SEDG: Stitch-Compatible End-to-End Layout Decomposition Based on Graph Neural Network

Yifan Guo[1], Jiawei Chen[1], Yexin Li[1], Yunxiang Zhang[1], Qing Zhang[1], Yuhang Zhang[2,1,*], Yongfu Li[1]
[1]*Department of Micro-Nano Electronics, Shanghai Jiao Tong University, Shanghai, China*
[2]*School of Integrated Circuit Science and Engineering, East China Normal University, Shanghai, China*
*Email: zhangyh@cee.ecnu.edu.cn

***Abstract*—Advanced semiconductor lithography faces significant challenges as feature sizes continue to shrink, necessitating effective Multiple Patterning Layout Decomposition (MPLD) algorithms. Existing MPLD algorithms are inefficient or cannot support stitch insertion to achieve finer-grained optimal decomposition. This paper introduces an end-to-end GNN-based framework that not only achieves high-quality solutions quickly but also applies to layouts with stitches. Our framework treats layouts as heterogeneous graphs and performs inference through a message-passing mechanism. We deliver ultra-competitive, near-optimal solutions that are 10× faster than the exact algorithm (e.g., integer linear programming) and 3× faster than approximate algorithms (e.g., exact-cover, semi-definite programming).**

***Index Terms*—Layout decomposition, graph neural networks, message passing, graph coloring.**

I. Introduction

The continued shrinkage of semiconductor process feature sizes presents increasing challenges for lithography processes [1]. When the distance between layout features is smaller than the minimum spacing limitation, it can lead to pattern distortion or bridging due to the lithographic resolution, resulting in circuit performance degradation or short circuits [2–5] (which is termed as "conflict"). Multiple Patterning Layout Decomposition (MPLD) is a critical technique in semiconductor manufacturing, developed in response to the limitations of lithography at sub-20 nm scales [6–8]. MPLD reduces the number of potential conflicts in the layout by dividing layout features onto different masks and splitting one feature into several sub-features (which is termed as "stitch insertion"), thus improving lithography quality [9]. It is important to develop efficient MPLD algorithms with the increase in layout complexity.

MPLD can be classified based on the number of layout layers used, including Double Patterning Layout Decomposition (DPLD), Triple Patterning Layout Decomposition (TPLD), Quadruple Patterning Layout Decomposition (QPLD) [8, 10]. Different algorithms are proposed to solve MPLD problems including mathematical programming, heuristic method, graph matching, and neural networks [6, 7, 11–17]. In mathematical programming, Integer Linear Programming (ILP) [12] excels in solving DPLD [18] and small TPLD [19] problems by finding optimal solutions. However, mathematical programming-based methods face an exponential explosion in runtime with the increasing complexity of layout. Semi-definite Programming (SDP) [12, 13] extends ILP by using vector programming to map three masks to three unit vectors positioned at 120° angles in a two-dimensional plane. This approach relaxes the problem into a semi-definite programming framework, enabling the discovery of approximate solutions in polynomial time, which enhances the efficiency of solving dense TPLD problems. Boolean Satisfiability (SAT) [7] approaches, inspired by the binary nature of variables in ILP, offer shorter runtimes than ILP for TPLD problems but are still inherently NP-complete. To mitigate runtime issues, a heuristic-based Exact-Cover (EC) [14, 20] algorithm is proposed, whereas it typically yields solutions of lower quality for complex layouts compared to mathematical programming methods. To improve runtime while ensuring the quality of layout decomposition, the Graph Matching [15, 16, 21] algorithm is proposed. It performs efficient layout decomposition by matching features that have been pre-decomposed in a graph library. It lacks flexibility as it heavily relies on the richness of the graph library and does not support stitch insertion. In [17], a graph neural network (GNN)-based MPLD algorithm is proposed to quickly find near-optimal solutions by learning the conflict relationships between layout features, reducing the runtime to 0.8% compared to ILP. Similar to graph matching, it does not support stitch insertion, which limits its applicability in complex graphs where stitches could be beneficial.

To fully utilize the advantages of different MPLD algorithms in processing different layouts, [17] proposes an adaptive MPLD framework. It first determines the type of input layouts by a neural network and then assigns them to the most suitable decomposers by another neural network. These networks are all trained in a supervised manner based on numerous labeled data from layout decomposition results of ILP and EC, which is highly time-consuming. Moreover, we notice that stitch graphs (i.e., graphs that need stitches to be decomposed optimally) only constitute less than 9% of layouts, yet they account for around 70% of the total runtime (mainly consumed by ILP and EC). This highlights that stitch graphs have been the critical bottleneck of the efficiency of the MPLD algorithm. To address the aforementioned issue, we propose SEDG, a stitch-compatible end-to-end layout de-

This work is supported in part by the National Natural Science Foundation of China under Grant No. 62304133 and No. 62350610271, in part by the Open Fund of State Key Laboratory of Integrated Chips and Systems, and in part by the Shanghai Explorer Program 24TS1400200.

