Efficient Hotspot Detection via Graph Neural Network

Shuyuan Sun[1], Yiyang Jiang[1], Fan Yang[1*], Bei Yu[2] and Xuan Zeng[1*]
[1]State Key Lab of ASIC & System, School of Microelectronics, Fudan University, China
[2]Department of Computer Science and Engineering,The Chinese University of Hong Kong, China

***Abstract*—Lithography hotspot detection is of great importance in chip manufacturing. It aims to find patterns that may incur defects in the early design stage. Inspired by the success of deep learning in computer vision, many works convert layouts into images, turn the hotspot detection problem into an image classification task. Traditional graph-based methods consume fewer computer resources and less detection time compared to image-based methods, but they have too many false alarms. In this paper, a hotspot detection approach via the graph neural network (GNN) is proposed. We also propose a novel representation model to map a layout to one graph, in which we introduce multi-dimensional features to encode components of the layout. Then we use a modified GNN to further process the extracted layout features and get an embedding of the local geometric relationship. Experimental results on the ICCAD2012 Contest benchmarks show our proposed approach can achieve over 10× speedup and fewer false alarms without loss of accuracy. On the ICCAD2020 benchmark, our model can achieve 2.10% higher accuracy compared with the previous approach.**

I. Introduction

With the continuous shrinking of feature size and the increasing circuit complexity, it is extremely difficult to keep the designed masks and the printed wafers consistent. Designers may accidentally introduce layout patterns that could cause fatal defects to the printed wafer because of effects incurred by the lithography process, particularly in advanced technology nodes. These patterns are called lithography hotspots, and the tape-out design should be hotspot-free to ensure that the circuit works correctly. Traditionally, it is required to perform the lithography simulation over the whole chip after each design adjustment to detect the hotspots. Because lithography simulation is highly time-consuming, many approaches have been proposed to replace lithography simulation in the early design stage for hotspot detection. The hotspot detection approaches can be classified into two categories: pattern matching-based and machine learning-based approaches. These approaches may not be as accurate as lithography simulation, but they can significantly save time in the early design stage and speed up the design cycle. Besides, they can narrow the search range for subsequent accurate hotspot detection.

For pattern matching-based methods, a density-based layout encoding scheme is proposed in [1]. Then, it applies principal components analysis (PCA) to discriminate between the hotspots and non-hotspots. In [2], a tangent space-based distance metric is proposed to classify the patterns. Generally, the pattern matching-based approach is faster but can only detect hotspot patterns which are in the pre-defined library.

For machine learning-based methods, [3] integrates the inception and attention modules into the detection flow. The classification loss and embedding triplet loss are used together to guide the learning procedure. In [4], the layout is transformed to the frequency domain via discrete cosine transform (DCT). The bias learning method is then used to alleviate the unbalance in hotspot and non-hotspot clips numbers. In [5], inspired by the truth that layout patterns are binary images and normal pictures are three-channel colored images, the authors proposed to leverage the binary neural network (BNN) to do the classification. Machine learning-based methods can achieve better accuracy but may suffer from more false alarms compared to pattern matching-based methods.

Hotspot detection approaches can also be classified based on the type of input: image-based and graph-based approaches. Most hotspot detection approaches, including all the previously mentioned methods, are image-based, which would be computationally intensive. There are a few works based on the abstract graph generated from the layout. In [6], Delaunay triangulation is used to extract features of hotspot patterns. In [7], the layout rectangles are represented as the graph nodes, and polygons with complex shapes are decomposed into rectangles. The shape of rectangles and the geometric relationship of layout are all encoded into one weight matrix. In [8], embeddings of critical patterns are represented as the eigenvector of the Laplacian matrix of the graph. The experimental results of [8] achieve 10× speeds up in the benchmark ICCAD2012 [9] compared with previous methods. Although graph-based approaches are mostly faster than image-based, they all suffer from unacceptable false alarms. We infer that the poor graph encoding scheme can not fully capture the information of layouts, which degrades the performance of graph-based models.

In this paper, we propose to obtain the embedding of layout via a modified graph neural network (GNN). Message passing-based GNNs iteratively transform and aggregate the neighborhood information. The node can capture further knowledge of the graph by stacking more layers of GNNs. The node embedding incrementally accumulates the local information of the graph. We use the multi-layer perceptron (MLP) to perform transformation on the feature vectors. The major contributions of this work are summarized as follows.

