Floorplanning with Edge-aware Graph Attention Network and Hindsight Experience Replay

BO YANG, University of Science and Technology of China, Hefei, China
QI XU, University of Science and Technology of China, Hefei, China
HAO GENG, ShanghaiTech University, Shanghai, China
SONG CHEN, University of Science and Technology of China, Hefei, China
BEI YU, The Chinese University of Hong Kong, NT, Hong Kong SAR
YI KANG, University of Science and Technology of China, Hefei, China

In this article, we focus on chip floorplanning, which aims to determine the location and orientation of circuit macros simultaneously, so the chip area and wirelength are minimized. As the highest level of abstraction in hierarchical physical design, floorplanning bridges the gap between the system-level design and the physical synthesis, whose quality directly influences downstream placement and routing. To tackle chip floorplanning, we propose an end-to-end reinforcement learning (RL) methodology with a hindsight experience replay technique. An edge-aware graph attention network (EAGAT) is developed to effectively encode the macro and connection features of the netlist graph. Moreover, we build a hierarchical decoder architecture mainly consisting of transformer and attention pointer mechanism to output floorplan actions. Since the RL agent automatically extracts knowledge about the solution space, the previously learned policy can be quickly transferred to optimize new unseen netlists. Experimental results demonstrate that, compared with state-of-the-art floorplanners, the proposed end-to-end methodology significantly optimizes area and wirelength on public GSRC and MCNC benchmarks.

CCS Concepts: • **Hardware → Partitioning and floorplanning;**

Additional Key Words and Phrases: Floorplanning, Reinforcement Learning, Graph Attention Network, Transformer

ACM Reference Format:
Bo Yang, Qi Xu, Hao Geng, Song Chen, Bei Yu, and Yi Kang. 2024. Floorplanning with Edge-aware Graph Attention Network and Hindsight Experience Replay. *ACM Trans. Des. Autom. Electron. Syst.* 29, 3, Article 56 (May 2024), 17 pages. https://doi.org/10.1145/3653453

This work is supported by USTC Research Funds of the Double First-Class Initiative under Grant YD2100002012. The authors would like to thank Information Science Laboratory Center of USTC for the hardware & software services.
Authors' addresses: B. Yang, Q. Xu (Corresponding author), Song Chen, and Y. Kang, School of Microelectronics, University of Science and Technology of China, Hefei, China; e-mails: yangbo19@mail.ustc.edu.cn, xuqi@ustc.edu.cn, songch@ustc.edu.cn, ykang@ustc.edu.cn; H. Geng, School of Information Science and Technology, ShanghaiTech University, Shanghai, China; e-mail: genghao@shanghaitech.edu.cn; B. Yu, Department of Computer Science and Engineering, The Chinese University of Hong Kong, NT, Hong Kong SAR; e-mail: byu@cse.cuhk.edu.hk.

ACM 1084-4309/2024/05-ART56
https://doi.org/10.1145/3653453

1 INTRODUCTION

In modern chip design, the back-end physical design dramatically affects the final performance of **integrated circuit (IC)** chips. Therefore, the earlier the physical design elements are considered in the design phase, the better the design optimization of the high-level architecture will be. As the first stage in the IC physical design, the input to floorplanning is a circuit netlist containing a set of macros with various sizes and their interconnection relationships with the aim of determining the location and orientation of macros simultaneously to optimize chip area and wirelength metrics.

Floorplanning has been proven to be an NP-hard problem with a vast solution space [1], even more than the number of atoms in the universe. Thus, exploring high-quality floorplan solutions in polynomial time is a big challenge. As a result, we need to rely on significantly specialized knowledge to design exact or approximate algorithms for solving floorplanning. Generally, traditional floorplanning methods are roughly divided into two categories. One category is the handcrafted heuristics, such as the partitioning method based on min-cut [2] and **simulated annealing (SA)**-based algorithm [3, 4]. However, these approaches cannot guarantee good performance and converge too slowly for large-scale netlists. Another class is the non-linear analytical solvers, i.e., ePlace [5] and RePlAce [6]. Through modeling the objective metrics as a differentiable function, a gradient descent algorithm is directly adopted to achieve the optimization. Although analytical solvers are widely used due to the short runtime and ability to address complex constraints, the solvers are not suitable for handling netlists with various macro sizes. In addition, both types of floorplanning approaches lack the transferability, meaning that the prior learned knowledge cannot be generalized to new unseen netlists. Hence, it is necessary to optimize the floorplan for each new design from scratch, significantly limiting the reduction in runtime.

Over the past decade, considerable improvements in computer systems and hardware have extensively promoted the development of **artificial intelligence (AI)** technologies. Meanwhile, AI techniques also support the relevant hardware design and shorten the development cycle. Recently, **reinforcement learning (RL)**-based approaches have been developed to solve chip floorplanning. He et al. [7] exploits a deep Q-learning algorithm to explore local search heuristics for floorplanning, achieving higher efficiency and better results than the SA method. The main idea of the work is to replace the probabilistic sampling process of solutions in SA with an RL strategy for space exploration. Based on a similar scheme, GoodFloorplan [8] further introduces the graph neural network to realize more efficient information encoding. But due to the characteristic of local heuristics, these methods are not able to transfer learned experience to unseen circuits and thus do not show great advantages over the traditional approach. Google's pioneered work proposes a deep RL strategy to address the macro placement [9], analogous to playing Go [10]. It first discretizes the chip canvas to grid cells, then the RL agent places macros onto the placable cells sequentially, and finally the reward signal is calculated to train the RL agent. Besides, to reduce the redundant output of the grid-based action space, Amini et al. [11] uses the **corner block list (CBL)** representation to encode the actions. However, this approach ignores the connection between actions, and the learning process becomes a partially observable **Markov Decision Process (MDP)** [12], making it difficult to generalize a reasonable policy due to the lack of information.

Although recent works have investigated RL-based floorplanning, these arts face training inefficiencies due to the high complexity of the floorplanning and the sparse reward challenge. Fundamentally different from prior methods, We propose an end-to-end RL methodology with a hindsight experience replay technique, which can efficiently explore the solution space and learn from previous poor-quality floorplans. An **edge-aware graph attention network (EAGAT)** is developed as an encoder to extract the netlist information. Compared with the traditional **graph neural network (GNN)**, the proposed EAGAT can not only encode the netlist's connection relations but

also realize the information interaction between macros. Besides, we build a hierarchical decoder based on the encoder part of Transformer architecture [13] to carry out the multi-actions. Because the aggregation pattern in transformer is global and input-dependent, each macro can capture the global information on the current floorplan structure. Our main contributions are summarized as follows:

— An end-to-end RL-based floorplanning methodology is proposed to greatly reduce the action space without representation loss, including a novel representation state and multi-action mechanism.
— An **edge-aware graph attention network (EAGAT)** is developed to encode edge information between macros (nodes) and further realize the feature interactions between nodes and edges.
— A transformer-based hierarchical policy network is leveraged to output multi-actions, enabling each macro to capture the entire floorplan structure information.
— A hindsight experience replay technique is presented for chip floorplanning, which allows sample-efficient learning from a huge floorplan solution space.

