Universal Symmetry Constraint Extraction for Analog and Mixed-Signal Circuits with Graph Neural Networks

Hao Chen, Keren Zhu, Mingjie Liu, Xiyuan Tang, Nan Sun, and David Z. Pan
ECE Department, The University of Texas at Austin, Austin, TX, USA
{haoc, keren.zhu, jay_liu, xitang}@utexas.edu, nansun@mail.utexas.edu, dpan@ece.utexas.edu

***Abstract*—Recent research trends in analog layout synthesis aim for a fully automated netlist-to-GDSII design flow with minimum human efforts. Due to the sensitiveness of analog circuit layouts, symmetry matching between critical building blocks and devices can significantly impact the overall circuit performance. Therefore, providing accurate symmetry constraints for automated layout synthesis tools is crucial to achieving high-quality layouts. This paper presents a novel graph-learning-based framework leveraging unsupervised learning to recognize circuit matching structures by making the most of numerous unlabeled circuits. The proposed framework supports both system-level and device-level symmetry constraints extraction for various large-scale analog/mixed-signal systems. Experimental results show that our framework outperforms state-of-the-art symmetry constraint detection algorithms with remarkable accuracy and runtime improvement.**

I. Introduction

The performance of modern analog/mixed-signal (AMS) designs is susceptible to parasitics, process variations, and layout-dependent effects due to the sensitiveness and complexity of AMS circuit layouts [1]. To guarantee the performance specification and circuit robustness, various geometrical matching constraints (e.g., symmetry, regularity, common-centroid) need to be carefully considered during the layout process [2].

Typically, these constraints are annotated by layout design experts with profound domain knowledge [3]. However, real-world AMS circuits have a wide range of specific circuit classes, device types, and even dozens of different topologies for a single functionality, imposing more challenges and thus making manual constraint-annotating a tedious and error-prone task. Thus, automatic constraint extraction is a desirable auxiliary to alleviate heavy human efforts and further improve the layout quality [4].

Recent research aims to achieve fully automated netlist-to-GDSII analog layout design flow with minimum human efforts. [5], [6] adopt optimization-based methodologies that utilize analog place-and-route (P&R) algorithms to realize optimized layout and minimize human efforts [7]–[10]. With distinctly specified constraints, these algorithms tend to generate diverse solutions that result in varying post-layout performance. Therefore, automatically generating precise and accurate symmetry constraints has become an essential procedure in the design flow. Figure 1 shows an example of a 2^{nd}-order continuous-time $\Delta\Sigma$ modulator (CTDSM). In Figure 1(a), all the devices and building blocks with symmetry matching requirements are specified with perfect constraints. By removing a symmetry constraint for a matched resistor pair, we can observe a dramatic change in the final layout that lead to performance degradation in signal-to-noise and distortion ratio (SNDR) of 3.1 dB and spurious-free dynamic range (SFDR) of 3.8 dB, as shown in Figure 1(b).

Recent advances in graph neural networks (GNNs) have demonstrated superior efficacy in learning graph structures and mining graph information [11], [12]. Applications of GNNs in the electronic design automation (EDA) domain have also been shown [13]–[15]. By formulating circuit netlists as graphs, we can rely on GNN models to extract symmetry constraints regarding device parameters and matching structures.

In this paper, we propose a novel unsupervised inductive graph-learning-based methodology for universal AMS symmetry constraint extraction. Leveraging the unsupervised learning technique with GNNs, our framework learns a strategy to extract the latent information of matching circuit structures, thus applicable to general AMS designs. The main contribution of this work is summarized as follows.

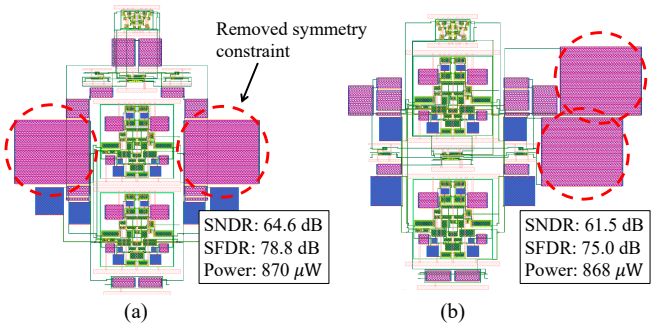

Fig. 1: Automated P&R layouts of a 2^{nd}-order CTDSM. (a) Layout with symmetry constraints specified accurately. (b) Layout with a symmetry constraint removed.

- A GNN-based framework is presented to identify universal symmetry constraints (i.e., system-level and device-level constraints) for AMS designs by extracting the information of matching structures. The source code is released on Github[†].
- The framework is *generalizable to every design* by learning feature representations over unsupervised loss function.
- A heterogeneous multigraph representation to model both active and passive elements for general AMS circuits is proposed.
- A circuit feature embedding algorithm is developed to support system-level symmetry constraint generation.
- Experimental results on real-world taped-out designs show that our framework outperforms previous work with remarkable quality improvement and significant runtime speedup.

The remainder of this paper is organized as follows. Section II provides a brief overview of related work. Section III gives the preliminaries and formulates the symmetry constraint extraction problem. Section IV details the proposed unsupervised inductive graph-learning framework. Section V presents the experimental results, and Section VI concludes the paper.

