the effectiveness of the ML-guided well-aware placement algorithm.

CKTS	Individual wells			WellGAN			Well-Aware Placement		
	Area	HPWL	RT	Area	HPWL	RT	Area	HPWL	RT
OTA1	360.2	72.3	**1.3**	318.0	68.7	3.2	**290.3**	**60.3**	3.6
OTA2	756.2	234.7	**4.8**	750.7	**203.1**	7.9	**599.0**	205.2	10.6
OTA3	1055.4	586.6	48.9	1325.6	**559.5**	43.2	**965.6**	651.3	**34.1**
OTA4	3255.2	837.1	**39.7**	3313.6	**799.6**	40.1	**3033.7**	866	42.6
COMP1	175.1	78.8	**2.0**	144.4	95.1	6.6	**82.2**	**61.8**	3.5
COMP2	192.2	93.1	**3.0**	194.2	105.0	5.6	**84.7**	**48.1**	3.6
BOOTSTRAP	177.9	64.5	**2.0**	130.8	83.4	5.0	**97.5**	**63.2**	4.8
RDAC	361.5	209.2	**12.4**	370.4	287.0	30.2	**144.3**	**137**	23.7
Norm.	1.82	1.26	**0.64**	1.74	1.46	1.33	**1.00**	**1.00**	1.00

Table 6: Comparison of area (μm^2), HPWL(μm), and runtime (RT(s)) [84].

3.3 Analog Routing

The routing stage completes the signal connections using available routing resources to meet various design constraints while meeting certain design objectives.

3.3.1 Incorporating Extracted Constraints into Analog Router

In addition to wirelength optimization and design rule handling, which is the main target of conventional digital routing, a comprehensive AMS router should consider more complicated design aspects, including voltage drop, current balancing, parasitics, and signal coupling. For better scalability, these design considerations are usually formulated into geometrical layout constraints. As matching is an essential concept in analog layouts, techniques for various constraints handling such as symmetry, common-centroid, topology-matching, and length-matching have been widely studied. In [72, 64], maze routing algorithms supporting mirror-symmetry constraints are proposed. In [57, 79], length-matching routing approaches for general routing topologies are presented. In [60], an ILP formulation for analog routing simultaneously considering symmetry, common-centroid, topology-matching, and length-matching is presented.

For real-world designs, besides the straightforward mirror symmetry constraints, variants of symmetry constraints are also frequently adopted to describe more sophisticated matching structures of nets. [11] extends the conventional symmetry constraints into four variants: mirror-, cross-, self-, and partial-symmetry, as shown in Fig. 21 for a bulk technology node.

The aforementioned geometrical constraints enforced on matching nets during the routing procedure are usually correlated with the placement constraints as placement results determine the overall wiring topologies. As a result, routing constraints can also be annotated by the methods such as the graph similarity and graph neural network frameworks described in the previous section.

3.3.2 GeniusRoute: ML-Guided Analog Routing

ML methods can also be useful in generating high-quality routing guidances. In [86], a new ML-guided analog routing paradigm, GeniusRoute, is proposed. Existing placement and routing algorithms for AMS circuits usually rely on human-defined heuristics or constraints, such as symmetry and signal flow, to mitigate layout-induced performance issues. In practice, the performance of analog circuits is sensitive to even minor layout changes. The proposed ML-guided analog routing uses a variational autoencoder (VAE) to learn the routing strategies from manual layout database and apply the learned knowledge to guide an automatic routing flow.

Data Representation and Pre-Processing The data in the GeniusRoute are represented as images. Three types of information from a manual layout are extracted to construct a channel in the image. First, the pins of all nets in layout are drawn in

