Versatile Multi-stage Graph Neural Network for Circuit Representation (Appendix)

Shuwen Yang
School of Intelligence Science and Technology
Peking University
swyang@pku.edu.cn

Zhihao Yang
School of Software and Microelectronics
Peking University
zhihaoyang@stu.pku.edu.cn

Dong Li
Huawei Noah's Ark Lab
lidong106@huawei.com

Yingxue Zhang
Huawei Noah's Ark Lab
yingxue.zhang@huawei.com

Zhanguang Zhang
Huawei Noah's Ark Lab
zhanguang.zhang@huawei.com

Guojie Song*
School of Intelligence Science and Technology
Peking University
gjsong@pku.edu.cn

Jianye HAO*
Huawei Noah's Ark Lab
haojianye@huawei.com

A Featurization

The featurization of *grid*, *cell*, *net* and *pin* is shown in Tab.1/2/3/4.

Table 1: *Grid* Features

Name	Type	Description
net density	$\mathbb{R}^2$	the density of *nets* in each *grid* (horizontally and vertically) [1]
pin density	$\mathbb{R}$	the density of *pins* in each *grid* [1]
node density	$\mathbb{R}$	the density of *cells* in each *grid* [1]

Table 2: *Cell* Features

Name	Type	Description
size	$\mathbb{R}^2$	width & height of the *cell*
degree	$\mathbb{N}$	# of *nets* connected with the *cell*

When conducting experiments on circuits in placement stage, grid features in Tab.1 are borrowed to supply node features by assigning each node with the features of the grid it's located in.

*Corresponding Author

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Table 3: *Net* Features

Name	Type	Description
degree	$\mathbb{N}$	# of *cells* connected with the *net*
span	$\mathbb{N}^2$	# of *grids* the *net* covers horizontally and vertically [2]

Note that the feature **span** is only available in placement stage.

Table 4: *Pin* Features

Name	Type	Description
pin offset	$\mathbb{R}^2$	the offset position of the *cell* (horizontally and vertically)
signal direction	0/1	this *pin* input/output signal to the *cell*

B Graphization of Circuit Design

B.1 Graphize Geometrical Information

For circuit designs in placement stage, a universal solution of graphizing geometry is to link the *cell*-pairs whose distances are lower than a threshold δ [3]:

$$\hat{\mathcal{E}}_G = \{(i,j)|i,j \in \mathcal{V} \wedge \sqrt{(\widetilde{\boldsymbol{p}}_x[i] - \widetilde{\boldsymbol{p}}_x[j])^2 + (\widetilde{\boldsymbol{p}}_y[i] - \widetilde{\boldsymbol{p}}_y[j])^2} < \delta\}$$
$$\widetilde{\boldsymbol{p}}_x = \boldsymbol{p}_x + \boldsymbol{s}_x/2 \qquad \widetilde{\boldsymbol{p}}_y = \boldsymbol{p}_y + \boldsymbol{s}_y/2 \tag{1}$$

where $(\widetilde{\boldsymbol{p}}_x[i], \widetilde{\boldsymbol{p}}_y[i])$ is the position of i-th *cell*'s center, and $\boldsymbol{s}_x, \boldsymbol{s}_y$ are the width/height of *cells*. However, the time cost of this solution is $O(|\mathcal{V}|^2)$ and makes it unpractical for very large circuits with millions of *cells* and *nets*.

To accelerate the geometry graphization, we cut the circuit into windows with size (w_x, w_y), scatter the *cells* into the windows and link the *cells* appearing in the same window, as it shows in Fig.1. Note that every *cell* has its 2D structure and may appear in multiple windows e.g. *cell* 4 in Fig.1(b).

