Efficient Model-based OPC via Graph Neural Network

Shuyuan Sun[1], Xuelian Chen[2], Fan Yang[1*], Bei Yu[3], Shang Li[1] and Xuan Zeng[1*]
[1]State Key Lab of ASIC & System, School of Microelectronics, Fudan University, China, [2]Cogenda Inc., China
[3]Department of Computer Science and Engineering, The Chinese University of Hong Kong, China

***Abstract*—As feature size continues to shrink and light source wavelengths remain unchanged, the optical diffraction effects seriously degrade chip yield. Optical proximity correction (OPC) has become an essential step for chip manufacturability. However, OPC cannot achieve satisfactory mask correction results in an affordable number of iterations on some layouts, delaying the chip design cycle. In this paper, we propose to use the Graph Neural Network (GNN) to predict the initial mask correction and speed up model-based OPC. We design a graph model to represent the chip layout, with segments represented as graph nodes and the diffraction effects between them represented as graph edges. The GNN is used to obtain the embedding of the segment, and we predict the shift value based on its output. The proposed GNN-based predictor enhances the original OPC to find a good correction solution for full-chip-size masks in fewer iterations. compared with the original OPC method, experimental results on 55nm and 32nm full chip designs show that the proposed GNN-based OPC method reduces the number of iterations up to 75% and the acquired mask of comparable quality.**

I. Introduction

Segment shift-based OPC is the most widely used type of OPC in practical scenarios, which is also called the model-based OPC. For advanced technology nodes, existing model-based OPC methods suffer from high computation costs and may not get satisfactory correction results in a limited number of iterations. It raises serious concerns due to the stringent demand for a fast design cycle. For instance, it may take over one week on thousands of CPU cores to complete the OPC task for a large design.

Many methods have been proposed to accelerate model-based OPC methods, which can be divided into two categories: pattern matching based and machine learning based. For pattern matching based approaches [1], [2], the OPC results of frequently occurring design cells are reused to accelerate the OPC. Considering designs containing mostly repetitive cells, such as SRAM, the pattern matching-based approach can significantly reduce the runtime of OPC. However, the improvement brought by pattern matching-based methods is limited for layouts with many different shapes.

For machine learning-based approaches, the work [3] proposes the concentric square sampling (CSS) method to obtain the feature vectors of segments, then uses linear regression to predict the shifts. In [4], a hierarchical Bayesian model combined with linear regression is used to avoid overfitting. It also proposes a concentric circle area pixel sampling (CCAS) method to simulate the physics of light propagation, as shown in Fig. 1. In [5], the features of the segments are generated using the CCAS method, and then a neural network is used to predict the movements of segments. In [6], the matrix-based concentric circle sampling (MCCS) method is proposed, and maximal circular mutual information is used to select important features. CCS, CCAS, and MCCS methods use nearby pixels of a segment to represent local geometric relations. In these pixel-based methods, each segment is an independent instance. But a full-chip OPC has tens of thousands of segments that need to adjust their positions. The time for data preparation, inference, and training of pixel-based methods will soon become prohibitive.

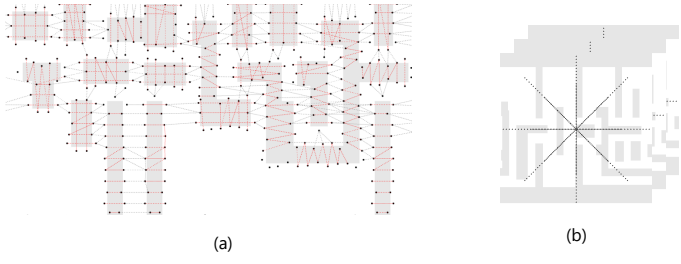

Fig. 1: (a) the constructed graph from the layout in our proposed approach. (b) the feature extracted from the layout using CCAS method. Each black node in (a) corresponding one clip like (b) in CCAS method.