- We leverage a modified GNN to learn a better embedding of the layout. It can significantly accelerate the hotspot detection flow. Besides, it is more adaptable compared to other deep learning-based methods. Traditional convolution neural networks must have fixed-size layout images as input, but the proposed approach can take in any size of layouts without adjustments to the GNN model.
- We build a graph model to represent the layout. Rectangles are represented by nodes, and edges are created between neighboring nodes. Node embeddings indicate the intrinsic properties of rectangles. Edge embeddings describe the relative position relationship between adjacent rectangles. In this way, the constructed graph could maximumly reserve the geometric information of one layout.

*Corresponding authors: {yangfan, xzeng}@fudan.edu.cn.

- We propose a graph neural network that can process both multi-type and multi-dimensional edges inspired by [10], [11].
- In the benchmark ICCAD2012 [9], our model can achieve over 10× speedup and fewer false alarm without significant loss of accuracy. In the more challenging benchmark ICCAD2020 [3], our model achieves average 2.52% higher accuracy compared with [3].

The rest of the paper is organized as follows. In Section II, we will present the background of hotspot detection and graph neural network. In Section III, we propose the GNN-based hotspot detection approach. In Section IV, the experimental results are presented to demonstrate the efficiency of the proposed method. In Section V, we conclude the paper.

II. Background

In this section, we will present some preliminary knowledge about hotspot detection and then review the background of graph neural networks (GNNs).

A. Problem Formulation

Definition 1 (Accuracy): The ratio of correctly predicted hotspots among the set of actual hotspots [9].

$$\text{Accuracy} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}}. \tag{1}$$

#TP denotes the number of clips that are truly positive predicted. #FN denotes the number of actual hotspot clips missed by the detector.

Definition 2 (False Alarm): the number of incorrectly predicted non-hotspots [9].

$$\text{FalseAlarm} = \#\text{FP}. \tag{2}$$

Problem 1 (Hotspot Detection): Given a collection of clips that contains hotspot and non-hotspot patterns, our goal is to train a classifier that can maximize the **accuracy** and minimize the **false alarm**.

B. Graph Neural Networks

Graph Neural Networks (GNNs) iteratively apply aggregation and transformation on nodes to extract multi-scale local information. Many variants of GNNs have been proposed recently, such as graph convolutional network (GCN) [12], RGCN [11] and EdgeConv [10].

GCN [12] applies one-order localized convolution for each node in the graph. In every GCN layer, each node transforms and aggregates the features of its neighboring nodes.

$$f(v_i)^{(l+1)} = \sigma\left(\frac{1}{c_i}\left(\sum_{j\in N(i)} W^{(l)} f(v_j)^{(l)} + W^{(l)} f(v_i)^{(l)}\right)\right), \tag{3}$$

where, $f(v_i)^{(l)} \in R^{d^{(l)}}$ represents the feature of node v_i in the l^{th} layer of the neural network, with $d^{(l)}$ being the dimensionality of the feature of the l^{th} layer. $W^{(l)} \in R^{d^{(l+1)}\times d^{(l)}}$ is the transformation matrix in l^{th} layer. $N(i)$ denotes the neighboring nodes of node i. $\sigma(\cdot)$ represents the activation function. c_i is the normalization constant.

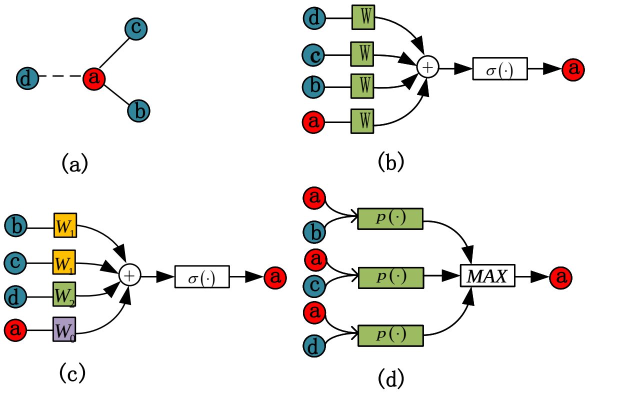

Fig. 1: (a) is the original graph, (b), (c), and (d) respectively explain how the embedding generators of GCN [12], RGCN [11] and EdgeConv [10]. In (b), W denotes the transformation matrix. In (c), W_r denotes the transformation matrix applied in relation r. In (d), $p(\cdot)$ denotes the edge generation function.

