On Advancing Physical Design using Graph Neural Networks (Invited Paper)

Yi-Chen Lu
yclu@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia USA

Sung Kyu Lim
limsk@ece.gatech.edu
Georgia Institute of Technology
Atlanta, Georgia USA

ABSTRACT

As modern Physical Design (PD) algorithms and methodologies evolve into the post-Moore era with the aid of machine learning, Graph Neural Networks (GNNs) are becoming increasingly ubiquitous given that netlists are essentially graphs. Recently, their ability to perform effective graph learning has provided significant insights to understand the underlying dynamics during netlist-to-layout transformations. GNNs follow a message-passing scheme, where the goal is to construct meaningful representations either at the entire graph or node-level by recursively aggregating and transforming the initial features. In the realm of PD, the GNN-learned representations have been leveraged to solve the tasks such as cell clustering, quality-of-result prediction, activity simulation, etc., which often overcome the limitations of traditional PD algorithms. In this work, we first revisit recent advancements that GNNs have made in PD. Second, we discuss how GNNs serve as the backbone of novel PD flows. Finally, we present our thoughts on ongoing and future PD challenges that GNNs can tackle and succeed.

1 INTRODUCTION

With the ever-increasing design complexity struggling commercial Electronic Design Automation (EDA) tools to deliver high-quality full-chip designs in a timely manner, Machine Learning (ML) algorithms have been widely leveraged across the entire Physical Design (PD) flow, from synthesis to sign-off, to achieve faster design convergence and better end-of-flow Power, Performance, and Area (PPA) metrics. To make accurate predictions or to discover hidden netlist characteristics that can drive better PD optimization, ML algorithms deeply rely on the input vectors that are representative of the underlying design. Given the fact that VLSI netlists are essentially graphs, where cells can be considered as nodes and nets can be viewed as edges, Graph Neural Networks (GNNs) become one of the most promising choices to encode netlist information in an efficient and systematic manner.

In general, GNNs follow a message passing scheme, where the goal is to transform the initial features of each node into better representations by aggregating the features from neighboring nodes. A feature vector of a node can be considered as a message which is iteratively transformed and passed onto its neighboring nodes. If a GNN model has k layers of neurons, the final representations of each cell are contributed by the features of its local k-hop neighborhood, which is also known as the *receptive field*. Nonetheless, the expressiveness of vanilla GNNs is known to suffer as the number of layers k increases beyond a certain threshold, which is known as the *over-smoothing* problem [5]. The main reason is that as the number of k increases, the receptive field of each node will have larger overlap, which in the end makes the final learned embeddings across different nodes indistinguishable (i.e., all the nodes have similar learned embeddings). To overcome the over-smoothing issue in PD, several works have either focused on improving the GNN message passing scheme [3] or enhancing the netlist transformation step [14] (more details will be discussed in later sections).

In the realm of PD, node embeddings learned by GNNs have been used to solve the downstream supervised learning, unsupervised learning, and reinforcement learning tasks including PD problems such as partitioning [13], placement [12, 14, 16], gate sizing [10, 11, 15], activity simulation [21], and PPA predictions [3, 8, 9, 20], which often overcome the limitations of traditional PD algorithms or basic deep neural networks (DNNs) models. The successes of GNNs in a variety of tasks have proven the effectiveness of using graph learning to address PD problems. Usually, GNNs serve as *feature encoders* of higher level of frameworks, which distills and refines the initial input features into better representations by considering the graph structural information for next stage predictors to make more accurate and reliable predictions.

Depending on the objective formulation, different GNN message passing and local neighborhood aggregation mechanisms have been developed to cope with each specific PD problem. Aside from common supervised learning-based approaches formulating the objectives regression or classification based metrics, recent works [11, 13, 14] have shown that graph representation learning conducted by GNNs is a promising way to optimize crucial PPA metrics in an unsupervised manner, which truly enables generalization across different designs and technology nodes. As today's commercial VLSI designs easily contain millions, if not, billions, of instances that are required to be jointly optimized within a constrained layout, conventional optimization algorithms using heuristics or analytical methods are struggling to keep up with the technology scaling, which often results in sub-optimal QoR metrics. Therefore, GNNs offer great opportunity to revisit classical PD problems in a learning-driven manner.