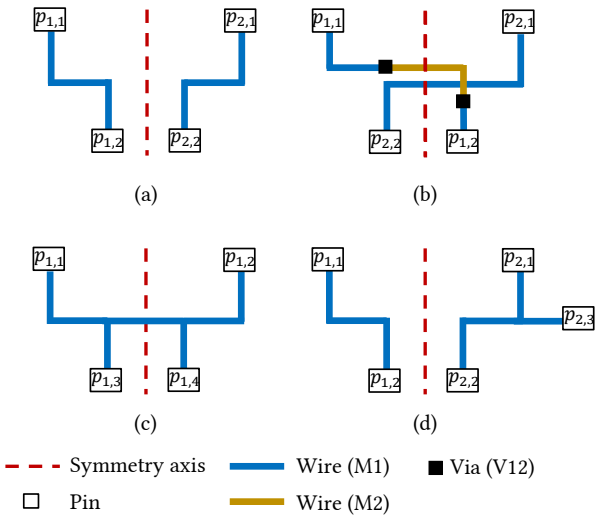


Fig. 21: Examples of variants of symmetry constraint. (a) Mirror-symmetry constraint. (b) Cross-symmetry constraint. (c) Self-symmetry constraint. (d) Partial-symmetry constraint [11].

the image to give a global view on the placement. Second, the pins of target nets are extracted. Third, the routing region of the nets are extracted to act as ground truth in the training procedure. In the inference flow, only the first two channels are presented.

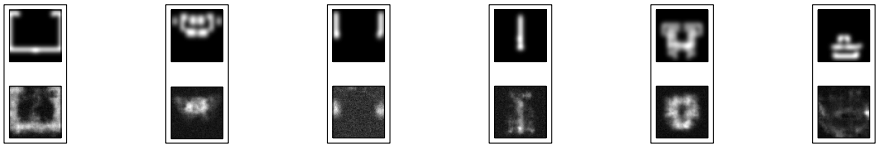
A training database is constructed by extracting the images from a set of manually routed layouts. To mitigate the shortage of training data, data augmentation is applied by flipping the images.

VAE-Based Routing Guidance Generation For each net type, a VAE model is trained to learn the routing strategy for the given net type. The proposed VAE-based method has two components: an encoder and a decoder. Encoder E_ϕ converts input data x into low-dimensional latent variable vector z, and decoder generates the routing guidance Y from z. VAE uses parametric distribution, usually Gaussian, to model X, Y and Z, i.e., $P(X|z,\theta) \sim \mathcal{N}(\mu(z),\sigma(z))$ and $z \sim \mathcal{N}(0,I)$, where θ denotes the trainable parameters. The training process maximizes the objective function

$$\log P(Y|z) - \mathcal{D}_{KL}[Q(z|X)||P(z)], \tag{26}$$

where $\log P(X|z)$ is a reconstruction log-likelihood of Y from X and $\mathcal{D}_{KL}$ is the Kullback-Leibler (KL) divergence measuring the dissimilarity between the learned distribution Q and training distribution P. The following reparameterization trick is often applied: first sample $\varepsilon \sim \mathcal{N}(0,I)$ and then compute $z = \mu(X) + \sigma^{1/2}(X) * \varepsilon$.

Semi-Supervised Training Algorithm In the GeniusRoute framework, the above VAE training scheme is extended to a semi-supervised approach to achieve better



(a) (b) (c) (d) (e) (f)

Fig. 22: Example of inferences of testing set. The upper row shows the ground truth, and the lower row shows the inference [86].

generalization amid limited training data. The proposed training algorithm takes the usage of unlabeled layout data and contains three stages.

In the first stage, the unlabeled training data used to train the network to reconstruct the layouts. The encoder in this stage can see more training data and learn a more generalized latent space mapping. The objective in this stage is to maximize the standard VAE objective function for reconstruction task:

$$\log P(X|z) - \mathcal{D}_{KL}[Q(z|X)||P(z)]. \tag{27}$$

In the second stage, labeled routing region data are used to train the network to learn the routing strategy. The encoder is fixed from the first stage and only the decoder weights are changed in this stage. The training loss at this stage minimizes the $\mathcal{L}^2$ norm of the distance between ground truth Y and inferred output $\hat{Y}$:

$$\|Y - \hat{Y}\|_2. \tag{28}$$

In the third stage, the GeniusRoute framework fine-tunes the entire model, including both the decoder and encoder. Since the network is already close to a nearly optimal point, the learning rate is set much lower than that in the first two stages. This stage maximizes the objective function shown in Equation (26).

Guided Analog Detailed Routing In the inference phase, GeniusRoute adopts the A^* search algorithm for detailed routing. It leverages the routing guidance generated by machine models to make routing decisions.