978-3-9826741-0-0/DATE25/© 2025 EDAA

composition framework based on graph neural network. While leveraging the GNN to efficiently solve the MPLD problem, SEDG also supports stitch insertion, thus eliminating the need for decomposer selectors and time-consuming algorithms in handling stitch graphs, thereby significantly reducing runtime. Our contributions are summarized as follows:

- We proposed a GNN-based method to process layout as heterogeneous graphs to support stitch insertion, thus achieving an end-to-end layout decomposition framework without complex decomposer selection.
- We conduct detailed experimental analysis on the framework including the impact of the number of node dimensions, number of iterations, and number of repetitions, providing directions for optimization and practical guidance to enhance the applicability and effectiveness.
- Validated on the ISCAS benchmark, we demonstrate our proposed framework excels in both solution quality and efficiency. It achieves better solutions than SDP while requiring only 26.65% of SDP's runtime. Compared to ILP and SDP, it delivers competitive solutions with significantly reduced runtimes, only 10.70% of ILP's and 31.29% of EC's.

II. Preliminaries

A. Multiple Patterning Layout Decomposition (MPLD)

1) Conflict and Stitch: In a given layout, conflicts arise when the spacing of two features on the same mask is smaller than the minimum allowable value. MPLD solves the conflict problem by assigning adjacent features to different masks. Fig. 1(a) shows an example of layout decomposition based on three masks (i.e., TPLD), where features a, b, and c are allocated to different masks (identified by different colors).

Under certain circumstances, some features cannot find mask assignments to avoid conflict. For example, feature d in Fig. 1(a) will always conflict with a, b, or c regardless of its mask assignment. To address such conflict, the concept of "stitch" has been introduced. The stitch splits a feature into several sub-features so that these sub-features can be assigned to different masks, thus avoiding conflicts. Fig. 1(b) and (c) demonstrate how introducing a stitch can resolve such unavoidable conflict. Employing stitches is typically less costly than resolving conflict in the manufacturing process.

2) Graph Format and Problem Formulations: MPLD involving k masks can be viewed as a variant of the k-coloring problem of a graph $G(V, CE, SE)$. As shown in Fig. 1, V is a node set representing layout features. CE is an edge set representing conflicts, determined by the layout's topology and the minimum allowable spacing. Before decomposition, numerous stitch candidates are generated, forming the edge set SE. The problem formulations are as follows:

$$\min \sum_{uv \in CE} C_{uv} + \alpha \sum_{uv \in SE} S_{uv}, \tag{1a}$$

$$C_{uv} = (\text{Color}(u) == \text{Color}(v)), \tag{1b}$$

$$S_{uv} = (\text{Color}(u) \neq \text{Color}(v)), \tag{1c}$$

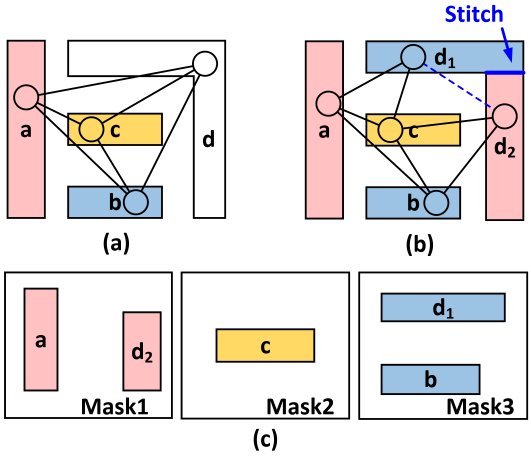

Fig. 1: An example of Triple Pattern Layout Decomposition. (a) A layout graph with only conflict edges. Conflict always exists regardless of the mask feature d assigned. (b) Introduction of stitch to alleviate conflicts. (c) Decomposition of a single layer into three masks.

where α represents the relative cost of stitch compared with conflict. Here, u and v are nodes within V, and $Color(u)$ denotes the mask or color assigned to node u.

3) OpenMPL: OpenMPL [10] is a robust open-source framework designed to address the complex challenges of MPLD. It seamlessly integrates a suite of optimization algorithms, including ILP, SDP, and EC. The process begins with the input layout undergoing graph simplification and stitch insertion, managed by OpenMPL. This preprocessing allows the framework to effectively decompose a large layout into manageable smaller components. Each component is then individually optimized using the integrated algorithms and subsequently reassembled to form a coherent, layered layout. This methodical approach facilitates efficient processing and enhanced accuracy in layout decomposition.