Despite being a relatively new modeling architecture, GNNs have revolutionized modern PD implementation flows without a doubt. The goal of this paper is to highlight the important PD advancements that have been achieved by GNNs and discuss about the remaining challenges waiting to be solved. The rest of the paper is organized as follow. Section 2 first presents the necessary background about GNNs and their common modeling approaches for solving PD problems. Section 3 presents the details and rationales on how graph learning-based frameworks have solved the conventional PD problems including partitioning for 3D ICs, placement, and gate sizing for timing optimization. Section 4 further illustrates how GNN serve as the backbone of novel commercial PD flows to achieve faster chip design turn-around time through early

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561094

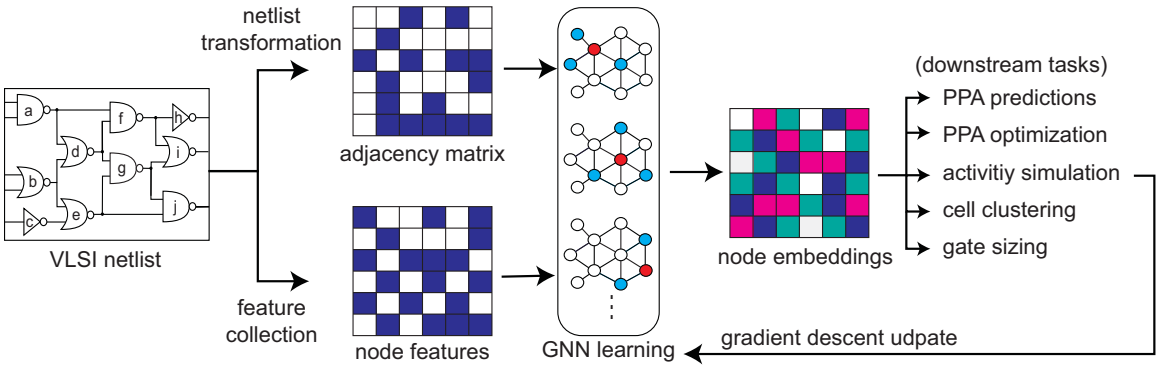

Figure 1: A common GNN modeling architecture in PD. First, given an initial netlist, an adjacency matrix denoting connectivity is constructed through netlist transformation and node features are collected to characterize the underlying design. Then, node representation learning is performed to transform the initial node features into better representations, which can be taken as the inputs to a variety of downstream tasks. Note that the entire framework can be end-to-end differentiable.

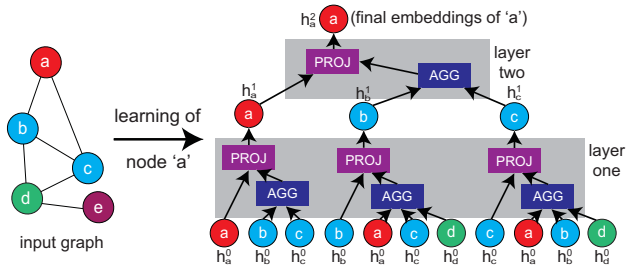

Figure 2: Illustration of node embedding process from [12]. Assuming a GNN model has two layers, the final representations of a node will only depend on nodes within 2-hop neighborhood, which is also known as the receptive field.

QoR predictions. Finally, we provide conclusion in Section 5 and present thoughts for future directions as well as remaining critical challenges.

2 BACKGROUND

2.1 Overview of GNNs in PD

While deep learning (DL) has brought tremendous success to PD in providing accurate predictions that boost productivity by distilling huge amount of design data, there is a fundamental challenge in applying popular DL algorithms such as vanilla neural networks or tree-based models to encode netlist information without spending huge amount of time and manual effort in feature engineering. The main reason is that these models often require a fixed form of input representations (e.g., assuming input vectors are in the same dimension), however, netlist come in various sizes and may have extremely different characteristics. To overcome this issue, recent works of ML in PD are seeking more systematic the effective technique for netlist encoding. GNNs thus become the second to none choices.

Since VLSI netlists are inherently hypergraphs, GNNs and their variants have been applied to solve a wide variety of PD problems throughout the entire design flow by constructing representative node embeddings from initial features and local neighborhood aggregation. Figure 1 demonstrates the common practice of applying GNNs to solve conventional PD problems. As shown in the figure, the main goal of GNNs is to construct node embeddings through representation learning, which takes the adjacency matrix denoting the connectivity among cells and initial features as inputs. The learned embeddings can be further leveraged in the downstream tasks to solve a wide range of problems.

