Layout Symmetry Annotation for Analog Circuits with Graph Neural Networks

Xiaohan Gao
CECA, CS Department
Peking University
xiaohangao@pku.edu.cn

Chenhui Deng
ECE Department
Cornell University
cd574@cornell.edu

Mingjie Liu
ECE Department
UT Austin
jay_liu@utexas.edu

Zhiru Zhang
ECE Department
Cornell University
zhiruz@cornell.edu

David Z. Pan
ECE Department
UT Austin
dpan@ece.utexas.edu

Yibo Lin*
CECA, CS Department
Peking University
yibolin@pku.edu.cn

ABSTRACT

The performance of analog circuits is susceptible to various layout constraints, such as symmetry, matching, etc. Modern analog placement and routing algorithms usually need to take these constraints as input for high quality solutions, while manually annotating such constraints is tedious and requires design expertise. Thus, automatic constraint annotation from circuit netlists is a critical step to analog layout automation. In this work, we propose a graph learning based framework to learn the general rules for annotation of the symmetry constraints with path-based feature extraction and label filtering techniques. Experimental results on the open-source analog circuit designs demonstrate that our framework is able to achieve significantly higher accuracy compared with the most recent works on symmetry constraint detection leveraging graph similarity and signal flow analysis techniques. The framework is general and can be extended to other pairwise constraints as well.

1 INTRODUCTION

Analog circuits require many layout constraints to guide the layout design for functionality and performance [1]. Due to the large variance in different analog circuits, in modern analog layout automation methodology, the workload of such annotation is approaching manual drawing of layouts, limiting the wide acceptance of the automation tools [2, 3].

Symmetry constraint is one of the substantial and representative constraints for layout design. To reject common-mode noise and enhance circuit robustness, analog circuits may adopt topologies with devices matching with each other. These topologies not only require devices designed with the same sizes, but also drawn symmetrically in the layout to avoid mismatching parasitics from process variations and layout dependent effects [1]. Meanwhile, as symmetry constraints widely apply to analog layout designs, the workload of annotating all of them is extremely high. Thus, automatic annotation of such constraints is desired to alleviate the designers' workload.

In 2017, DARPA announced the Electronic Resurgence Initiative (ERI) and started to sponsor the ALIGN and MAGICAL projects for fully automated analog layout generation with "no-human-in-the-loop" through its IDEA program [2–4]. Automatic annotation of analog layout constraints is a key preprocessing step to guarantee the performance of analog placement and routing in both projects.

On the other hand, in real-world, analog circuits contain a large number of variants even for a single functionality. For example, there are more than 100 operational transconductor amplifier (OTA) topologies appeared in the textbook and research papers [1, 5]. Examples include telescopic, folded-cascode, and Miller-compensated. Designers may also develop their customized structures to boost the performance. This makes the automatic annotation challenging and difficult to generalize.

Existing efforts on symmetry constraint annotation can be categorized into two major types: 1) circuit analysis [6]; 2) graph matching [7–11]. The circuit analysis approaches usually require expensive circuit simulation to detect symmetry and matching constraints [6]. Graph matching based approaches include signal flow based graph automorphism, pattern matching with circuit template library, and graph similarity [7–11]. The performance of these approaches is either highly-correlated to the coverage of pattern library, or sensitive to the selection of similarity threshold.

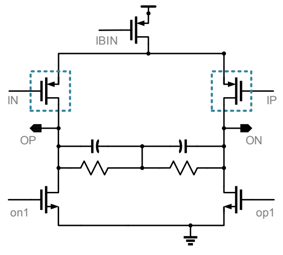
There are also other works on analog sizing based on rules and expert knowledge [12–14]. Recently, Kunal et al. [5] proposed a graph learning based approach to annotating the primitive blocks in the circuit hierarchy. The symmetry constraints are still encoded in the primitive circuit library.

In this work, we propose a graph learning framework for symmetry constraint detection. We leverage graph neural networks to learn a general set of rules of symmetry constraints from existing circuits rather than relying on pre-constructed template libraries or expensive circuit simulation. The major contributions of the work are summarized as follows.