The routing guidance is considered as cost functions in A^* search, together with other common routing objective such as wire length and penalty of vias. The cost from the routing guidance is composed of two parts: the penalty of violating the guidance (violating cost), and the cost of routing in the region of other nets demand (competition cost). During the automatic routing process, the nets are encouraged to be routed over the ML-generated guidance.

Fig. 22 shows output examples of the model inference on testing sets. Fig. 22(a) and (c) are outputs of the clock model, Fig. 22(b) and (e) are from differential nets model, and Fig. 22(c) and (f) are from the power and ground model. The ML models not only learn well in manual placements, but are also capable of generating reasonable outputs for machine-generated placement for clock and differential nets.

Table 7 shows the comparison of simulated performance of a comparator. The GeniusRoute result has comparable performance to manual routing and outperforms the router in [72]. Compared to the same router without ML guidance, the proposed method achieve 67% reduction in input-referred offset, with comparable or better results in other metrics.

	Schematic	Manual	[72]	W/o guide	This work
Offset (μV)	/	480	1230	2530	830
Delay (ps)	102	170	180	164	163
Noise (μV_{rms})	439.8	406.6	437.7	439.7	420.7
Power (μW)	13.45	16.98	17.19	16.82	16.80

Table 7: Comparison of post-layout simulation results for a comparator [86].

In the experiments, the GeniusRoute is trained on a dataset consisting of comparators and amplifiers which are representative analog circuit architectures and share similar layout strategies. However, AMS circuits include many different circuit architectures. The layout strategy for those different circuit types can be significantly diverse. How to extending the ML-guided routing to other circuit types is an interesting open question for the future research.

4 Conclusion and Future Directions

In this chapter, we provide an overview of the current state of the art of applying machine learning to analog layout design automation, from subcircuit recognition and constraint generation, to placement and routing. Various machine learning techniques, including supervised, semi-supervised, graph neural network, and reinforcement learning, have been applied to different stages of analog layouts.

The semi- or fully-automated open-source analog layout generators ALIGN and MAGICAL have shown great initial success and provided solid foundations for future research and development. An example that incorporates ALIGN into a larger optimization flow is described in [44]: the netlist is sized using a neural network model and iteratively improved with fast layout generation in the inner loop of optimization. A transfer learning method for improving the accuracy of the neural network based on post-silicon measurements is also proposed. The approach is exercised on a VCO circuit, with silicon results. Another example that incorporates MAGICAL into a larger AMS system is described in [46], where an automated end-to-end successive approximation register (SAR) ADC compiler OpenSAR is built. It leverages automated placement and routing in MAGICAL to generate analog building blocks. Meanwhile, capacitor digital-to-analog converter (CDAC) arrays are designed using a template-based layout generator and digital blocks are done by commercial digital tools. They are then integrated by the hierarchical MAGICAL.

There are still many challenges and research opportunities in applying ML to analog layout.

Integrating with Layout Synthesis Flow The ML-based analog layout automation algorithms have demonstrated many successes in experiments. How to make the whole flow robust and consistent when incorporating ML models is pending a good answer. ML predictions are not always correct and sometimes stochastic. A mature flow might need to compensate the imperfectness of ML-based techniques and leverage their advantages.

In [48] and MAGICAL 1.0 [9], in-loop simulations are involved on building-block-level to guarantee the quality. However, as the simulation costs grow significantly for system-level designs, more scalable and sparser in-loop simulations, or better predictive models will be needed.

Analog layout dataset and data-efficient ML A key challenge for machine learning is always lack of quality training data. Many ML for analog layout techniques require manual labeled data to train the neural network model. However, manually labeling layout data is expensive. How to tackle the limited data is an ongoing open question. There are several possible answers. In [86], a semi-supervised training algorithm is presented to achieve higher data efficiency. More data-efficient ML algorithms might mitigate the issue. In [47], the MAGICAL framework is utilized to generate training data. Machine-generated or automatic labeled data might be a possible solution.

Understanding the schematics Many existing ML for analog layout techniques use GNN to model the circuit netlist. However, it is still an open question whether GNN is powerful enough to learn the graph topology [78]. Typical convolutional GNN focuses on learning the node-wise features. However, in circuit designs, the topology and interconnections are crucial to circuit functionality. There are several ongoing GNN research trends targeting learning the topology and structures, such as motif-based methods [56] and generative graph models [36].