Figure 2 further demonstrates the detailed local neighborhood aggregation during the representation learning. It is shown that due to the strategy of aggregation, the receptive field of a vanilla GNN will be limited by its number of layers, which means if a GNN has k layers, then the final representations of each cell will only be contributed by the neighboring nodes within k-hop neighborhood. As aforementioned, due to the issue of *over-smoothing*, increasing the number of layers k may not always be beneficial to the learning, as the learned embeddings across different nodes may be indistinguishable due to the high overlapping of their receptive fields. It is suggested in [2] that $k = 3$ often yields good empirical results.

2.2 GNN Message Passing

In general, given a graph $G = (V, E)$ and node features h, the GNN node representations at level k (e.g., h^k) can be obtained from the ones at the previous level $k-1$ as:

$$\begin{aligned} h^{k-1}_{N_k(v)} &= \text{AGG\_OP}\left(\{\mathbf{W}^{agg}_k h^{k-1}_u, \forall u \in N_k(v)\}\right), \\ h^k_v &= \sigma\left(\mathbf{W}^{proj}_k \cdot \text{concat}[h^{k-1}_v, h^{k-1}_{N_v(v)}]\right), \end{aligned} \tag{1}$$

where AGG_OP denotes the aggregation operation where popular choices include sum, average, min and max operations; σ denotes the sigmoid function; h^k_v denotes the representation vector of node v at level k, $N_k(v)$ denotes the neighbors sampled at k-hop which is subject to the sampling size s_k, W^{agg}_k and W^{proj}_k denote the aggregation and projection matrices respectively, which can be viewed as the neural layer at level k. Finally, $\{h^0\}$ represent the initial node features and for a GNN model with K layers, h^K denote the final obtained node embeddings for downstream tasks. It should be noted

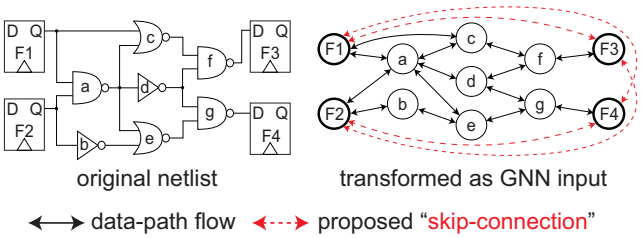

Figure 3: Netlist transformation from [14]. Compared with the clique-based technique adopted by many previous works, the transformed graph now only focuses on preserving the connections on timing paths while introducing additional "skip-connection" edges between launch and capture flops.

that several works [3, 14] have enhanced the vanilla GNN architecture to more precisely address PD problems. In the following sections, we will present more details on recent GNNs advancement and achievements made in the realm of PD.

2.3 Netlist Transformation

Since a VLSI netlist is inherently a hypergraph $G = (V, E)$ and the node representation learning conducted by GNNs relies on an adjacency matrix $A \in R^{|V|x|V|}$ where each element $A_{ij} \in \{0, 1\}$ denotes whether learning messages can be passed from node i to node j, a netlist transformation is needed prior to GNN representation learning. Most of the previous GNN-related works [1, 10, 12, 13] adopt the clique-based model for netlist transformation, where cells on the same net are assigned message passing connections to each other. Although the clique-based model is the most popular approach to transform hypergraphs to graphs in PD, it is pointed out in [14] that this netlist transformation may weaken the expressiveness of GNNs as it introduces a large amount of edges and thus it may be difficult to identify critical connections among cells. Considering the fact that not every edge is equally important, [14] proposes a novel netlist transformation technique as shown in Figure 3, where the main highlight is that only connections on timing paths are preserved in the transformed graph and additional "skip-connection" edges are introduced between launch and capture flops.

2.4 Initial Feature Collection

One of the main reasons why GNNs are so powerful is that they perform effective netlist encoding based on netlist characteristics. This allows downstream modules to globally and systematically perform optimization decisions based on design characteristics rather than heuristics as most of the conventional algorithms offer in circuit design. Therefore, prior to the graph learning process, it is essential to determine an initial feature vector for each node (or edge) of the underlying netlist. The initial feature vectors are expected to provide insights related to the problem that is being solved. Popular node features include timing information (slack, transition), power consumption (cell internal power, net switching power), parasitics, library information (constraints), physical information (x, y locations) etc. During the graph learning process, these initial features of a design instance will be transform into better representations by considering the local graph structure and its neighboring nodes.