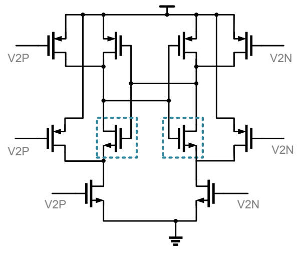
- We propose a graph learning framework for symmetry constraint detection. The framework is general and can be extended to other pairwise constraints as well.
- We map the symmetry constraint detection to a binary classification problem by measuring the similarity of node pair.
- We propose a path-based feature to mimic electric potential in circuit analysis.

*Corresponding author

ASPDAC '21, January 18–21, 2021, Tokyo, Japan
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431545



(a) OTA output stage



(b) Latch

Figure 1: Example of symmetry pairs with different neighboring structures in the circuits.

- We develop a probability-based filtering technique to effectively reduce the false positive rates.
- Experimental results on open-source analog circuits demonstrate that our framework can achieve > 90% TPR and < 1% FPR, comparing to the < 70% TPR and > 2% FPR achieved by the recent graph matching based algorithm S^{3}DET [11] and the signal flow analysis based algorithm [3], respectively.

The rest of the paper is organized as follows. Section 2 formulates the symmetry constraint detection problem; Section 3 explains the algorithm details; Section 4 demonstrates the experimental results; Section 5 concludes the paper.

2 PRELIMINARIES

2.1 Layout Symmetry Constraint

In this work, we tackle device-level symmetry constraints within analog circuit blocks. We can generalize our approach to large analog circuits providing the design hierarchy in the input netlist files. If we define the netlist of an analog circuit as a graph $G = (V, E)$ with vertices V representing devices like transistor, diode, capacitor, resistor, etc., and hyperedges E representing the interconnections, then we can define *symmetry pair* as a pair of devices/vertices (v_i, v_j) that need to be placed symmetrically to a central line in the layout. Figure 1 shows an example of symmetry pairs in two circuits, the output stage of an operational transconductance amplifier (OTA) and a latch. We refer one symmetry pair as a device-level symmetry constraint.

2.2 Graph Neural Networks

With wide application of deep learning, neural networks are known to be an effective and efficient model for tasks like classification and regression. However, neural networks like ANN and CNN only take vectors or tensors as input data, which are difficult to work on graphs. Defferrard et al. generalize convolutional neural networks (CNNs) from regular grid (e.g., images) to general graphs via graph convolutional filters [15]. Later, Kipf and Welling simplify the convolutional operator and propose graph convolutional network (GCN) [16].

Graph learning techniques can be categorized into *transductive* and *inductive* settings. Under the transductive setting, the embedding of each node is directly optimized and thus the training process requires to see all the nodes. The original GCN [16] is one kind of transductive approaches. The inductive approaches learn a general rule from the training graphs through sample and aggregation of structural information and node attributes [17], as shown in Figure 2. The learned model can be applied to unseen data. In this work, we would like to build a model to detect layout symmetry constraints of unseen graphs, so an inductive approach like GraphSage [17] is adopted.

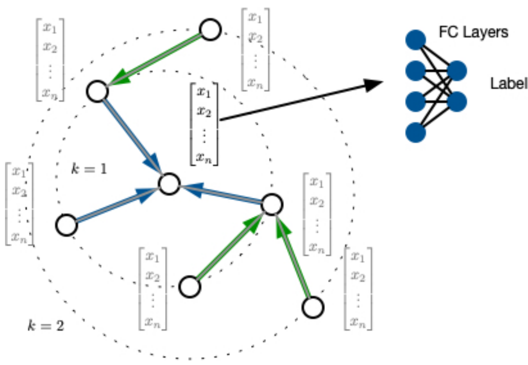


Figure 2: Sample and aggregation to obtain node embeddings. The node embeddings can be used as the input to fully connected (FC) layers for classification.

2.3 Problem Formulation

We define the layout symmetry detection problem as follows. Given a set of analog circuits with device sizes and labeled symmetry pairs, our objective is to build a graph learning model to predict the symmetry pairs from the circuit netlists with maximum accuracy.