The rest of this article is organized as follows: Section 2 presents the preliminaries and then gives the floorplanning problem formulation. Section 3 describes the proposed end-to-end floorplanning methodology in detail. Section 4 records the experimental results, followed by the conclusions in Section 5.

2 PRELIMINARIES

2.1 Reinforcement Learning

Reinforcement learning can be formulated as an infinite-horizon **Markov Decision Process (MDP)**, which is defined by a tuple $(S, A, p_0, p, R, \gamma)$. Considering an MDP, the initial state $s_0 \in S$ is sampled from an initial distribution p_0. At each timestep t, the agent takes an action $a_t \in A$ according to the policy $\pi(a_t|s_t)$. When the environment receives a_t, it produces a reward signal $r_t \in R$ and transfers to a next state $s_{t+1} \in S$ based on the transition distribution p. The goal of RL is to optimize the expected discounted cumulative return $E_{\pi_\theta}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t]$, where θ denotes the network parameters and $\gamma \in (0, 1)$ is the discount factor.

2.2 Floorplanning Formulation

Let $M = \{m_1, \ldots, m_n\}$ be a set of rectangular macros, where each macro m_i has a width w_i and a height h_i, and $N = \{n_1, \ldots, n_m\}$ indicates a netlist, which describes the connection relations among macros. Besides, let o_i represent the orientation of macro m_i; $o_i = 1$ if the macro is rotated by 90 degrees, else $o_i = 0$. A floorplan f needs to simultaneously assign the coordinate (x_i, y_i) and the orientation o_i to each macro m_i to minimize area and wirelength metrics with no overlap between any two macros. The objective function of floorplanning is defined in Equation (1):

$$\min_f \ \mathrm{A}(f) + \eta \cdot \mathrm{W}(f), \tag{1}$$

where η is a coefficient, and $\mathrm{A}(\cdot)$ and $\mathrm{W}(\cdot)$ refer to the area and wirelength function, respectively. The wirelength is estimated by the **half-perimeter wirelength (HPWL)** model, which is the most common approximation method for wirelength. The specific calculation of HPWL in Reference [14] is as follows:

$$HPWL = \sum_i \left\{ \left(\max_{b \in i}\{x_b\} - \min_{b \in i}\{x_b\} \right) + \left(\max_{b \in i}\{y_b\} - \min_{b \in i}\{y_b\} \right) \right\}, \tag{2}$$

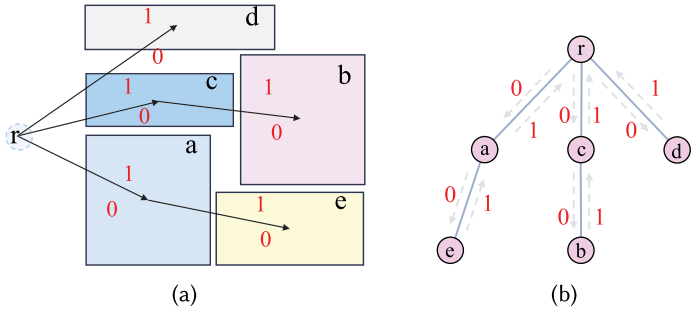


Fig. 1. (a) The O-tree representation of a floorplan with five blocks is (0011001101, *aecbd*) and the root r is a dummy node; (b) the encoding of the corresponding O-tree.

where i and b are the indexes of the hypernet and the macro, and x_b and y_b represent the coordinates of the macro b.

2.3 O-tree Representation

O-tree is a rooted ordered tree representing a non-slicing floorplan, which is encoded as a tuple (T, π) [15]. The bit string T is a realization of the tree structure, where a "0" means a traversal descends an edge in the tree, and a "1" indicates when it ascends an edge in the tree. The permutation π is the label sequence when we traverse the tree in the **depth-first search (DFS)** order. Thus, given a floorplan with n rectangular blocks, $n(2 + \lceil \lg n \rceil)$ bits are required to store the tuple with $2n$ bits for T and $n \lg n$ bits for π. Given an O-tree representation, it takes only linear time to construct the placement and its constraint graph, i.e., $O(n)$. An example of O-tree representation is shown in Figure 1(a).

3 END-TO-END METHODOLOGY

In this section, we elaborate the proposed end-to-end floorplanning methodology. First, a reinforcement learning model is built for the floorplanning problem. Then, the developed end-to-end floorplanning network is illustrated in detail. Finally, a novel reinforcement learning technique called hindsight experiential replay is fully described.

3.1 Reinforcement Learning Formulation

3.1.1 Netlist Graph. As depicted in Section 2.2, a netlist is a description of the connectivity of an integrated circuit with each net containing macros that need to be interconnected by metal wires in the physical design. Generally, it is difficult for deep neural networks to directly extract the connection relationships in netlist. But, inspired by the simple fact that circuit netlist is a graph, as shown in Figure 2, the circuit topology representation can be generated through the graph neural network. Occasionally, a metal wire may connect more than two macros in a net. To represent the case in the graph, an edge is introduced between any two macros to form a clique [16]. For example, a net with k macros builds a k-clique, and all the edges in the clique receive a weight of $2/k$. We define the summation of the weights received by an edge as the edge's connection degree. Thus, the circuit netlist is represented by an undirected graph $G(V, E)$, where V denotes the set of circuit macros (nodes), while E is an $n \times n$ adjacency matrix with $E_{ij} = c_{ij}$ if nodes i and j are connected by an edge (c_{ij} refers to the connection degree), otherwise $E_{ij} = 0$.

3.1.2 Action Design. Considerable action space brings a critical challenge for deep reinforcement learning algorithms. Thus, to apply reinforcement learning to the floorplanning task with

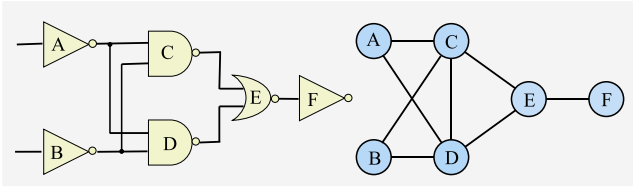


Fig. 2. The graph structure of a gate-level circuit.