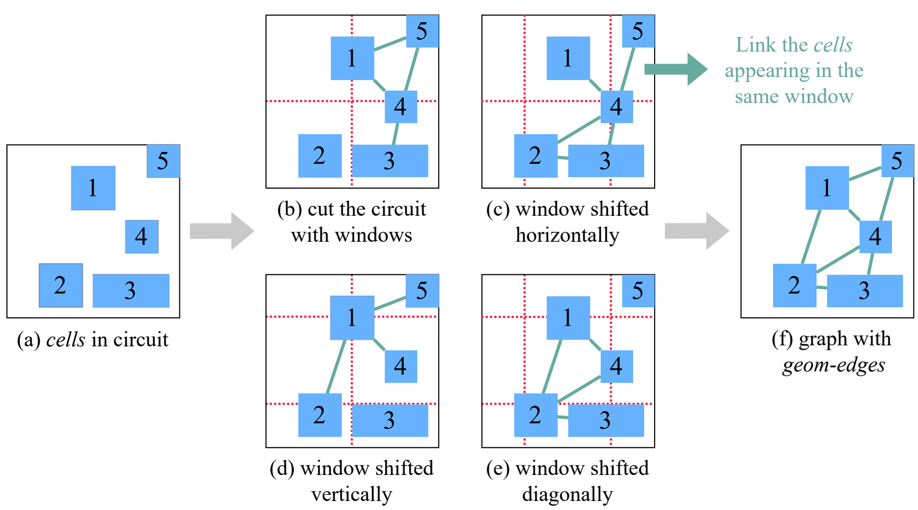

Figure 1: Link the adjacent *cells* by scattering them into shift windows.

However, it is still possible that two adjacent *cells* are incidentally split into different windows and can't get linked e.g. *cell* 2 and 3 in Fig.1(b). To avoid this, we shift the windows by $w_x/2$ horizontally or $w_y/2$ vertically and add new links (see Fig.1(c)(d)(e)). Finally, as it shows in Fig.1(f), we get *geom-edges* $\mathcal{E}_G$:

$$\mathcal{E}_G = \{(i,j)|i,j \in \mathcal{V} \wedge i,j \text{ appear in the same window}\} \quad (2)$$

$\mathcal{E}_G$ is actually a sampling of the *cell*-pairs with distance lower than $\delta = \sqrt{w_x^2 + w_y^2}$, and the possibility of being sampled is inversely proportional to their distance. However, the time consumption of constructing $\mathcal{E}_G$ is still $O(|\mathcal{V}|^2/|\mathcal{W}|)^2$. To further lower the consumption, for every *cell* i, we only link it to at most c others appearing in the same window and produce an $\mathcal{E}_G^* \subseteq \mathcal{E}_G$ to substitute $\mathcal{E}_G$. As c is a constant, the time cost of constructing $\mathcal{E}_G^*$ is $O(|\mathcal{V}|)$ (see proof in Appendix.B.2). To further enhance the information in *geom-edges*, we store the *cell*-pair distances as raw features in $\mathcal{E}_G$.

To show the robustness of our geometry graphization method, we test the sensitivity of window size (w_x, w_y) and link capacity c in Sec.C

B.2 Time Consumption

Inside a circuit design, despite of several "macros" occupying the margins, most *cells* are not big enough to appear in multiple windows in Fig.1, and they are normally distributed to the windows to avoid placement conflicts [4]. Therefore, we assume that the average number of *cells* appear in a window is $|\mathcal{V}|/|\mathcal{W}|$. The time consumption of constructing $\mathcal{E}_G^*$ is $4|\mathcal{W}| \cdot (\frac{|\mathcal{V}|}{|\mathcal{W}|} \cdot c) = 4c|\mathcal{V}| = O(|\mathcal{V}|)$.

Moreover, the time costs of constructing $\mathcal{V}, \mathcal{U}, \mathcal{E}_T$ are $O(|\mathcal{V}|), O(|\mathcal{U}|), O(|\mathcal{P}|)$ respectively, so we can produce circuit graph $\mathcal{G}$ in $O(|\mathcal{V}| + |\mathcal{U}| + |\mathcal{P}|)$.

C Parameter Sensitivity Experiment

Table 5: Congestion prediction result in placement stage under different window size (w_x, w_y)

(w_x, w_y)	Time (s/epoch)	Cell-level			Grid-level		
		pearson	spearman	kendall	pearson	spearman	kendall
(8,10)	28.46	**0.889**	0.721	0.584	0.692	0.742	0.549
(16,20)	27.85	**0.889**	**0.729**	**0.591**	0.694	0.740	0.547
(32,40)	27.07	0.887	0.714	0.575	0.697	**0.770**	**0.577**
(64,80)	23.96	0.888	0.717	0.578	**0.698**	0.762	0.570
(128,160)	25.06	0.888	0.724	0.584	0.694	0.758	0.564