2.5 Common Learning Objectives

2.5.1 Supervised Learning. This category includes the works whose goals are making specific predictions as close as target groundtruths, where the objective is either in a regression or a classification format. Common examples include PPA predictions, parasitic extractions, and activity simulations. Despite that having a rich amount of data often leads supervised models into better prediction accuracy, the generalization of these models is still a major concern as they inevitably will be limited by the designs or technologies that are trained upon. Recently, many research groups have discovered that by using GNNs as feature encoders, the supervised learning-based modules often yield better prediction results and generalizability [8, 9]. In other words, instead of using raw features that are manually defined, leveraging GNNs as feature transformers has been shown to improve accuracy of supervised learning.

2.5.2 Unsupervised Learning. This category refers to the works whose objective formulations do not involve any target groundtruths aside from the inputs, which is more preferred in the realm of EDA as data collection is often extremely expensive. Previous works [12–14] have successfully leveraged GNNs to perform unsupervised learning for placement optimization, where curated loss functions are devised to mimic traditional placement objectives including wirelegnth, congestion, timing, and power metrics. In these applications, GNNs not only serve as feature encoders but also as standalone optimization algorithms where the training is directly optimizing design metrics.

2.5.3 Reinforcement Learning (RL). This category represents the works whose goals are to iteratively maximize rewards under designated environments. Previous works [10, 16] have demonstrated that GNNs can be leveraged as RL state vector encoders which provide essential information and guidance on improving the rewards for placement [16] or timing [10] optimization. A subtle difference of the GNN usage from those in supervised learning is that GNNs in RL applications are expected to capture the time-wise node feature variations on same graph structures, which is because the node features are continuously changing as different actions are taken.

3 GNN APPLICATIONS IN PHYSICAL DESIGN

After describing the necessary background of how GNNs work and how they can be applied to solve a wide variety of PD problems, we now delve into specific applications and illustrate in detail about how GNNs have advanced PD.

3.1 GNN Circuit Partitioning for 3D ICs

Circuit partitioning is a topic that has been studied extensively for the past several decades. The renowned algorithm hMetis [4] has shown that multi-level partitioning can achieve good partitioning results in terms of cutsize. However, for the circuit partitioning stage in state-of-the-art 3D design flows, cutsize is no longer the priority metric to evaluate partitioning results. In 3D ICs, min-cut partitioning algorithms may not fully realize the potential that 3D integration provides as they may under-utilize 3D connections. To overcome this issue, [13] propose a design-aware, unsupervised, 3D tier partitioning framework using GNNs. The key idea is that instead of focusing on minimizing connectivity as conventional

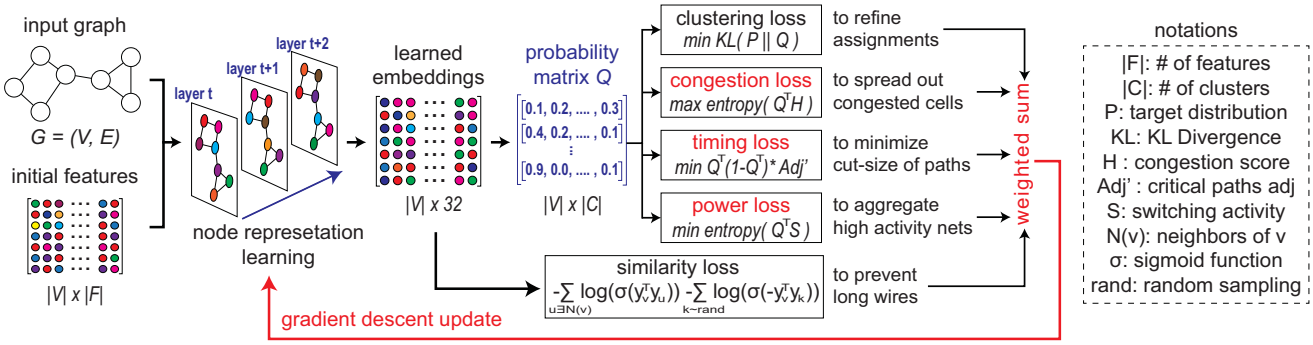

Figure 4: PPA-directed unsupervised deep graph clustering framework from [14]. A GNN module is jointly trained with a clustering module to optimize PPA metrics as ML loss functions bin an end-to-end manner.