Recently, another resolution enhancement technique (RET) called inverse lithography (ILT) has attracted much attention. In [7], Generative Adversarial Network (GAN) is employed to generate initial corrections to reduce the running time of ILT. In [8], a level set algorithm is introduced for mask optimization, and GPU is used to accelerate lithography simulations. In [9], each pixel on the mask constructs a graph using the pixels sampled by the CCAS method, then a GNN is used to predict the pixel value. However, the high computation cost and discontinuity problem prevent ILT from being widely applied to full-chip size mask correction. In [10], ILT is performed on a via layer of a larger layout, and good correction results are obtained. It is a nice try because vias locate sparsely in the layout, so there would be fewer graphics discontinuity problems. But there will be more discontinuous shapes near the boundaries of the ILT correction window since there are dense shapes in the M1 layer. The discontinuity problem of the ILT on the M1 layer will be severe. Another disadvantage is that the masks produced by ILT have poor manufacturability compared to the model-based OPC. ILT adjusts the pixels of the mask according to the difference between the litho result and the target image. It provides better flexibility in optimizing mask patterns but results in unwanted shapes that make masks difficult to produce. On the contrary, Model-based OPC may lose some flexibility since it only adjusts positions of segments, but its moderate runtime makes it practical to be applied to the full-chip design. In real-world scenarios, commercial OPC tools would choose the model-based OPC to correct the entire mask and then apply ILT to fix hot spots in smaller areas.

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

In this work, we propose to use GNN to predict the initial shifts of segments, aiming to speed up the OPC process. The outputs of the GNN-based predictor are taken as the initial input for the following model-based OPC. In Fig. 1, we show the extracted CCAS features by compared to the proposed graph model. The proposed method is designed for segment shift-based OPC on the full-chip size mask. and The main contributions of this work are summarized as follows.

- We introduce a graph model to represent the mask layout. The graph model can efficiently capture the features of the segments with significant less runtime for feature extraction compared to the pixel-based methods. It effectively reduces the complexity of the extraction of features compared to the pixel-based methods.
- We leverage a modified GNN to process the obtained features, then make predictions based on its output embedding. We also propose two different graph convolutional kernels for different types of geometric relationships between segments.
- Experimental results on 55nm and 32nm full chip designs show that the proposed GNN-based OPC method can reduce the number of iterations up to 75%, compared with the original OPC method, with comparable quality.

The rest of the paper is organized as follows. In Section II, we present the preliminary knowledge of model-based OPC and GNN. In Section III, we illustrate the proposed GNN-based OPC prediction method. In Section IV, the experimental results are demonstrated to show the efficacy of the proposed method. In Section V, we conclude the paper.

II. Background

In this section, we first review knowledge about OPC and the Graph Neural Network (GNN), then give definitions of measurement metrics and the formulation of the problem.

A. OPC

In Fig. 2(b), blue lines draw the target shape, and the grey contour is its aerial image after lithography simulation. The aerial image is significantly different from the designed shape. It shows the importance of OPC for improving the layout manufacturability. The model-based OPC technique first fractures edges into small segments denoted in orange in Fig. 2. Then under the guidance of lithography simulations, segments move inward or outward to compensate for unfavorable features caused by lithography effects. The model-based OPC repeatedly manipulates segments and tries to minimize the differences between the aerial image and the target design.

All shapes within distance D (around $1000 \sim 2000$nm) will affect the lithographic image of segments. As the feature size continues to shrink and the mask pattern is denser, more shapes should be considered in the OPC process, which makes the OPC more difficult to converge. In the model-based OPC, the shift of segments may oscillate and require more OPC iterations to have a satisfactory correction solution in advanced technology nodes.

B. Graph Neural Network

GNNs are very efficient at handling graph type data. In each layer of the GNN, all nodes perform the same computation simultaneously, and the operation is defined as the convolution kernel of the GNN. Therefore, the size of the input graph can be arbitrary since all nodes share the same parameterized kernel.

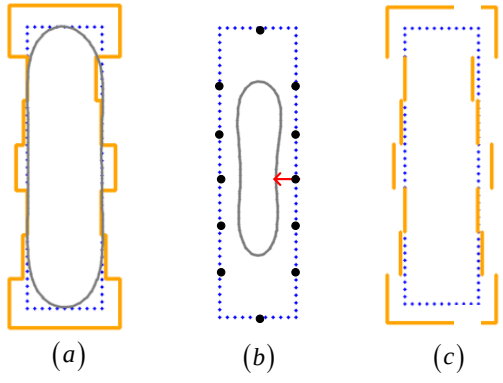

Fig. 2: The left one draws the corrected shape, its optical image and target shape. Optical image is drawn in grey lines and target shape is drawn in blue line. The corrected shape is in orange. The middle one is the original mask shape and its optical image. The right one shows that segments move outward and inward to compensate for the lithographic effects.