Table 6: Congestion prediction result in placement stage under different link capacity c

c	Time (s/epoch)	Cell-level			Grid-level		
		pearson	spearman	kendall	pearson	spearman	kendall
2	23.97	**0.889**	0.724	0.586	0.695	0.742	0.549
5	27.07	0.887	0.714	0.575	**0.697**	**0.770**	**0.577**
10	29.15	0.881	0.707	0.569	0.694	0.762	0.566
20	30.96	**0.889**	**0.725**	**0.587**	**0.697**	0.745	0.552

We test the sensitivity of window size (w_x, w_y) and link capacity c, as it shows in Tab.5 and Tab.6. The results indicate that the choice of these hyper-parameters has little influence on the model's function, so the robustness of Circuit GNN is claimed.

[2] $|\mathcal{W}|$ is the average # of windows after splitting the circuit.

D Model Sensitivity Experiment

D.1 Message Function

The choice of $\Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_T}\mathcal{U}}$ **and** $\Phi_{msg}^{\mathcal{U}\xrightarrow{\mathcal{E}_T}\mathcal{V}}$ On the one hand, there are usually more geom-edges than topo-edges in Circuit Graph (3.4M geom-edges and 1.9M topo-edges in *superblue19*), so we use edge-weight summation rather than inner product, which is $F_{\mathcal{U}}$ (hidden dimension of net) times more expansive in computation. On the other hand, it is also reasonable for geom-edges to use edge-weight summation because geometrically closer cells have a stronger relationship. Still, we test the performance when topo-edges use edge-weight summation (**topo. e.**) or geom-edges use inner product (**geom. i.**):

Table 7: Result of exchanging topological and geometrical message functions.

Baseline	Time (s/epoch)	Node-level			Grid-level		
		pearson	spearman	kendall	pearson	spearman	kendall
topo. e.	25.35	0.886	0.707	0.570	0.694	0.743	0.552
geom. i.	37.71	0.886	**0.717**	**0.579**	0.689	0.734	0.542
Ours	27.07	**0.887**	0.714	0.575	**0.697**	**0.770**	**0.577**

D.2 Information Fusing Strategy

We hope to keep most of the informative values when fusing the topological and geometrical information, while sum-pooling and mean-pooling may revise them. Concatenation is not considered because we hope to keep the same hidden dimension in each layer. The results below show that using sum-pooling and mean-pooling has worse spearman & kendall (Grid-level) and only marginal improvement in other metrics:

Table 8: Result of different information fusing functions.

Baseline	Time (s/epoch)	Node-level			Grid-level		
		pearson	spearman	kendall	pearson	spearman	kendall
Ours (sum pool)	29.00	0.887	**0.717**	**0.580**	**0.699**	0.756	0.564
Ours (mean pool)	29.38	**0.888**	0.715	0.577	0.697	0.755	0.563
Ours	27.07	0.887	0.714	0.575	0.697	**0.770**	**0.577**

D.3 Readout Representation

We concatenate the raw features to enrich the representations. The results below show that excluding raw features only causes a marginal performance drop:

Table 9: Result of removing raw features in readout representation.

Baseline	Time (s/epoch)	Node-level			Grid-level		
		pearson	spearman	kendall	pearson	spearman	kendall
Ours (w/o. raw feat.)	27.45	**0.892**	0.713	0.574	0.697	0.759	0.567
Ours	27.07	0.887	**0.714**	**0.575**	**0.697**	**0.770**	**0.577**

D.4 Position Encoding

Directly encoding the cell positions as features leads to very bad generalization because raw 3D positions do not satisfy translation and rotation invariances[5]. Here are the results: (**Ours (pos.**

encode) is the modification we made which encodes the cell positions into node features instead of using *geom-edges*.)

Table 10: Result of using direct position encoding and *geom-edges*.