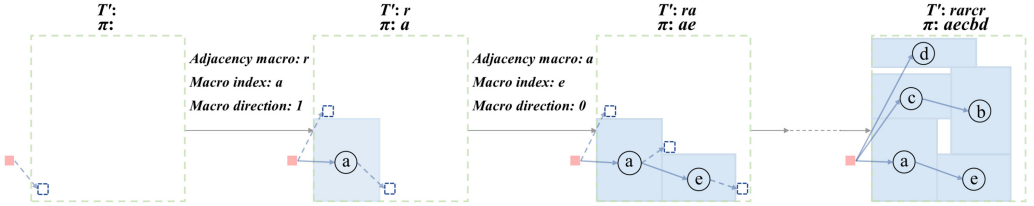


Fig. 3. The diagram of the proposed MDP model, where the RL agent sequentially places circuit macros onto the chip.

high computation complexity, it is crucial to design a suitable action mechanism to simplify the action space. In the past few decades, several representations methods for non-slicing floorplan structures have been presented, including Sequence Pair [1], O-tree [15], B*-tree [17], **Transitive Closure Graph with a packing sequence (TCG-S)** [18], and **Corner Block List (CBL)** [19]. In this work, we choose the well-designed O-tree representation, which is more concise than others. Figure 1(b) depicts the horizontal O-tree for the non-slicing floorplan shown in Figure 1(a). However, the O-tree floorplan representation must be encoded as a $2(n-1)$ bits string T and a permutation sequence π obtained by the DFS on the tree. To simplify the representation, we replace the bit string T with another string T' to fit the output of the neural network. For example, in Figure 1(b), given the root (virtual) node r representing the left boundary of the floorplan, string T' is $\{rarcr\}$, where the ith component refers to the parent node of the ith element in π. As a result, only $2n\lceil \lg n \rceil$ bits are used to store the new tuple (T', π). Meanwhile, based on the new representation, we can directly determine the positions of each macro on the floorplan.

Since the RL agent sequentially places circuit macros onto the chip, we propose a multi-action mechanism to determine the position and orientation of a macro at each timestep. That is, for a current macro to be placed, we should establish its position relationship with all placed macros and simultaneously determine its placed orientation. For a better understanding, Figure 3 illustrates the sequential decision process to progressively generate the tuple (T', π). At each timestep, the first sub-action is to select one macro, called **adjacency macro**, whose bottom-right corner is the new insertion position. It is important to ensure that the **Depth-First Search (DFS)** order is maintained while traversing the positions. The next sub-action determines which macro should be placed at the current step. This selection is referred to as the **macro index**. Finally, the last sub-action, called **macro direction**, involves assigning an orientation to the chosen macro, i.e., rotated (1) or not (0).

3.1.3 Floorplan Structure Extraction. Given the original netlist graph G, the EAGAT encoder only extracts topological information between macros, while features of the floorplan structure cannot be perceived. Since the floorplan structure reflects the relative positions between macros, it directly influences the chip area. To capture the structure information, previous works [8, 9, 11] introduce the coordinate of the macro into the node feature. But, in fact, the receptive field of the

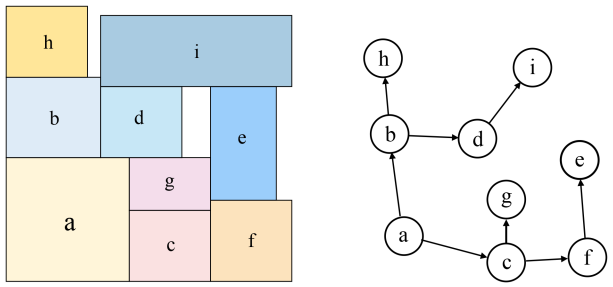


Fig. 4. The binary tree-based floorplan structure.

nodes in a graph neural network can only increase gradually with the number of layers. With a limited number of layers, even two closely placed macros may have no information interaction. In this article, we develop a trick to explicitly construct the structure information without altering the topological information extraction. As shown in Figure 4, the partial floorplan structure is expressed as a binary tree structure proposed in Reference [17]. For macro m_i with a width w_i and a height h_i, we specify the macro's coordinate as (x_i, y_i). If the macro m_j is located on the right-hand side and adjacent to macro m_i in the floorplan, i.e., $x_j = x_i + w_i$ and $y_j = y_i$, then we set m_j as the left child of m_i in the binary tree. Similarly, if m_j is the right child of m_i, then the macro m_j is located above and adjacent to the macro m_i, with $x_j = x_i$ and $y_j = y_i + h_i$.

After obtaining the binary tree-based floorplan structure, we directly convert it into another undirected graph $\hat{G}$, which is used to supplement the lack of floorplan structure information in netlist graph G. For example, if an edge in $\hat{G}$ connects node i and node j but is not attached in G, we then add a corresponding undirected edge in G. The connection degree of the added edge is set to zero, i.e., $c_{ij} = 0$, indicating the original connection relations among circuit macros are not broken. As a result, we can use the graph neural network to extract two kinds of features on the renewed G, namely, topological information and structural information. As the macros are placed sequentially, the netlist graph G will be updated at each timestep.

3.1.4 Reinforcement Learning Model. In this work, we target the chip floorplanning problem, and the RL agent sequentially places circuit macros to feasible positions on the layout, as presented in Figure 3. Based on the key components described above, the reinforcement learning model for floorplanning is defined as follows:

— **State.** Since the renewed netlist graph contains both topological and structural information, we use it to represent a state. As described in Section 3.1.3, the netlist graph is dynamically updated at each timestep. To learn effective representations for the netlist graph, node and edge features are defined. The features of a node include the size and the coordinate of the corresponding macro. In addition, we add an extra dimension to indicate whether the macro has been placed and mask the coordinate information if the macro is not placed. The edge features reflect the connection degree between nodes.

— **Action.** As introduced in Section 3.1.2, an action consists of three sub-actions: placement position, ID of the macro to be placed (macro index), and orientation information. The placement position is represented by the adjacency macro, because we only need to determine which macro to place on the bottom-right to determine the placement position. Therefore, we only need to select an index of an unplaced macro to determine the position. Compared to the action design that generates the macros' coordinates directly, the multi-action

mechanism significantly reduces the parameter space of the policy network, accelerating the convergence of the end-to-end floorplanning methodology.

— **State Transition.** The policy decides the state transition, so it is deterministic.