II. Related Work

Among prior art on automatic AMS symmetry constraint extraction, [16]–[18] conduct sensitivity analysis to identify symmetry and matching pairs. The simulation-based methodologies are generalizable to various circuit structures and performance metrics in the analog domain. Nevertheless, these approaches suffer from expensive circuit simulation, thus impractical in large systems such as analog-to-digital converter (ADC).

As circuits can be modeled as hypergraphs naturally, graph matching based techniques have also been investigated. The work [19] presents a pattern matching algorithm on structural signal flow graphs to generate hierarchical matching constraint groups. The work [6] performs signal flow analysis on circuit graphs in addition to pattern matching. Though the graph matching algorithms show decent results on simpler designs

978-1-6654-3274-0/21/$31.00 ©2021 IEEE

[†]https://github.com/baloneymath/AncstrGNN

TABLE I: Comparisons of symmetry constraint extraction methods among recent work and the proposed framework.

	ICCAD'19 [6]	ASP-DAC'20 [20]	ICCAD'20 [21]	This work
Circuit representation	Graph	Graph	Bipartite graph	Heterogeneous multigraph
Training method	N/A	N/A	Supervised	Unsupervised
Circuit embedding	N/A	Heuristic	GNN	GNN
Sizing consideration	N/A	N/A	Devices	Devices + Circuits
Device-level matching	Heuristic patterns	N/A	Heuristic patterns	Cosine similarity
System-level matching	N/A	K-S test	GED	Cosine similarity
Unified for device- and system-level	N/A	N/A	No	Yes

and are computationally efficient, they fail to recognize symmetries for complicated circuit topologies with hierarchical structures. In [20], a spectral method computing graph similarity is proposed to detect system-level symmetry constraints. Still, the scalability is restricted by heavy-loaded statistical computation. In [21], a GNN-assisted hierarchical symmetry constraint annotation framework is shown. Symmetry constraints are determined by estimating graph edit distance (GED) between matching pairs using a GNN. However, the supervised learning method requires a large amount of labeled data to train an accurate prediction model and limits its applicability to specific circuit types. Great efforts are required to extend the trained model for unseen and more complex circuit structures.

Graph representations are used for circuit prototyping in [6], [19]–[21]. The work [19] formulates circuit netlists with directed graphs to describe detailed pin-to-pin connections and signal flows. However, the constructed graph consists of a vast vertex set, thus unsuitable for real-world complex AMS systems. The work [6], [20] adopts a simplified graph representation with pins modeled as vertices. Nevertheless, the vertex set is still large and might have information loss due to graph isomorphism. In [21], a bipartite graph representation is applied with devices and nets being disjoint vertex sets. Though it has decreased vertex number and can capture connection types by setting labels on edges, the complexity of exploring neighboring structures is significantly higher.

Capitalizing on the sizing information is crucial for symmetry constraint extraction. Though previous approaches consider sizing for device-level symmetry constraints, they cannot handle the sizing of subcircuits, which is essential for system-level symmetry constraint generation. Figure 2 shows an example with three system-level symmetry constraints. Without sizing consideration, an algorithm tends to cause false alarms by annotating all the inverters as a symmetry group since they have identical topologies. Our framework captures both the sizing of devices and subcircuits to improve the solution quality.

Table I summarizes the aforementioned symmetry constraint extraction approaches and compares them with our proposed unsupervised inductive learning framework. Compared with previous approaches, the proposed heterogeneous multigraph circuit representation enjoys a reduced vertex set for better efficiency, without missing any detailed connecting information. The proposed unsupervised GNN-learning-based symmetry constraint extraction method shows excellent solution quality and great generalizability. Besides, our framework does not require manual labels for training, thus extensible to new circuits with minimum efforts.

III. Preliminaries

In this section, we introduce the definitions of system symmetry constraint and device symmetry constraint. Then, we formulate the AMS symmetry constraint extraction problem.

A. Symmetry Constraints for AMS Circuits

To optimize performance for AMS circuit layouts, symmetry constraints are specified for pairs of matched modules to enforce matching. A

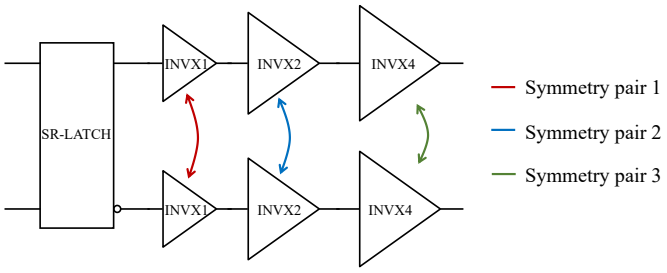

Fig. 2: Part of a clock circuit in a SAR ADC with symmetry constraints considering sizing.