The convolution kernel of GNN transforms and aggregates the features of nodes and edges. Depending on the size of the information window that needs to be considered, we can stack different numbers of GNN layers to extract local information at multiple scales.

Many variants of GNNs have been proposed, and the difference between them mainly lies in the design of convolution kernels. Graph Convolutional Networks (GCN) [11] and RGCN [12] are two representative examples. GCN [11] applies a linear transformation to all adjacent nodes, then uses "mean" aggregation to obtain feature vectors.

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

where $f(v_i)^{(k+1)}$ denotes the feature of node v_i at k-th layer, $j \in N(i)$ represents all connected nodes of node v_i. $W^{(k)}$ represents the transformation matrix formed by learnable parameters at k-th layer. c_i is the normalization factor.

RGCN [12] applies different transformations to nodes connected by different types of edges, unlike GCN which uses the same transformation for all connections,. The computation of $RGCN$ is formulated in Eq.(2).

$$f(v_i)^{(k+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in N(i,r)} \frac{1}{c_{i,r}} W_r{}^{(k)} f(v_j)^{(k)} + W_0^{(k)} f(v_i)^{(k)}\right), \tag{2}$$

where R is the relation type set of the graph. $j \in N(i,r)$ means node v_j is the neighboring node of node v_i in type of r. $W_r^{(k)}$ stands for the transformation matrix for type r connection at k-th layer.

C. Problem Formulation

Definition 1 (RMSE Loss). Root-Mean-Square Error (RMSE) is an important metric to evaluate the quality of regression models. The calculation of RMSE is shown as

$$RMSE = \sqrt{\frac{\sum_{i}^{N} (y_i - \hat{y}_i)^2}{N}}, \tag{3}$$

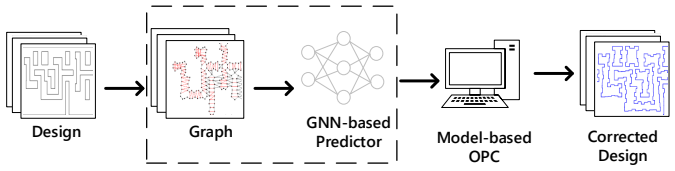

Fig. 3: The hybrid model-based OPC flow.

where y_i represents the ground truth value and $\hat{y}_i$ is the predicted value. N is the total number of testing data.

Definition 2 (R2 Score). $R2$ Score is also called the coefficient of determination, which is closely related to the RMSE loss. The definition of R2 is shown as

$$R2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}, \tag{4}$$

where $\bar{y}$ is the mean of all ground truth values. It measures the portion of variations in the dependable variables from the independent variables. When all the predictions take the mean of the true values, R2 score equals to zero.

Definition 3 (Edge Placement Error). Edge Placement Error (EPE) is an important metric to evaluate mask printability. As is shown in the middle image of Fig. 2, measuring points are sampled evenly on the horizontal and vertical edges of the target shape. We measure the minimum distance D from measuring points to lithographic contours. In Fig. 2, the red pointer gives one example of measuring the distance D. If the distance D is greater than the threshold distance thr, we mark it as an EPE violation. We set $thr = 15$ in our settings.

$$EPE(x, y) = \begin{cases} 1, D(x, y) \geq thr, \\ 0, D(x, y) < thr. \end{cases} \tag{5}$$

Problem 1 (Accelerate the Model-based OPC) Our goal is to shorten the runtime of model-based OPC by giving good initial moving values of segments, helping it give good correction results in a limited number of iterations. The prediction accuracy is evaluated by the RMSE loss and the R2 score. The mask printability is assessed by the amount of EPE violations.

III. Algorithms

In this section, we first introduce a hybrid model-based OPC pipeline, explaining how the original OPC improves its performance using our proposed method. Then we build a graph model to represent the layout geometry. Finally, we show how to use a GNN to obtain the final node embedding and use it to predict shifts of segments.

A. Hybrid Model-based OPC Flow

Model-based OPC first splits the edges of polygons into multiple movable segments, and then iteratively adjusts the positions of segments under the guidance of lithography simulations. It is time-consuming for OPC to obtain satisfactory calibration results on a chip size layout. Finding a good solution in a limited number of iterations is a challenge for OPC on some layouts. Therefore, we aim to reduce the runtime of model-based OPC by accurately predicting segment movements. The proposed hybrid OPC flow is shown in Fig. 3.

Furthermore, we propose a hybrid OPC online prediction scheme to fine-tune the GNN model during the OPC process.