B. Graph Neural Network

Graph Neural Networks [22] are a powerful class of neural networks designed specifically for processing data represented as graphs. These networks are particularly effective in capturing the dependencies within structured data through their nodes and edges, allowing them to handle complex relational information. The fundamental principle of GNNs is to aggregate features from a node's neighbors through a process known as message passing [23]. Each node in the graph updates its state iteratively by combining its features with aggregated features from its neighbors. Long Short-Term Memory (LSTM) [24] networks are a specialized type of recurrent neural network (RNN) [25] designed to address the shortage of learning long-range dependencies within sequence data. LSTM has been widely used in GNNs as iterations can also be considered as time sequence data. This architecture enables GNNs to learn node-level, edge-level, and graph-level representations, making them versatile for a wide range

of applications, including combinatorial optimization [26], recommendation systems [27], and bioinformatics [28].

III. Our Proposed Framework

A. Overview

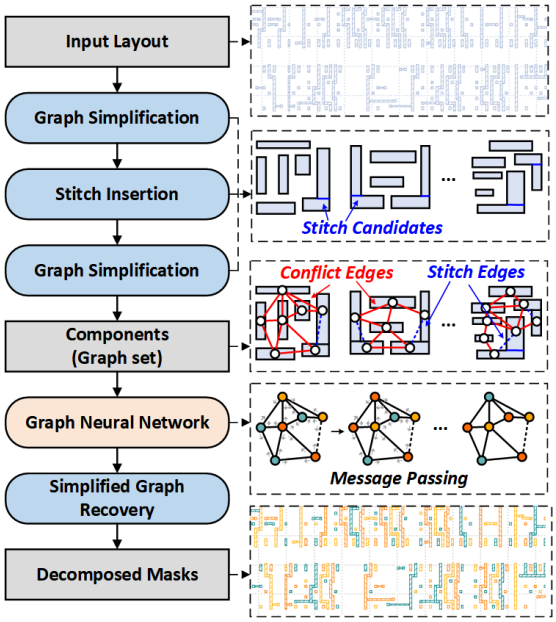

Fig. 2: Overview of our proposed framework.

Fig. 2 illustrates our proposed framework. Initially, the input layout is preprocessed by graph simplification algorithms including independent component computation (ICC) [12], iterative vertex removal (IVR) [12]. After the first simplification, stitch candidates are inserted [12]. Note that some stitch candidates help to decrease the number of conflicts, while others will be redundant. Another graph simplification is performed to remove part of potential redundant stitches using stitch redundancy removal (SRR) [10]. This preparatory step is essential before the neural network can perform color mapping on the graphs. Subsequently, graphs are processed by the graph neural network, with each graph optimized individually and all graphs computed in parallel as batches. After several rounds of message passing, every node in the graphs is mapped to a certain color, i.e., every feature in the layout is assigned to a specific mask. A detailed illustration of GNN is in Section III-B. Finally, the simplified graphs are recovered to the decomposed layouts, which can be regarded as the reverse process of graph simplifications. To fairly evaluate the proposed GNN-based framework, graph simplification, stitch insertion, and simplified graph recovery are implemented by OpenMPL.

B. GNN Structure

Unlike existing GNN decomposer [17], which handles layouts as homogeneous graphs with only conflict edges, our proposed GNN treats layouts as heterogeneous graphs with both conflict and stitch edges and features a more detailed design for message passing. Fig. 3 illustrates the structure of our GNN and the message passing mechanism including three phases.

1) Node Embedding: We initiate our process by randomly assigning "color beliefs" to each node. The color belief for each node is a vector of length 3 that represents the probability of the node being assigned each color. The color beliefs are then transformed into node features through a linear embedding layer (lines 2, 3 in Algorithm 1). We use a vector with a dimension of d for each node.

2) Message Passing: The proposed GNN is based on a Message Passing Neural Network (MPNN). It involves two primary processes: Aggregation (i.e., message generation process, lines 5-7 in Algorithm 1) and Combination (i.e., node features update process, line 8 in Algorithm 1).

During the aggregation process, each node processes its features via simple linear layers to generate messages, which are then transmitted to neighboring nodes. Messages from conflict and stitch neighbors are summed separately according to the following formulas:

$$\text{msg}_c^{(i)} = M_c \cdot F_V^{(i)} \cdot W_c, \tag{2a}$$

$$\text{msg}_s^{(i)} = M_s \cdot F_V^{(i)} \cdot W_s, \tag{2b}$$

$$\text{msg}^{(i)} = \text{concat}(\text{msg}_c^{(i)}, \text{msg}_s^{(i)}) \cdot W_{\text{msg}}, \tag{2c}$$

where i is the round of message passing. M_c and M_s are $n \times n$ adjacency matrices representing conflict and stitch relations respectively, n is the number of nodes in V, $F_V^{(i)}$ is the $n \times d$ matrix of features for all nodes in V, d is the dimension of the features of each node. W_c and W_s are $d \times d$ trainable weight matrices for conflict and stitch messages respectively. Afterward, the concatenated messages $\text{msg}_c^{(i)}$ and $\text{msg}_s^{(i)}$ are merged and transformed through a trainable weight matrix W_{msg}, which is a $2d \times d$ matrix, to form the final messages $\text{msg}^{(i)}$. This step effectively integrates the information from both types of neighbors, enhancing the model's ability to capture complex interactions between nodes.