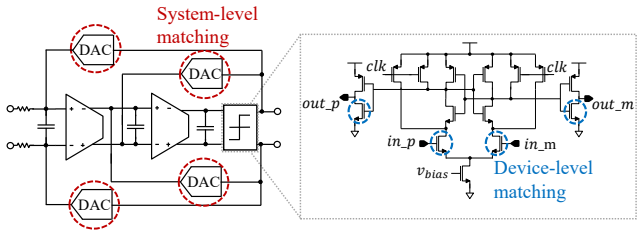

Fig. 3: (a) System-level symmetry constraints. (b) Device-level symmetry constraints.

symmetry constraint is a three-tuple $s = (T_c, t_i, t_j)$ that specifies the matching requirement between a pair of matched modules (t_i, t_j) under the circuit hierarchy T_c, where $t_i, t_j \in T_c$. The matched pair (t_i, t_j) should be placed and routed symmetrically regarding some joint axes. For a symmetry constraint s, if exists any other subcircuit under T_c and that t_i, t_j are building blocks (e.g., operational transconductance amplifier (OTA), comparator) or passive devices (e.g., capacitors, resistors), then s is a *system-level* symmetry constraint. Otherwise, s is defined as a *device-level* symmetry constraint. Any module pair (t_i, t_j) with t_i and t_j under different circuit hierarchies or having nonidentical types is considered *invalid*.

Figure 3(a) shows an example of system-level symmetry constraints of two digital-to-analog converter (DAC) pairs on a 3$^{\text{rd}}$-order CTDSM. Figure 3(b) annotates some device-level symmetry constraints of a comparator in the CTDSM.

B. Problem Formulation

The AMS symmetry constraint extraction problem is formulated as follows.

Problem 1 (AMS Symmetry Constraint Extraction): Given a circuit netlist N with hierarchy tree structure $T = \{t_i | 1 \leq i \leq |T|\}$, where t_i is a primitive element if t_i is a leaf; otherwise, t_i is a building block, generate a set of symmetry constraints such that every matched pair (t_i, t_j) in the corresponding circuit hierarchy $T_c \subseteq T$ is specified with a valid symmetry constraint.

IV. Algorithms

The overall flow of our GNN-based symmetry constraint extraction framework is shown in Figure 4. In the flow, the input circuit netlist is first transformed into a proposed heterogeneous multigraph. Each vertex in the constructed multigraph is then initialized with a feature vector considering device types and parameters. After the pre-processing stage, the core of our framework consists of three main phases: 1) unsupervised inductive GNN learning, which samples and aggregates the neighboring features of each vertex to extract accurate structural information, 2) circuit feature embedding, which determines the feature of each subcircuit by detecting representative sub-structures within the circuit graph, and 3) classification, which calculates the cosine similarity between valid symmetry pair candidates to generate final symmetry constraints.

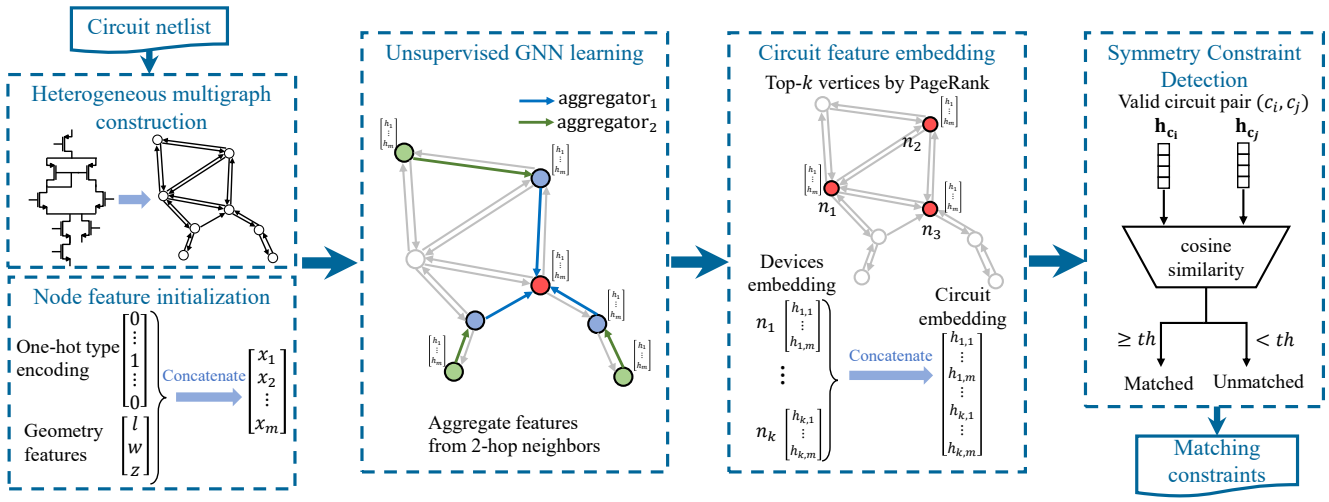

Fig. 4: Computation flow of the proposed GNN-based symmetry constraint extraction framework.

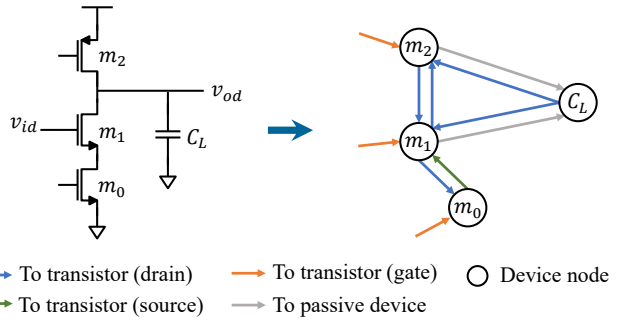

Fig. 5: Example of the proposed heterogeneous multigraph representation for an analog circuit.

