Understanding Graphs in EDA: From Shallow to Deep Learning

Yuzhe Ma CUHK
Zhuolun He CUHK
Wei Li CUHK
Lu Zhang CUHK
Bei Yu CUHK

ABSTRACT

As the scale of integrated circuits keeps increasing, it is witnessed that there is a surge in the research of electronic design automation (EDA) to make the technology node scaling happen. Graph is of great significance in the technology evolution since it is one of the most natural ways of abstraction to many fundamental objects in EDA problems like netlist and layout, and hence many EDA problems are essentially graph problems. Traditional approaches for solving these problems are mostly based on analytical solutions or heuristic algorithms, which require substantial efforts in designing and tuning. With the emergence of the learning techniques, dealing with graph problems with machine learning or deep learning has become a potential way to further improve the quality of solutions. In this paper, we discuss a set of key techniques for conducting machine learning on graphs. Particularly, a few challenges in applying graph learning to EDA applications are highlighted. Furthermore, two case studies are presented to demonstrate the potential of graph learning on EDA applications.

ACM Reference Format:
Yuzhe Ma, Zhuolun He, Wei Li, Lu Zhang, and Bei Yu. 2020. Understanding Graphs in EDA: From Shallow to Deep Learning . In *Proceedings of the 2020 International Symposium on Physical Design (ISPD '20), March 29-April 1, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3372780.3378173

1 INTRODUCTION

Modern electronic design automation (EDA) flow is constituted by multiple stages. On each step, dedicated optimization is performed to achieve desired quality of results (QoR). As the integration keeps increasing, more and more design constraints are imposed and lead to performance bottleneck and severe runtime overhead. Various optimization techniques have been proposed to improve or renovate the existing methodologies in EDA flow.

Graph is one of the core subjects in enormous EDA problems and optimization algorithms. It is a mathematical structure that models pairwise relationships among different items, which makes it a natural and powerful representation for many fundamental objects in EDA applications, such as Boolean functions, netlists and layout. Over the past few decades, a lot of problems are investigated by leveraging graph abstraction and a rich set of elegant graph algorithms are developed to solve these problems [1–7]. It can be observed that graph algorithms can assist the problem-solving of

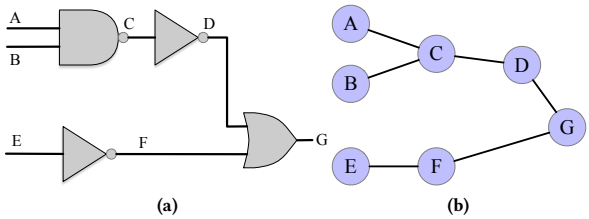

Figure 1: (a) A circuit; (b) The graph representation.

EDA in a few ways. Firstly, since a problem is abstracted into a graph representation, numerous well-known graph algorithms can be directly applied or slightly modified to form a solution. However, it may be still non-trivial to develop an effective approach for those complicated problems after being modeled with graphs. Besides, mathematical programming is another useful approach to utilize. By defining an objective function and a set of constraints over the constructed graph, mathematical optimization methodologies can be applied to derive a solution, in which certain highly optimized software and libraries can be of great help. It is commonly seen many problems are tackled with integer linear programming (ILP) [5, 6, 8], linear programming [9, 10], etc. However, given the fact that many EDA problems are NP-hard and problem sizes are usually very large, the efficiency will be a major concern. Heuristic algorithms or approximation algorithms over graphs have been intensively developed to achieve a balance between performance and efficiency [11, 12].

As traditional optimization becomes more and more sophisticated, data-driven learning techniques have drawn much attention in hardware design automation. Classical machine learning techniques require manual extraction of the features which are fed into a downstream learning model for training, and several such attempts have been made in EDA applications [13–17]. Deep learning has demonstrated that feature representation can be automatically learned in the presence of a large amount of data, which achieves noticeable performance gain in EDA area [18–21]. However, a few issues may emerge when it comes to graphs. On one hand, applying classical machine learning approaches on graphs relies on non-trivial heuristics for feature extraction to encode structural information, which consumes lots of endeavors to achieve desired performance. On the other hand, it is not straightforward to transfer conventional grid-based operations (e.g., convolution, pooling, etc.) in deep neural networks to tackle irregular structured data like graphs. Recently, many approaches have emerged in graph learning research covering a wide range of sub-fields, including node classification [22–24], graph generation [25–27], model robustness [28], etc. Inspired by such recent progress in graph learning, there are also a few attempts trying to apply graph learning techniques to solve certain problems in EDA [29], which demonstrate the great

ISPD '20, March 29-April 1, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7091-2/20/03…$15.00
https://doi.org/10.1145/3372780.3378173

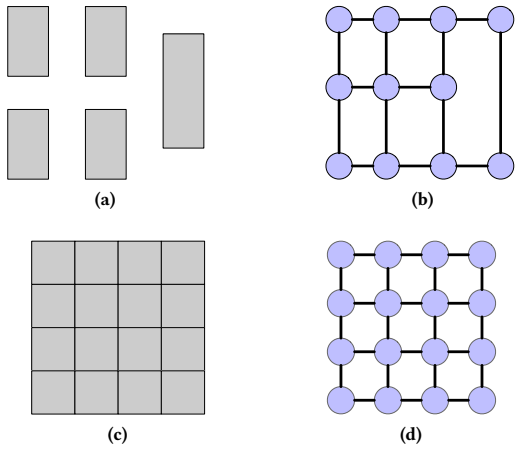

Figure 2: Examples of routing graphs. (a) Channel model; (b) Channel graph; (c) Grid model; (d) Grid graph.