As shown in Equation (3), GCN does not distinguish the types of edges. In RGCN [11], different transformation matrices are used for different types of edges.

$$f(v_i)^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in N(i,r)} \frac{1}{c_{i,r}} W_r{}^{(l)} f(v_j)^{(l)} + W_0^{(l)} f(v_i)^{(l)}\right), \tag{4}$$

where $N(i,r)$ denotes the set of neighboring nodes of node i with edge type $r \in R$. R denotes the set of types of edges. W_0 denotes the transformation matrix of previous node embedding. $c_{i,r}$ is the normalization constant for edge type r. W_r is the transformation matrix for edge type r.

In [10], the EdgeConv is proposed to further leverages the feature of edges.

$$f(e_{ij})^{(l)} = p\left(f^{(l)}(v_i), f^{(l)}(v_j)\right), \tag{5}$$

where $f(e_{ij})^{(l)}$ denotes the feature vector of edge e_{ij} in l^{th} layer. $p(\cdot)$ is a function to generate the edge embedding with the features of connected nodes as inputs. The features of edges are then used to update the features of nodes. It use 'max' aggregation instead of 'mean'.

$$f(v_i)^{(l+1)} = \max_{j\in N(i)} f(e_{ij})^{(l)}. \tag{6}$$

An illustration of GCN [12], RGCN [11] and EdgeConv [10] is shown in Figure 1.

III. Proposed GNN-based Hotspot Detector

In this section, we will present our proposed GNN-based lithography hotspot detector (GNN-HSD). We creatively propose a novel approach to map a layout to one graph. A modified graph neural network (GNN) is adopted to further process features of nodes and edges. In this way, we get a good representation of the layout and use it to do the hotspot classification.

A. Graph Representation of Layout

In Figure 2, we show a small cut from one actual layout clip. Polygons represent all graphics on the layout. In Figure 2, the rectangle a and rectangle b together form one polygon of the

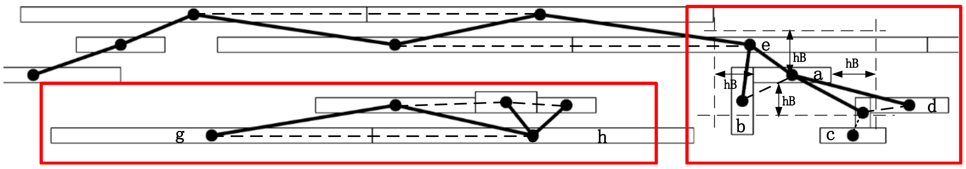

Fig. 2: An example of the graph representation of layout: dashed lines denote edges of type r_0, and solid lines denote edges of type r_1. The situation in the right box explains how to construct edges of type r_1. The situation in the left box explains how to decompose long rectangles, which also can be seen as the creation process of edges of type r_0.

layout. We use a graph $G = (V, E, R)$ to represent the layout pattern. Here, V denotes the node set. Each node corresponds to a rectangle in layout. For complex polygons, we decompose them into rectangles. For rather long rectangles, we further decompose them into small ones. Thus, there would not be too much information of neighboring nodes that have to be aggregated.

In the graph $G = (V, E, R)$, E represents the edges between the nodes. R denotes the type set of edges. There are two types of edges, i.e., $R = r_0, r_1$ in our built layout graph. The edges of type r_0 connect rectangles belonging to the same polygon. In other words, the decomposed rectangles are connected by edges of type r_0. Edges of type r_1 connect rectangles that are adjacent but not fractured from the same polygon. A threshold of distance hB is introduced here. If two rectangles overlap after inflating distance of hB in all directions, an edge of type r_1 will connect this two rectangles. Rectangles that are too close would affect each other during lithography, thus more possibly resulting in lithography hotspots. The threshold hB of distance is determined by the technology. For example, we can set hB to half of the feature size of the technology. In this way, we can construct a layout graph with two types of edges. GNN can identify these two types of neighbors and treat them differently in the information propagation process.

We show an example of the graph representation in Figure 2. We decompose all polygons in the layout and represent them as nodes in the graph. In the left bottom box of Figure 2, the long rectangle is decomposed into small rectangles g and h. Then we create edges between adjacent rectangles (nodes). In the right box in Figure 2, the distances between rectangles e, d, and a are lower than the threshold hB. Thus, in the built layout graph, nodes e and d are connected to node a through edges of type r_1. Note that rectangle c is far away from rectangle a, so they are not connected.

Compared to [8], we further decompose long rectangles into rectangles. It can efficiently represent the complex relationships of patterns. Also, we introduce more features of edges and nodes into the layout graph. In [8], a single weight value is used to represent the connection between two rectangles. It only uses a constant value to represent the connection between rectangles. Our methods use multi-dimensional feature vectors of nodes and edges, which may be more representative than the weight encoding method. Since more features are introduced in our proposed approach, the false alarms can be greatly suppressed, as proved by the experimental results.