A. Heterogeneous Mulitgraph Construction

Circuit schematic can be formulated into graphs naturally. To construct a circuit graph that models the nets connections with more details, we propose a heterogeneous directed multigraph (i.e., graph with parallel edges) representation for general AMS designs. With an input netlist, we transform the circuit into a heterogeneous multigraph $G = (V, E)$, where V is a set of vertices and E is a set of directed edges. The set V consists of primitive devices in the circuit, and each vertex $v \in V$ is marked with its corresponding device type. A directed edge $e = (u, v, \tau_v) \in E$ signifies the interconnection from vertex u to v with edge type τ_v, where $\tau_v \in P$ denotes the type of the port of v connected by e, and $P = \{p_{gate}, p_{drain}, p_{source}, p_{passive}\}$ is a set of port types. Note that parallel edges are permitted since G is a multigraph.

Example 1: Figure 5 illustrates a vivid example of the proposed heterogeneous multigraph representation. As shown in the figure, the four devices m_0, m_1, m_2, and C_L are mapped to four vertices, respectively. Since the drain of m_1 is connected to the drain of m_2, an edge $e_1 = (m_1, m_2, p_{drain})$ is added to the graph. Similarly, another edge $e_2 = (m_1, C_L, p_{passive})$ is created.

Algorithm 1 sketches the detailed graph construction procedure. We first initialize a multigraph $G = (V, E)$. In ines 2-4, we traverse the input netlist hierarchy tree T and add the leaf nodes (devices) to V. After building the vertex set V, we iterate through the circuit nets and adopt a clique-based edges construction, as shown in Lines 5-13 in Algorithm 1. Note that we avoid constructing self-loops in G. After enumerating all the nets, the edge set E is constructed and we return the multigraph G for circuit representation.

Algorithm 1 *ConstructHeterogeneousGraph*(N)

Input: A circuit netlist N with hierarchy tree T.
Output: The heterogeneous multigraph representation G.

```
1:  Initialize a heterogeneous multigraph G = (V, E).
2:  for each tree node t_i ∈ T do
3:      if t_i is a leaf node then                ▷ t_i is a primitive element
4:          V := V ∪ {t_i};
5:  for each net n_i ∈ N do
6:      for each device port p_i ∈ n_i do
7:          for each device port p_j ∈ n_i and p_i ≠ p_j do
8:              u := CorrespondingVertex(p_i);                ▷ u ∈ V
9:              v := CorrespondingVertex(p_j);                ▷ v ∈ V
10:             if u ≠ v then                  ▷ avoid self loops in E
11:                 E := E ∪ {(u, v, τ_v), (v, u, τ_u)};
    return G;
```

TABLE II: Initial features of a vertex v in the heterogeneous multigraph G representing the input circuit.

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.

B. Node Feature Initialization

After the heterogeneous multigraph construction, we determine an initial feature vector for each vertex. Table II summarizes the features and their dimension. The features consist of the device type information of a vertex and its shape parameters. The type of a device (e.g., nch_lvt, pch_lvt, cfmom) is converted to a 15-dimensional one-hot vector. As modern analog devices are much more sophisticated, there could be tens of design parameters to precisely describe a device's shape. However, using all the parameters will restrict the learning model's ability to identify matching of nonidentical circuit structures and cause extra efforts extending the model to new circuits. To extract the shape information of a device without jeopardizing the generalizability of our learning model, we build a vector comprising length, width, and number of metal layers to approximate the exact shape details. These two vectors are then concatenated to form the feature vector of a vertex for GNN learning.

C. Unsupervised Inductive GNN Learning

To make the GNN model generalizable, we leverage unsupervised learning to train a universal inductive strategy which can be applied on

unseen circuits.

With the constructed heterogeneous multigraph $G = (V, E)$ of a circuit, we train our GNN model to sample and aggregate neighboring features for each vertex $v \in V$. By iteratively aggregating the features of neighbor vertices, the GNN model will recognize the localized interconnection and the peripheral structures of each vertex v. Inspired by the core idea of [22], we set the feature aggregating function to aggregate the features of K-hop neighbors of a vertex as follows to comprehend the connections with different edge types.

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

where $h_v^{(k)}$ symbolizes the feature vector of vertex v at the k^{th} layer of the GNNs, $GRU(\cdot,\cdot)$ is the computation function of a gated recurrent unit, $\mathcal{N}_{in}(v)$ signifies the in-neighbors of v, and $W_{e_{uv}}$ is the linear transformation matrix corresponding to the edge e_{uv}. To be specific, there is a matrix set $\mathcal{W}$ with $|\mathcal{W}| = 4$ since we define four edge types in Section IV-A. Note that the total number of layers K in the learning model is corresponding to the features aggregation of K-hop neighbors. In our implementation, we set $K = 2$, and the output dimension D of each neural network is set to 18.