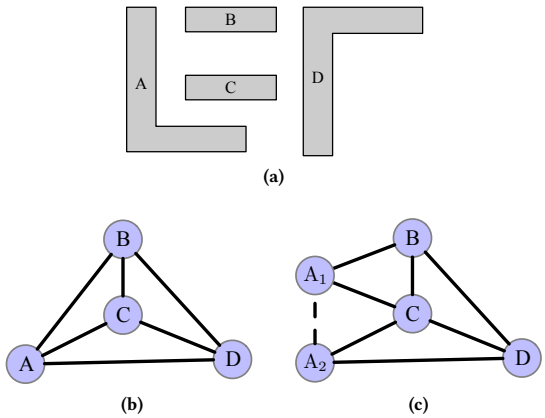

Figure 3: An example of decomposition graph in layout decomposition problem. (a) Layout pattern; (b) Decomposition graph; (c) Decomposition graph with stitch edge.

potential of graph-based learning methods on overcoming performance bottleneck in the existing design flow and push forward the advance in the industry.

In this paper, a number of general graph-based learning techniques are briefly introduced. Particularly, several special properties that are critical in circuits design are highlighted, including hypergraph, graph heterogeneity and scalability. Then we focus on applying graph-based learning techniques to assist EDA applications. Two case studies of graph learning in EDA are presented, including testability analysis for design-for-testing and timing model selection in a netlist.

The rest of this paper is organized as follows. Section 2 reviews a set of graph-based problems and widely used graph algorithms in the EDA field. Section 3 introduces some fundamental idea of graph learning and some conventional graph learning algorithms. Section 4 presents two case studies and Section 5 concludes the paper.

2 TRADITIONAL GRAPH-BASED METHODOLOGIES IN EDA APPLICATIONS

Graph model is widely leveraged in a wide range of applications in the modern design flow, which can greatly simplify the problem formulation and algorithm analysis. On top of that, many problems in typical EDA flow can be addressed effectively, e.g., technology mapping [1, 30, 31], testability analysis [2], circuit partitioning [3, 32], placement [4, 33], etc. The most intuitive modeling for a circuit is a graph whose nodes represent gates and edges represent wires, as shown in Figure 1. In addition, different ways of graph construction may be applied based on different characteristics in applications. In logical verification, a Boolean function is modeled with a rooted, directed graph [34]. Global routing leverages graph to capture the adjacencies and capacities of the routing region, in which channel graph model [35] (Figure 2(a) – Figure 2(b)) and grid graph model are applied [8, 36] (Figure 2(c) – Figure 2(d)). In detailed routing, horizontal and vertical constraint graphs are used to model the relative positions of different nets in a channel routing instance [37]. In the post-layout stage, graph representation is still a powerful method. Layout decomposition is a challenging problem when using multiple patterning lithography for manufacturing in advanced technology nodes. The problem can be modeled using a conflict graph and a stitch graph [5, 6, 38–40], as shown in Figure 3.

Since many problems can be modeled as graphs, a convenient way to solve these problems is to apply graph algorithms directly. For example, critical path extraction can be transformed into finding the longest path in a weighted and directed graph, which can be solved based on Bellman-Ford algorithm [41, 42]. The construction of a power network with a minimum wirelength can be modeled as a minimum tree construction (MST) problem [42]. Other widely applied graph algorithms in EDA include network flow for placement [9, 10], graph partitioning [32], graph coloring for layout decomposition [5, 6, 12, 39, 43, 44], etc. Besides, many NP-complete and NP-hard problems are prevalent in EDA field. Considering the scale of the problem is huge in modern designs, it is difficult to derive an optimal solution for these problems. Therefore, heuristic algorithms are also commonly applied, which empirically yield good solutions. Previous studies have shown that heuristic algorithms can achieve high quality results efficiently in floorplanning [45, 46] and layout decomposition [12, 47, 48]. In addition, mathematical optimization is another category of approach which optimizes the value of an objective function and subject to a set of constraints. It is also used intensively to solve graph problems in EDA applications. For example, integer linear programming (ILP) is used for solving a variety of problems such as routing [49], layout decomposition [6, 50], etc.

Apart from traditional analytical or heuristic algorithms, a few attempts have exploited machine learning techniques to tackle graph-based problems. Most of these approaches utilized classical machine learning models which are "shallow" learning in contrast to conventional deep neural networks.

Samanta *et al.* [51] performed wire and buffer sizing based on support vector regression (SVR) to minimize variation on non-tree clock networks. In [52], a machine learning approach is proposed to model wire delay and slew based on the timing graph, which utilizes a set of classical analytical values extracted from the timing graph as features for regression to obtain the wire delay/slew. A

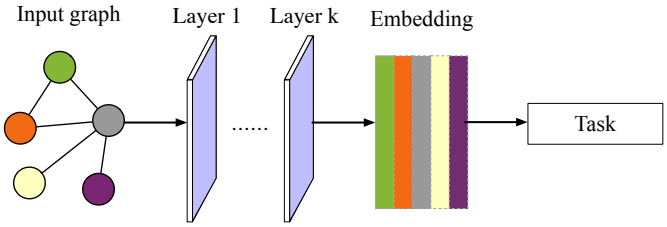

Figure 4: Graph neural network with k layers for embedding generation. Obtained embedding is fed to downstream tasks.

Gaussian Process Regression-based active learning flow is proposed for high performance adder design space exploration based on prefix graph representation [16]. A timing failure prediction technique is proposed in [53] given the information of netlist, timing constraints, and floorplan. In place-and-route (P&R) stage, routability is a critical issue and has large impact on the final quality. Some previous works investigate routability estimation with a particular routing graph model based on manually extracted features. Qi *et al.* [54] and Zhou *et al.* [55] applied multivariate adaptive regression splines (MARS) to detailed routing congestion estimation. Pui *et al.* [17] proposed a hierarchical hybrid model for congestion estimation in FPGA placement, which consists of linear regression and support vector regression.

3 GRAPH REPRESENTATION WITH DEEP LEARNING METHODOLOGIES

The main challenge of conducting learning algorithms on graphs is how to encoding structural information of graphs, which have been intensively investigated in the machine learning community. In this section, we introduce a bunch of graph representation methods based on neural networks, and highlight some challenges on how to apply to EDA applications.