In the combination process, each node receives these messages and updates its features using an LSTM layer, which processes the incoming messages as input and updates node features as the hidden state. To ensure each node has full awareness of the entire graph, we conduct several rounds of message passing (line 4 in Algorithm 1).

3) Node Decoding: After all rounds, the node features are decoded into color beliefs through a linear layer (line 10 in Algorithm 1). The linear layer has a dimension $d \times 3$, allowing each node to obtain a color belief vector of length 3. The color of the node is assigned based on the value corresponding to the highest color belief in the vector.

C. Repetitions

To prevent the neural network from converging to a local optimum during the coloring process, which is relevant to the coloring initialization [29], we repeat the message passing in GNN multiple times. Each repetition features an independent

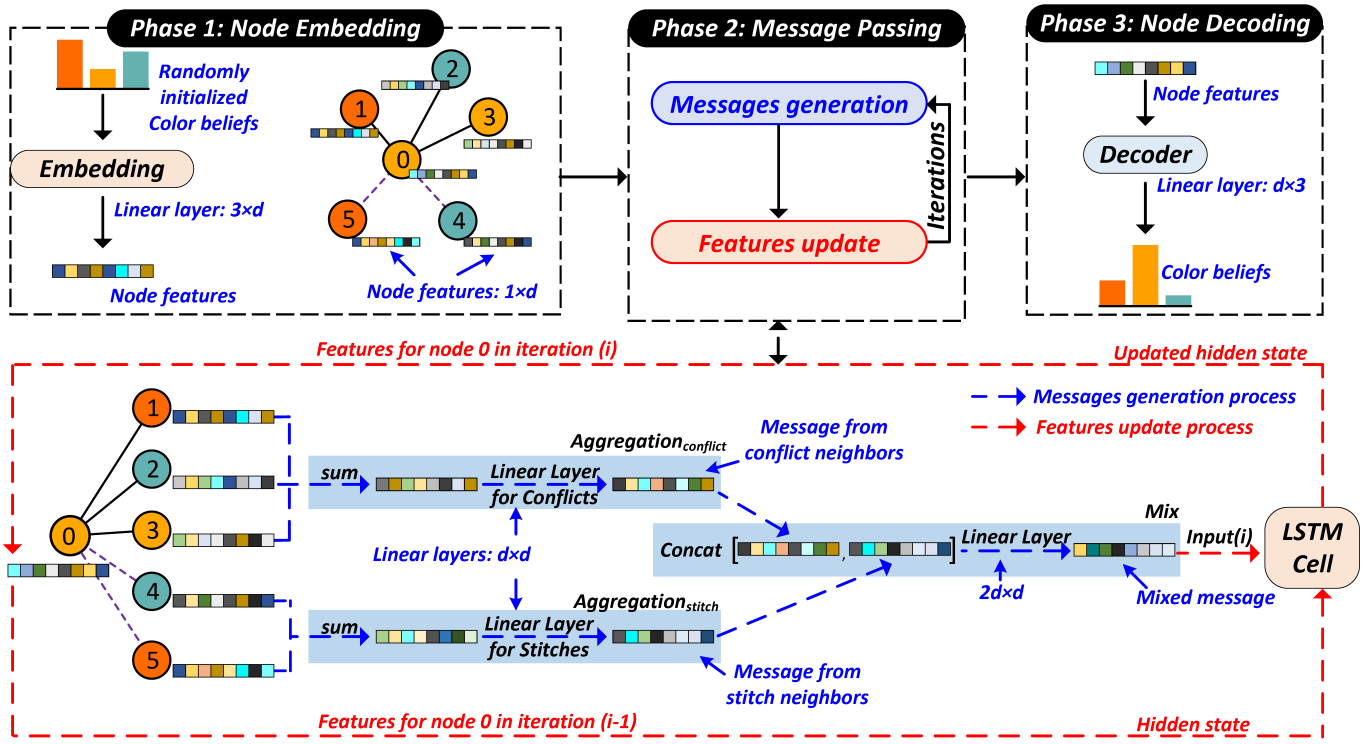

Fig. 3: GNN structure and detailed illustrations of the message passing mechanism. Features of each node are updated through several rounds of iterations and then decoded into color beliefs. Each iteration contains a message generation process and a features update process. Each node acts as both a message generator and a receiver. Node 0 is selected to illustrate the process.