For a circuit $G = (V, E)$ with $|V|$ devices, we need to make $O(|V|^2)$ pairwise predictions to obtain all symmetry pairs. The number of actual symmetry pairs is usually much smaller than the non-symmetry ones. Thus, this is a biased learning problem. We adopt *true positive rate* (TPR), *false positive rate* (FPR) and *F1-score* as the holistic accuracy measures. The definitions of TPR, FPR, and F1-score are as follows,

$$\begin{aligned} TPR &= \frac{TP}{P}, \quad FPR = \frac{FP}{N}, \\ F1\text{-}score &= \frac{2TP}{2TP + FN + FP}, \end{aligned} \tag{1}$$

where TP stands for the amount of truly predicted positive labels, P for the amount of positive labels, FP for the amount of incorrectly predicted positive labels, and N for the amount of negative labels. The target of the learning task is to maximize TPR and minimize FPR, or equivalently to maximize the F1-score.

3 ALGORITHM

In this section, we explain the framework of our symmetry detection approach and algorithmic details. Figure 3 shows the overall flow, which consists of three main stages: pre-processing, GraphSage-based detection model, and post-processing. The pre-processing stage takes raw SPICE analog netlists as input and constructs graph representations for each circuit. In this stage, we also extract feature vectors from the type information of the netlists and structure of the graph. In the detection model stage, we map the symmetry constraint detection problem into a binary classification problem. We train GraphSage [17] in a supervised manner, which can predict potential 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. The detailed algorithms are presented as follows.

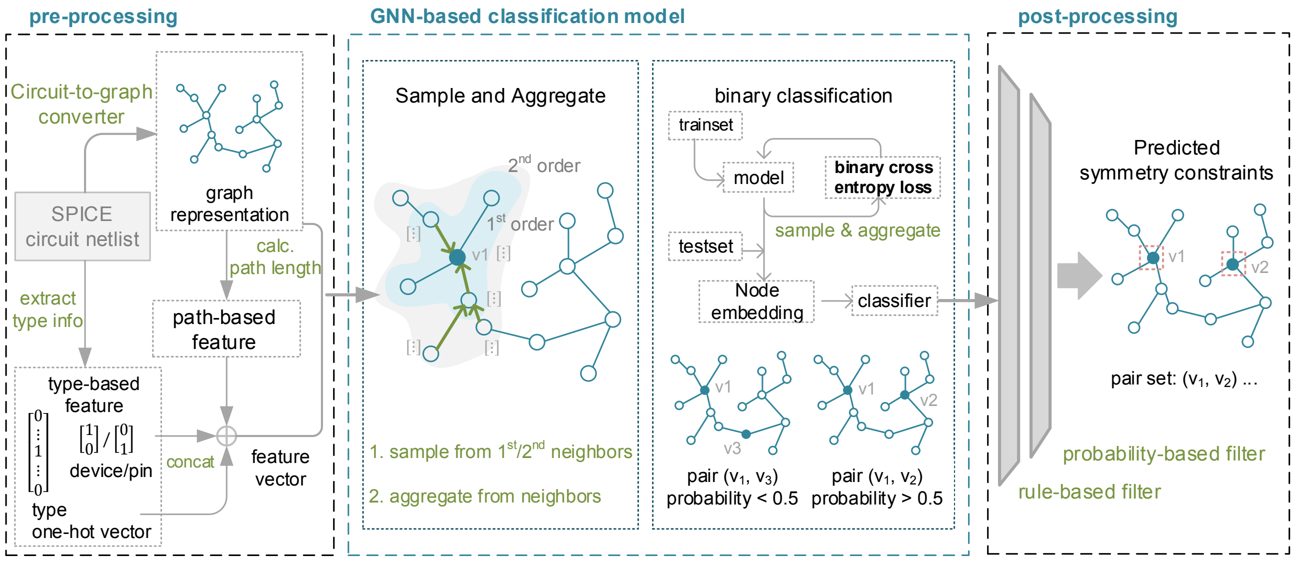


Figure 3: The overall flow.