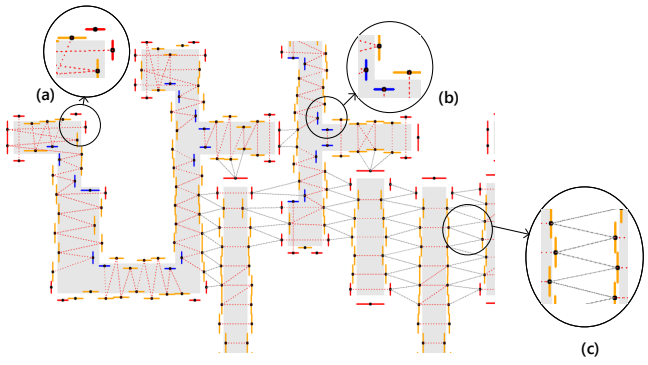

Fig. 4: A graph is constructed from a layout. (a) and (b) are enlarged views of the convex and concave corner segments. (c) is a magnified view of spatial connections (e_{sp} edges).

Due to the large size of the entire chip, model-based OPC divides the layout into small tiles. The OPC will be executed sequentially on these tiles. After completing OPC on N tiles, we retrain the GNN-based predictor using obtained correction results on these tiles. Therefore, the proposed hybrid OPC flow has good adaptability to various mask designs.

B. Graph Representation of Layouts

Fig. 4 is a small snippet from a layout, explains how we convert the layout to a graph $G = (V, E)$. V and E represent node and edge sets respectively. Segments are represented as graph nodes. Furthermore, we divide the nodes into three types: convex, concave, and ordinary. In Fig. 4, the segments represented by these three types of nodes are marked in red, blue, and orange. Fig. 4(a) and Fig. 4(b) respectively shows an enlarge view of convex and concave segments. The one-hot encoding is used to encode segment types and concatenate them with segment length and its ratio to edge length to form node embeddings representing segments.

We define three types of edges to represent the geometric relationship between the three types of line segments. For direct contact line segments, we use edges e_{seq} to connect them, which is called sequential connection. Edges e_{sp} connect adjacent segments belonging to different polygons, which we call spatial connections. We show an example of e_{sp} edges in Fig. 4(c) and mark them with grey dashed lines. An edge e_{sy} connects adjacent segments that belong to the same polygon but without directly touch, called a symmetric connection. In Fig. 4, we mark edges e_{sy} in red dashed lines.

The relative positions between segments are important for describing the local geometric relationships. We use the difference between the center points of the segment on the x-axis and the y-axis as the features of edge e_{seq}. The geometric relationship between two segments that in sequential connection is quite

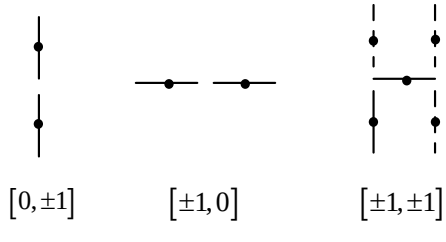

Fig. 5: An illustration for features of sequential connected segments.

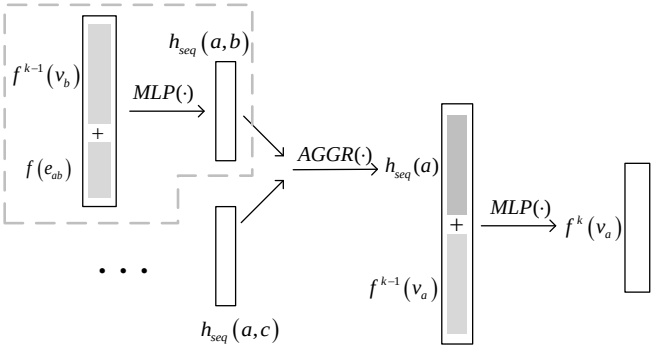

Fig. 6: The computation graph of $SeqConv$.

regular. They are either in a straight line or perpendicular to each other, as shown in Fig. 5. For simplicity, We give the definition for features of edge e_{seq} as

$$f\left(e_{seq}\left(a,b\right)\right)=\left[p\left(x_b-x_a\right),p\left(y_b-y_a\right)\right], \tag{6}$$

$$p\left(z\right)=\begin{cases}1, z>0\\0, z=0\\-1, z<0\end{cases}, \tag{7}$$

where $f\left(e_{seq}\left(a,b\right)\right)$ represents the feature vector of edge $e_{sq}\left(a,b\right)$. a and b are two segments connected via edge e_{seq}. Note that the sequential connection is directional. $e_{sq}\left(a,b\right)$ means b connected to a. x_a and y_a respectively stands for the x and y coordinates of center point of the segment a.