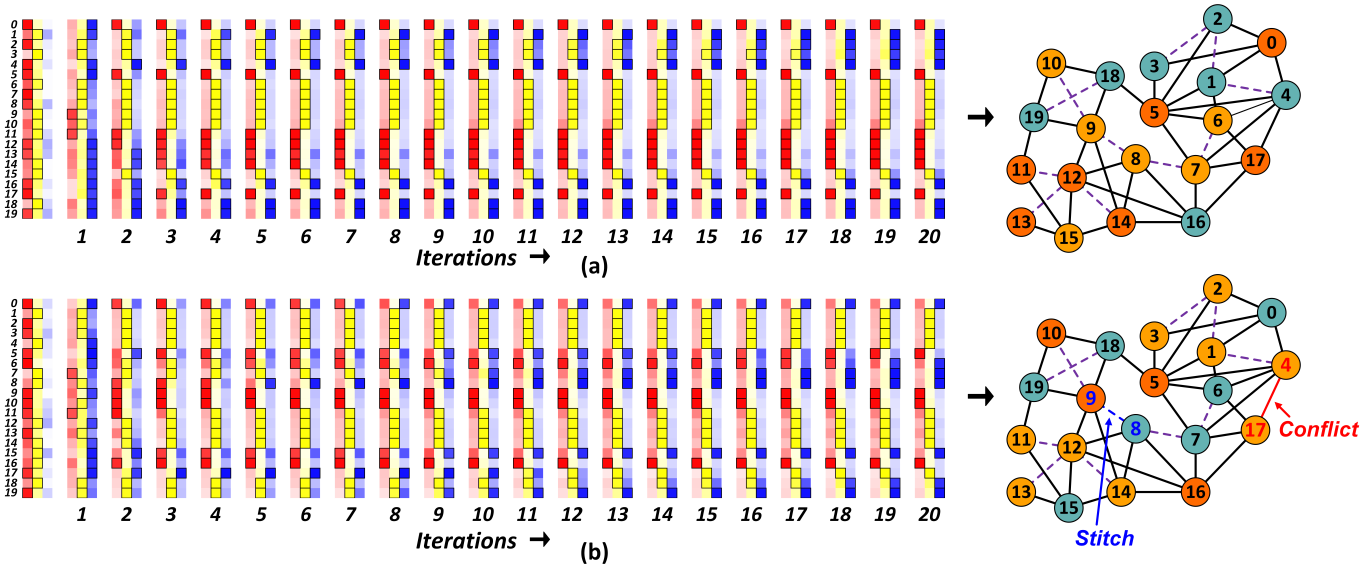

Fig. 4: Changes in the color beliefs for each node with iterations, starting from random initializations. (a) and (b) represent two independent repetitions over 20 iterations. Due to variance in initialization, variations in the final costs are observed.

random initialization, aiming to explore the solution space more thoroughly and to seek a global optimum.

As illustrated in Fig. 4, for a specific component of the layout, our neural network was run twice (i.e., 2 repetitions), with each node's color confidence initialized independently and randomly. In Fig. 4(a), after 20 iterations, the coloring scheme converged to an optimal solution with a cost of 0. In Fig. 4(b), despite 20 iterations, there is a stitch between nodes 8 and 9 and a conflict between nodes 4 and 17, which could not be optimized by minor local adjustments,

indicating that the model converged to a local optimum under the initialization conditions of Fig. 4(b). This demonstrates the need for multiple repetitions to potentially achieve higher-quality solutions.

Algorithm 1 Our Proposed GNN-based Layout Decomposer

Require: M_c, $M_s \leftarrow$ Adjacency matrices. M_c, M_s represent conflicts and stitches respectively;
Require: $iter \leftarrow$ Number of iterations;
Require: $rpt \leftarrow$ Number of repetitions;
Require: $Emb \leftarrow$ Trained layer for node embedding;
Require: Agg_c, $Agg_s \leftarrow$ Layers for generating messages;
Require: $Mix \leftarrow$ Layer to mix conflict and stitch messages;
Require: $Com \leftarrow$ LSTM layer for updating node features;
Require: $Decoder \leftarrow$ Layer for decoding node features into color-beliefs;
Ensure: $C_b \rightarrow$ Color-beliefs for each node in graph;
Ensure: $C \rightarrow$ Colors scheme for graph;

```
for i ∈ {1, ..., rpt} do
    C_b ← Random initialization for each node;
    E_v ← Features for each node by Emb(C_b);
    for j ∈ {1, ..., iter} do
        msg_c ← Agg_c(E_v, M_c);
        msg_s ← Agg_s(E_v, M_s);
        msg ← Mix(msg_c, msg_s);
        E_v ← Com(msg, E_v);
    end for
    C_b ← Decoder(E_v);
    C_i ← Choose colors with highest beliefs in C_b;
end for
return the best scheme in {C_1, ..., C_rpt};
```

D. Loss Function

The loss function we employ is the contrastive loss [30], formulated as follows:

$$\sum_{\{uv\}\in CE} \mathrm{ReLU}^2(m_1 - D_{\mathrm{uv}}) + \alpha \sum_{\{uv\}\in SE} \mathrm{ReLU}^2(D_{\mathrm{uv}} - m_2) \tag{3}$$

where D_{uv} represents the Euclidean distance between the color beliefs of nodes u and v. The margins m_1 and m_2 are set for conflict and stitch edges, respectively. We have chosen the values 1 and 0 for these margins, based on the principle that the color belief distances should be greater between nodes connected by conflict edges and closer between nodes connected by stitch edges. The parameter α is a constant, maintained at the same value as in the problem formulation of MPLD.