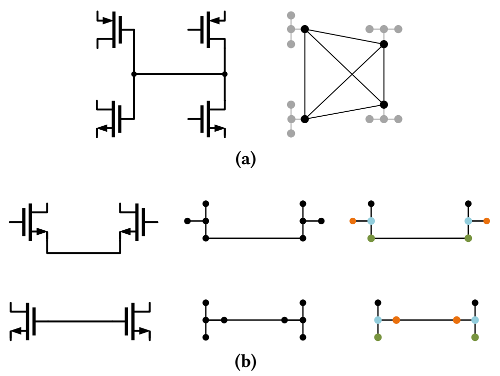


Figure 4: Simplified graph representation. (a) Convert hyperedge to clique. (b) Solve the isomorphic problem.

Algorithm 1 Path-based feature extraction

```
Input: Graph representation $G$ of analog circuits, VSS node $v_{vss}$
Output: Path-based feature $p$ for each pin node
 1: function AssignWeight($G$)
 2:     for edge $e : (v_1, v_2)$ of graph $G$ do
 3:         if $v_1$ is device or $v_2$ is device then
 4:             set weight of edge $e$ to 0.5
 5:         else
 6:             set weight of edge $e$ to 0
 7: end function
 8: function ExtractFeature($G, v_{vss}$)
 9:     AddWeight($G$)
10:     for pin node $v$ of graph $G$ do
11:         $p \leftarrow$ ShortestPathLength($v_{vss}, v$)
12: end function
```

3.1 Graph Representation & Feature Extraction

The framework takes SPICE netlists as input. We adopt simplified graph representation from S^3DET [11]. As shown in Figure 4(a), both device instances and device pins are recognized as graph nodes and there are edges connecting pin nodes with their device nodes. The pins nodes form a clique if they share the same net, that is, there is an edge between any two pins of the net.

3.1.1 Type-based feature. This simplified graph representation has a problem that circuits with different topologies may have isomorphic graph representations, as shown in Figure 4(b). We improve the representation by incorporating the type information of each node to ensure unique topologies for different interconnections.

We encode the type information as part of the node feature. We use a two-dimensional vector to indicate whether a node is a device or a pin, where $[0, 1]$, $[1, 0]$ stand for a device and a pin, respectively. Then, we transfer 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) into a 13-dimensional one-hot vector. We introduce a power node and a GND node as auxiliary nodes and set their types to IO, which our model can utilize for distinguishing them.

3.1.2 Path-based Feature. We also propose a novel path-based feature 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/GND-sourced path lengths. Taking Figure 1(b) as an example, the annotated symmetry constraint in dotted rectangles is a pair of NMOS nodes. 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. We observe that the path length from the GND node to the corresponding pins of the two nodes are also the same.

Algorithm 1 describes how to compute this feature. Given a graph representation G and the VSS nodes (we can extract these special nodes from the naming convention in the netlists), we assign weight to each edge and then calculate the shortest path length from VSS node to each node. If there are multiple VSS/GND nodes in one circuit, we add an extra node as new VSS node and add edges with weight 0 between the extra node and VSS/GND nodes.

Table 1: The components of node features.

Component	Device-or-pin	Type (one-hot)	Path-based feature
Dimension	2	13	1
Property	local	local	global

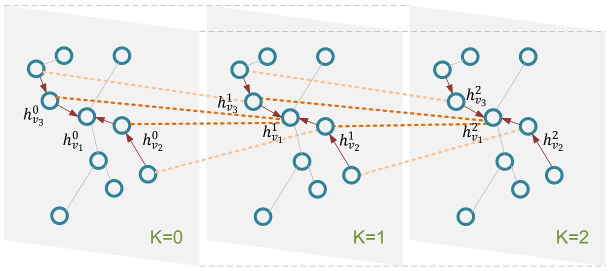


Figure 5: Sample and aggregate.