min-cut based algorithms, [13] strives to identify the high similarity cells based on their attributes and local graph structure. The entire framework consists of two stages. First, GNNs are utilized to transform the hand-picked features that include hierarchy, timing, and power information into higher dimensional representations by minimizing the per-cell similarity-based loss function as:

$$\mathcal{L}(h_v) = -\sum_{u \in N(v)} \log(\sigma(h_v^\top h_u)) - \sum_{i=1}^{M} \mathbb{E}_{n_i \sim Neg(v)} \log(\sigma(-h_v^\top h_{n_i})), \quad (2)$$

where $Neg(v)$ denotes the negative sampling distribution of node v, and M denotes the negative sampling size (negative sampling denotes the selection of cells that should be dissimilar to the current target cell). After the graph representation learning is completed, the weighted k-means clustering algorithm [7] is utilized to cluster the GNN learned representations into two clusters (they assume 3D ICs to be 2-tier). Finally, it is shown that with the GNN learning-based tier partitioning approach, the end-of-flow 3D full-chip PPA metrics can be improved significantly compared with the existing bin-based min-cut tier partitioning algorithm.

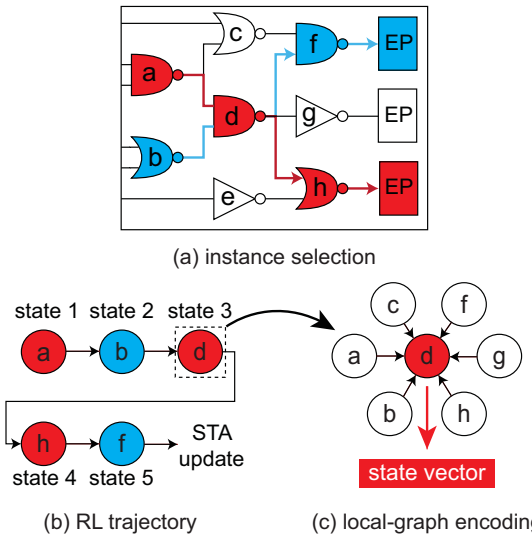

Figure 5: Illustration of RL gate sizing process with GNN state encoding from [10].

3.2 Placement Optimization using GNNs

In modern industrial PD flows, placement optimization via placement guidance has become a must-use technique to high quality placement. The placement guidance technique assist placement optimization in commercial tools by informing placers about the design instances that should be placed close to each other in order to optimize crucial PPA metrics. During placement, commercial placers will spend effort in grouping the cells that are suggested to be in a common cluster nearby each other. Nonetheless, performing placement guidance requires in-depth design-specific knowledge, which is only achievable by a handful of experienced designers who are familiar with the underlying design. To construct placement guidance (i.e., cell clustering constraints) in a systematic and automated manner, [12, 14] propose to identify critical cell clusters using GNNs.

The proposed GNN-based placement optimization framework [14] is shown in Figure 4. The input to the framework is a globally placed netlist along with its PPA evaluations which include congestion, timing, and power metrics. The key idea of the framework is to discover the cell clusters that are important to improve the underlying netlist if being optimized with extra effort during placement optimization. For the first time, [14] demonstrates that GNNs can not only be used as feature encoder, but also be leveraged as standalone optimization algorithms with proper objective formulations.

3.3 Timing Optimization using RL and GNN

It is widely acknowledged that RL is a promising paradigm that achieves super-human performance on many EDA tasks. A recent work [16] shows that RL algorithms equipped with GNNs can be used for macro placement to improve design turn-around time and final PPA metrics. In addition, RL has also been applied to solve analog transistor sizing [19], global routing [6], and technology mapping [17]. In this work, we particularly illustrate how RL can be leveraged to solve the VLSI gate sizing for timing optimization problem and how GNNs play an indispensable role to achieve good performance.

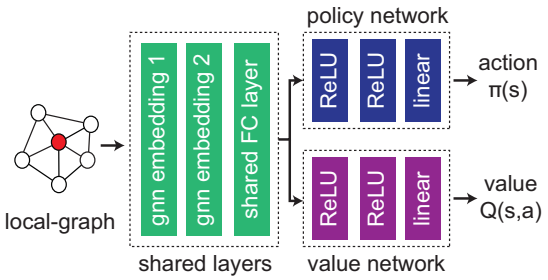

Figure 6: RL agent architecture from [10]. A two-layer GNN module is utilized as a state encoder. The encoded state vector is taken as the inputs of the policy and value networks to decide the action and to estimate the reward.