IV. Result and Discussion

A. Experiment Setup

Our proposed framework is implemented using PyTorch [31] and the OpenMPL framework. We utilize the ISCAS benchmark [12], a standard widely employed in related studies [7, 10, 12, 14, 17]. The minimum distance is set to 120 nm for the initial ten cases and 100 nm for the subsequent five cases. Although the ISCAS benchmark comprises only 15 cases, each case can be divided into multiple independent graphs, resulting in a total of more than 6,500 graphs under our specific minimum distance settings. For our experiments, 256 graphs are randomly selected as the training set, 64 graphs are selected as the validation set, and tests are then conducted on the benchmark. The training dataset comprises less than 4% of the total number of graphs in the benchmark, which ensures the generalization capability of our model. All cases are configured with three colors, a simplicity level [10] of 3, and a stitch relative cost α of 0.1. The experiments are conducted on a Linux machine equipped with an Intel Core processor at 2.5 GHz and an NVIDIA 2080Ti GPU.

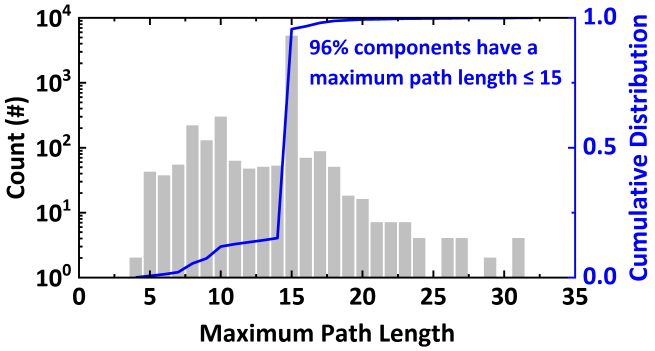

Fig. 5: Distribution of maximum path lengths in ISCAS benchmark.

B. Analysis Model Hyper-parameter

1) Model and Training Hyper-parameters: The number of rounds for message passing is fixed at 20, as we observed that the lengths of the longest paths in each graph typically range between ten and twenty, as demonstrated in Fig. 5. Twenty iterations ensure that messages pass from one end to the other end in long paths. Node features are dimensioned at 32 because among the four dimensions tested (16, 32, 64, and 128), 32 performed significantly better than 16. Using dimensions of 64 or 128 did not show significant improvement over 32 on the benchmark, while it increased the number of parameters and memory cost, as shown in Table III.

2) Inference Hyper-parameters: Fig. 6 shows the performance of the trained model on the ISCAS benchmark under different repetitions and iterations. In Fig. 6(a), the number of iterations for the model is fixed at 20, with repetitions ranging from 5 to 30. It is observed that our model significantly outperforms SDP in both cost and runtime with few repetitions. As the number of repetitions increases, the model's cost-runtime curve surpasses EC, and the cost approaches that of ILP in a very short time. In Fig. 6(b), repetitions are fixed at 10, while the number of iterations ranges from 10 to 30, showing that the cost converges after the number of iterations exceeds 20. To balance cost and runtime, we use 10 repetitions and 20 iterations for evaluations on the ISCAS benchmark.