References

1. ALIGN: Analog layout, intelligently generated from netlists (Software repository, accessed November 1, 2021). https://github.com/ALIGN-analoglayout/ALIGN-public
2. Afacan, E., Lourenço, N., Martins, R., Dündar, G.: Review: Machine learning techniques in analog/RF integrated circuit design, synthesis, layout, and test. Integration **77**, 113–130 (2021)
3. Ajayi, T., Cherivirala, Y.K., Kwon, K., Kamineni, S., Saligane, M., Fayazi, M., Gupta, S., Chen, C.H., Sylvester, D., Blaauw, D., Dreslinski, Jr., R., Calhoun, B., Wentzloff, D.D.: Fully autonomous mixed signal SoC design & layout generation platform. In: IEEE Hot Chips Symposium (2020)
4. Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., Wang, W.: SimGNN: A neural network approach to fast graph similarity computation. In: ACM International Conference on Web Search and Data Mining, pp. 384–392 (2019)
5. Balasa, F., Maruvada, S.C., Krishnamoorthy, K.: Efficient solution space exploration based on segment trees in analog placement with symmetry constraints. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2002)
6. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
7. Chakravarti, I.M., Roy, J., Laha, R.G.: Handbook of methods of applied statistics. Wiley New York (1967)
8. Chang, E., Han, J., Bae, W., Wang, Z., Narevsky, N., Nikolic, B., Alon, E.: BAG2: A process-portable framework for generator-based AMS circuit design. In: IEEE Custom Integrated Circuits Conference (CICC) (2018)

9. Chen, H., Liu, M., Tang, X., Zhu, K., Mukherjee, A., Sun, N., Pan, D.Z.: MAGICAL 1.0: An open-source fully-automated AMS layout synthesis framework verified with a 40-nm 1 GS/s $\Delta\Sigma$ ADC. In: IEEE Custom Integrated Circuits Conference (CICC) (2021)
10. Chen, H., Liu, M., Xu, B., Zhu, K., Tang, X., Li, S., Lin, Y., Sun, N., Pan, D.Z.: MAGICAL: An open-source fully automated analog IC layout system from netlist to GDSII. IEEE Design & Test (2020)
11. Chen, H., Zhu, K., Liu, M., Tang, X., Sun, N., Pan, D.Z.: Toward silicon-proven detailed routing for analog and mixed signal circuit. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2020)
12. Chen, H., Zhu, K., Liu, M., Tang, X., Sun, N., Pan, D.Z.: Universal symmetry constraint extraction for analog and mixed-signal circuits with graph neural networks. In: ACM/IEEE Design Automation Conference (DAC) (2021)
13. Cohn, J., Garrod, D.J., Rutenbar, R.A., Carley, L.R.: KOAN/ANAGRAM II: New tools for device-level analog placement and routing. IEEE Journal Solid-State Circuits **26**(3), 330–342 (1991)
14. Crossley, J., Puggelli, A., Le, H.P., Yang, B., Nancollas, R., Jung, K., Kong, L., Narevsky, N., Lu, Y., Sutardja, N., An, E.J., Sangiovanni-Vincentelli, A.L., Alon, E.: BAG: A designer-oriented integrated framework for the development of ams circuit generators. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 74–81 (2013)
15. De Ranter, C.R.C., Van der Plas, G., Steyaert, M.S.J., Gielen, G.G.E., Sansen, W.M.C.: CYCLONE: Automated design and layout of RF LC-oscillators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **21**(11), 1161–1170 (2002)
16. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Neural Information Processing Systems (NeurIPS), pp. 3844–3852 (2016)
17. Dey, R., Salem, F.M.: Gate-variants of gated recurrent unit (gru) neural networks. In: IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1597–1600 (2017)
18. Dhar, T., Kunal, K., Li, Y., Madhusudan, M., Poojary, J., Sharma, A.K., Xu, W., Burns, S.M., Harjani, R., Hu, J., Kirkpatrick, D.A., Mukherjee, P., Sapatnekar, S.S., Yaldiz, S.: ALIGN: A system for automating analog layout. IEEE Design & Test (2021)
19. Dhar, T., Poojary, J., Li, Y., Kunal, K., Madhusudan, M., Sharma, A.K., Manasi, S.D., Hu, J., Harjani, R., Sapatnekar, S.S.: Fast and efficient constraint evaluation of analog layout using machine learning models. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 158–163 (2021)
20. Dhillon, I., Guan, Y., Kulis, B.: Weighted graph cuts without eigenvectors: A multilevel approach. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) **29**(11), 1944–1957 (2007)
21. Eick, M., Strasser, M., Lu, K., Schlichtmann, U., Graeb, H.E.: Comprehensive generation of hierarchical placement rules for analog integrated circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **30**(2), 180–193 (2011)
22. Gao, X., Deng, C., Liu, M., Zhang, Z., Pan, D.Z., Lin, Y.: Layout symmetry annotation for analog circuits with graph neural networks. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC) (2021)
23. Gong, L., Cheng, Q.: Exploiting edge features for graph neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9211–9219 (2019)
24. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Neural Information Processing Systems (NeurIPS) (2014)
25. Graeb, H.E. (ed.): Analog Layout Synthesis: A Survey of Topological Approaches. Springer, New York, NY, USA (2010)
26. Guerra, D., Canelas, A., Póvoa, R., Horta, N., Lourenço, N., Martins, R.: Artificial neural networks as an alternative for automatic analog IC placement. In: IEEE International Conference on Synthesis, Modeling, Analysis and Simulation Methods, and Applications to Circuit Design (SMACD) (2019)