B. Features of Nodes and Edges

For graph node representation, we take five intrinsic features of the rectangle to form a embedding of one node. The five features of the rectangle are stated as follows.

TABLE I: Notions in this paper.

$G = (V, E, R)$	The graph generated from layout
e_r	Edges that satisfied with relation r
r_0	Edge relation, if the edge connect nodes from the same polygon
r_1	Edge relation, if the edge connect nodes from different polygons
v_i	Node i belong to the node set V
$N(v_i)$	All neighboring nodes of v_i
$N(v_i, r)$	Neighboring nodes of v_i, connected by type r edges
$f(\cdot)$	The feature vector of a node or an edge
$h(\cdot)$	The feature vector of a neighbor

- x_range: the length of rectangles in the x-direction;
- y_range: the length of rectangles in the y-direction;
- min_dist: the minimum distance to the nearest connected rectangle with the edge of type r_1;
- shape_complexity: the number of edges of type r_0. It is used to measure the complexity of the polygon it belongs to. Note that edges generated by cutting long polygons are not considered here;
- env_complexity: the number of edges of type r_1. It is used to measure the complexity of its local environment.

We concatenate the above five features into one vector, which is taken as the embedding of one node. Unlike [10], we cannot extract the features of nodes by simply adding or subtracting the coordinates of the rectangles. Because rectangles, unlike points, have shapes and more complex relationships. By only adding or subtracting the center points of two rectangles, the information of the shapes and relationships of rectangles is missing. We can see from Figure 3, even though rectangles B and C share the same center point, their spatial relationships to rectangle A are different.

For edges of type r_0, their features are defined by positions of the junction points of two rectangles. For edges of type r_1, their features are defined based on the 'projection box' of two rectangles. Illustrated in Figure 3, the 'projection box' of two rectangles A and B is the gray rectangle between them. Taking

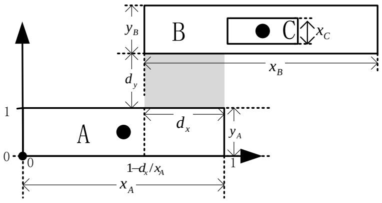

Fig. 3: Illustration of the edge feature. $f(e_{A,B})$=(1, 1-d_x/x_A, 1, d_y/y_A), $f(e_{B,A})$=(1, 0, d_x/x_B, -d_y/y_B).

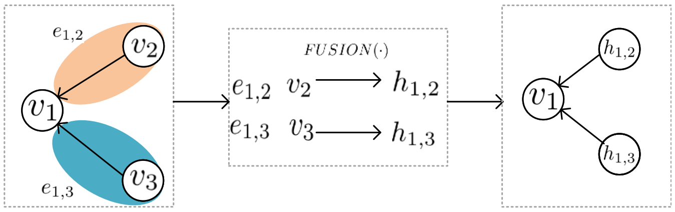
$e_{1,2}$
v_2
v_1
v_3
$e_{1,3}$
$FUSION(\cdot)$
$e_{1,2}$ $v_2 \longrightarrow h_{1,2}$
$e_{1,3}$ $v_3 \longrightarrow h_{1,3}$
v_1
$h_{1,2}$
$h_{1,3}$

Fig. 4: The generation of neighboring feature vectors.

the edge of type r_1 between rectangles A and B as an example, its feature $f(e_{A,B,r_1})$ is defined as (f_0, f_1, f_2, f_3). A coordinate system origin from the lower-left point of rectangle A is built. f_0 denotes the projection relationship of two rectangles A and B. One rectangle defines two intervals along the x and y directions, respectively. If the x intervals of A and B overlap, f_0 equals to 1. If the y intervals of A and B overlap, f_0 equals to 0. If both x and y intervals of two rectangles overlap, which indicates these two rectangles overlap and they belong to the same polygon. These two rectangles would be connected by an edge of type r_0 instead of r_1. f_1 and f_2 denote the coordinates of the overlap interval along x and y directions, respectively. f_3 denotes the projection distance between rectangles A and B. And f_1, f_2, and f_3 are normalized by the length of rooted rectangle in x and y directions. We show an example of how to get features of an edge of type r_1 in Figure 3.

C. Embedding Generation

After obtaining the initial features of nodes and edges, we use a modified graph neural network (GNN) to generate the final representations of graph nodes. Note that the initial feature vector x_v of a node only indicates the intrinsic properties of one rectangle. To represent the local geometric relationships via the generated node embeddings, we apply the message passing-based GNN on the layout graph. At each layer of GNNs, each graph node performs transformation and aggregation on feature vectors of its neighboring nodes. By stacking K layers of GNN, each graph node can learn a K-depth local knowledge of the graph.