3.1 Graph Learning with Neural Networks

Graph-based learning is a new approach to machine learning with a wide range of applications [56]. Before performing a certain task, representation of node or graph should be obtained first, which is known as embedding and can be fed to downstream models, as shown in Figure 4.

The blossoms of deep learning in various disciplines have promoted the application of neural networks in graph learning. Typically, neural networks expertise in extracting latent representations from Euclidean data, such as an image (a grid of pixel) or a text (a sequence of letters). While a graph lies in non-Euclidean domain, which could be quite irregular. Therefore, it is natural and necessary to extend deep learning approaches to graph data.

The seminal works of graph neural network (GNN) are mostly inspired by recurrent neural networks, where nodes recurrently exchange information with adjacent nodes until a stable state is reached. Formally, the hidden state of a node v is updated recurrently as following:

$$\boldsymbol{h}_v^{(t+1)} = \sum_{u \in \mathcal{N}_v} f(\boldsymbol{h}_u^{(t)}, \boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv}), \tag{1}$$

where $\boldsymbol{h}_v^t$ is the hidden state of node v at time step t, $\mathcal{N}_v$ the set of adjacent vertices of v, $\boldsymbol{x}_v$ the feature of node v, $\boldsymbol{x}_{uv}$ the feature of edge uv, and f the parametric function for local state transition, which should be carefully design (specifically, a contraction map [57]) to ensure convergence. Despite the conceptual significance, public interest towards recurrent GNNs was limited due to the restricted expressive power of a contractive operation and the heavy computational burden to reach its equilibrium.

Given the drawbacks of recurrent GNNs, the emergence of Graph Convolutional Networks (GCNs) is no surprise. Taxonomically, GCNs fall into two categories, viz., spectral-based and spatial-based, with the former based on graph spectral analysis, while the latter inherits the paradigm of message passing from recurrent GNNs. We introduce the two approaches in the following paragraphs.

Graph convolution is defined [58] on Fourier domain in the spectral approaches, where the eigen-decomposition of graph Laplacian is computed. Specifically, graph Laplacian is defined as $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{\frac{1}{2}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top$, where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{D}$ and $\boldsymbol{A}$ are degree matrix and adjacent matrix of the graph, respectively. Let $g_\theta : \mathbb{R} \to \mathbb{R}$ be a filter defined on graph spectrum $\boldsymbol{\Lambda}$ and $f : \boldsymbol{V} \to \mathbb{R}$ be features of nodes, graph convolution is given by:

$$g_\theta * f = V(g_\theta(\boldsymbol{\Lambda}) \odot \boldsymbol{V}^\top f), \tag{2}$$

which respects the Convolution Theorem. Equivalently, we write $g = \mathrm{diag}(g_\theta(\boldsymbol{\Lambda}))$ and a convolutional layer with multiple (f_l) channels is defined by:

$$\boldsymbol{H}^{(l+1)} = \sigma(\sum_{i=1}^{f_l} \boldsymbol{V}(g^i \boldsymbol{V}^\top \boldsymbol{H}^{(l)})), \tag{3}$$

where $\boldsymbol{H}^{(l)}$ is the output of previous convolutional layer with $\boldsymbol{H}^{(0)} := \boldsymbol{X}$ the collection of node features, g^i the i-th trainable filter, and $\sigma(\cdot)$ a nonlinear activation function. Note that the filters in spectral domain may not be localized, which could be alleviated with some smoothing techniques [58]. Further, the computation complexity of this line of methods is reduced through approximation and simplification [22, 59].

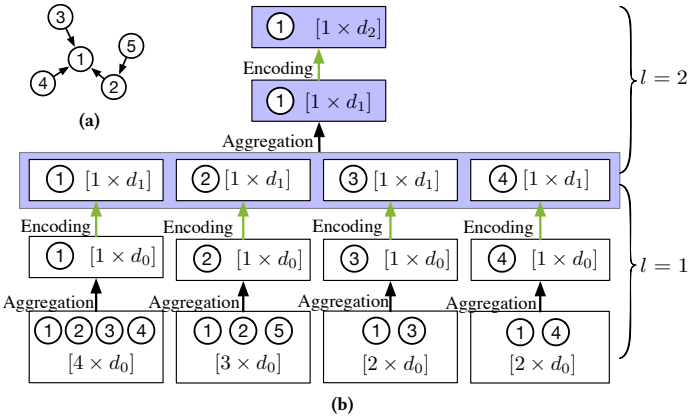

Figure 5: An illustration to compute the embedding for a node with $l = 2$. (a) Graph; (b) Procedure to compute the embedding for node 1.

Spatial-based graph convolutions are defined based on the spatial relationship of nodes, where information is propagated and aggregated in a message passing scheme.

A representative work is GraphSAGE [23], which can generate node embedding by leveraging node feature information from the neighborhood. The fundamental procedure consists of two steps, i.e., aggregation and encoding, which can be formulated as:

$$\boldsymbol{h}_{\mathcal{N}(v)}^{(l)} \leftarrow \text{AGGREGATE}_l(\{\boldsymbol{h}_u^{(l-1)}, \forall u \in \mathcal{N}(v)\}), \tag{4}$$

$$\boldsymbol{h}_v^{(l)} \leftarrow \sigma(\boldsymbol{W}^{(l)} \cdot \boldsymbol{h}_{\mathcal{N}(v)}^{(l)}), \tag{5}$$

where AGGREGATE_l in Equation (4) is the aggregation function applied to node v and its neighborhood $\mathcal{N}(v)$. Equation (5) is encoding operation consisting of an embedding projection and a non-linear activation. An example illustrating a 2-layer network for generating the embedding for node 1 is depicted in Figure 5, with encoding dimensions in l_1 and l_2 are d_1 and d_2, respectively. Specifically, if mean function is selected as the aggregation function, the aggregation in layer l is equivalent to Equation (6).