27. Gusmão, A., Passos, F., Póvoa, R., Horta, N., Lourenço, N., Martins, R.: Semi-supervised artificial neural networks towards analog ic placement recommender. In: IEEE International Symposium on Circuits and Systems (ISCAS) (2020)
28. Hamilton, W., Ying, Z., Leskovec, J.: Inductive Representation Learning on Large Graphs. In: Neural Information Processing Systems (NeurIPS), pp. 1024–1034 (2017)
29. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. In: IEEE Data Engineering Bulletin (2017)
30. Harjani, R., Rutenbar, R.A., Carley, L.R.: A prototype framework for knowledge-based analog circuit synthesis. In: ACM/IEEE Design Automation Conference (DAC), pp. 42–49 (1987)
31. Harjani, R., Rutenbar, R.A., Carley, L.R.: OASYS: A framework for analog circuit synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **8**(12), 1247–1266 (1989)
32. K.-L. J. Wong, C.-K. K. Yang: A serial-link transceiver with transition equalization. In: IEEE International Solid-State Circuits Conference (ISSCC), pp. 223–232 (2006)
33. Karmokar, N., Madhusudan, M., Sharma, A.K., Harjani, R., Lin, M.P.H., Sapatnekar, S.S.: Common-centroid analog circuit layout. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC) (2022)
34. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing (SISC) **20**(1), 359–392 (1998)
35. Kipf, T.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
36. Kipf, T.N., Welling, N.: Variational graph auto-encoders. arxiv:1611.07308 (2016)
37. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. Neural Information Processing Systems (NeurIPS) **25**(2), 1097–1105 (2012)
38. Kunal, K., Dhar, T., Madhusudan, M., Poojary, J., Sharma, A., Xu, W., Burns, S.M., Hu, J., Harjani, R., Sapatnekar, S.S.: GANA: Graph convolutional network based automated netlist annotation for analog circuits. IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE) (2020)
39. Kunal, K., Madhusudan, M., Sharma, A.K., Xu, W., Burns, S.M., Harjani, R., Hu, J., Kirkpatrick, D.A., Sapatnekar, S.S.: ALIGN: Open-source analog layout automation from the ground up. In: ACM/IEEE Design Automation Conference (DAC), pp. 77–80 (2019)
40. Kunal, K., Poojary, J., Dhar, T., Madhusudan, M., Harjani, R., Sapatnekar, S.S.: A general approach for identifying hierarchical symmetry constraints for analog circuit layout. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2020)
41. Li, Y., Lin, Y., Madhusudan, M., Sharma, A., Xu, W., Sapatnekar, S.S., Harjani, R., Hu, J.: A customized graph neural network model for guiding analog IC placement. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2020)
42. Li, Y., Lin, Y., Madhusudan, M., Sharma, A., Xu, W., Sapatnekar, S.S., Harjani, R., Hu, J.: Exploring a machine learning approach to performance driven analog IC placement. In: IEEE Annual Symposium on VLSI (ISVLSI), pp. 24–29 (2020)
43. Liou, G.H., Wang, S.H., Su, Y.Y., Lin, M.P.H.: Classifying analog and digital circuits with machine learning techniques toward mixed-signal design automation. In: IEEE International Conference on Synthesis, Modeling, Analysis and Simulation Methods, and Applications to Circuit Design (SMACD), vol. 15, pp. 173–176 (2018)
44. Liu, J., Su, S., Madhusudan, M., Hassanpourghadi, M., Saunders, S., Zhang, Q., Rasul, R., Li, Y., Hu, J., Sharma, A.K., Sapatnekar, S.S., Harjani, R., Levi, A., Gupta, S., Chen, M.S.W.: From specification to silicon: Towards analog/mixed-signal design automation using surrogate NN models with transfer learning. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2021)
45. Liu, M., Li, W., Zhu, K., Xu, B., Lin, Y., Shen, L., Tang, X., Sun, N., Pan, D.Z.: S^3DET: Detecting system symmetry constraints for analog circuits with graph similarity. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC) (2020)
46. Liu, M., Tang, X., Zhu, K., Chen, H., Sun, N., Pan, D.Z.: OpenSAR: An open source automated end-to-end SAR ADC compiler. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2021)