TABLE I: Decomposition Cost and Runtime Comparison in ISCAS Benchmark

Circuit	ILP				SDP				EC				Ours			
	st#	cn#	Cost	RT (s)	st#	cn#	Cost	RT (s)	st#	cn#	Cost	RT (s)	st#	cn#	Cost	RT (s)
c432	4	0	0.4	0.126	4	0	0.4	0.013	4	0	0.4	0.028	4	0	0.4	0.141
c499	0	0	0.0	0.020	0	0	0.0	0.008	0	0	0.0	0.039	0	0	0.0	0.141
c880	7	0	0.7	0.126	8	0	0.8	0.013	7	0	0.7	0.041	7	0	0.7	0.136
c1355	3	0	0.3	0.093	3	0	0.3	0.012	3	0	0.3	0.057	3	0	0.3	0.141
c1908	1	0	0.1	0.045	1	0	0.1	0.010	1	0	0.1	0.082	1	0	0.1	0.145
c2670	6	0	0.6	0.109	6	0	0.6	0.022	6	0	0.6	0.143	10	0	1.0	0.145
c3540	8	1	1.8	0.180	8	1	1.8	0.039	8	1	1.8	0.164	10	1	2.0	0.142
c5315	9	0	0.9	0.156	9	0	0.9	0.034	9	0	0.9	0.199	11	0	1.1	0.142
c6288	204	1	21.4	5.465	203	7	27.3	0.429	204	1	21.4	0.495	212	1	22.2	0.225
c7552	23	0	2.3	0.473	23	0	2.3	0.073	21	1	3.1	0.308	25	0	2.5	0.169
s1488	2	0	0.2	0.055	2	0	0.2	0.017	2	0	0.2	0.074	2	0	0.2	0.138
s38417	54	19	24.4	2.712	46	27	31.6	1.170	54	19	24.4	1.004	57	19	24.7	0.224
s35932	40	44	48.0	7.191	20	64	66.0	3.589	48	44	48.8	2.609	52	44	49.2	0.445
s38584	116	36	47.6	7.293	105	48	58.5	3.332	117	36	47.7	2.627	120	36	48.0	0.463
s15850	97	34	43.7	6.227	83	48	56.3	2.961	100	34	44.0	2.495	106	34	44.6	0.446
Average			12.83	2.018			16.47	0.781			12.96	0.691			13.13	0.216
Ratio			1.00	1.000			1.28	0.387			1.01	0.342			1.02	0.107

TABLE II: ISCAS Benchmark Analysis and Cost Comparison on Stitch Graphs

Circuit	$\|G\|$	$\|s.G\|$	s.Ratio	ILP		Ours	
				st#	cn#	st#	cn#
c432	4	4	100.00%	4	0	4	0
c499	4	0	0.00%	0	0	0	0
c880	7	7	100.00%	7	0	7	0
c1355	4	3	75.00%	3	0	3	0
c1908	3	1	33.33%	1	0	1	0
c2670	9	6	66.67%	6	0	6	0
c3540	14	8	57.14%	8	0	8	0
c5315	15	9	60.00%	9	0	9	0
c6288	192	175	91.15%	204	0	209	0
c7552	37	22	59.46%	23	0	23	0
s1488	8	2	25.00%	2	0	2	0
s38417	663	54	8.14%	54	0	54	0
s35932	1993	40	2.01%	40	0	43	0
s38584	1916	116	6.05%	116	0	116	0
s15850	1641	97	5.91%	97	0	97	0
Average	434	36.27		38.3	0	38.8	0

TABLE III: Comparison of Different Node Feature Dimensions

Dim	Cost	Parameters#	Inference Memory
16	1.04×	3363	0.80×
32	**1.00×**	**12867**	**1.00×**
64	1.00×	50307	1.76×
128	1.00×	198915	2.54×

C. Comparison With Other State-of-the-Art Methods

Table I compares our proposed framework with other methods. The solution quality of our proposed framework on the ISCAS benchmark surpasses that of SDP and is comparable to EC. While maintaining high-quality solutions, our method significantly outperforms existing algorithms in efficiency, reducing runtime to 10.70% of ILP, 26.65% of SDP, and 31.29% of EC, respectively.

We also analyzed each layout in the ISCAS benchmark in Table II, where G represents the set of graphs after graph simplification and stitch insertion, and $s.G$ is a subset of G that only includes graphs with stitches in the optimal coloring scheme, namely stitch graphs. We specifically extracted these stitch graphs, which are the graph sets that the traditional GNN decomposer cannot handle, and tested them with our proposed framework. The results are shown in Table II. First, we noticed that the composition of graph sets varies across different layouts, e.g., in the layout s35932, only 40 out of 1993 graphs contain stitches in the final coloring scheme, whereas, in the layout c6288, 172 out of 192 graphs ultimately contain stitches. Secondly, we found that our proposed framework achieved optimal solutions in most layouts except for s35932 and c6288.

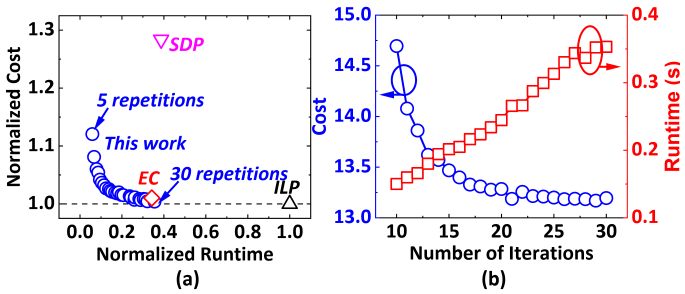

Fig. 6: Relationship between runtime and cost with different (a) number of repetitions and (b) number of iterations.

V. Conclusion

We introduced an end-to-end GNN-based framework for MPLD that supports stitch insertion, effectively addressing limitations in existing methods. By modeling layouts as heterogeneous graphs with conflict and stitch edges, our approach leverages message passing to efficiently compute high-quality decomposition solutions. Experiments on the ISCAS benchmark show that our method achieves solution quality comparable to algorithms like ILP and EC while significantly reducing runtime, requiring only a fraction of their computational time.