$$\begin{aligned} \boldsymbol{H}_{\mathcal{N}(v)}^{(l)} &= \boldsymbol{A} \cdot \boldsymbol{H}^{(l-1)} \\ &= \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & w_1 & w_1 & w_1 & 0 \\ w_2 & 1 & 0 & 0 & w_1 \\ w_2 & 0 & 1 & 0 & 0 \\ w_2 & 0 & 0 & 1 & 0 \\ 0 & w_2 & 0 & 0 & 1 \end{bmatrix} \end{array} \times \begin{bmatrix} \boldsymbol{h}_1^{(l-1)} \\ \boldsymbol{h}_2^{(l-1)} \\ \boldsymbol{h}_3^{(l-1)} \\ \boldsymbol{h}_4^{(l-1)} \\ \boldsymbol{h}_5^{(l-1)} \end{bmatrix}, \end{aligned} \tag{6}$$

where $\boldsymbol{A}$ is the adjacency matrix of the graph, and $\boldsymbol{H}^{(l)}$ contains the embeddings for every node in the graph, w_1 and w_2 are weights for input edges and output edges, respectively. Then the two-step process can be calculated with matrix calculation:

$$\boldsymbol{H}^{(l)} = \sigma((\boldsymbol{A} \cdot \boldsymbol{H}^{(l-1)}) \cdot \boldsymbol{W}^{(l)}). \tag{7}$$

Note that random walk is often introduced as a sampling technique. For more representative work, see [60–62]. We refer readers to [57] for a more comprehensive survey of graph learning.

3.2 Challenges in EDA Applications

3.2.1 Scalability. Unlike conventional graph learning tasks, graph learning for EDA problems is prone to runtime overhead considering that the scale of circuits keeps soaring. Similar to conventional CNNs, the most time-consuming process in the computation of a GCN is the embedding generation. To tackle the issue of scalability, several attempts have been made for efficient graph representation learning. In [63], a forward computation method with personalized PageRank is investigated to incorporate neighborhood features without aggregation procedure. Besides, it is pointed out that the inefficiency might be caused by duplicated computation under the GraphSAGE-like framework [64]. To address this, PinSAGE [64] is proposed to select important neighbors by random walk instead of aggregating all the neighbors, and a MapReduce pipeline is leveraged for maximizing the inference throughput of a trained model. Recently, GraphZoom [65] is proposed for improving both accuracy and scalability of unsupervised graph embedding algorithms, which is a multi-level spectral framework. In addition to designing specific algorithms and models, there are also a few third-party libraries like DGL [66] for users to make the network scalable.

3.2.2 Hypergraph. Another significant distinction is that hypergraph is commonly applied in EDA applications. Different from a regular edge that connects exactly two vertices, a hyperedge may connect more than two vertices. The basic idea to extend GCN to handle hypergraphs is approximating the structural information indicated by a hyperedge with normal edges. One way to perform convolution on hypergraph is proposed in [67], which can be written as:

$$\boldsymbol{H}^{(l+1)} = \sigma(\boldsymbol{D}^{-1/2}\boldsymbol{Q}\boldsymbol{W}\boldsymbol{B}^{-1}\boldsymbol{Q}^{\top}\boldsymbol{D}^{-1/2}\boldsymbol{H}^{(l)}\boldsymbol{P}), \tag{8}$$

where $\boldsymbol{D}, \boldsymbol{B}$ are the degree matrices of the vertex and hyperedges, respectively, $\boldsymbol{P}$ and $\boldsymbol{W}$ are trainable weights and $\boldsymbol{Q}$ is the incidence matrix of the hypergraph. Intuitively, Equation (8) works through clique expansion [67] which is one of the widely used methods to handle hyperedge by replacing it of size k with a k-clique. There are also a few variations of clique expansion such as attention-based clique expansion [68], which assigns different weights to generate normal edges by an attention mechanism. However, performing clique expansion on hyperedge of size k requires a complexity of $\mathcal{O}(k!)$ in terms of the number of edges.

In order to reduce the complexity and improve the efficiency, the transformation procedure can be reduced by selection. Specifically, a few normal edges are selected in the edge set generated by clique expansion instead of keeping all the edges. The selection criteria is based on the assumption that nodes in the same hyperedge share similar features. Intuitively, an edge can be omitted if the representations of two nodes are already similar during training, while those nodes with relatively distinct representations should remain connected. Therefore, one criterion is proposed in [69], which is to select node pairs connected by the same hyperedge with maximal feature difference. The procedure can be formulated as:

$$(i, j) = \arg\max_{i,j \in e} ||(\boldsymbol{h}_i - \boldsymbol{h}_j)||_2. \tag{9}$$

where $\boldsymbol{h}_i, \boldsymbol{h}_j$ are the features of node i and node j, respectively. By connecting (i, j) and $\{(i, u), (j, u) : u \in S_e\}$ where $u \in S_e$ is also called "mediator", the complexity can be reduced to $\mathcal{O}(k)$. To do so, the graph structure changes dynamically during training since the node representations are going to be updated. [70] also proposes a fast selection criteria which uses the input feature of each node to compare the difference and selects the edges at the beginning such that the graph structure is fixed regardless of potential change on node representations during training, which requires a constant number of edges for approximation.

3.2.3 Heterogeneous Graphs. Most of the existing works on graph representation focus on homogeneous graphs, in which all vertices are of the same type and all edges only represent one kind of relation. However, graphs in EDA can be constructed in a heterogeneous manner, in which there may exist different types of nodes and edges. For example, in the multiple patterning lithography problem, a typical graph contains two types of edges: stitch edge and conflict edge, as shown in Figure 3(c). A conflict edge implies

the connected nodes tend to be assigned different colors, while a stitch edge implies the connected nodes should have the same color. To address the issue of heterogeneity, several methodologies have been proposed based on the fundamental knowledge of learning on homogeneous graphs. The main difference lies in the selection of neighborhood in the feature aggregation step.