47. Liu, M., Zhu, K., Gu, J., Shen, L., Tang, X., Sun, N., Pan, D.Z.: Towards decrypting the art of analog layout: Placement quality prediction via transfer learning. In: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE) (2020)
48. Liu, M., Zhu, K., Tang, X., Xu, B., Shi, W., Sun, N., Pan, D.Z.: Closing the design loop: Bayesian optimization assisted hierarchical analog layout synthesis. In: ACM/IEEE Design Automation Conference (DAC) (2020)
49. Ma, Q., Xiao, L., Tam, Y.C., Young, E.F.Y.: Simultaneous handling of symmetry, common centroid, and general placement constraints. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **30**(1), 85–95 (2011)
50. Martins, R.M.F., Lourenço, N.C.C., Horta, N.C.G.: Generating Analog IC Layouts with LAYGEN II. Springer, New York, NY, USA (2010)
51. Massier, T., Graeb, H., Schlichtmann, U.: The sizing rules method for CMOS and bipolar analog integrated circuit synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **27**(12), 2209–2222 (2008)
52. McCulloh, I., Armstrong, H., Johnson, A.N.: Social network analysis with applications. Wiley & Sons, Inc (2013)
53. Meissner, M., Hedric, L.: FEATS: Framework for explorative analog topology synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **34**(2), 213–226 (2015)
54. Meng, Q., Harjani, R.: A 4GHz instantaneous bandwidth low squint phased array using subharmonic ILO based channelization. In: IEEE European Solid-State Circuits Conference (ESSCIRC), pp. 110–113 (2018)
55. Mirza, M., Osindero, S.: Conditional generative adversarial nets. arxiv:1411.1784 (2014)
56. Monti, F., Otness, K., Bronstein, M.M.: MotifNet: A motif-based graph convolutional network for directed graphs. In: IEEE Data Science Workshop (DSW), pp. 225–228 (2018)
57. Mustafa Ozdal, M., Wong, M.D.F.: A length-matching routing algorithm for high-performance printed circuit boards. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **25**(12), 2784–2794 (2006)
58. Ochotta, E., Rutenbar, R.A., Carley, L.R.: ASTRX/OBLX: Tools for rapid synthesis of high-performance analog circuits. In: ACM/IEEE Design Automation Conference (DAC), pp. 24–30 (1994)
59. Ohlrich, M., Ebeling, C., Ginting, E., Sather, L.: SubGemini: Identifying subcircuits using a fast subgraph algorithm. In: ACM/IEEE Design Automation Conference (DAC), pp. 31–37 (1993)
60. Ou, H., Chien, H.C., Chang, Y.: Nonuniform multilevel analog routing with matching constraints. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **33**(12), 1942–1954 (2014)
61. Ou, H.C., Chien, H.C.C., Chang, Y.W.: Simultaneous analog placement and routing with current flow and current density considerations. In: ACM/IEEE Design Automation Conference (DAC) (2013)
62. Ou, H.C., Tseng, K.H., Liu, J.Y., Wu, I.P., Chang, Y.W.: Layout-dependent effects-aware analytical analog placement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **35**(8), 1243–1254 (2016)
63. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab, Stanford, CA (1999)
64. Pan, P., Chen, H., Cheng, Y., Liu, J., Hu, W.: Configurable analog routing methodology via technology and design constraint unification. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 620–626 (2012)
65. Pang, Y., Balasa, F., Lampaert, K., Cheng, C.K.: Block placement with symmetry constraints based on the o-tree non-slicing representation. In: ACM/IEEE Design Automation Conference (DAC) (2000)
66. Razavi, B.: Design of Analog CMOS Integrated Circuits, 1st edn. McGraw-Hill, Inc., New York, NY, USA (2001)
67. Sapatnekar, S.S.: Timing. Springer, Boston, MA, USA (2004)