To perform unsupervised learning on the GNN model, we define a loss function $\mathcal{L} : \mathbb{R}^D \to \mathbb{R}$, where D is the dimension of the vertex feature vector. The function $\mathcal{L}$ takes the final feature representation $z_v = h_v^{(K)}$ of a vertex v as input and compute the cross-entropy loss as follows.

$$\begin{aligned}\mathcal{L}(z_v) = &- \sum_{u \in \mathcal{N}_{in}(v)} \log(\sigma(z_u^\intercal z_v)) \\ &- \sum_{i=1}^{B} \mathbb{E}_{\tilde{u} \sim \mathit{Neg}(v)} \log(1 - \sigma(z_{\tilde{u}}^\intercal z_v)),\end{aligned} \tag{2}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic sigmoid function, $\mathit{Neg}(v)$ denotes the negative sampling distribution with respect to v, and B is the total number of negative samples. Practically, $\mathcal{N}_{in}(v)$ selects the 1-hop in-neighbors of v, and B is set to 5. By minimizing the overall loss $\mathcal{L}_{tot} = \sum_{v \in V} \mathcal{L}(z_v)$, the GNN model learns the strategy to improve feature similarity between each vertex v and its 1-hop neighbors, while enlarging the discrepancy between v and the negative samples $\mathit{Neg}(v)$.

Intuitively, as the training procedure aggregates neighboring features and optimizes the loss $\mathcal{L}_{tot}$, the final feature representation of each vertex v contains the information of the localized circuit structure centered at v, including the device types, geometric shapes, and interconnections. Therefore, those devices with symmetry matching requirements tend to possess alike final feature representations since they would have similar neighboring structures.

D. Circuit Feature Embedding

With the trained feature vectors of the vertices, we obtain the device-level information. To detect the symmetry matching between building blocks, we perform circuit feature embedding to determine the feature representation for each subcircuit.

For each subcircuit t, we analyze its multigraph representation G_t to produce its feature embedding using the vertex feature vectors trained in the previous stage. Since some nonidentical subcircuits still require symmetry matching (e.g., capacitor/resistor arrays with different interconnections), we characterize the subcircuit by selecting some of the most representative vertices in the multigraph, instead of using the entire multigraph. To select the top-M representative vertices within G_t, we introduce the PageRank algorithm [23] to find the targets. In our implementation, M is set to 10. If the devices in the subcircuit t is less than M, we set $M = |V_t|$.

Algorithm 2 provides the circuit feature embedding procedure. Firstly, we construct a simplified directed graph $G'_t = (V_t, E'_t)$ with an updated edge set E'_t (Lines 1-4). The edges in E'_t are directed edges without types, and no parallel edge is allowed in G'_t (i.e., at most two edges can exist between every two vertices). With the simplified directed graph G'_t, we calculate the PageRank value of the vertices in V_t and sort V_t in the descending order with repsect to the calculated values (Lines 5-6). The PageRank computation for each vertex is defined as follows.

$$PR(v) = \frac{1-\gamma}{|V_t|} + \gamma \cdot \sum_{u \in \mathcal{N}_{in}(v)} \frac{PR(u)}{|\mathcal{N}_{out}(v)|}, \tag{3}$$

where γ is the damping factor, $\mathcal{N}_{in}(v)$ denotes the in-neighbors of v, and $\mathcal{N}_{out}(v)$ indicates the out-neighbors of v. Then, we select the top-M vertices with the highest PageRank values and set the concatenation of their feature vectors as the subcircuit feature embedding, as sketched in Lines 7-10.

Algorithm 2 *EmbedCircuitFeature*(t, G_t, Z)

Input: A subcircuit t in the circuit hierarchy tree T (i.e., t is not a leaf node), its heterogeneous multigraph representation $G_t = (V_t, E_t)$, and the trained vertex feature vectors Z.
Output: The subcircuit feature embedding z_t.
1: Initialize a directed graph $G'_t = (V_t, E'_t)$, $E'_t = \emptyset$.
2: **for each** $e = (u, v, \tau_v) \in E_t$ **do**
3: **if** $(u, v) \notin E'_t$ **then** ▷ avoid parallel edges
4: $E'_t := E'_t \cup \{(u, v)\}$;
5: Compute $PR(v_i)$ for all $v_i \in V$;
6: Sort V_t in the descending order of $PR(v_i)$ for all $v_i \in V_t$;
7: $z_t := \mathbf{0}$;
8: **for** $i = 1$ to M **do** ▷ pick the top-M PageRank vertices
9: $z_{v_i} := \mathit{FeatureVector}(v_i) \in Z$;
10: $z_t := \mathit{Concatenate}(z_t, z_{v_i})$;
return z_t;

Algorithm 3 *DetectSymmetryConstraints*(N)

Input: A circuit netlist N with hierarchy tree T.
Output: A set of valid symmetry constraints S.
1: $S = \emptyset$; $\lambda_{th} := \mathit{SimilarityThreshold}(N)$;
2: **for each** circuit hierarchy $T_c \in T$ **do**
3: **for each** valid pair $(t_i, t_j) \in T_c$ **do**
4: $z_{t_i} := \mathit{FeatureVector}(t_i)$;
5: $z_{t_j} := \mathit{FeatureVector}(t_j)$;
6: $\lambda_{sim} := \mathit{CosineSimilarity}(z_{t_i}, z_{t_j})$;
7: **if** $\lambda_{sim} > \lambda_{th}$ **then**
8: $S := S \cup \{(T_c, t_i, t_j)\}$;
return S;

E. Symmetry Constraint Detection

In this step, we determine the final symmetry constraints for the matched devices and subcircuits. Algorithm 3 shows the detection procedure. For each circuit hierarchy $T_c \subseteq T$, we compare the feature vectors' similarity of every candidate pair $(t_i, t_j) \in T_c$. A candidate pair is a valid module pair as defined in Section III-A. Before calculating the similarity, we set a similarity threshold λ_{th}. Since larger subcircuits are more probable to have nonidentical matching structures, for system-level constraints, we define λ_{th} according to the maximum subcircuit size $|\widehat{N}_{sub}|$ in the circuit N as follows.