The intuition is that most symmetry pairs have the similar DC potential, according to our observation. Thus, we propose this path-based feature as a symbolic electric potential at each pin for capturing the global positions of nodes in the circuit. This feature also helps alleviate the issue of lacking global information in common graph learning [17] when generating node embeddings. Table 1 summarizes the three features and their dimensions. They are concatenated into a feature vector for graph learning.

3.2 Graph Learning Methodology

At this stage, we convert the symmetry constraint detection problem to a binary classification problem. Targeting the classification of node pairs, We train a graph neural network and apply the trained model to new circuits to generate node embeddings, which will be used on predicting the probabilities of symmetry constraint. We enhance the expressivity by replacing the explicit symmetry patterns with node embeddings, which are dense vectors distilled from local graph structure and node features. The distillation process is trainable and the graphs with node features generated at the previous stage will serve as the input of the process, so our node embedding model will comprehensively consider the information of neighbors and the structure of the graph.

We adapt GraphSage [17] for our node embedding model and customize the loss function for our problem. As an embedding framework, GraphSage has proved its superiority in problems like node classification, clustering, and link prediction, which shows its strong generalization ability for downstream tasks. Next, We explain the detailed forward propagation and how we map the symmetry constraint detection to binary classification.

3.2.1 Node Embedding. A complete procedure of a neural network involves forward propagation and back propagation. We will depict the previous part in this subsection and leave the details about the objective function to the next subsection. The key components of the forward propagation are sampling and aggregation. To explain the two basic components, we demonstrate the Figure 5.

The illustration takes a two-step sampling-and-aggregation inference as an example. K denotes the aggregation depth, that is, the number of neighborhood layers, and we set $K = 2$ in this figure. h_v^k denotes the node representation of node v at current step $K = k$. As for node v_1, its sampled first-order neighbors are node v_2 and v_3. At step $K = 1$, the algorithm aggregates node representation from sampled neighbors for each node. To make it specific, new node representation $h_{v_1}^1$ is a combination of previous representations $h_{v_1}^0$, $h_{v_2}^0$ and $h_{v_3}^0$. The algorithm also simultaneously proceeds sampling and aggregation for other nodes like v_2 and v_3. As the process iterates, node v_1 can obtain feature information from further neighborhoods.

Aggregator functions are crucial to the sampling and aggregation process. The Mean aggregator we adopted 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. We take ReLU as our activation function $\sigma(\cdot)$ in aggregation process. The Equation 2 summarizes the mean aggregator,

$$h_v^k = \sigma(W \cdot \{h_v^{k-1} \oplus MEAN(h_u^{k-1}, \forall u \in \mathcal{N}(v)\})). \quad (2)$$

3.2.2 Binary Classification. We formulate a new objective function to adapt the node embedding methodology to symmetry constraint annotation. We can designate a label for each node pair, which represents whether the node pair is a symmetric pair, 1 for symmetric, 0 for non-symmetric. Note that annotating the latent relationships between devices can be transformed into predicting the labels for device node pairs, the task is essentially a binary classification.

We split the mapping into two steps. The first step associates the node embeddings with the value of labels, while the second step finds a suitable loss function for back propagation. We figure out a straightforward valid approach for the first step, known as bilinear scoring function [18, 19], which is basically a mutation operator, defined as follows:

$$prob = \sigma(z_1^T W z_2), \quad (3)$$

where z_1, z_2 denotes the node embeddings of node v_1, v_2 and W is a trainable weight matrix. $\sigma(\cdot)$ is another activation function involved. The probability $prob$ that the label of pair (v_1, v_2) is 1 is determined by the mutation operator.

To train the aggregator functions mentioned before, we apply binary cross entropy loss, a classic loss function of binary classification. The loss function is designed for our target,

$$loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(prob_i) + (1 - y_i) \cdot \log(1 - prob_i), \quad (4)$$

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

3.3 Filters

Our model is able to annotate most of the symmetry constraints by predicting symmetric labels for node pairs. To further reduce the false alarm rate (to diminish the cases that a non-symmetric pair is labeled as symmetry constraint), we raise two filters.