For the spatial and symmetrical connections, the geometric relationships between segments are more complex than sequential ones. The coordinates difference of center points of connected segments are used but without the reduction function $p(\cdot)$ shown in Eq. (7). Besides, we concatenate the vertical distance d_{ab} between connected segments to the embeddings of edge e_{sp} and e_{sy}.

$$f\left(e_{sp}\left(a,b\right)\right)=\left[x_b-x_a,y_b-y_a,d_{ab}\right]. \tag{8}$$

Eq. (8) formulates the embedding computation for edge e_{sp}, which also applies to the edge e_{sy}. Neighboring segments also need to be found in the OPC tool. So we can directly use the built-in functions in the OPC tool to efficiently construct the edge e_{sp} and the edge e_{sy}.

C. Graph Neural Network Design

In Section III-B, we build a graph to represent the layout. The initial node embedding of each segment can only represent its own attributes, we have to make good use of the connection in the graph. We leverage a message passing-based Graph Neural Network (GNN) to update the node embeddings via stacking layers. In one GNN layer, each node would transform and aggregate the features of its connected nodes to update self-representation. Each graph node could obtain a K-depth local knowledge by stacking K layers of GNN. We assume that the output node embeddings of the GNN can well encode the local geometric relationships of the segments.

All nodes on the graph share the same convolution kernel of GNNs. The difference between various message passing-based GNNs lies in the design of convolution kernels. The framework of most GNNs is shown in the Algorithm 1. The $Conv(\cdot)$ function stands for different designs of the convolution kernel. In GCN [11] and RGCN [12], the designs of $Conv(\cdot)$ function are respectively shown in Eq. (1) and Eq. (2).

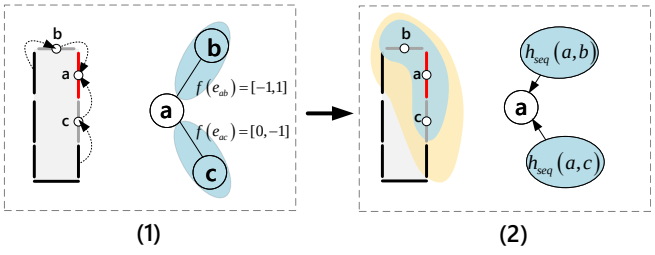

Fig. 7: An illustration for $SeqConv$.

Here we propose two convolution kernels, i.e. $SeqConv$ and $SpaceSymmConv$, to better encode the local neighborhood of segments. The span range of the edge e_{seq} is smaller than the edge e_{sp} and e_{sy}. If we process all nodes with different transformations in one convolution operation like RCN [12], we will get too much global information and lack local knowledge. Therefore, there are two convolutions $SeqConv$ and $SpaceSymmConv$ for processing the information passed through edge e_{seq} and edge e_{sp}/e_{sy} respectively. This modification to the convolution kernel also gives more flexibility compared to RCN [12].

Algorithm 1 the framework of message passing-based GNNs

Input : Graph $G(V,E)$, input node features x_v, $v\in V$; input edge features x_e, $e\in E$; depth K;
Output : vector representations z_v for all nodes $v\in V$;
$f^0(v)\leftarrow x_v$, $v\in V$;
$f(e)\leftarrow x_e$, $e\in E$;
for $k=1...K$ **do**
  $f^k\left(v_i\right)=Conv\left(G,f^{k-1}\left(v_i\right),f^{k-1}\left(v_j\right),f\left(e_{ij}\right)\right),e_{ij}\in E$;
end for
$z_v\leftarrow f^K(v),\forall v\in V$;
return z_v;

To consider both the relative positions and properties of segments, we concatenate the embeddings of edges with the embeddings of connected nodes to form a feature vector. The obtained feature vectors are then processed using a learnable MLP (Multilayer Perceptron) and output embeddings representing the node-edge pairs. The computation graph of this operation for the sequential connection is shown in the grey dashed box in Fig. 6. This operation is represented in Eq. (9), where the resulting embedding h_r is called the neighborhood feature vector. E_r denotes the edge set of type r, and $H(\cdot)$ represents the learnable MLP.

$$h_r(i,j)=H\left[f\left(v_j\right),f\left(e_{ij}\right)\right],e_{ij}\in E_r. \tag{9}$$

For each edge type, a node may have multiple connected nodes. Therefore, we take the element-wise mean of $\{h_r(i,j),\exists e_r(i,j)\in E_r\}$ to obtain a general vector $h_r(i)$. This operation is denoted as $AGGR(\cdot)$ in Fig. 6.