$$\lambda_{th} = \min(0.999, \alpha + \frac{\beta}{1 + |\widehat{N}_{sub}|}), \tag{4}$$

where α and β are constants. In our implementation, we set $\alpha = \beta = 0.95$. For device-level constraint detection, we set $\lambda_{th} = 0.99$. A candidate pair will be annotated as a valid constraint if the cosine similarity λ_{sim} of

their features z_{t_i} and z_{t_j} is greater than λ_{th}, where λ_{sim} is calculated as follows.

$$\lambda_{sim} = \frac{z_{t_i} \cdot z_{t_j}}{\|z_{t_i}\| \|z_{t_j}\|}. \tag{5}$$

If $\lambda_{sim} > \lambda_{th}$, we construct a new symmetry constraint $s = (T_c, t_i, t_j)$. After enumerating all the valid pairs, we obtain the final symmetry constraint set (Lines 2-8).

V. Experimental Results

The proposed framework for AMS symmetry constraint extraction is implemented in Python with the PyTorch library. All experiments are conducted on a Linux workstation with an Intel i9 3.3GHz CPU with 128GB memory, and an Nvidia Titan Xp GPU. To demonstrate the effectiveness and scalability of our proposed framework, we conduct experiments on a wide range of AMS circuit classes. We obtain five large-scale ADC architectures from experienced analog circuit designers. Table III lists the statistics of the ADCs. We also collect 15 block-level circuits from the open-source datasets [5], [6], including variants of operational transconductance amplifiers (OTAs), comparators (COMPs), DACs, etc. Table IV summarizes the statistics of the block-level designs. These circuits form the training dataset for unsupervised GNN learning.

To evaluate the solution quality, we adopt comprehensive measurements covering true positive rate (TPR), false positive rate (FPR), positive predictive value (PPV), accuracy (ACC), and F_1-score over the valid pairs. The formulas of these metrics are shown as follows.

$$\begin{aligned} &\text{TPR} = \frac{\text{TP}}{\text{TP+FN}}, \ \text{FPR} = \frac{\text{FP}}{\text{FP+TN}}, \ \text{PPV} = \frac{\text{TP}}{\text{TP+FP}}, \\ &\text{ACC} = \frac{\text{TP+TN}}{\text{TP+FP+TN+FN}}, \ F_1\text{-score} = \frac{2\text{TP}}{2\text{TP+FP+FN}}, \end{aligned} \tag{6}$$

where TP, FP, TN, and FN abbreviate the amount of the true positives, false positives, true negatives, and false negatives of the symmetry constraints generated by our framework compared with the ground truth (i.e., the constraints given by circuit designers), respectively.

TABLE III: Statistics of the five ADC benchmarks.

Benchmark	Architechture	#Devices	#Nets	#Valid Pairs
ADC1	2^{nd}-order CT $\Delta\Sigma$	285	122	148
ADC2	3^{rd}-order CT $\Delta\Sigma$	345	162	104
ADC3	3^{rd}-order CT $\Delta\Sigma$	347	163	82
ADC4	SAR	731	372	776
ADC5	Hybrid CT $\Delta\Sigma$ SAR	1233	586	1177

TABLE IV: Statistics of the block-level circuit benchmarks.

Benchmark	#Circuits	#Devices	#Nets	#Valid Pairs
OTA	6	133	109	770
COMP	6	145	109	1060
DAC	2	22	30	43
LATCH	1	24	14	132
Total	15	324	262	2005

A. System-level Constraint Extraction

We compare the system-level constraint extraction results with the state-of-the-art framework [20] that generates system-level symmetry constraints based on graph similarity. The executable of [20] is obtained from its authors and executed on our machine.

Table V shows the comparison between the system-level symmetry constraint extraction result. Comparing the solutions, our framework consistently outperforms [20] in all the five ADC designs. On average, our framework achieves a far superior F_1-score improved by 15.8% and 4.6%, 20.6%, and 6.2% higher TPR, PPV, ACC, respectively. Obtaining a higher F_1-score indicates that the symmetry constraints generated by our framework are precise, and more truly matched building blocks are covered. Besides, our framework achieves an extremely low FPR of 0.7% with a 4.1% reduction compared with [20], implying that the proposed algorithm rarely produces unnecessary constraints. Furthermore, one can notice that the proposed graph-learning-based framework achieves significant runtime improvement with about 218× reduction on average. In large cases such as ADC4 and ADC5, our framework can further achieve up to 483× and 350× speedup, respectively.

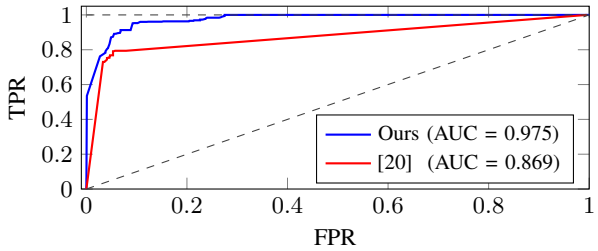

Fig. 6: ROC curves of [20] and our framework on the merged dataset with five ADCs.