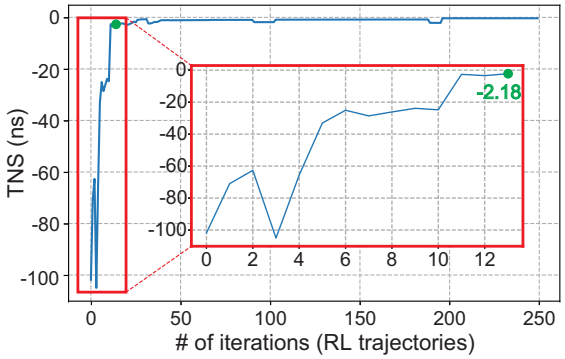

Figure 7: RL gate sizing iterations on an industrial benchmark from [10].

Figure 5 presents a high-level overview of RL gate sizing process in [10], where GNNs are applied to encode local graph information of each selected instance into an RL state vector. The GNN encoded state vector is expected to characterize an instance behaviour during timing optimization, and is further taken as the input to the downstream agent to decide what actions to take in order to maximize the associated rewards. The detailed architecture is shown in Figure 6 and a complete sizing process on an industrial benchmark is shown in Figure 7. Finally, it is shown in [10] that although the RL sizing framework equipped with graph representation learning using GNNs adopts a fundamentally different approach to perform gate sizing, the timing optimization results are comparable, or even better, with the default algorithm from an industry-leading commercial tool.

3.4 GNNs for Fast ECO Power Optimization

The most common usage of GNNs lie in supervised learning, where the goal is to make predictions either in node or graph level. Here, we describe a recent work [11] on how GNNs can be utilized to expedite signoff power optimization using Engineering Change Orders (ECOs). At signoff ECO, V_{th}-assignment is one of the most popular techniques to improve power dissipation as it introduces minimum changes to the overall placed and routed layout. However, this signoff power optimization process is extremely time-consuming and

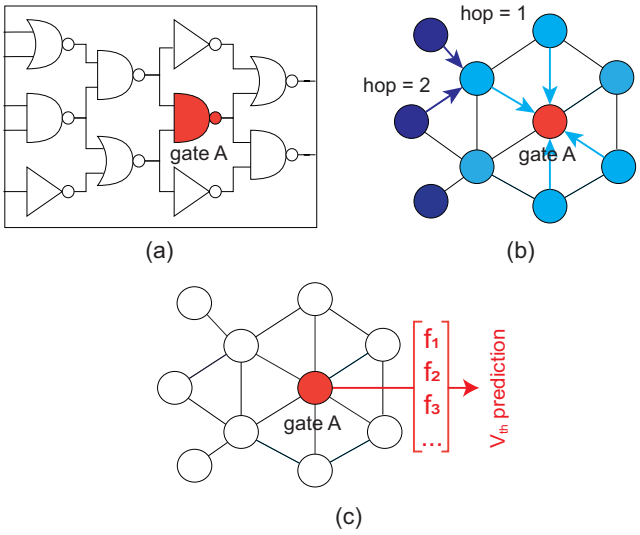

Figure 8: High-level view of the ECO-GNN framework from [11]. The framework adopts supervised learning philosophy to generate tool-accurate V_{th}-assignment prediction on each design instance.

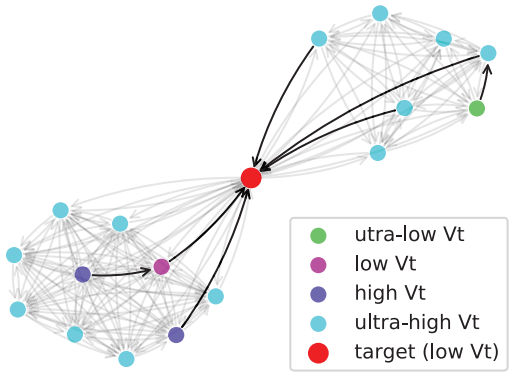

Figure 9: GNN explanation result from [11]. Important message passing edges and neighboring nodes that have more contributions to the target node are identified.

the power improvement is obscure before spending a huge amount of time in ECO iterations, which provides a great opportunity to apply learning-based algorithms in expediting the process.