In $SeqConv$, we concatenate the neighborhood feature vector $h_{seq}(i)$ with the embedding $f^{k-1}\left(v_i\right)$ of node i, and use another MLP to generate the final representation of nodes in this GNN layer. Fig. 6 shows the computation graph of $SeqConv$. Fig. 7 gives a small example of $SeqConv$, illustrated on both the layout side and the graph side. In Fig. 7(2), the geometry of the segment within the blue area can be represented by the embedding of node a obtained after one $SeqConv$. And

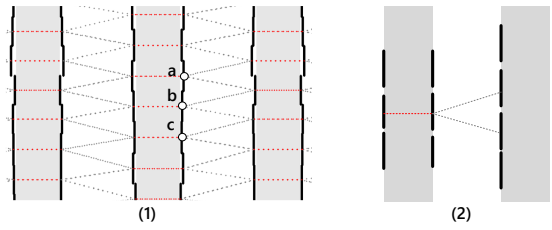

Fig. 8: Fragments a, b, and c in (1) have the same geometric neighborhoods as shown in (2), but with different shifting values after OPC.

segments wthin the yellow region could be reached by node a after two layers of $SeqConv$.

In $SpaceSymmConv$, we first calculate the neighborhood feature vectors of h_{sp} and h_{sy} using Eq. (9). Then we update the node embeddings using an MLP with $h_{sp}(i)$, $h_{sy}(i)$ and $f(v_i)$ as input. The computation graph of $SpaceSemConv$ is similar to that of $SeConv$ shown in Fig. 6, and the only difference is the input of the second MLP. We first use the $SeqConv$ to get a node embedding representing the partial shapes of polygon edges and then use $SpaceSymmConv$ to obtain information about the wider spatial extent.

Generally, we stacked two layers of $SeqConv$ followed by one layer of $SpaceSymmConv$. For advanced nodes, we stacked three layers of $SeqConv$, followed by two layers of $SpaceSymmConv$. We believe that the feature vector z_v obtained by GNN can capture the local geometric information of segments. We predict the movement of segments based on the outputs of GNN, making the predicted values are close to those obtained by the model-based OPC.

From the observations of the layout shapes after OPC, it is found that the shifts of segments are not only related to the local environments, but also affected by the movements of its adjacent segments. In Fig. 8, the geometric neighborhood of fragments a, b and c share the same structure as shown in (2), but their shifting values after the OPC are quite different.

Therefore, we make two times of predictions and use the attention mechanism to improve the prediction accuracy on shifting values. We have the first prediction y_1 based on the feature vectors z_v generated by the GNN. The attention module uses the movement value of the first prediction y_1 and the feature vector z_v as input to make the second movement prediction y_2. Experimental results show that it can greatly improve regression accuracy.

IV. Experimental Results

The GNN-based predictor is implemented in Python with the Pytorch-geometric toolkit [13]. An Nvidia RTX 2080 Ti GPU is used for training and testing. We conduct experiments on two full-chip size layouts to verify the effectiveness of our approach. One is on a 55nm CPU design with 837×553 μm^2 size. And the other is on the case 3 of the ICCAD 2016 dataset [14], which is a 32nm design. The ICCAD 2016 dataset was originally proposed for hotspot classification, but we use it to validate that our approach is also applicable at the full-chip size. This layout contains 9779 polygons and is of size 40288×25830 nm^2.

On the 55nm CPU layout, we integrate the proposed GNN-based predictor with a commercial OPC tool. Experimental results show that our method effectively reduces the number of iterations of the original OPC, while also improving the OPC quality. For the ICCAD 2016 layout of 32nm, we use the lithography simulator from the ICCAD 2013 benchmark suite [15]. We re-implemented the OPC method according to [16] as the raw OPC tool for the ICCAD 2016 layout.

TABLE I: Time analysis of graph construction and prediction.