— **Reward.** The floorplanning problem aims to minimize the weighted sum of area and wirelength, as defined in Equation (1). The most straightforward approach is to adopt negative objective function values as the reward signal in each floorplanning step. However, this leads to a sparse reward problem, one of the biggest challenges in reinforcement learning [20]. To address the problem, a dense reward strategy is designed, where the change in the normalized area gap of the partial floorplan over two consecutive timesteps is set as a reward signal, as shown in Equation (4).

$$W_t = \max_{i \in \mathcal{P}_t}(x_i + w_i), \tag{3a}$$

$$H_t = \max_{i \in \mathcal{P}_t}(y_i + h_i), \tag{3b}$$

$$g_t = \left(W_t \cdot H_t - \sum_{i \in \mathcal{P}_t} w_i \cdot h_i\right) / (W_t \cdot H_t), \tag{3c}$$

$$r_t = \begin{cases} g_{t-1} - g_t, & \text{if } t < n \\ g_{t-1} - g_t - \eta \cdot \mathrm{W}(f), & \text{if } t = n, \end{cases} \tag{3d}$$

where $\mathcal{P}_t$ is the set of macros already placed at timestep t, W_t (H_t) refers to the width (height) of the current floorplan, and n denotes the number of macros in the netlist. Equation (3c) calculates the normalized area gap g_t for the partial floorplan at timestep t. The reward r_t is defined as the cost difference between normalized area gaps over two consecutive timesteps. For the last timestep, r_t requires an additional negative wirelength value calculated by the wirelength function $\mathrm{W}(f)$, where f denotes the final floorplan structure. Thus, the cumulative reward at timestep t can be written as:

$$\sum_{t}^{n} r_t = g_{t-1} - g_n - \eta \cdot \mathrm{W}(f), \tag{4}$$

where η denotes a coefficient and the value is set to 0.5 during the training and inference. Based on the reward design, the agent can consistently achieve a meaningful reward signal, regardless of whether the objective cost changes.

3.2 Hierarchical Floorplanning Network

In this section, the proposed **Hierarchical Floorplanning Network (HFN)** is described in detail. As shown in Figure 5, HFN starts with an embedding layer, followed by an **edge-aware graph attention network (EAGAT)** encoder. Compared with the traditional graph attention model that only focuses on node-level features and cannot achieve feature interaction between nodes and edges, the proposed EAGAT encoder can effectively realize the feature interactions between nodes and edges. The detailed calculation processes are depicted in Equations (5) to (6). Then, based on the node features extracted by the EAGAT encoder, a transformer-based hierarchical policy network is performed to output three sub-actions; meanwhile, a **multi-layer perceptron (MLP)**-based value network provides a value estimation. Since the value network is omitted from Figure 5, we refer to the policy network as a decoder in the subsequent sections.

3.2.1 EAGAT Encoder. Most popular network architectures share a similar Embedding & GNN paradigm [21], which we refer to as the base model, as shown in the middle part of Figure 5. The

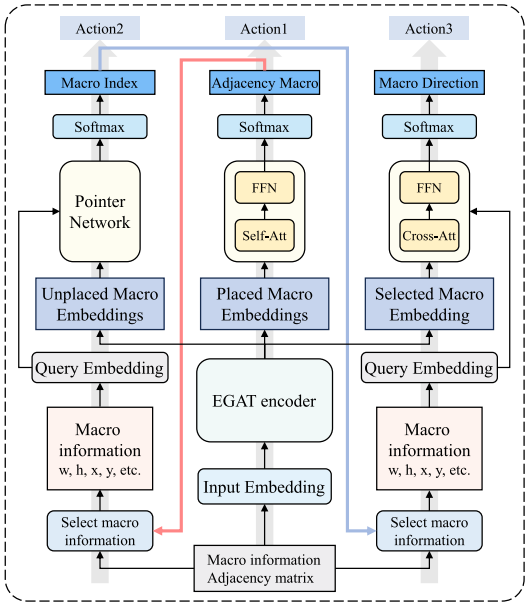


Fig. 5. The architecture of the Hierarchical Floorplanning Network (HFN). The arrows mark the forward process that produces three sub-actions in sequence.

input to HFN is a renewed netlist graph, which contains macro information and an adjacency matrix (edge information). Then, the embedding layer transforms the input low-dimensional numerical vectors into high-dimensional dense embeddings using linear layers. Given the generated node and edge embeddings, the developed EAGAT encoder further extracts the features of nodes and edges. Because edge features capture the connectivity information between nodes, when edge features are incorporated into the node representations, information interaction between highly connected nodes will be adequate. As a result, highly connected nodes are placed closely to achieve the wirelength optimization. Besides, we use the attention mechanism to perform the process of node feature aggregation. Compared with the average pooling operation, the aggregation mechanism adaptively learns the weights between a node and its neighborhoods. Figure 6 shows the proposed EAGAT encoder, where each layer l mainly consists of two calculation components: multi-head attention and position-wise feed-forward networks.

First, node embeddings are linearly projected into query $\boldsymbol{Q}^l$, key $\boldsymbol{K}^l$, and value $\boldsymbol{V}^l$. Similarly, edge embeddings are linearly projected into edge feature $\boldsymbol{E}^l$ and bias term $\boldsymbol{B}^l$. Then, we calculate the attention score of a node i with its neighbor node j in the kth head as follows:

$$
\begin{aligned}
a_{ij}^{k,l} &= \frac{\boldsymbol{Q}_i^{k,l^{\mathrm{T}}} \boldsymbol{K}_j^{k,l}}{\sqrt{d_k}} + B_{ji}^{k,l}, \\
s_{ij}^{k,l} &= \frac{\exp(a_{ij}^{k,l})}{\sum_{t \in \mathcal{N}_i} \exp(a_{it}^{k,l})},
\end{aligned}
\tag{5}
$$

where $\boldsymbol{Q}_i^{k,l}, \boldsymbol{K}_j^{k,l} \in \mathbb{R}^{d_k}$ are obtained by linear projection of input node embeddings $\boldsymbol{h}_i^l$ and $\boldsymbol{h}_j^l$. $B_{ji}^{k,l}$ is the kth element of the bias $\boldsymbol{B}_{ji}^l \in \mathbb{R}^H$ (H: the number of heads), which is formed by the learned linear transformation of the input edge embedding $\boldsymbol{e}_{ji}^l$. Since the bias term $B_{ji}^{k,l}$ is added to the

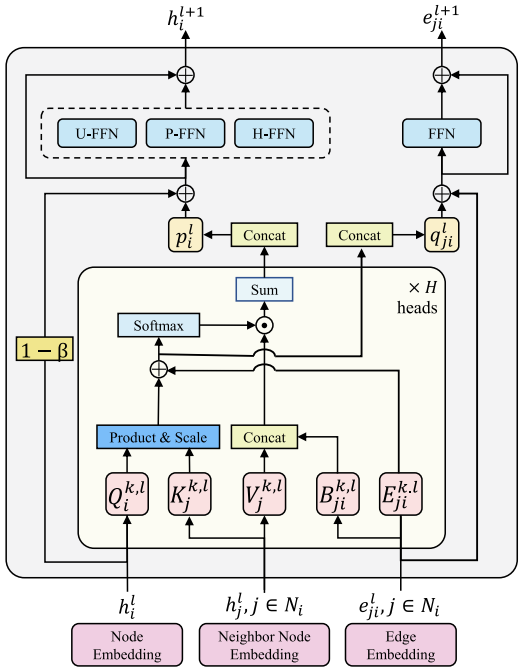


Fig. 6. The proposed EAGAT encoder.

scaled dot product between the queries and the keys, edge features influence the attention process. Besides, $\mathcal{N}_i$ denotes the set of the neighbors of node i in the netlist graph. The calculated attention score $s_{ij}^{k,l}$ indicates the weight on the edge from node j to node i.