Considering there are multiple types of nodes in heterogeneous graphs, feature aggregation will naturally involve aggregating with the same node type and with different node types. Zhang *et al.* proposed HetGNN [71] to capture both structure and content heterogeneity in heterogeneous graphs. Aggregating the nodes with the same type in the neighborhood is done as :

$$\boldsymbol{e}_v^{(k+1)} = \frac{\sum_{u \in \mathcal{N}_t(v)} \mathcal{F}(\boldsymbol{h}_u^{(k)})}{|\mathcal{N}_t(v)|}, \tag{10}$$

where $\mathcal{N}_t(v)$ is the neighborhood of node v with the same type t. $\mathcal{F}(\cdot)$ is a user-defined transformation function, e.g., Bi-LSTM is used in [71] and linear transformation is applied in [62]. Therefore, there are in total $|O_V|$ embedding in the graph, where O_V is the set of all the node types in a graph. Then, embedding of different types are combined through an attention mechanism as follows

$$\boldsymbol{h}_v^{(k+1)} = \sum_{t \in O_V} \alpha_t \boldsymbol{e}_v^{(k+1)}, \tag{11}$$

where α_t denotes the importance of node type t to node v.

Apart from node heterogeneity, there could also be multiple types of edges in a graph, which denote different relationships between items. In [72], a relational graph convolutional network (R-GCN) is proposed to deal with different types of relations (edges) in a graph. Essentially, a forward computation of a node in R-GCN is performed as

$$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} \boldsymbol{W}_r^{(l)} \boldsymbol{h}_j^{(l)} + \boldsymbol{W}_0^{(l)} \boldsymbol{h}_i^{(l)}\right), \tag{12}$$

where $\mathcal{N}_i^r$ denotes the set of neighbor indices of node i under relation $r \in R$. $\frac{1}{c_{i,r}}$ is a normalized constant which can be predetermined or learned, according to [72].

Besides feature aggregation by node type and edge type, Wang *et al.* proposed HetGAN [73] and used the concept of meta-path to select neighbors, where meta-path indicates composite relation and is able to represent some semantic relationship, e.g., Author-Paper-Author is one kind of meta-path in an academic graph and represents a co-author relationship. HetGAN uses node-level attention and semantic-level (meta-path level) attention to learn the importance of each node and each meta-path, respectively.

$$\boldsymbol{z}_i^{\Phi} = \sigma(\sum_{j \in \mathcal{N}_i^{\Phi}} \alpha_{ij}^{\Phi} \cdot \boldsymbol{h}_j), \tag{13}$$

where $\boldsymbol{z}_i^{\Phi}$ is the embedding of node i for meta-path Φ. $\mathcal{N}_i^{\Phi}$ denotes the meta-path based neighbors of node i. α_{ij}^{Φ} is weight coefficient which is calculated through attention mechanism. Feature aggregation via Equation (13) is based on single meta-path, which is semantic-specific and able to capture a particular kind of semantic information. To combine different semantic information reflected by different meta-paths, the importance of each semantic specific embedding should be identified, which is calculated as:

$$w_{\Phi_i} = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \mathbf{q}^{\top} \cdot \tanh\left(\mathbf{W} \cdot \boldsymbol{z}_i^{\Phi} + \mathbf{b}\right), \tag{14}$$

which essentially is a non-linear transformation with an attention vector $\boldsymbol{q}$. Then, the importance coefficient is normalized by softmax function. One limitation of HetGAN is that the meta-path should be pre-defined manually, which might not be able to capture all meaningful meta-paths. Yun *et al.* proposed Graph Transformer Networks (GTN) [74], in which meta-path is represented by matrix multiplication of soft adjacency matrices.

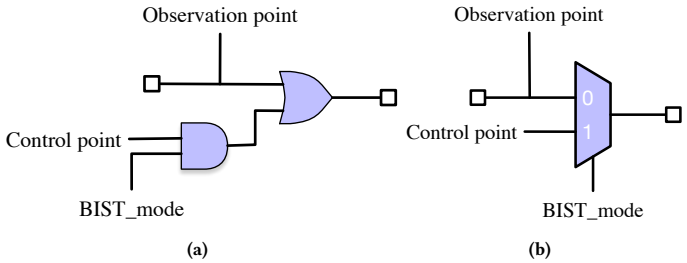

Figure 6: Example of the test point insertion: (a) TPI with AND/OR gate; (b) TPI with a multiplexer.

4 CASE STUDIES

In this section, we present two case studies on applying graph learning into EDA applications, including test point insertion and timing model selection.

4.1 Test Point Insertion

4.1.1 Problem Background. Built-in self test (BIST) is an important technique in design-for-testing, whose purpose is to design additional features into integrated circuits to allow them to perform self-testing such that the controllability and the observability can be improved. Test point insertion (TPI) is a broadly used approach that involves adding extra control points (CPs) or observation points (OPs) to the circuit. CPs can be used for setting signal lines to desired logic values, while OPs are added as scan cells to make a node observable. An example demonstrating two kinds of insertion is given in Figure 6. In this case study, GCN is applied to performs observation points insertion in a given netlist to improve the observability of a design. Essentially, it can be cast as a binary classification problem which is to determine whether an observation point should be added on the output port or not for each gate in the design. A comprehensive study is in [29].

Table 1: Statistics of test point insertion benchmarks

Design	Bench1	Bench2	Bench3	Bench4
#Nodes	1384264	1456453	1416382	1397586
#Edges	2102622	2182639	2137364	2124516

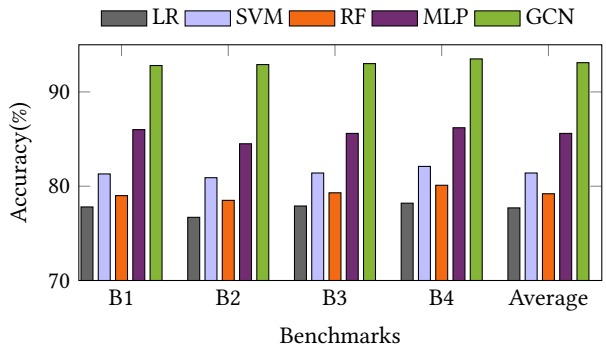

Figure 7: Accuracy comparison with classical machine learning algorithms.