Rule-based filter. The first filter is based on rules that the sizes and types of two device-level symmetric nodes are supposed to be identical. The second filter focuses on eliminating the redundant pairs on account of probabilities predicted before.

Probability-based filter. After the rule-based filter, we perform filtering based on the probability given by the graph neural network model. We make a rational assumption that a device node can appear in no more than one pair of nodes with device-level symmetry constraint. Taking the assumption as a starting point, we

Algorithm 2 Probability-based Filter

```
Input: Probability list $Prob$ along with given pairs
Output: Symmetry pair set $Set_{sym}$
1: function ProbFilter($Prob$)
2:     $Prob_{sorted} \leftarrow$ Sort($Prob$)
3:     for node pair $(v_i, v_j)$ of $Prob_{sorted}$ do
4:         if $v_i, v_j \notin Set_{sym}$ nodes set then
5:             add $(v_i, v_j)$ to $Set_{sym}$
6: end function
```

can remove the misjudged constraints which have nodes recurring in annotated pairs. The motivation of Algorithm 2 is selecting the possible node pair with the highest probability every time, which derives from the greedy algorithm. For each analog circuit, the filter sorts all candidate pairs and always pick the pair in the front of the queue to join the symmetry constraint set.

4 EXPERIMENTAL RESULTS

4.1 Baseline models

We compare our model with two other methods of symmetric constraint detection for analog circuits, S^3DET [11] and signal flow analysis (SFA), both developed in an open-source analog layout generator MAGICAL [3]. The S^3DET algorithm leverages graph similarity to match components with similar neighboring structures. While it is originally validated for system symmetry constraints, the algorithm is generic and can be adapted to device-level symmetry detection as well. The SFA implementation incorporates a set of conventional algorithms for device-level symmetry annotation: it first detect seed symmetry pairs like differential pairs with pattern matching; then, it performs signal flow analysis [1] to traverse the graph starting from the seed pairs and search for the rest symmetry pairs.

To differentiate from the original S^3DET for system symmetry constraints, we name our device-level implementation as S^3DET-dl. We modify the subgraph extraction step of the original method to fit device-level symmetry constraint detection. S^3DET-dl extracts neighbor subgraphs for two device nodes, and apply Kolmogorov-Smirnov test to judge the similarity of two subgraphs.

4.2 Datasets

Our datasets are based on two sets, S^3DET dataset and Minnesota ALIGN dataset. The former is the dataset used by the S^3DET project [11] and we obtain from the authors. The latter is obtained from the open-source repository of the Minnesota ALIGN team [2].

The SPICE files of the two datasets both encode the hierarchy identified by the designers. In the netlists, the representation of a circuit can be seen as a tree structure. The root node of the tree represents the circuit itself, and other nodes correspond to a sub-circuit. The sub-circuit corresponding to an upper node nests the sub-circuits of its child nodes. The leaf nodes represent sub-circuit blocks that are no longer divided in the design. In our experiment, we detect the device-level symmetry constraints, which appear in the leaf-level sub-circuits. We extract the leaf-level sub-circuits with a certain scale and with symmetry constraints in them to construct new datasets, S^3-leaf and ALIGN-leaf.

The extracted datasets have a variety of circuit architectures, including OTA (operational transconductance amplifier), comparator, DAC (digital-to-analog converter), and latch, etc. In addition to the difference in circuit types, there are difference in the scale of the circuits in our datasets and the distribution of device types, which our model needs to handle as well.

Table 2: Statistics of the datasets.

Datasets	Circuits	Nodes	Edges	Valid pairs	Pos/Neg
S^3-leaf	10	1378	9149	1522	89/1433
ALIGN-leaf	5	580	2134	576	48/528
OTA	5	684	3422	750	45/705

We split the datasets by using S^3-leaf as trainset and ALIGN-leaf as testset, since the baseline model SFA encodes general patterns covering symmetry constraints in S^3-leaf. We name the entire dataset composed of all the circuits as MIXED. We also select the OTA circuits from all circuits to form a new dataset, named OTA. We can verify the ability of our model to fit symmetry constraint detection for a single circuit type. For dataset OTA, we randomly split trainset and testset in a 3:2 ratio at circuit level. We make sure that there is no information leak from the testset to training. In our datasets, we define valid pair as a pair meeting the type rule. The valid pairs are candidates for symmetry constraints. Within valid pairs, there are positive ones with symmetry constraint and negative ones without symmetry constraint. Table 2 summarizes the statistics of our datasets.