To achieve this, [11] proposes to train a GNN-based framework with groundtruth data pre-generated from commercial tools, where the framework directly predicts the end V_{th}-assignment results of each design instance after the optimization. Ideally, with a large amount of training data, a well-trained framework will generate tool-accurate assignments prediction results on unseen designs that are not utilized in training. Figure 8 shows a high-level view of the GNN-based signoff power optimization framework named ECO-GNN [11]. For each design instance, the framework will predict

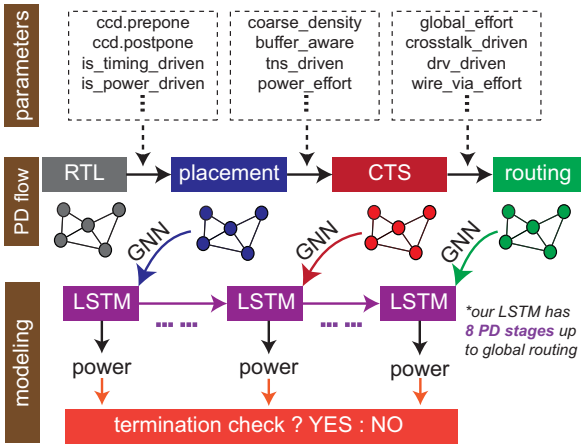

Figure 10: GNN integration in a commercial PD flow from [8]. GNN dynamically encodes netlists at different stages into graph vectors which are taken as inputs of a LSTM model that provides decision on early termination.

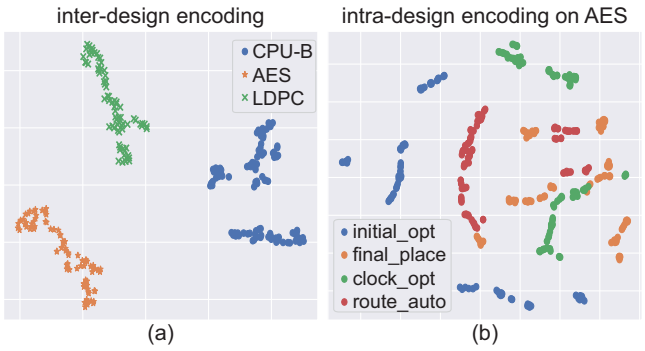

Figure 11: t-SNE visualization of GNN netlist encoding from [8]. (a) Each dot is a complete PD run of an unseen netlist. (b) Each dot denotes a netlist at a specific PD stage.

the V_{th}-assignment that the reference commercial tool will make during ECO optimization. As for each instance, the number of available assignment choices is discrete and finite, the training loss is defined as the cross-entropy loss between the predictions made by the framework and the groundtruths from the reference tool. In [11], a total of 14 designs are utilized in the experiments with a train/test splitting ratio of 9:5. It is demonstrated that after training the framework ECO-GNN with 9 designs, it can generate accurate prediction results on the remaining 5 testing designs instantly with an F1-score as high as 0.9, which improves the tool optimization runtime by as much as 14 times.

Finally, to understand the rationality behind predictions made by the GNN framework. [11] further proposes a GNN explanation technique to explain the prediction made behind each cell. The key idea of the explanation technique is to trace the important message passing flows and to identify the neighboring nodes that have more contribution to the prediction of the target node. Figure 9 shows the explanation result of a target node (colored in red) whose prediction is made as "low V_{th}". It is observed that although the majority of the neighboring nodes are ultra-high V_{th}, but cells with lower V_{th} types have higher importance to the target node. As a result, low V_{th} is assigned to the target node, which aligns well with common design knowledge that cells in lower V_{th} types have larger capacitance and thus impose tighter constraints to the target cell.

4 GNN INTEGRATION IN PD FLOWS

After discussing GNN applications on individual PD tasks, we now describe a recent advancement [8] on integrating GNNs into a complete PD flow which helps to reduce chip design turn-around time by providing early termination check at a series of intermediate stages. Figure 10 demonstrates an overview of the integration, where GNN is taken as a netlist encoder and provides information to a LSTM model that makes decision of whether terminating the underlying PD implementation based on the criterion of end-of-flow power estimation. The key highlight is that GNN is leveraged to encode netlists in different stages, where the graph structures and the node features are dynamically (and may be drastically) changing from stage to stage. To further quantify the effectiveness of GNN representation learning, [8] further visualizes the GNN encoding using a dimension reduction technique, t-SNE [18], as shown in Figure 11. It is shown that the GNN module not only clearly differentiates different designs, but also distinguishes netlists from different PD stages even under a same design, which has proven the success of leveraging GNNs as netlist encoders.