Based on the attention mechanism, node and edge embeddings are updated as:

$$\begin{aligned} \boldsymbol{p}_i^{k,l} &= \sum_{j \in N_i} s_{ij}^{k,l} \text{concat}\left(\boldsymbol{V}_j^{k,l}, E_{ji}^{k,l}\right), \\ \boldsymbol{p}_i^{l} &= \text{concat}\left(\overset{H}{k=1} \boldsymbol{p}_i^{k,l}\right) + (1-\beta)\, \boldsymbol{h}_i^l, \\ \boldsymbol{q}_{ji}^{l} &= \text{concat}\left(\overset{H}{k=1} a_{ij}^{k,l}\right) + \boldsymbol{e}_{ji}^l, \end{aligned} \tag{6}$$

where $\boldsymbol{V}_j^{k,l} \in \mathbb{R}^{d_k}$ is the learned linear transformed node embedding. $E_{ji}^{k,l}$ is the kth element of the edge feature $\boldsymbol{E}_{ji}^l \in \mathbb{R}^H$, obtained by transforming edge embedding. $\boldsymbol{h}_i^l$ and $\boldsymbol{e}_{ji}^l$ refer to the input node and edge embedding of the layer l, and β denotes a learnable parameter. Besides, concat represents the concatenation operation.

Following the multi-head attention module, the feed-forward sublayer consists of two consecutive position-wise fully connected linear layers with a non-linear activation in between, such as Leaky_ReLU [22]. Note that at each timestep, two different types of nodes are fed into the HFN, i.e., placed and unplaced nodes. To produce specific information about different types of nodes, inspired by VLMo [23] and mixture-of-experts networks [24], we introduce a mixture of **modality experts (MoE)** as a substitute for the **feed-forward network (FFN)**, where each MoE block captures more interaction by switching to different modality experts. Three modality experts are defined: **unplaced expert (U-FFN), placed expert (P-FFN)**, and **hybrid expert (H-FFN)**, as shown in the dashed lines in Figure 6. In the shallow layer of the HFN architecture, we adopt

U-FFN and P-FFN to encode the corresponding node features, respectively. H-FFN is then used at the deep layer to capture more modality interaction. The final node and edge features are calculated as follows:

$$\begin{aligned} \boldsymbol{h}_i^{l+1} &= \mathrm{MoE}(\boldsymbol{p}_i^l) + \boldsymbol{p}_i^l, \\ \boldsymbol{e}_{ji}^{l+1} &= \mathrm{FFN}(\boldsymbol{q}_{ji}^l) + \boldsymbol{q}_{ji}^l, \end{aligned} \tag{7}$$

where MoE represents the defined three modality experts. Since edge features are inserted into the calculation of attention scores, which are further adopted to update both node and edge embeddings, the information interactions between nodes and edges are effectively realized. More importantly, due to the inner product operation, effective feature crosses are achieved, providing additional interaction information beyond individual features [25].

3.2.2 Transformer-based Hierarchical Policy Network. Given the extracted features of nodes in the netlist graph, the transformer-based hierarchical policy network needs to output three sub-actions, namely, placed position, ID of the macro to be placed, and orientation information, as depicted in Figure 5.

First, the encoder part of Transformer [13] is adopted to determine a placed position on the current partial floorplan, which contains a self-attention layer and a feed-forward network. The inputs to the transformer model are all legal positions that satisfy the DFS order, as described in Section 3.1.2. Since choosing a placed location corresponds to selecting one of all placed macros whose lower right corner is placed there, the output is a adjacency macro representing the first sub-action. Besides, we apply the softmax function on the network output to get the probability distribution of all legal positions.

Next, we exploit the pointer network [26] to determine which macro to be placed, i.e., to output the macro index as the second sub-action. The macro information chosen by the adjacency macro, which includes size and coordinates, is transformed into a query vector $\hat{\boldsymbol{q}} \in \mathbb{R}^{d_v}$ through a linear layer, while the all unplaced macro features from the encoder are mapped into key vectors. The calculation procedures of the pointer network are described in Equation (8).

$$u_i = \tanh\left(\frac{\hat{\boldsymbol{q}}^{\mathrm{T}}\hat{\boldsymbol{k}}_i}{\sqrt{d_v}}\right), \quad p_i = \frac{\exp(u_i)}{\sum_{j\in\mathcal{U}_t}\exp(u_j)}, \tag{8}$$

where $\hat{\boldsymbol{k}}_i \in \mathbb{R}^{d_v}$ is the transformed feature vector of the unplaced macro i, and $\mathcal{U}_t$ denotes the set of all unplaced macros at timestep t.

Finally, the cross-attention mechanism [13] is further employed to decide the orientation of the currently placed macro. We first permute the possible orientations and corresponding positions of the macro selected by the macro index and then linearly embed the information into queries. Meanwhile, the macro features output by the EAGAT encoder are linearly projected into keys and values. By performing the cross-attention with the softmax activation function, the probability distribution of all possible orientations is produced. Once the placed macro's orientation is achieved, we directly insert it into the placed position generated by the first sub-action.

Throughout the entire forward pass of placing one macro, the data stream passes sequentially through the encoder and the three parts of the decoder. Since the previous sub-actions are embedded as a query to the next sub-action decoder, the multi-action mechanism is still a strict MDP model.

3.3 Hindsight Experience Replay

Unlike current model-free RL algorithms, humans can learn almost as much from undesirable outcomes as from desired ones [27]. For example, a standard RL algorithm would learn little from the

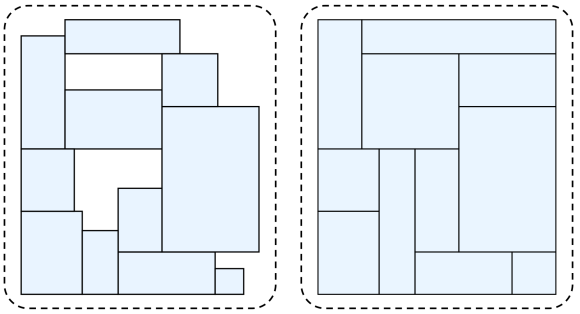


Fig. 7. The process of the hindsight experience replay for floorplanning.

cases of failed trajectories, hindering the exploration of solution space. The ineffective exploration is infinitely magnified in solving floorplanning with a vast solution space, resulting in a highly long learning time. To realize sample-efficient learning, a hindsight experience replay technique [27] is presented for floorplanning. The idea behind it is elementary: After experiencing some episodes, we store them in the replay buffer and change the sizes of the macros and the connections between them to reach a designed goal. The updated state is then passed back to the neural network and enforces the network to perform the previous actions repeatedly. Thus, the process is similar to supervised learning.