#Polygon	W/H (nm)	Build Graph	Prediction
38	2048	0.02s	0.064s
447	2048 × 4	0.20s	0.56s
1293	2048 × 6	0.36s	1.03s
2164	2048 × 8	0.69s	1.83s

TABLE II: Regression Performance.

Layout	Area ($\mu m \times \mu m$)	Pixel-based MSE	Pixel-based R2	Pixel-based Inference time(s)	Graph-based MSE	Graph-based R2	Graph-based Inference time(s)
$xt184$	14×20	6.6964	0.7703	8.044	**6.0813**	**0.8345**	**0.05**
$sram$	122×81	2.7014	0.9217	11.53	**2.6155**	**0.9662**	**2.031**
Ratio		1.081	0.940	9.406	1	1	1

A. GNN predictor performance

To measure the efficiency and accuracy of our approach, we conduct the following experiments. Table I shows graph construction time and graph neural network inference time in layout of different sizes. The "W/H" denotes the width/height of the cut square layouts. The "#Polygon" denotes the number of polygons. The layouts are cut from the ICCAD 2016 layout. The results in Table I show that the runtime of our approach is trivial to the entire OPC process, which usually takes hundreds of seconds.

To evaluate the accuracy of predictions, we use two 65nm designs $xt184$ and $sram$ to form the training and testing dataset. For pixel-based approach, we use the CCAS method proposed in [4] to extract features. However, the accuracy of linear regression used in [4] is not acceptable. To improve the regression accuracy, we replace the linear regression in [4] by a neural network with two hidden layers. Dropout operation and L2 norm regulation are used to further improve the regression accuracy. In the $sram$ layout, the data used for graph-based predictor is 6 times smaller than the pixel-based one. In both layouts, the graph-based approach is more than 5x faster than the pixel-based one. We set the batch size to 1024 for the pixel-based predictor and 4 for the graph-based one. Note that the RMSE loss on the $xt184$ is bigger than on the $sram$ for both predictor. Because there are many repetitive shapes in the $sram$ layout, and the testing part of $xt184$ is most different from the training part. We use the $xt184$ layout to test the generality of the predictors and the $sram$ layout to evaluate the representation ability for the layout. This proves that the graph-based approach can achieve the same or better representation of the layout than the pixel-based one.

B. Full-chip OPC on CPU and ICCAD 2016 Layout

For full-chip size OPC, we need to divide the layout into small tiles to suit the input size of the optical model. Because of the movement of a fragment is determined by the litho intensity of its center point. Since these points are sparsely located in the layout, the OPC tools uses some sample rate to calculate the lithographic map. Therefore, the OPC tile size is usually larger than the input size of the optical model to make the OPC process more efficient. But with more graphics need to be considered, complete the OPC process in limited iterations would also be harder. We have demonstrated the accuracy of

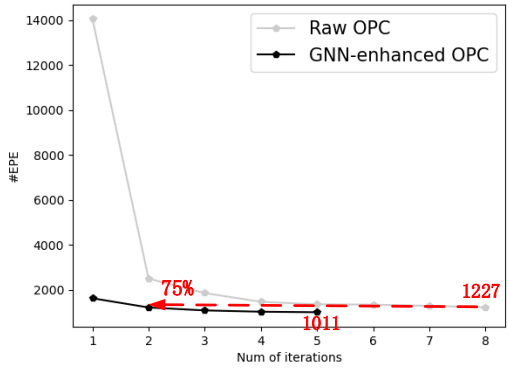

Fig. 9: The EPE descending curve of the CPU layout.

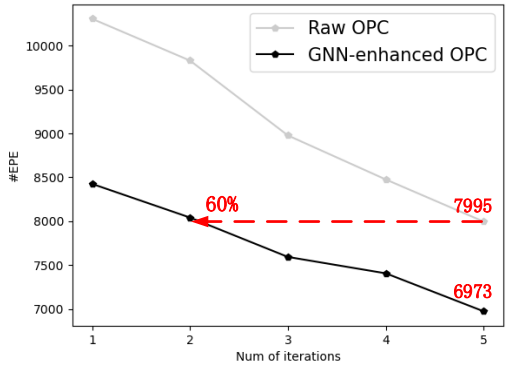

Fig. 10: The EPE descending curve of the ICCAD 2016 layout.

our GNN-based predictor before, and here we test whether it can boost the raw OPC and improve its performance.