5 DISCUSSION AND CONCLUSION

In this paper, we have revisited recent advancement made by GNNs and discussed the key rationales behind their success. It is shown that GNNs can not only be utilized as feature encoders, but also as optimization algorithms that can directly improve design PPA metrics. In spite of the superior results achieved, there are still a few challenges waiting to be solved to broaden the impact of GNNs in PD. First, as aforementioned, on one hand GNNs suffer from the over-smoothing problem as the number of layers increases, and on the other hand the receptive field of a GNN model is limited by its number of layers. This directly results in a dilemma on choosing the feasible number of layers, which has been empirically set between two to four in many applications. Nonetheless, a more serious issue is that as the over-smoothing problem prohibits GNNs architecture from going deep, common message passing mechanism may not be able to capture the concept of "path" easily. To overcome this issue, [3] proposes a level-by-level message passing technique to ensure the information of standpoints of timing paths can always be passed onto the endpoints. However, timing paths in VLSI designs can easily have thousands of levels, which makes this technique not feasible as information may be extremely diluted at the endpoints.

With the successes that GNNs and their variants have brought, it is no doubt that GNNs will continuously play an important role in the realm of PD. Going forward, we believe GNNs for PPA optimization using unsupervised learning will be an even more critical topic for future research to investigate in, as this will allow GNNs to become truly standalone algorithms that are not limited by any design or technology. We believe the pioneer works presented in this paper shall clearly demonstrate on how GNNs can advance PD.

REFERENCES

[1] A. Agnesina, K. Chang, and S. K. Lim. Vlsi placement parameter optimization using deep reinforcement learning. In Proceedings of the 39th International Conference on Computer-Aided Design, pages 1–9, 2020.

[2] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 3438–3445, 2020.

[3] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In Proceedings of the 59th Annual Design Automation Conference 2022. ACM, 2022.

[4] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 7(1):69–79, 1999.

[5] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In Thirty-Second AAAI conference on artificial intelligence, 2018.

[6] H. Liao, W. Zhang, X. Dong, B. Poczos, K. Shimada, and L. Burak Kara. A deep reinforcement learning approach for global routing. Journal of Mechanical Design, 142(6), 2020.

[7] S. Lloyd. Least squares quantization in pcm. IEEE transactions on information theory, 28(2):129–137, 1982.

[8] Y.-C. Lu, W.-T. Chan, V. Khandelwal, and S. K. Lim. Driving early physical synthesis exploration through end-of-flow total power prediction. In Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD, 2022.

[9] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim. Doomed run prediction in physical design by exploiting sequential flow and graph learning. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9. IEEE, 2021.

[10] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 733–738. IEEE, 2021.

[11] Y.-C. Lu, S. Nath, S. S. K. Pentapati, and S. K. Lim. A fast learning-driven signoff power optimization framework. In 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9. IEEE, 2020.

[12] Y.-C. Lu, S. Pentapati, and S. K. Lim. The law of attraction: Affinity-aware placement optimization using graph neural networks. In Proceedings of the 2021 International Symposium on Physical Design, pages 7–14, 2021.

[13] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim. Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics. In 2020 57th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2020.

[14] Y.-C. Lu, T. Yang, S. K. Lim, and H. Ren. Placement optimization via ppa-directed graph clustering. In Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD, 2022.

[15] U. Mallappa and C.-K. Cheng. Gra-lpo: Graph convolution based leakage power optimization. In 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), pages 697–702. IEEE, 2021.

[16] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, et al. A graph placement methodology for fast chip design. Nature, 594(7862):207–212, 2021.

[17] G. Pasandi, S. Nazarian, and M. Pedram. Approximate logic synthesis: A reinforcement learning-based technology mapping approach. In 20th International Symposium on Quality Electronic Design (ISQED), pages 26–32. IEEE, 2019.

[18] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(11), 2008.

[19] H. Wang, K. Wang, J. Yang, N. Sun, H. Lee, and S. Han. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In ACM/IEEE 57th Design Automation Conference (DAC), pages 1–6. IEEE, 2020.

[20] Z. Xie, R. Liang, X. Xu, J. Hu, Y. Duan, and Y. Chen. Net 2: A graph attention network method customized for pre-placement net length estimation. In 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), pages 671–677. IEEE, 2021.

[21] Y. Zhang, H. Ren, and B. Khailany. Grannite: Graph neural network inference for transferable power estimation. In 2020 57th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2020.