In GCN [12], edges can only take binary or scalar values to describe connections in the graph. While in the constructed graph, the feature vectors of edges $f(e_{i,j})$ are multi-dimensional. To fully capture the inner interaction between graph edges and their connected nodes, a two-layer multi-layer perceptron (MLP) with dropout rate is introduced, which is denoted as the function $FUSION(\cdot)$. Function $FUSION(\cdot)$ injectively maps the combination of edge $e_{i,j}$ and its connected node v_j to a new feature vector $h_{i,j}$, as shown in Figure 4. Because its output embeddings concerns both the edge and the connected node, we call it the neighbor vectors.

After that, we take the elementwise mean of the vectors in $\left\{h_{i,j}^k, \forall v_j \in N\left(v_i, r\right)\right\}$ to derive a general vector $h_{N(v_i,r)}^k$, this operation is denoted as $AGGR(\cdot)$ in Algorithm 1. The mean aggregator $AGGR$ is proposed in [13]. The output of $AGGR(\cdot)$ is denoted as $h_{N(v_i,r)}^k$ which stands for a certain type of neighbors. $N(v_i, r)$ denotes the set of neighbor indices of node v_i under relation $r \in R$. As mentioned in Section III-A, there are two types of edges corresponding to different connections between rectangles in the constructed layout graph. To distinguish them, we separately deal with these two types of neighbors via two trainable networks $FUSION_r(\cdot)$ and respectively generate neighbor vectors. Finally, we concatenate all vectors in $\left\{h_{N(v_i,r)}^k, \forall r \in R\right\}$ and get the neighborhood vector $h_{N_{(v_i)}}$, which stands for all neighbors of v_i in the scope.

Algorithm 1 GNN embedding generation algorithm

```
Input : Graph $G(V, E, R)$; input node features $x_v$, $v \in V$; input edge features $x_e$, $e \in E$; depth K; relations $r \in R$;
Output : vector representation $z_g$ for the whole graph;
$f^0(v) \leftarrow x_v$, $v \in V$;
$f^0(e) \leftarrow x_e$, $e \in E$;
for $k = 1 ... K$ do
    for $r$ in $R$ do
        $h_{i,j}^k = FUSION_r\left(f^{k-1}(v_j), f^{k-1}(e_{i,j,r})\right)$;
        $h_{N(v_i,r)}^k = AGGR\left(\{h_{i,j}^k, \forall v_j \in N\left(v_i, r\right)\}\right)$;
    end for
    $h_{N(v_i)}^k = CONCAT\left(\left\{h_{N(v_i,r)}^k, \forall r \in R\right\}\right)$;
    $f^k(v) = UpdateNode\left(f^{k-1}(v), h_{N(v)}^k\right)$;
    $f^k(e) = UpdateEdge\left(f^{k-1}(e), h_{N(v)}^k\right)$;
end for
$z_v \leftarrow f^K(v), \forall v \in V$;
$z_g \leftarrow GlobalMaxPool\left(\{z_v\}\right)$;
return $z_g$;
```

The generation procedure of neighborhood vector $h_{N_{(v_i)}}$ injectively maps the neighboring information of a node to a low-dimensional vector. We incorporate the neighborhood vector $h_{N_{(v_i)}}^k$ with the previous layer node vector $f(v_i)^{k-1}$ to update the node embeddings. In the first layer of the modified GNN, the local geometric situation and the intrinsic properties of the rectangle are merged to update the node embeddings. In Algorithm 2, we leverage a one-layer MLP to do the node update. The following Equation (7) demonstrates the node update. Note that function $q(\cdot)$ also is represented as the function $UpdateNode(\cdot)$ in Algorithm 2.

$$f\left(v_i^{(k)}\right) = q\left(h_{N(v_i)}^{(k)}, f\left(v_i^{(k-1)}\right)\right). \qquad (7)$$

In traditional GNNs, edge features are commonly fixed during the iteration. And as node embeddings repeatively get updated, fixed edge embeddings can not well interpret the relationship between nodes. In [10], the spatial information of two connected nodes are used to update the edge embeddings via Equation (5). In our case, the neighbor vector $h_{N(v)}$ carries the geometric information. So we replace node embeddings with the neighborhood vectors $h_{N(v)}$. To utilize the edge embeddings