4.3 Experimental Setup

Parameters used in our models and baselines are listed in Table 3. When training our graph neural network model, we sample nodes from their neighborhood at each iteration. As described in Section 3, we define the neighbor domain of a node in a layered manner and the depth K of inner-loop depends on the number of neighbor layers. For the convenience of experiments, we fix the parameter K to 4. That is, for each node in the testset, we explore four layers of neighbor domain in order to determine its node embedding value. This is also consistent with the setting in S^3DET which extracts the neighbor subgraphs by extending two device nodes.

Beyond the number of aggregator functions, there is another important parameter, sampling size S, for the sampling process. The sample size denotes the number of sampled nodes in each neighbor layer, which can partially reflect the weight of that layer for node embedding. The values of S are supposed to ensure that the algorithm can consider all pin node information for one node. Also, if we choose a scale too large, the generated node embedding will be inclined to the pin node since some nodes connect to pin node with high degrees.

For the baseline model S^3DET-dl, we need to set its parameter tol. The parameter tol is the threshold for similarity score calculated in S^3DET-dl. If the similarity score of a pair of nodes is greater than tol, the pair will be annotated as symmetry constraints. We choose the value which achieves the best results in testset for S^3DET-dl.

As shown in Table2, the device pairs with symmetry constraints only account for a small percentage of all device pairs. If we let all pairs of trainset participate in training, the training data will be extremely imbalanced. In order to learn a valid model, we turn to resampling techniques. Resampling is a method that adding some instances to the minority class or removing some instances of the majority class to alleviate the imbalance issue. We designate

Table 3: Parameter setup

Parameters	Source	Value	Meaning
K	ours (MIXED, OTA)	4	aggregation depth
S	ours (MIXED, OTA)	15	sample scale
r	ours (MIXED, OTA)	2	proportion(neg/pos)
e	ours (MIXED, OTA)	500	max training epoch
bs	ours (MIXED, OTA)	256	batch size
lr	ours (MIXED, OTA)	0.001	learning rate
hd	ours (MIXED, OTA)	20	hidden dimension
tol	S^{3}DET-dl, MIXED	0.99	similarity tolerance
tol	S^{3}DET-dl, OTA	0.9999	similarity tolerance

Table 4: Symmetry constraint annotation results

Dataset	MIXED			OTA		
Metric	TPR	FPR	F1-score	TPR	FPR	F1-score
S^{3}DET-dl	0.667	0.0722	0.485	0.556	0.0741	0.556
SFA	0.667	0.0203	0.676	0.778	**0.0185**	0.824
ours-A	**0.917**	0.0833	0.579	**1.000**	0.0556	0.857
ours-B	0.806	0.0167	0.784	0.889	**0.0185**	0.889
ours-full	**0.917**	**0.0074**	**0.904**	**1.000**	**0.0185**	**0.947**

a smaller proportion r of negative instances to positive instances to resample the training data.

4.4 Performance

We conduct experiments on two tasks MIXED and OTA. Table 4 shows the comparison results of our models to other models in symmetry constraint annotation. Beyond the aforementioned baselines, we also compare our model (denoted as ours-full) to the model without probability-based filter (denoted as ours-A) and the model without path-based feature (denoted as ours-B). We use TPR, FPR and F1-score as evaluation metrics.

A good model is expected to maximize TPR and F1-score and minimize FPR. Our model outperforms other baselines in all three metrics. We achieve a very low FPR (0.74% on MIXED and 1.85% on OTA), which means that we rarely predict the node pairs without symmetry constraints as positive. Probability-based filter promotes removing false positive predictions. With extremely low FPR, we can minimize the impact of over constraining the downstream tasks like placement and routing. Comparing to the model without path-based feature, our complete model achieves 11.1% improvement in TPR for both tasks. Especially in the OTA task, our model detects all the symmetry constraints.