Figure 7 illustrates the process of updating a state. To satisfy area optimality, we constantly adjust each macro's top and right-hand edges to fill in as much whitespace as possible without violating the density constraint. Accordingly, to optimize the wirelength metric, closely placed macros are allocated to more connection degrees. In this case, swapping the position of any two macros will increase the wirelength, which has been proved in Reference [28]. Through these operations, the updated higher-quality floorplans are stored in the buffer. By the supervised learning method, we force RL agents to learn the stored knowledge and improve exploration efficiency. Since the hindsight experience replay technique can be combined with an arbitrary off-policy RL algorithm, no additional optimization tasks are introduced into our end-to-end floorplanning methodology.

4 EXPERIMENTAL RESULTS

This section evaluates the performance of the proposed end-to-end RL methodology using academic benchmarks and presents several ablation experiments.

4.1 Experiment Setting

The implementation of our methodology is in `Python` with `Pytorch` [29], and we execute it on a Linux server with one Nvidia GeForce RTX 3090 GPU. We first train the RL model using the hindsight experience replay technique and then test it on representative MCNC [30] and GSRC [31] benchmarks, including five MCNC circuits (ami33, ami49, apte, hp, and xerox), and six GSRC circuits (n10, n30, n50, n100, n200, and n300). Besides, we manually generated a larger netlist "design" to further verify the effectiveness of our method. The macros in the netlist have the same size ranges as the publicly available benchmarks n200 and n300, while the connection relationships are adjusted to be consistent with the actual netlist connections. We have open-sourced the netlist on GitHub.[1] Since the macro sizes in these benchmarks vary considerably, which poses a significant challenge to the end-to-end methodology, a fast post-processing is further adopted to fine-tune macros' coordinates and orientations.

[1] https://github.com/yangbo19/Floorplanning-with-EGAT-and-HER

Table 1. Comparison with State-of-the-arts on MCNC and GSRC Benchmarks
(Area Unit:$\times 10^6 \mu m^2$, Wire Unit:$\times 10^5 \mu m$, RT Unit:s)

Circuit	#Macro	#Net	SA			ICCD'20 [7]			KDD'22 [11]			Ours		
			Area	Wire	RT	Area	Wire	RT	Area	Wire	RT	Area	Wire	RT
apte	9	97	47.310	3.43	38	47.080	4.03	16	–	–	–	**47.080**	**3.22**	24
xerox	10	203	20.640	6.62	99	20.420	6.33	17	–	–	–	**20.403**	**5.80**	25
hp	11	83	9.400	2.62	44	**9.210**	**1.95**	12	–	–	–	9.233	1.98	31
ami33	33	123	1.275	**0.59**	82	1.240	0.69	43	–	0.82	–	**1.235**	0.64	102
ami49	49	408	39.053	14.22	165	38.650	17.24	67	–	13.75	–	**38.028**	**13.25**	242
n10	10	118	0.238	0.18	32	0.239	0.17	18	–	0.41	–	**0.234**	**0.14**	30
n30	30	349	0.228	0.48	67	0.223	0.49	52	–	1.12	–	**0.218**	**0.46**	74
n50	50	485	0.221	0.99	132.3	0.215	1.02	89	–	1.63	–	**0.211**	**0.95**	301
n100	100	576	0.205	1.54	396	0.195	1.55	389	–	3.37	–	**0.190**	**1.37**	1,513
n200	200	1,274	0.207	3.34	1,102	0.215	3.48	785	–	3.52	–	**0.197**	**3.26**	3,875
n300	300	1,632	0.329	5.44	2,062	0.340	5.25	3,767	–	**4.77**	–	**0.301**	4.86	8,322
design	500	2,746	0.694	8.12	5,203	0.688	7.72	6,203	–	–	–	**0.662**	**7.11**	16,101
average	109	675	9.983	3.96	785	9.892	4.16	955	–	–	–	9.833	3.59	2,553

4.2 Baselines

To verify the effectiveness of our end-to-end floorplanning methodology, in addition to comparing the **Simulated Annealing (SA)** algorithm, we also compare with two state-of-the-art reinforcement learning-based floorplanners, including ICCD'20 [7] and KDD'22 [11].

SA: Simulated annealing-based floorplanning relies on the representation of the geometric relationship among macros. The objective metric is optimized by introducing perturbations into the floorplan representation through an annealing schedule process. Although the SA algorithm adopts a hill-climbing technique to escape from the locally optimal solution, when faced with the complex floorplanning problem, there is still a high probability of getting trapped in local optima even if the number of iterations increases. For a more fair comparison, we conduct a batch of experiments to adjust the parameters of simulated annealing and report the best results. The initial temperature is chosen within the range of 10^6 and 10^8, while the termination temperature is set at 10^{-11}. The number of iterations ranges from 50 to 200, and the cooling rate is fixed at 0.97. Upon reaching the termination temperature, the SA algorithm is halted, and the result is recorded. The **Sequence Pair (SP)** is adopted as the floorplan representation. Additionally, the neighborhood function remains consistent with Reference [7].

ICCD'20 [7]: ICCD'20 exploits a Q-learning algorithm to explore the local search heuristic for floorplanning. As in SA, perturbations are randomly sampled based on the current solution. Then, the manual features are defined for each solution and fed into an MLP to estimate the acceptance probability. Finally, the solution with the highest probability is selected at inference time. The process is continued until a predefined termination condition is reached. Thus, the runtime of ICCD'20 is dependent on the termination condition.

KDD'22 [11]: Following Google's approach, KDD'22 also formulates floorplanning as a sequential decision-making process and each time the network needs to simultaneously output the ID and position of the macro to be placed. Besides, the CBL floorplan representation is exploited to reduce the redundant output of the grid-based action space.

4.3 Comparison with Baselines

Table 1 compares our methodology with all baselines on GSRC, MCNC and "design" benchmarks. Columns "#Macro" and "#Net" are the number of macros and nets in each benchmark circuit. Columns "Area" and "Wire" represent the area and wirelength costs of the generated floorplan, while "RT" denotes the average runtime of the approaches. As the areas are not reported in KDD'22,

Table 2. Area Minimization on MCNC and GSRC Benchmarks (Area Unit:$\times 10^6 \mu m^2$, RT Unit:s)

Circuit	#Macro	#Net	SA		ICCD'20 [7]		Ours	
			Area	RT	Area	RT	Area	RT
ami33	33	123	1.224	79	1.224	55	1.222	98
ami49	49	408	37.960	155	38.133	72	37.487	233
n100	100	576	0.187	407	0.185	399	0.181	1,429
n200	200	1,274	0.196	1,048	0.198	902	0.190	3,598
n300	300	1,632	0.299	2,525	0.306	3,542	0.289	7,996
average	137	803	7.973	843	8.009	994	7.874	2,671

Table 3. Wirelength Minimization on MCNC and GSRC Benchmarks (Wire Unit:$\times 10^5 \mu m$, RT Unit:s)