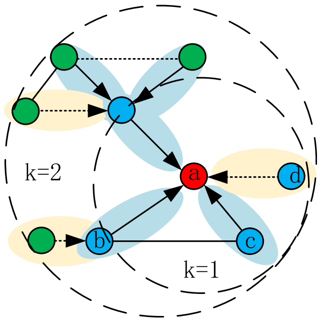

Fig. 5: The aggregation of a two-layer GNN.

of the previous layer, we add a single self-connection to each edge. Note that the initial edge vector x_e describes the relative position relationship between two adjacent rectangles. Equation (8) is the modified edge update function. In Algorithm 2, function $UpdateEdge(\cdot)$ further expands Equation (8). The matrices W_1 and W_2 in Algorithm 2 and W in Equation (8) all perform dimensional conversions.

$$f\left(e_{ij}^{(k)}\right) = p\left(h_{N(v_i)}^{(k)}, h_{N(v_j)}^{(k)}\right) + W \cdot f\left(e_{ij}^{(k-1)}\right). \tag{8}$$

Algorithm 2 Node & Edge update functions

```
**Input :** node feature $f(v)$; edge feature $f(e_{ij})$; neighbor feature $h_{N(v)}$; transform matrices $W$;
**Output :** updated node and edge features $f_{new}$;
**Function 1:** $UpdateNode\left(f(v), h_{N(v)}\right)$;
$f_{new}(v) = MLP\left(CONCAT\left(f(v), h_{N(v)}\right)\right)$;
**Function 2:** $UpdateEdge\left(h_{N(v)}, f(e_{ij})\right)$;
$h'_{N(v)} = W_2 \cdot h_{N(v)}$;
$f_{new}\left(e_{ij}\right) = MLP\left(CONCAT\left(h'_{N(v_i)}, h'_{N(v_j)}\right)\right)$;
$f_{new}\left(e_{ij}\right) = f_{new}\left(e_{ij}\right) + W_1 \cdot f\left(e_{ij}\right)$;
```

The goal of all previous operations is to deliver an embedding to best represent the local geometric situation for each graph node. Each node in the graph applies the transformation and aggregation to its neighboring nodes. More importantly, all nodes share the same model to process information simultaneously. The network used in the GNN is similar to the convolution kernel used in CNN. No matter how big the graph is, the GNN model size is fixed. And if the size of the input layout changes, we do not need dimensional adjustments to the GNN-based model to adapt to the variation.

Finally, we need to find a feature vector to represent the layout graph. In Algorithm 1, we apply 'GlobalMaxPool' on the last layer embeddings z_v of all graph nodes. The 'GlobalMaxPool' means to take the maximum value on each dimension of the node embeddings $z_v \in R^{N \times F}$. F is the dimension of the output feature vector and N is the number of nodes. In [14], it is pointed out that 'max' aggregation is good at extracting the primary information of the graph. Therefore, we believe the obtained graph feature vector can represent the geometric relationship of the layout clip better.

D. Training Graph Neural Network

Now we get $z_g \in R^{1 \times F}$ as the graph embedding. We believe it can deliver a good representation of the topology of the layout graph. In this subsection, we will introduce how to calculate the prediction loss and backward propagate it.

We leverage a two-layer MLP to convert $z_g \in R^{1 \times F}$ into a two-dimensional output vector $[x_n, x_h]$, which can be viewed as a one-hot encoding of the non-hospot/hotspot label. Then apply softmax function to do normalization:

$$x'_n = \frac{e^{x_n}}{e^{x_n} + e^{x_h}}, \quad x'_h = \frac{e^{x_h}}{e^{x_n} + e^{x_h}}. \tag{9}$$

The groundtruth label of the graph is defined as follow:

$$[y_n, y_h] = \begin{cases} [1,0], \text{graph label is nonhotspot} \\ [0,1], \text{graph label is hotspot} \end{cases} \tag{10}$$

TABLE II: Benchmark Statistics.

Benchmarks	Training Set		Testing Set		Size/Clip
	#HS	#NHS	#HS	#NHS	(μm^2)
ICCAD2012	1204	17096	2524	13503	3.6 × 3.6
Via-1	3430	10290	2267	6878	2.0 × 2.0
Via-2	1029	11319	724	7489	2.0 × 2.0
Via-3	614	19034	432	12614	2.0 × 2.0
Via-4	39	23010	26	15313	2.0 × 2.0
Via-Merge	50700	63653	3449	42294	2.0 × 2.0

The detection loss function is defined as follow:

$$L = -\left(y_n \cdot \log\left(x'_n\right) + y_h \cdot \log\left(x'_h\right)\right). \tag{11}$$

It is worth mentioning that the ICCAD2012 dataset [9] is quite unbalanced. The number of non-hotspot clips is much bigger than the number of hotspot clips. In this paper, we repeat the hotspot clips several times into the original training dataset. In this way, we can achieve a balance in the number of two classes clips. Also, we adopt the bias learning proposed in [4]. The ground-truth for non-hotspot clips is changed to $[1-\varepsilon, \varepsilon]$. We set ε=0.38 in this paper. The biased learning increases the detection accuracy but also introduces more false alarms at the same time.