REFERENCES

[1] B. Yu and D. Z. Pan, *Design for manufacturability with advanced lithography*. Springer, 2016.

[2] W. Lu, Y. Zhang, Q. Zhang, X. Zhang, and Y. Li, "Litho-neuralODE: improving hotspot detection accuracy with advanced data augmentation and neural ordinary differential equations," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 387–392.

[3] Q. Zhang, Y. Zhang, J. Li, W. Lu, and Y. Li, "Litho-neuralODE 2.0: improving hotspot detection accuracy with advanced data augmentation, DCT-based features, and neural ordinary differential equations," *Integration*, vol. 85, pp. 10–19, 2022.

[4] Q. Zhang, Y. Zhang, W. Lu, H. Huang, Z. Zhong, C. Zhou, and Y. Li, "Litho-asymvnet: super-resolution lithography modeling with an asymmetric v-net architecture," *Science China Information Sciences*, vol. 66, no. 12, p. 229406, 2023.

[5] Y. Zhang, G. He, F. Zhang, Y. Li, and G. Wang, "The study of lithographic variation in resistive random access memory," *Journal of Semiconductors*, vol. 45, no. 5, p. 052303, 2024.

[6] D. Z. Pan, L. Liebmann, B. Yu, X. Xu, and Y. Lin, "Pushing multiple patterning in sub-10nm: are we ready?" in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[7] H. Liu, P. Liao, M. Zou, B. Pang, X. Li, M. Yuan, T.-Y. Ho, and B. Yu, "Layout decomposition via boolean satisfiability," in *ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[8] B. Yu and D. Z. Pan, "Layout decomposition for quadruple patterning lithography and beyond," in *Proceedings of the Annual Design Automation Conference*, 2014, p. 1–6.

[9] K. Yuan, J.-S. Yang, and D. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," in *Proceedings of International Symposium on Physical Design*, 2009, p. 107–114.

[10] W. Li, Y. Ma, Q. Sun, L. Zhang, Y. Lin, I. H.-R. Jiang, B. Yu, and D. Z. Pan, "OpenMPL: An open-source layout decomposer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2331–2344, 2021.

[11] Y. Ma, X. Zeng, and B. Yu, "Methodologies for layout decomposition and mask optimization: a systematic review," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017, pp. 1–6.

[12] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 1–8.

[13] T. Matsui, Y. Kohira, C. Kodama, and A. Takahashi, "Positive semidefinite relaxation and approximation algorithm for triple patterning lithography," in *Algorithms and Computation*. Springer International Publishing, 2014, pp. 365–375.

[14] I. H.-R. Jiang and H.-Y. Chang, "Multiple patterning layout decomposition considering complex coloring rules and density balancing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 12, pp. 2080–2092, 2017.

[15] B. Yu, D. Z. Pan, T. Matsunawa, and X. Zeng, "Machine learning and pattern matching in physical design," in *Asia and South Pacific Design Automation Conference*, 2015, pp. 286–293.

[16] J. Kuang and E. F. Y. Young, "An efficient layout decomposition approach for triple patterning lithography," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[17] W. Li, Y. Ma, Y. Lin, and B. Yu, "Adaptive layout decomposition with graph embedding neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 5030–5042, 2022.

[18] Y. Xu and C. Chu, "GREMA: graph reduction based efficient mask assignment for double patterning technology," in *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, 2009, pp. 601–606.

[19] B. Yu, Y.-H. Lin, G. Luk-Pat, D. Ding, K. Lucas, and D. Z. Pan, "A high-performance triple patterning layout decomposer with balanced density," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 163–169.

[20] H.-Y. Chang and I. H.-R. Jiang, "Multiple patterning layout decomposition considering complex coloring rules," in *Proceedings of Annual Design Automation Conference*, 2016, pp. 1–6.

[21] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres, "Inexact graph matching by means of estimation of distribution algorithms," *Pattern Recognition*, vol. 35, no. 12, pp. 2867–2880, 2002.

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.

[23] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Message passing neural networks," *Machine Learning Meets Quantum Physics*, pp. 199–214, 2020.

[24] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[25] S. Grossberg, "Recurrent neural networks," *Scholarpedia*, vol. 8, no. 2, p. 1888, 2013.

[26] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 130, pp. 1–61, 2023.

[27] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[28] X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang, "Graph neural networks and their current applications in bioinformatics," *Frontiers in Genetics*, vol. 12, p. 690049, 2021.

[29] D. Gamarnik and M. Sudan, "Limits of local algorithms over sparse random graphs," in *Proceedings of the Conference on Innovations in Theoretical Computer Science*, 2014, p. 369–376.

[30] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 1735–1742.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: an imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.