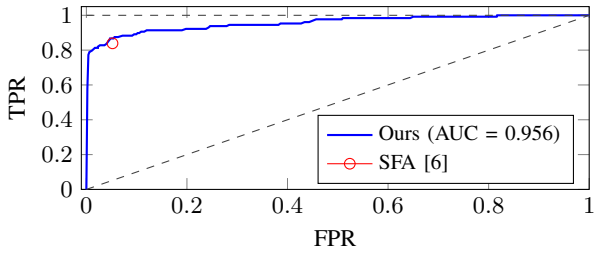

Fig. 7: ROC curve of our results on the merged dataset with the block-level circuits, and the point of [6] in the ROC space.

To cover a more thorough analysis of the two approaches, we combine the five ADCs into a merged dataset and plot the receiver operating characteristic (ROC) curves, as shown in Figure 6. One can observe that our framework obtains a much larger area under the curve (AUC). In fact, the ROC curve of [20] is fully enclosed by ours.

B. Device-level Constraint Extraction

We further investigate the device-level constraint extraction results. Table VI details the comparisons of evaluation metrics between the signal flow analysis (SFA) method in [6] and our framework. Our results show that, on average, we achieve 4.5% reduction in FPR, 19.7%, 3.9%, 9.8% improvement in PPV, ACC, and F_1-score, respectively. Observe that the SFA algorithm obtains higher TPR than ours, the main reason is because it tends to mark more unnecessary constraints and produce more false alarms. However, a lower FPR is desired in practice since specifying incorrect constraints can conflict with proper constraints. In that case, the analog P&R engines might fail to get a feasible solution. Also, our framework outperforms [6] significantly in F_1-score, which is a more generalized metric that accounts for both TPR and PPV. A higher F_1-score implies better model quality. Note that the 15 block-level circuits are all relatively small, and thus the runtime of our approach is dominated by loading the GNN model. When combining all the circuits to a merged dataset, our framework still completes the constraint extraction for all the circuits within 3 seconds. Furthermore, the scalability of our framework has already been verified by generating constraints for large-scale designs in Table III.

Figure 7 plots the ROC curve of our framework on the merged dataset consisting of the 15 block-level circuits. Note that [6] can only produce a single point in the ROC space since it is a heuristic algorithm instead of a probability-based method. As shown in Figure 7, the ROC curve of our framework results in a high AUC of 0.956, and encloses the point produced by the SFA algorithm.

TABLE V: Comparison of TPR, FPR, ACC, F_1-score, and runtime(s) for system-level symmetry constraint extraction.

Benchmark			S^3DET [20]						This work					
Design	#Devices	#Nets	TPR	FPR	PPV	ACC	F_1-score	Runtime	TPR	FPR	PPV	ACC	F_1-score	Runtime†
ADC1	285	122	1.000	0.036	0.667	0.966	0.800	36.70	1.000	0.000	1.000	1.000	1.000	2.71
ADC2	345	162	1.000	0.044	0.765	0.962	0.867	30.98	1.000	0.000	1.000	1.000	1.000	2.45
ADC3	347	163	1.000	0.125	0.526	0.890	0.690	49.58	1.000	0.014	0.909	0.988	0.952	2.74
ADC4	731	372	0.619	0.000	1.000	0.812	0.765	1717.81	0.880	0.005	0.994	0.938	0.934	3.55
ADC5	1233	586	0.864	0.036	0.836	0.946	0.850	1795.52	0.835	0.015	0.920	0.958	0.875	5.14
Average	-	-	0.897	0.048	0.759	0.915	0.794	726.12	**0.943**	**0.007**	**0.965**	**0.977**	**0.952**	**3.32**

† Runtime with GNN model training time excluded.

TABLE VI: Comparison of TPR, FPR, ACC, F_1-score, and runtime(s) for device-level symmetry constraint extraction.

Benchmark			SFA [6]						This work					
Design	#Devices	#Nets	TPR	FPR	PPV	ACC	F_1-score	Runtime	TPR	FPR	PPV	ACC	F_1-score	Runtime†
OTA1	12	14	0.667	0.000	1.000	0.941	0.800	<0.1	0.333	0.000	1.000	0.882	0.500	2.17
OTA2	20	20	0.875	0.171	0.333	0.833	0.483	<0.1	0.625	0.049	0.556	0.922	0.588	2.17
OTA3	12	12	0.667	0.083	0.667	0.867	0.667	<0.1	0.333	0.000	1.000	0.867	0.500	2.17
OTA4	36	36	0.667	0.131	0.170	0.861	0.271	<0.1	0.667	0.007	0.800	0.981	0.727	2.18
OTA5	38	18	0.833	0.004	0.909	0.989	0.870	<0.1	0.667	0.011	0.727	0.975	0.696	2.18
OTA6	15	9	0.571	0.000	1.000	0.870	0.727	<0.1	1.000	0.000	1.000	1.000	1.000	2.11
COMP1	47	31	1.000	0.108	0.197	0.895	0.329	<0.1	1.000	0.011	0.700	0.989	0.824	2.17
COMP2	8	16	1.000	0.000	1.000	1.000	1.000	<0.1	1.000	0.000	1.000	1.000	1.000	2.18
COMP3	34	22	0.875	0.016	0.778	0.978	0.824	<0.1	1.000	0.004	0.941	0.996	0.970	2.19
COMP4	22	16	0.625	0.057	0.455	0.921	0.526	<0.1	0.625	0.019	0.714	0.956	0.667	2.18
COMP5	17	12	1.000	0.143	0.500	0.875	0.667	<0.1	1.000	0.000	1.000	1.000	1.000	2.17
COMP6	17	12	1.000	0.000	1.000	1.000	1.000	<0.1	1.000	0.000	1.000	1.000	1.000	2.17
DAC1	10	11	1.000	0.000	1.000	1.000	1.000	<0.1	1.000	0.000	1.000	1.000	1.000	2.17
DAC2	12	19	1.000	0.000	1.000	1.000	1.000	<0.1	1.000	0.000	1.000	1.000	1.000	2.17
LATCH1	24	14	0.800	0.074	0.471	0.917	0.593	<0.1	0.600	0.000	1.000	0.970	0.750	2.18
Average	-	-	**0.839**	0.052	0.699	0.930	0.717	<**0.1**	0.790	**0.007**	**0.896**	**0.969**	**0.815**	2.17