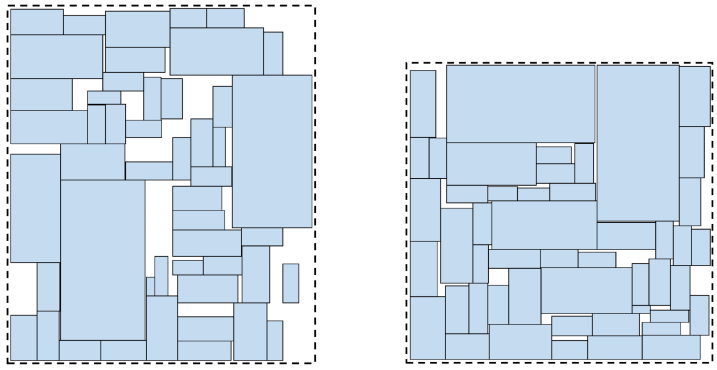
Circuit	#Macro	#Net	SA		ICCD'20 [7]		Ours	
			Wire	RT	Wire	RT	Wire	RT
ami33	33	123	0.39	88	0.40	49	0.36	93
ami49	49	408	7.12	172	7.33	69	7.03	244
n100	100	576	1.22	403	1.25	411	1.09	1,498
n200	200	1,274	2.96	1,109	2.99	912	2.91	3,735
n300	300	1,632	4.63	2,133	4.54	3,744	4.47	8,134
average	137	803	3.26	781	3.30	1,037	3.17	2,741

we only list the wirelength results. The better results are emphasized in bold in the table. We can see that, compared to SA, ICCD'20 [7], and KDD'22 [11], our methodology achieves 10.58%, 12.18%, and 73.99% wirelength improvements, and 3.88%, 2.99% reductions in area, demonstrating that our methodology can effectively produce high-quality chip floorplan. In particular, thanks to the proposed EAGAT architecture, the wirelength optimization is quite obvious. Note that the wirelength improvement is derived as follows: First, we separately calculate the wirelength improvements of our method over other works on each benchmark. Then, the improvement values on all benchmarks are averaged to achieve the final results. Besides, since the macro sizes vary considerably in different circuits, the experimental results also reflect that our method can generalize to different types of netlists. The reason is that the built RL model greatly reduces the action space without representation loss, as described in Section 3.1. While the runtime of our method is longer than other non-end-to-end approaches, i.e., SA and ICCD'20, it is negligible compared to the long runtime of the entire chip design process. We further verify the performance of the proposed methodology on the area and wirelength minimization problems. For the area minimization, we train our RL model with the objective function defined in Equation (1) but only take the area optimization into account and correspondingly remove rewards related to area from the reward function during inference. That is, for area optimization, we set η in Equation (4) to 0. A similar procedure is for wirelength minimization; we only consider the final wirelength as the reward for wirelength optimization. The results are listed in Table 2 and Table 3. We notice similar performance results to those in Table 1, which further demonstrate the superiority of our methodology.

To visualize the comparison results, Figure 8 shows the floorplan of ami49 circuit generated by SA and our end-to-end methodology, respectively. Obviously, our method produces a more compactly placed floorplan with less whitespace.

4.4 Transferability Study

To verify the transferability of our proposed floorplanning methodology, another experiment is conducted based on two settings, as shown in Table 4. We first divide the training netlists into



(a) area: $39.053 \times 10^6 \mu m^2$ (b) area: $38.028 \times 10^6 \mu m^2$

Fig. 8. Visualization of ami49 floorplans from (a) SA algorithm; (b) the proposed end-to-end methodology.

Table 4. Comparison of the Transferability Performance of the Proposed Methodology and ICCD'20 [7]

Setting	Circuit	Method	Area		Wire		RT	
			Trans.	Orig.	Trans.	Orig.	Trans.	Orig.
$Group_1$ ↓ $Group_2$	ami33	Ours	1.242	1.235	0.66	0.64	113	102
		ICCD'20	1.257	1.240	0.73	0.69	49	43
	ami49	Ours	38.503	38.028	13.10	13.25	251	242
		ICCD'20	39.102	38.65	17.11	17.24	78	67
$Group_2$ ↓ $Group_1$	n100	Ours	0.195	0.190	1.47	1.37	1,598	1,513
		ICCD'20	0.208	0.195	1.69	1.55	402	389
	n200	Ours	0.204	0.197	3.31	3.26	3,903	3,875
		ICCD'20	0.231	0.215	3.59	3.48	831	785
	n300	Ours	0.315	0.301	4.88	4.86	8,411	8,322
		ICCD'20	0.352	0.34	6.33	5.25	4,014	3,767
$Group_3$ ↓ $Group_4$	apte	Ours	47.245	47.080	3.34	3.22	29	24
		ICCD'20	47.313	47.080	4.15	4.03	21	16
	xerox	Ours	20.489	20.403	6.53	5.80	31	25
		ICCD'20	20.633	20.420	6.52	6.33	24	17
	hp	Ours	9.394	9.233	2.01	1.98	39	31
		ICCD'20	9.454	9.210	1.98	1.95	17	12
$Group_4$ ↓ $Group_3$	n10	Ours	0.242	0.234	0.16	0.14	38	30
		ICCD'20	0.245	0.239	0.18	0.17	23	18
	n30	Ours	0.224	0.218	0.50	0.46	82	74
		ICCD'20	0.231	0.223	0.48	0.49	59	52
	n50	Ours	0.215	0.211	0.99	0.95	346	301
		ICCD'20	0.219	0.215	1.05	1.02	105	89

$Group_i \rightarrow Group_j$ means the model is trained on $Group_i$ and tested on the netlists with the different distribution as $Group_j$ and vice versa. (Area unit:$\times 10^6 \mu m^2$, Wire unit:$\times 10^5 \mu m$, RT unit:s.)

two groups, where $Group_1$ ($Group_2$) represents that the number of the macro is greater (less) than 100, and the training netlists with the identical distribution include n100, n200, and n300 (ami33 and ami49). In addition, to further validate the transferability between different types of designs, we partition two additional groups, where $Group_3$ comprises the training netlists with the same distribution as GSRC, including n10, n30, and n50. $Group_4$ is composed of training netlists with an identical distribution to MCNC, including apte, xerox, and hp. Columns "Trans." and "Orig." denote

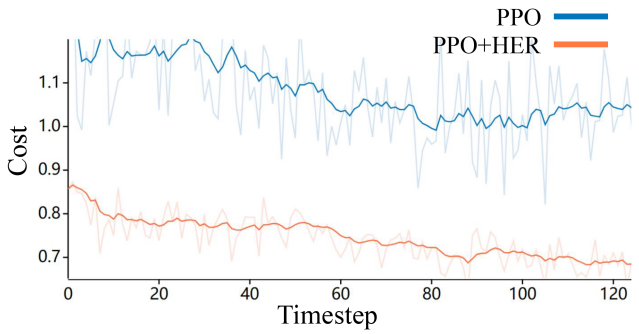


Fig. 9. Comparison of agents trained using PPO algorithm with and without HER technique on the ami49 netlist.