IV. Experimental Results

Our approach is implemented in Python with the Pytorch-geometric toolkit [15]. An nVIDIA RTX 2080 Ti GPU is used for training and testing. To verify the efficiency and robustness of our approach, two benchmarks are employed. One is the ICCAD2012 benchmark [9], and the other is the ICCAD2020 benchmark [3]. Details of these two datasets are listed in TABLE II. "#HS" denotes the total number of hotspots, and "#NHS" denotes the total number of non-hotspots. Note that, the ICCAD2020 benchmark contains clips of the layout of vias. It is composed of four small datasets (the Via-1 to Via-4) ang a big merged one (Via-Merge). For more detailed information of the ICCAD2020 benchmark, please refer to [3].

A. Naive HSD via Graph Model

To illustrate the improvements brought in by GNN, we construct a naive lithography hotspot detection (HSD) model without GNN as a comparison. In the naive method, after the feature extraction in Section III-A and Section III-B, we feed the embedding matrix of nodes $X_n \in R^{N \times I_n}$ and of edges $X_e \in R^{E \times I_e}$ directly to a MLP for classification. Here, N and E denote the number of graph nodes and edges, respectively. $I_n = 5$ and $I_e = 4$ represent the dimensions of input feature vector of graph nodes and edges. Inspired by [16], we apply a shared single-layer feedforward neural network $a(\cdot)$ to get the attention coefficient of nodes and edges. Equations (12) and (13) show how to get the coefficients of nodes, which also applies to the edges.

$$c_{v_i} = a\left(f\left(v_i\right)\right), \tag{12}$$

$$\alpha_{v_i} = softmax\left(c_{v_i}\right) = \frac{\exp(c_{v_i})}{\sum\limits_{j \in N} \exp(c_{v_j})}. \tag{13}$$

After obtaining the attention coefficients α , we respectively compute the weighted sum of feature vectors of nodes and

TABLE III: Performance Comparisons for the ICCAD2012 and ICCAD2020 Benchmarks.

Benchmark	DAC'19 [5]			TCAD'19 [4]			ICCAD'20 [3]			NAIVE-HSD			GNN-HSD		
	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)
ICCAD2012	**98.54**	3260	561.28	98.40	3535	502.70	98.42	2481	143.79	87.20	5985	3.37	98.42	1731	3.2
Via-1	89.85	1886	57.76	71.50	773	43.36	93.42	1589	19.83	89.46	4603	1.75	**95.41**	1722	1.93
Via-2	73.00	1222	21.66	65.06	1290	40.02	86.32	1100	13.22	89.23	5492	1.70	**90.33**	1507	1.31
Via-3	73.38	3406	43.15	48.15	760	60.23	88.2	2105	20.69	86.57	6373	1.95	**89.81**	1928	1.63
Via-4	73.08	15288	51.98	76.92	155	67.44	80.77	152	20.70	76.92	1482	2.38	**84.62**	400	1.82
Via-Merge	90.42	9295	105.30	88.01	7633	165.85	92.2	6453	59.74	88.34	17969	5.62	**93.33**	5502	4.76
Average	83.06	5726.17	140.19	74.67	2357.67	146.60	89.89	2313.33	46.33	86.29	6984	2.80	**91.99**	**2131.67**	**2.44**
Ratio	0.90	2.68	57.45	0.81	1.11	60.08	0.98	1.09	18.99	0.94	3.28	1.15	**1.00**	**1.00**	**1.00**

edges. Then we concatenate the final embedding of nodes and edges together to get the embedding of the graph.

$$f(v)=\sum_{i \in N} \alpha_{v_i} f(v_i), f(e)=\sum_{j \in E} \alpha_{e_j} f(e_j), \tag{14}$$

$$f(G)=concat(f(v), f(e)). \tag{15}$$

The following training process is the same as mentioned in Section III-D. In TABLE III, “GNN-HSD” and “NAIVE-HSD” respectively correspond to our proposed layout representation model with and without GNN. Without GNN to aggregate the information of neighborhood, “NAIVE-HSD” suffers from unacceptable false alarms in both the ICCAD2012 and the ICCAD2020 benchmarks. This proves that GNN can deliver a better representation of layouts and play an important role in the hotspot detection task.