† Runtime with GNN model training time excluded.

VI. Conclusion

This paper has presented a novel graph-learning-based symmetry constraint extraction framework for analog/mixed-signal circuits. An efficient heterogeneous multigraph representation has been proposed for interconnection modeling. A circuit feature embedding algorithm has been shown to represent a circuit with the most representative substructures. With the unsupervised inductive learning technique, the proposed framework is generalizable to every design. Experimental results have demonstrated the efficiency and effectiveness of the proposed framework in detecting both system-level and device-level symmetry constraints.

Acknowledgement

This work is supported in part by the NSF under Grant No. 1704758, and the DARPA IDEA program.

References

[1] B. Razavi, *Design of Analog CMOS Integrated Circuits*, 1st ed. USA: McGraw-Hill, Inc., 2001.

[2] A. Hastings and R. A. Hastings, *The Art of Analog Layout*, 1st ed. USA: Prentice Hall, 2001.

[3] J. Scheible and J. Lienig, "Automation of analog ic layout: Challenges and solutions," in *Proc. ISPD*, 2015, pp. 33–40.

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

[5] K. Kunal, M. Madhusudan, A. K. Sharma, W. Xu, S. M. Burns, J. Hu, D. A. Kirkpatrick, and S. S. Sapatnekar, "ALIGN: Open-source analog layout automation from the ground up," in *Proc. DAC*, 2019, pp. 1–4.

[6] B. Xu, K. Zhu, M. Liu, Y. Lin, S. Li, X. Tang, N. Sun, and D. Z. Pan, "MAGICAL: Toward fully automated analog ic layout leveraging human and machine intelligence," in *Proc. ICCAD*, 2019, pp. 1–8.

[7] I.-P. Wu, H.-C. Ou, and Y.-W. Chang, "QB-trees: Towards an optimal topological representation and its applications to analog layout designs," in *Proc. DAC*, 2016, pp. 1–6.

[8] K. Zhu, H. Chen, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Effective analog/mixed-signal circuit placement considering system signal flow," in *Proc. ICCAD*, 2020, pp. 1–8.

[9] H.-C. Ou, H.-C. Chang Chien, and Y.-W. Chang, "Nonuniform multilevel analog routing with matching constraints," *IEEE TCAD*, vol. 33, no. 12, pp. 1942–1954, 2014.

[10] H. Chen, K. Zhu, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Toward silicon-proven detailed routing for analog and mixed-signal circuits," in *Proc. ICCAD*, 2020, pp. 1–8.

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2016, pp. 1–14.

[12] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NeurIPS*, 2017, pp. 1024–1034.

[13] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *Proc. DAC*, 2020, pp. 1–6.

[14] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samandi, and S. K. Lim, "TP-GNN: A graph neural network framework for tier partitioning in monolithic 3D ICs," in *Proc. DAC*, 2020, pp. 1–6.

[15] Y. Li, Y. Lin, M. Madhusudan, A. Sharma, W. Xu, S. S. Sapatnekar, R. Harjani, and J. Hu, "A customized graph neural network model for guiding analog ic placement," in *Proc. ICCAD*, 2020, pp. 1–8.

[16] E. Charbon, E. Malavasi, and A. Sangiovanni-Vincentelli, "Generalized constraint generation for analog circuit design," in *Proc. ICCAD*, 1993, pp. 408–414.

[17] U. Choudhury and A. Sangiovanni-Vincentelli, "Constraint generation for routing analog circuits," in *Proc. DAC*, 1990, pp. 561–566.

[18] E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE TCAD*, vol. 15, no. 8, pp. 923–942, 1996.

[19] M. Eick, M. Strasser, K. Lu, U. Schlichtmann, and H. E. Graeb, "Comprehensive generation of hierarchical placement rules for analog integrated circuits," *IEEE TCAD*, vol. 30, no. 2, pp. 180–193, 2011.

[20] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "S^3DET: Detecting system symmetry constraints for analog circuits with graph similarity," in *Proc. ASPDAC*, 2020, pp. 193–198.

[21] K. Kunal, J. Poojary, T. Dhar, M. Madhusudan, R. Harjani, and S. S. Sapatnekar, "A general approach for identifying hierarchical symmetry constraints for analog circuit layout," in *Proc. ICCAD*, 2020, pp. 1–8.

[22] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proc. ICLR*, 2016, pp. 1–20.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.