The CPU layout is a real chip design of 837×553 μm^2 size, and the used commercial OPC tool divides it into 896 tiles of 27×27 μm^2 size. After the OPC results of N tiles are obtained, the GNN model is retrained with the new correction results. In this way, the prediction model will keep being updated during the OPC process to adapt the design. Compared to the original OPC with 8 iterations, GNN-enhanced OPC use only two iterations acquiring comparable correction result, shown in Fig. 10.

For the ICCAD 2016 layout, we modify the original litho simulator from [15] to simulate larger mask size. As this layout is comparable small, we split it into 24 tiles of size 8.2×8.2 μm^2 size. Since the OPC time for each tile is small and there are only limited tiles, we use 8 tiles for model training, then use the trained model give prediction to the rest tiles. In Fig. 9, the GNN-enhanced OPC reduces over 1000 EPE violation than the raw OPC after 5 iterations. Compared to the original OPC, the GNN-enhanced OPC achieves a similar number of EPE violations using only 40% running time, as shown in Fig. 10.

Our method may not suitable for isolated shapes and small size mask correction, because the original OPC on them is capable converge in limited iterations. But for full-chip-scale mask correction, the original OPC cannot give a corrected mask of good printability in a affordable number of iterations. Experimental results show that our GNN-based predictor provides good initial values for model-based OPC and acquire comparable mask quality in much fewer OPC iterations. Furthermore, Our method is practical and easy to be integrated into the original OPC flow.

V. Conclusion

In this paper, we propose a graph model to represent the layout, then use GNN to generate the final embedding of segments. The generated embeddings are used to predict the shifts of segments and accelerate the iteration procedure of OPC. Compared to traditional methods that predict the shift of fragments using their surrounding pixels, our graph-based approach has better prediction accuracy and consumes much less inference time. Experimental results show that our approach could significantly reduce the number of iterations of the original model-based OPC to obtain comparable correction results on full-chip designs.

Acknowledgement

This research is supported partly by National Key R&D Program of China 2020YFA0711900, 2020YFA0711903, partly by National Natural Science Foundation of China (NSFC) research projects 62090025, 62141407, 61974032, 61929102, and The Research Grants Council of Hong Kong SAR (No. CUHK14209420 and CUHK14208021).

References

[1] Tom Wang and etc. Pattern centric opc flow: a special ret flow with fast turn-around-time. In *Optical Microlithography XXI*, volume 6924, page 69243V. International Society for Optics and Photonics, 2008.

[2] Piyush Verma and et al. Pattern-based pre-opc operation to improve model-based opc runtime. In *Photomask Technology 2014*, volume 9235, page 923506. International Society for Optics and Photonics, 2014.

[3] Allan Gu and Avideh Zakhor. Optical proximity correction with linear regression. *IEEE Transactions on Semiconductor Manufacturing*, 21(2):263–271, 2008.

[4] Tetsuaki Matsunawa, Bei Yu, and David Z Pan. Optical proximity correction with hierarchical bayes model. In *Optical Microlithography XXVIII*, volume 9426, page 94260X. International Society for Optics and Photonics, 2015.

[5] Bo-Yi Yu and et al. Deep learning-based framework for comprehensive mask optimization. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 311–316, 2019.

[6] Bentian Jiang, Evangeline FY Young, and et al. A fast machine learning-based mask printability predictor for opc acceleration. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 412–419, 2019.

[7] Haoyu Yang and et al. Gan-opc: Mask optimization with lithography-guided generative adversarial nets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2822–2834, 2019.

[8] Ziyang Yu, Guojin Chen, Yuzhe Ma, and Bei Yu. A gpu-enabled level set method for mask optimization. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1835–1838. IEEE, 2021.

[9] Shengen Zhang, Xu Ma, and et al. Fast optical proximity correction based on graph convolution network. In *Optical Microlithography XXXIV*, volume 11613, pages 190–197. SPIE, 2021.

[10] Guojin Chen, Bei Yu, and et al. Damo: Deep agile mask optimization for full chip scale. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

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

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

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

[14] Rasit O Topaloglu. Iccad-2016 cad contest in pattern classification for integrated circuit design space analysis and benchmark suite. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–4. IEEE, 2016.

[15] Shayak Banerjee and et al. Iccad-2013 cad contest in mask optimization and benchmark suite. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 271–274. IEEE, 2013.

[16] Jian Kuang, Evangeline FY Young, and et al. A robust approach for process variation aware mask optimization. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1591–1594. IEEE, 2015.