Finally, we evaluate the runtime in training and testing, as summarized in Table 5. We can see that the inference time per circuit is comparable to the other two approaches and the one-time training time is also very fast.

5 CONCLUSION

In this work, we propose a graph learning based framework for layout symmetry annotation in analog circuits. By extraction of both local and global features from the graph representations of the analog netlists, we develop a graph neural network with a dedicated training technique to learn the node similarity on imbalanced data. We further propose a rule-based filter and a probability-based filter to effectively reduce the false positive rate. The experimental results show that we can achieve higher true positive rate (> 90%) and lower false positive rate (< 1%) compared with the recent detection algorithms based on graph matching and signal flow analysis. In the future, besides on further improving the accuracy, we plan to extend the experiments to other pairwise constraints such as matching and even non-pairwise constraints that involving more than two devices, like concentric symmetry, matching, etc.

Table 5: Runtime results (s)

Dataset	MIXED			OTA		
Time	Train time	Inference time per circuit		Train time	Inference time per circuit	
		avg	max		avg	max
S^{3}DET-dl	-	0.36	0.84	-	0.03	0.09
SFA	-	0.66	0.92	-	0.43	0.49
ours	102.70	0.06	0.11	37.80	0.10	0.12

ACKNOWLEDGE

This project is supported in part by the National Key Research and Development Program of China (No. 2019YFB2205001).

REFERENCES

[1] B. Razavi, *Design of Analog CMOS Integrated Circuits*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2001.
[2] K. Kunal, M. Madhusudan, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, 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.
[3] 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*. IEEE, 2019, pp. 1–8.
[4] "DARPA IDEA program," https://www.darpa.mil/attachments/eri_design_proposers_day.pdf.
[5] K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. Sharma, W. Xu, S. M. Burns, J. Hu, R. Harjani, and S. S. Sapatnekar, "GANA: Graph convolutional network based automated netlist annotation for analog circuits," in *Proc. DATE*, 2020.
[6] E. Charbon, E. Malavasi, and A. Sangiovanni-Vincentelli, "Generalized constraint generation for analog circuit design," in *Proc. ICCAD*, Nov 1993, pp. 408–414.
[7] 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, Feb 2011.
[8] Q. Hao, S. Dong, S. Chen, X. Hong, Y. Su, and Z. Qu, "Constraints generation for analog circuits layout," in *ICCCAS*, vol. 2, June 2004, pp. 1339–1343 Vol.2.
[9] Z. Zhou, S. Dong, X. Hong, Q. Hao, and S. Chen, "Analog constraints extraction based on the signal flow analysis," in *Proc. ASICON*, vol. 2, Oct 2005, pp. 825–828.
[10] P. Wu, M. P. Lin, and T. Ho, "Analog layout synthesis with knowledge mining," in *Proc. ECCTD*, Aug 2015, pp. 1–4.
[11] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "S3DET: Detecting system symmetry constraints for analog circuits with graph similarity," in *Proc. ASPDAC*. IEEE, 2020, pp. 193–198.
[12] T. Massier, H. Graeb, and U. Schlichtmann, "The sizing rules method for cmos and bipolar analog integrated circuit synthesis," *IEEE TCAD*, vol. 27, no. 12, pp. 2209–2222, Dec 2008.
[13] R. Harjani, R. A. Rutenbar, and L. R. Carley, "A prototype framework for knowledge-based analog circuit synthesis," in *Proc. DAC*, 1987, pp. 42–49.
[14] T. Massier, H. Graeb, and U. Schlichtmann, "The sizing rules method for cmos and bipolar analog integrated circuit synthesis," *IEEE TCAD*, vol. 27, no. 12, pp. 2209–2222, 2008.
[15] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. NeurIPS*, 2016, pp. 3844–3852.
[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[17] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NeurIPS*, 2017, pp. 1024–1034.
[18] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.
[19] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.