B. Results Comparison

TABLE III summarizes the performances of the proposed approach and the state-of-the-art approaches for the ICCAD2012 and the ICCAD2020 benchmarks. The results of other approaches are from [3]. Columns “DAC'19 [5]”, “TCAD'19 [17]”, and “TCAD'19 [4]” denote the results of selected baseline approaches respectively. Columns “GNN-HSD” and “NAIVE-HSD” correspond to the results of our proposed approach with and without GNN separately. “GNN-HSD” outperforms ICCAD'20 averagely with 2.10% on detection accuracy and about 19× speed up.

Fast detection speed is the main advantage of taking graphs as the input. Previous graph-based approaches like [8], suffer from serious false alarms. In [8], a fixed number of nodes is required to process the layout. Therefore, [8] has to divide the whole layout into smaller blocks to do the detection. The false alarm number reaches 40183 in ICCAD2012 in total, which is unacceptable. Our proposed method solves the problem of too many false alarms of graph-based approaches. In the ICCAD2012 benchmark, the false alarm number of “GNN-HSD” is only 1731 in total.

In the ICCAD2012 benchmark, our proposed approach detects faster than other listed approaches over 20 times. In the ICCAD2020 benchmark, “GNN-HSD” gains a better accuracy in all datasets. Note that DAC'19 [5] exhibits a slightly better detection accuracy, but it performs poorly on the ICCAD2020 dataset. In the Via-4 dataset, the false alarm of DAC'19 is 15288, while false alarm of “GNN-HSD” is only 676. The experimental results show good robustness in our proposed approach. In conclusion, our proposed “GNN-HSD” model can detect fast and accurately in the ICCAD2012 and the ICCAD2020 benchmarks.

V. Conclusion

In this paper, GNN-based approach is applied to the hotspot detection problem. It achieves more than 10× speed up with no loss of accuracy and fewer false alarms in both the ICCAD2012 and ICCAD2020 benchmarks. In the future, we hope to automatically generate features of graph nodes and edges from the basic information of the rectangles in the layout.

Acknowledgement

This research is supported partly by National Key R&D Program of China 2020YFA0711900, 2020YFA0711903, the National Natural Science Foundation of China (NSFC) Research Projects under Grants 61822402, 61774045, 62090025, 61929102, 62011530132, and The Research Grants Council of Hong Kong SAR (No. CUHK14209420 and CUHK14208021).

References

[1] Wan-Yu Wen and et al. A fuzzy-matching model with grid reduction for lithography hotspot detection. *IEEE TCAD*, 33(11), 2014.

[2] Fan Yang and et al. Improved tangent space-based distance metric for lithographic hotspot classification. *IEEE TCAD*, 36(9), 2016.

[3] Hao Geng, Haoyu Yang, and et al. Hotspot detection via attention-based deep layout metric learning. In *Proc. ICCAD*, 2020.

[4] Haoyu Yang, Jing Su, and et al. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE TCAD*, 38(6), 2018.

[5] Yiyang Jiang, Fan Yang, and et al. Efficient layout hotspot detection via binarized residual neural network. In *Proc. DAC*, 2019.

[6] Izumi Nitta and et al. A fuzzy pattern matching method based on graph kernel for lithography hotspot detection. In *Design-Process-Technology Co-optimization for Manufacturability XI*, volume 10148, page 101480U. International Society for Optics and Photonics, 2017.

[7] Andrew B Kahng, Chul-Hong Park, and Xu Xu. Fast dual graph-based hotspot detection. In *Photomask Technology*, volume 6349, 2006.

[8] Fan Yang and et al. Efficient svm-based hotspot detection using spectral clustering. In *Proc. ISCAS*, 2017.

[9] J Andres Torres. ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite. In *Proc. ICCAD*, 2012.

[10] Yue Wang and et al. Dynamic graph CNN for learning on point clouds. *ACM TOG*, 38(5), 2019.

[11] Michael Schlichtkrull, Thomas N Kipf, and et al. Modeling relational data with graph convolutional networks. In *European semantic web conference*, 2018.

[12] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[13] William L Hamilton and et al. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[14] Keyulu Xu and et al. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[15] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop*, 2019.

[16] Veličković and et al. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[17] Ying Chen and et al. Semisupervised hotspot detection with self-paced multitask learning. *IEEE TCAD*, 39(7), 2019.