Baseline	Time (s/epoch)	Node-level			Grid-level		
		pearson	spearman	kendall	pearson	spearman	kendall
GAT	13.90	0.777	0.267	0.200	0.215	0.399	0.280
GAT (pos. encode)	16.21	0.777	0.263	0.197	0.210	0.397	0.279
Ours (w/o. geom.)	21.62	0.779	0.289	0.217	0.315	0.468	0.329
Ours (pos. encode)	22.55	0.766	0.328	0.292	0.228	0.475	0.411
Ours	27.07	**0.887**	**0.714**	**0.575**	**0.697**	**0.770**	**0.577**

E Additional Experiment Tables

Table 11: Net wirelength prediction in logic synthesis stage (↓ means "lower is better")

Baseline	Time (s/epcoh)	pearson	spearman	kendall	MAE↓	RMSE↓
MLP	2.16	0.150	0.192	0.096	0.633	0.854
Net$^{\mathbf{2f}}$	15.29	0.225	0.362	0.248	**0.606**	0.830
Net$^{\mathbf{2a}}$	16.75	0.172	0.227	0.153	0.614	**0.821**
Ours (w/o. geom.)	15.66	**0.484**	**0.547**	**0.418**	0.619	**0.821**

Table 12: Congestion prediction result in placement stage (in precision, recall and F1-score)

Baseline	Time (s/epoch)	Cell-level			Grid-level		
		precision	recall	F1-score	precision	recall	F1-score
GAT (w. geom.)	16.21	0.718	**1.000**	0.836	0.669	**1.000**	0.802
pix2pix	4.46	-	-	-	0.695	0.996	0.814
LHNN	305.47	-	-	-	0.807	0.907	0.855
Ours (w/o. topo.)	21.54	0.876	0.899	0.879	0.865	0.851	0.860
Ours	27.07	**0.884**	0.900	**0.892**	**0.887**	0.857	**0.872**

F Number of Parameters and Inference Time

The # of parameters and inference time on *superblue19* are listed below:

Table 13: # of parameters and inference time on congestion prediction in logic synthesis stage.

Model	GCN	GraphSAGE	GAT	CongestionNet	Ours (w/o geom.)
# parameter	205K	204K	205K	280K	426K
Inference Time (s)	2.74	2.69	3.18	2.99	3.09

Table 14: # of parameters and inference time on congestion prediction in placement stage.

Model	pix2pix	LHNN	Ours
# parameter	992K	54K	480K
Inference Time (s)	0.35	65.21	4.09

Table 15: # of parameters and inference time on wirelength prediction in logic synthesis stage.

Model	MLP	Net$^{\mathbf{2f}}$	Net$^{\mathbf{2a}}$	Ours (w/o geom.)
# parameter	4K	12K	37K	642K
Inference Time (s)	0.45	0.69	1.35	1.61

Table 16: # of parameters and inference time on wirelength prediction in placement stage.

Model	MLP	Net$^{\mathbf{2f}}$	Net$^{\mathbf{2a}}$	LHNN	Ours
# parameter	4K	13K	39K	54K	694K
Inference Time (s)	0.6	1.13	2.39	21.41	3.51

When working on *superblue19*, DREAMPlace spends around 285s to output congestion and wirelength from netlist input, where it takes 226.2s to produce placement result of *superblue19*, 55.2s to output congestion and 3.63 to calculate wirelength, respectively.

References

[1] Mohamed Baker Alawieh, Wuxi Li, Yibo Lin, Love Singhal, Mahesh A. Iyer, and David Z. Pan. High-definition routing congestion prediction for large-scale fpgas. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 26–31, 2020.

[2] Bowen Wang, Guibao Shen, Dong Li, Jianye Hao, Wulong Liu, Yu Huang, Hongzhong Wu, Yibo Lin, Guangyong Chen, and Pheng Ann Heng. Lhnn: Lattice hypergraph neural network for vlsi congestion prediction, 2022.

[3] Tong Xia, Junjie Lin, Yong Li, Jie Feng, Pan Hui, Funing Sun, Diansheng Guo, and Depeng Jin. 3dgcn: 3-dimensional dynamic graph convolutional network for citywide crowd flow prediction. *ACM Trans. Knowl. Discov. Data*, 15(6), jun 2021.

[4] A. Mirhoseini, A. Goldie, M. Yazgan, and J. Jiang. A graph placement methodology for fast chip design. In *Nature 594*, page 207–212, 2021.

[5] Shuwen Yang, Ziyao Li, Guojie Song, and Lingsheng Cai. Deep molecular representation learning via fusing physical and chemical information. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 16346–16357. Curran Associates, Inc., 2021.