4.1.2 Implementation Details. SCOAP [75] is leveraged to set initial features for each node, which is a classical quantitative heuristic measurement for testability evaluation. Specifically, each node is associated with a four-dimensional vector $[LL, C0, C1, O]$. LL represents the logic level of the corresponding gate. $[C0, C1, O]$ correspond to controllability-0, controllability-1 and observability, respectively, which are calculated with SCOAP method.

In order to demonstrate the superiority of the GCN model, we compare the classification accuracy between GCN and another four classical learning models, including logistic regression (LR), random forest (RF), support vector machine (SVM) and multi-layer perceptron (MLP). Since classical machine learning models require handcrafted features extracted from a graph, neighborhood features are manually integrated by collecting the features of the nodes in the fan-in cone and fan-out cone. 500 nodes in fan-in cone and 500 nodes in fan-out cone are collected using breadth-first-search. Every time a node is visited, the feature of this node is concatenated to the current feature vector. The node embedding generation is conducted similar to GraphSAGE, consisting of three aggregation layers and three encoding layers whose dimensions are 32, 64 and 128, respectively. The classification is performed with a set of fully-connected layers whose dimensions are 64, 64, 128 and 2. Four industrial benchmarks (Bench1 – Bench4) are used in the experiments, whose statistics are shown in Table 1. It can be seen that the sizes of graphs are all in million scale. To preserve the evaluation principle of a machine learning model, each time three designs are used for training and the remaining one is used for testing. The accuracy comparison is presented in Figure 7. GCN achieves significantly higher accuracy than all other classical machine learning models on average for all test designs.

Data visualization can facilitate us to justify whether the representation of a node is discriminative or not. Furthermore, we visualize different node embedding generated with different network depth, which denotes the representation after integrating features of the nodes in 1-hop neighborhood, 2-hop neighborhood and 3-hop neighborhood, respectively. In this experiment, we visualize the feature representation obtained from different encoders using t-SNE [76] for 1000 nodes, including 500 positive nodes and 500 negative nodes, as shown in Figure 8. It can be observed that the representations obtained for positive class and negative class become more discriminative as search depth increases.

4.2 Timing Model Selection

4.2.1 Problem Background. Gate sizing is a commonly used method to optimize the timing of a circuit, which is an intermediate step to resize instances. In modern design flow, different nets require different models for delay calculation, such as wire delay model and buffer delay model with various parameters, to achieve accurate outputs, shown as Figure 9. Conventional gate sizing flow suffers from the inaccurate selection of delay model for each net in the circuit, which relies on heuristics and is usually conservative. In this case study, our goal is to train a classification model such that the selection can be more accurate than heuristics.

Table 2: Statistics of timing model selection benchmarks

Design	#Nodes	#Edges	#POS	#NEG
D1	49559	109118	2961	46598
D2	46548	105534	2168	44380
D3	45986	95423	2783	43203
D4	41943	90992	1808	40135

4.2.2 Experiment Details. A netlist is represented as directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Each node $v \in \mathcal{V}$ represents a driver node and each edge $e \in \mathcal{E}$ represents the connection between two nodes. The dataset consists of four 7nm designs. An initial attribute feature vector with a dimension of 14 is supplied to each driver node, including fan-out number, instance location, sensitivity, slew, arrival time, slack, capacitance and resistances of net and sink, and delays of net and arc. Labels of nodes are generated by comparing and analyzing the netlists before and after the global optimization step in an industrial tool. Statistic of designs are summarized in Table 2. #POS and #NEG represent the number of nets using buffer delay model and wire RC delay model, respectively.

Given the dataset, a GCN can be trained based on the GraphSAGE-like framework [23]. A single GCN is trained for this task, which contains two steps of aggregation-encoding process and a fully connected layer with hidden dimension of 64. Similar to many classification problems in EDA applications, data imbalance is a severe issue. To resolve that, a two-stage GCN is leveraged which is similar to [29], and both models share the same structure. After the first GCN model is trained, the parameters of the first model are fixed and the second one starts to be trained, which is initialized by the parameters obtained from the first stage.

Table 3 shows the results of numerical sizing baseline, single-stage GCN and two-stage GCN. In every round of train-and-test, we select one design as the testing dataset, while the other three designs are split into training and validating dataset. The results show that the two-stage GCN achieves the highest F1-score among the three methods, which demonstrates the effectiveness of the GCN approach.

5 CONCLUSION

In this paper, we discussed a few key techniques of extending deep learning approaches to handle irregular structure data and highlight several challenges that are commonly encountered in EDA applications. Two case studies on timing model selection and

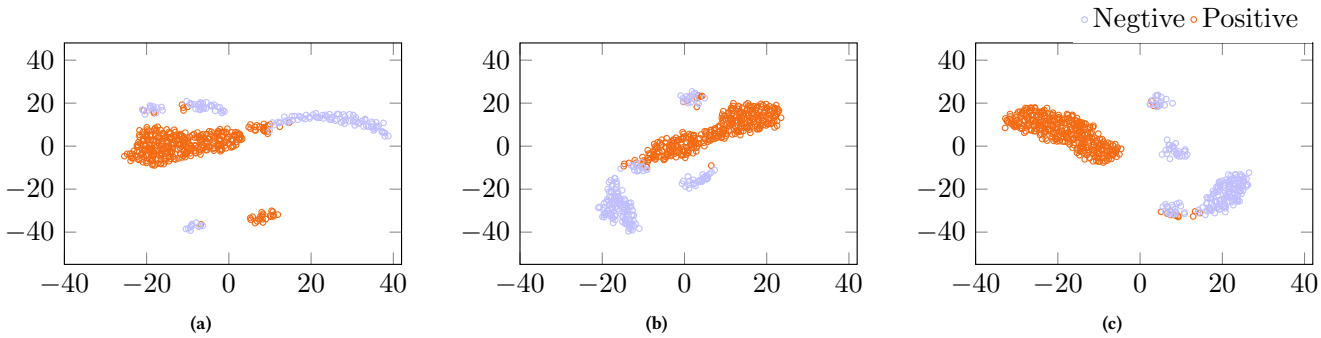

Figure 8: Visualization of node embedding with different search depth L. (a) L=1; (b) L=2; (c) L=3.