68. Vapnik, V.: Statistical learning theory. Wiley, New York, NY, USA (1998)
69. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph Attention Networks. arXiv:1710.10903 (2017)
70. Wu, C.Y., Graeb, H., Hu, J.: A pre-search assisted ILP approach to analog integrated circuit routing. In: IEEE International Conference on Computer Design (ICCD), pp. 244–250 (2015)
71. Wu, P.H., Lin, M.P.H., Chen, T.C., Yeh, C.F., Li, X., Ho, T.Y.: A novel analog physical synthesis methodology integrating existent design expertise. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) **34**(2), 199–212 (2015)
72. Xiao, L., Young, E.F.Y., He, X., Pun, K.P.: Practical placement and routing techniques for analog circuit designs. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 675–679 (2010)
73. Xu, B., Basaran, B., Su, M., Pan, D.Z.: Analog placement constraint extraction and exploration with the application to layout retargeting. In: ACM International Symposium on Physical Design (ISPD) (2018)
74. Xu, B., Li, S., Pui, C.W., Liu, D., Shen, L., Lin, Y., Sun, N., Pan, D.Z.: Device layer-aware analytical placement for analog circuits. In: ACM International Symposium on Physical Design (ISPD) (2019)
75. Xu, B., Li, S., Xu, X., Sun, N., Pan, D.Z.: Hierarchical and analytical placement techniques for high-performance analog circuits. In: ACM International Symposium on Physical Design (ISPD) (2017)
76. Xu, B., Lin, Y., Tang, X., Li, S., Shen, L., Sun, N., Pan, D.Z.: WellGAN: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout. In: ACM/IEEE Design Automation Conference (DAC) (2019)
77. Xu, B., Zhu, K., Liu, M., Lin, Y., Li, S., Tang, X., Sun, N., Pan, D.Z.: MAGICAL: Toward fully automated analog IC layout leveraging human and machine intelligence. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2019)
78. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (ICLR) (2019)
79. Yan, T., Wong, M.D.F.: BSG-Route: A length-matching router for general topology. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 499–505 (2008)
80. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: ACM International Conference on Knowledge Discovery and Data Mining (KDD), pp. 974–983 (2018)
81. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Neural Information Processing Systems (NeurIPS), pp. 4800–4810 (2018)
82. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: On approximating graph edit distance. VLDB Endowment **2**(1), 25–36 (2009)
83. Zhu, K., Chen, H., Liu, M., Pan, D.Z.: Automating analog constraint extraction: from heuristics to learning. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC) (2022)
84. Zhu, K., Chen, H., Liu, M., Tang, X., Shi, W., Sun, N., Pan, D.Z.: Generative-adversarial-network-guided well-aware placement for analog circuits. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC) (2022)
85. Zhu, K., Chen, H., Liu, M., Tang, X., Sun, N., Pan, D.Z.: Effective analog/mixed-signal circuit placement considering system signal flow. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2020)
86. Zhu, K., Liu, M., Lin, Y., Xu, B., Li, S., Tang, X., Sun, N., Pan, D.Z.: GeniusRoute: A new analog routing paradigm using generative neural network guidance. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2019)