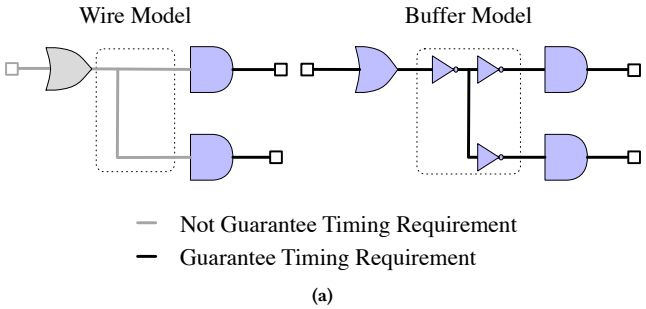

Figure 9: Example of timing model selection. (a) Wire RC delay model; (b) Buffer delay model.

Table 3: Results on the benchmarks

Design	Model	F1-score	Precision	Recall
D1	Baseline	0.502	0.597	0.433
	GCN	0.561	0.466	0.706
	GCN-2	**0.581**	0.523	0.652
D2	Baseline	0.529	0.528	0.530
	GCN	0.462	0.326	0.791
	GCN-2	**0.574**	0.532	0.623
D3	Baseline	0.527	0.660	0.438
	GCN	0.526	0.396	0.782
	GCN-2	**0.538**	0.437	0.699
D4	Baseline	0.549	0.542	0.556
	GCN	0.497	0.364	0.785
	GCN-2	**0.556**	0.454	0.715
Average	Baseline	0.527	0.582	0.490
	GCN	0.511	0.388	0.766
	GCN-2	**0.565**	0.493	0.669

test point insertion demonstrated the promising functionalities of graph learning in the circuits design domain.

Despite that significant improvements have been achieved, there are still lots of mysteries to be uncovered. For example, paths in a graph can reveal important properties of a graph (e.g., critical paths in a circuit), which is distinct from current developments based on neighborhood aggregating. Dealing with paths in graph with learning techniques may potentially broaden the availability of graph learning in the EDA domain. In addition, conventional learning algorithms focused on classification or regression tasks which typically cannot yield the final solution to a combinatorial optimization problem directly. Leveraging graph learning to solve combinatorial problems might be a new direction for graph learning to play a role in the EDA domain and beyond.

ACKNOWLEDGMENT

The authors would like to thank Dr. Qinghua Liu from Cadence Design Systems and Dr. Mark H. Ren from NVIDIA Research for their valuable support and insightful comments on the completion of case studies. This work is supported by The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

REFERENCES

[1] K.-C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "Dag-map: Graph-based fpga technology mapping for delay optimization," *IEEE Design & Test of Computers*, vol. 9, no. 3, pp. 7–20, 1992.
[2] K.-T. Cheng and C.-J. Lin, "Timing-driven test point insertion for full-scan and partial-scan BIST," in *Proc. ITC*, 1995, pp. 506–514.
[3] N. Selvakkumaran and G. Karypis, "Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization," *IEEE TCAD*, vol. 25, no. 3, pp. 504–517, 2006.
[4] B. Hu and M. Marek-Sadowska, "Fine granularity clustering-based placement," *IEEE TCAD*, vol. 23, no. 4, pp. 527–536, april 2004.
[5] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE TCAD*, vol. 29, pp. 939–952, June 2010.
[6] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE TCAD*, vol. 34, no. 3, pp. 433–446, March 2015.
[7] D. Z. Pan, B. Yu, and J.-R. Gao, "Design for manufacturing with emerging nano-lithography," *IEEE TCAD*, vol. 32, no. 10, pp. 1453–1472, 2013.
[8] M. Cho and D. Z. Pan, "BoxRouter: a new global router based on box expansion and progressive ILP," in *Proc. DAC*, 2006, pp. 373–378.
[9] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert, and D. Z. Pan, "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," *IEEE TCAD*, 2017.
[10] H. Li, W.-K. Chow, G. Chen, E. F. Young, and B. Yu, "Routability-driven and fence-aware legalization for mixed-cell-height circuits," in *Proc. DAC*, 2018, pp. 1–6.
[11] G. Chen and E. F. Young, "Salt: provably good routing topology by a novel steiner shallow-light tree algorithm," *IEEE TCAD*, 2019.
[12] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," *IEEE TCAD*, vol. 33, no. 3, pp. 397–408, March 2014.
[13] H. Zhang, B. Yu, and E. F. Y. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *Proc. ICCAD*, 2016, pp. 47:1–47:8.
[14] H. Geng, W. Zhong, H. Yang, Y. Ma, J. Mitra, and B. Yu, "Sraf insertion via supervised dictionary learning," *IEEE TCAD*, 2019.

[15] W.-H. Chang, L.-D. Chen, C.-H. Lin, S.-P. Mu, M. C.-T. Chao, C.-H. Tsai, and Y.-C. Chiu, "Generating routing-driven power distribution networks with machine-learning technique," in *Proc. ISPD*, 2016, pp. 145–152.
[16] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A pareto driven machine learning approach," *IEEE TCAD*, vol. 38, no. 12, pp. 2298–2311, 2018.
[17] C.-W. Pui, G. Chen, Y. Ma, E. F. Young, and B. Yu, "Clock-aware ultrascale fpga placement with machine learning routability prediction," in *Proc. ICCAD*, 2017, pp. 929–936.
[18] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018, pp. 131:1–131:6.
[19] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," *IEEE TCAD*, vol. 38, no. 6, pp. 1175–1187, 2019.
[20] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. ICCAD*, 2018, pp. 80:1–80:8.
[21] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "DeePattern: Layout pattern generation with transforming convolutional auto-encoder," in *Proc. DAC*, 2019, pp. 148:1–148:6.
[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *Proc. ICLR*, 2016.
[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NIPS*, 2017, pp. 1024–1034.
[24] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. KDD*, 2018, pp. 974–983.
[25] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proc. CVPR*, 2017, pp. 5410–5419.
[26] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. NIPS*, 2018, pp. 6410–6421.
[27] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *Proc. ICML*, 2018, pp. 5694–5703.
[28] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *Proc. ICML*, 2018, pp. 1123–1132.
[29] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High performance graph convolutional networks with applications in testability analysis," in *Proc. DAC*, 2019, p. 18.
[30] R. J. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *Proc. DAC*, 1990, pp. 613–619.
[31] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*, 2010, pp. 24–40.
[32] C. J. Alpert, A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hypergraph partitioning with fixed vertices," *IEEE TCAD*, vol. 19, no. 2, pp. 267–272, 2000.
[33] B. Yu, X. Xu, J.-R. Gao, Y. Lin, Z. Li, C. Alpert, and D. Z. Pan, "Methodology for standard cell compliance and detailed placement for triple patterning lithography," *IEEE TCAD*, vol. 34, no. 5, pp. 726–739, May 2015.
[34] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE TC*, vol. 100, no. 8, pp. 677–691, 1986.
[35] J. Cong and P. H. Madden, "Performance driven global routing for standard cell design," in *Proc. ISPD*, vol. 14, no. 16, 1997, pp. 73–80.
[36] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow," in *Proc. ISPD*, 2000, pp. 19–25.
[37] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE TCAD*, vol. 1, no. 1, pp. 25–35, 1982.
[38] K. Yuan, J.-S. Yang, and D. Z. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," *IEEE TCAD*, vol. 29, no. 2, pp. 185–196, Feb. 2010.
[39] H.-Y. Chang and I. H.-R. Jiang, "Multiple patterning layout decomposition considering complex coloring rules," in *Proc. DAC*, 2016, pp. 40:1–40:6.
[40] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," in *Proc. ICCAD*, 2017, pp. 81–88.
[41] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.
[42] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.
[43] B. Yu and D. Z. Pan, "Layout decomposition for quadruple patterning lithography and beyond," in *Proc. DAC*, 2014, pp. 53:1–53:6.
[44] 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 *Proc. ICCAD*, 2013, pp. 163–169.
[45] C. K. Cheng, S. Z. Yao, and T. C. Hu, "The orientation of modules based on graph decomposition," *IEEE TC*, vol. 40, pp. 774–780, June 1991.
[46] C.-W. Sham, F. Y. Young, and C. Chu, "Optimal cell flipping in placement and floorplanning," in *Proc. DAC*, 2006, pp. 1109–1114.
[47] J. Kuang and E. F. Y. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proc. DAC*, 2013, pp. 69:1–69:6.
[48] Y. Yang, W.-S. Luk, D. Z. Pan, H. Zhou, C. Yan, D. Zhou, and X. Zeng, "Layout decomposition co-optimization for hybrid e-beam and multiple patterning lithography," *IEEE TCAD*, vol. 35, no. 9, pp. 1532–1545, 2016.
[49] M. Cho and D. Z. Pan, "BoxRouter: a new global router based on box expansion and progressive ILP," *IEEE TCAD*, vol. 26, no. 12, pp. 2130–2143, 2007.
[50] Y. Lin, X. Xu, B. Yu, R. Baldick, and D. Z. Pan, "Triple/quadruple patterning layout decomposition via linear programming and iterative rounding," *JM3*, vol. 16, no. 2, 2017.
[51] R. Samanta, J. Hu, and P. Li, "Discrete buffer and wire sizing for link-based non-tree clock networks," *IEEE TVLSI*, vol. 18, no. 7, pp. 1025–1035, 2009.
[52] A. B. Kahng, S. Kang, H. Lee, S. Nath, and J. Wadhwani, "Learning-based approximation of interconnect delay and slew in signoff timing tools," in *Proc. SLIP*, 2013, pp. 1–8.
[53] W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. D. MacDonald, and S. Nath, "Learning-based prediction of embedded memory timing failures during initial floorplan design," in *Proc. ASPDAC*, 2016, pp. 178–185.
[54] Z. Qi, Y. Cai, and Q. Zhou, "Accurate prediction of detailed routing congestion using supervised data learning," in *Proc. ICCD*, 2014, pp. 97–103.
[55] Q. Zhou, X. Wang, Z. Qi, Z. Chen, Q. Zhou, and Y. Cai, "An accurate detailed routing routability prediction model in placement," in *Proc. ASQED*, 2015, pp. 119–122.
[56] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications," *IEEE TKDE*, vol. 30, no. 9, pp. 1616–1637, 2018.
[57] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.
[58] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.
[59] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.
[60] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.
[61] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.
[62] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
[63] A. Bojchevski, J. Klicpera, B. Perozzi, M. Blais, A. Kapoor, M. Lukasik, and S. Günnemann, "Is pagerank all you need for scalable graph neural networks?" 2019.
[64] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 974–983.
[65] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng, "Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding," *arXiv preprint arXiv:1910.02370*, 2019.
[66] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.
[67] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proc. AAAI*, vol. 33, 2019, pp. 3558–3565.
[68] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *arXiv preprint arXiv:1901.08150*, 2019.
[69] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, "Hypergcn: A new method for training graph convolutional networks on hypergraphs," in *Advances in Neural Information Processing Systems*, 2019, pp. 1509–1520.
[70] T.-H. H. Chan and Z. Liang, "Generalizing the hypergraph laplacian via a diffusion process with mediators," *Theoretical Computer Science*, 2019.
[71] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proc. KDD*. ACM, 2019, pp. 793–803.
[72] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.
[73] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network." ACM, 2019, pp. 2022–2032.
[74] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," in *Proc. NIPS*, 2019, pp. 11 960–11 970.
[75] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia controllability/observability analysis program," in *Proc. DAC*, 1980, pp. 190–196.
[76] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.