Table 5. Ablation Study of the MOE Block (Area Unit:$\times 10^6 \mu m^2$, Wire Unit:$\times 10^5 \mu m$, RT Unit:s)

Circuit	#Macro	#Net	FFN			MoE		
			Area	Wire	RT	Area	Wire	RT
ami33	33	123	1.263	0.67	133	1.235	0.64	102
ami49	49	408	38.533	15.22	271	38.028	13.25	242
n100	100	576	0.197	1.57	1,645	0.190	1.37	1,513
n200	200	1,274	0.211	3.28	3,987	0.197	3.26	3,875
n300	300	1,632	0.317	5.10	8,573	0.301	4.86	8,322
average	137	803	8.104	5.17	2,922	7.990	4.68	2,811

the transferability result and the original data, as listed in Table 1. As shown in the table, when the network model is tested on the netlists with different distributions, there is a certain degradation in performance. However, our methodology still behaves better than ICCD'20 [7], demonstrating that the proposed approach can generalize the prior learned knowledge to new unseen netlists.

4.5 Ablation Studies

4.5.1 Hindsight Experience Replay. An ablation experiment is performed in Figure 9 to explore the impact of the **hindsight experience replay (HER)** technique on the performance. "PPO" and "PPO+HER" refer to the models trained using the **proximal policy optimization algorithm (PPO)** [32] without and with the HER technique, respectively. The experiment is performed on the ami49 netlist. As shown in the figure, the PPO algorithm with the HER technique starts from a lower floorplan cost at the beginning of the inference process and ultimately produces better results. It demonstrates that the HER technique can not only generalize the prior knowledge of other netlists but also combine well with RL learning algorithms to achieve significant performance gains.

4.5.2 MoE Block. We also investigate how the MoE block affects performance. As described in Section 3.2.1, a mixture of **modality experts (MoE)** is proposed to capture information interaction between different types of nodes. In the comparison case, we replace the MoE blocks in the EAGAT encoder with a simple **feed-forward network (FFN)**, that is, all types of node features are aggregated through the same neural network for non-linear transformation. As shown in Table 5, the MoE blocks produce better floorplan results, demonstrating that MoE blocks used in the EAGAT encoder positively contribute to the network architecture.

5 CONCLUSION

In this work, we have proposed an end-to-end RL-based floorplanning methodology, which jointly learns an EAGAT encoder and a hierarchical policy network with a transformer backbone. The developed EAGAT encoder realizes the efficient interactions of node and edge information. Meanwhile, a mixture of modality experts are introduced in the encoder to capture specific information for each type of node. In addition, the proposed hindsight experience replay technique allows the RL model to benefit directly from valuable experience and be trained efficiently, which provides a good training scheme for applying reinforcement learning to combinatorial optimization problems. Experimental results demonstrate that our methodology outperforms previous state-of-the-art methods on various circuit benchmarks.

REFERENCES

[1] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. 1996. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 15, 12 (1996), 1518–1524.

[2] K.-S. The and D. F. Wong. 1991. Area optimization for higher order hierarchical floorplans. In *IEEE International Conference on Computer Design (ICCD'91)*. 520–521.

[3] Tung-Chieh Chen and Yao-Wen Chang. 2005. Modern floorplanning based on fast simulated annealing. In *ACM International Symposium on Physical Design (ISPD'05)*. 104–112.

[4] Song Chen and Takeshi Yoshimura. 2008. Fixed-outline floorplanning: Block-position enumeration and a new method for calculating area costs. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 27, 5 (2008), 858–871.

[5] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace-MS: Electrostatics-based placement for mixed-size circuits. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 34, 5 (2015), 685–698.

[6] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. 2018. Replace: Advancing solution quality and routability validation in global placement. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 38, 9 (2018), 1717–1730.

[7] Zhuolun He, Yuzhe Ma, Lu Zhang, Peiyu Liao, Ngai Wong, Bei Yu, and Martin D. F. Wong. 2020. Learn to floorplan through acquisition of effective local search heuristics. In *IEEE International Conference on Computer Design (ICCD'20)*. 324–331.

[8] Qi Xu, Hao Geng, Song Chen, Bo Yuan, Cheng Zhuo, Yi Kang, and Xiaoqing Wen. 2021. GoodFloorplan: Graph convolutional network and reinforcement learning-based floorplanning. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 41, 10 (2021), 3492–3502.

[9] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.

[10] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[11] Mohammad Amini, Zhanguang Zhang, Surya Penmetsa, Yingxue Zhang, Jianye Hao, and Wulong Liu. 2022. Generalizable floorplanner through corner block list representation and hypergraph embedding. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD'22)*. 2692–2702.

[12] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Conference on Neural Information Processing Systems (NIPS'17)*. 6000–6010.

[14] Khushro Shahookar and Pinaki Mazumder. 1991. VLSI cell placement techniques. *ACM Comput. Surv.* 23, 2 (1991), 143–220.

[15] Pei-Ning Guo, Chung-Kuan Cheng, and Takeshi Yoshimura. 1999. An O-tree representation of non-slicing floorplan and its applications. In *ACM/IEEE Design Automation Conference (DAC'99)*. 268–273.

[16] Sung Kyu Lim. 2008. *Practical Problems in VLSI Physical Design Automation*. Springer.

[17] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. 2000. B*-trees: A new representation for non-slicing floorplans. In *ACM/IEEE Design Automation Conference (DAC'00)*. 458–463.

[18] Jai Ming Lin and Yao Wen Chang. 2004. TCG-S: Orthogonal coupling of P*-admissible representations for general floorplans. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 23, 6 (2004), 968–980.

[19] Xianlong Hong, Sheqin Dong, Gang Huang, Yici Cai, Chung-Kuan Cheng, and Jun Gu. 2004. Corner block list representation and its application to floorplan optimization. *IEEE Trans. Circ. Syst. II: Expr. Briefs* 51, 5 (2004), 228–233.
[20] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. 2020. Agent57: Outperforming the Atari human benchmark. In *International Conference on Machine Learning (ICML'20)*. 507–517.
[21] Md Shamim Hussain, Mohammed J. Zaki, and Dharmashankar Subramanian. 2022. Global self-attention as a replacement for graph convolution. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD'22)*. 655–665.
[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision (ICCV'15)*. 1026–1034.
[23] Hangbo Bao, Wenhui Wang, Li Dong, Qiang Liu, Owais Khan Mohammed, Kriti Aggarwal, Subhojit Som, Songhao Piao, and Furu Wei. 2022. VLMo: Unified vision-language pre-training with mixture-of-modality-experts. In *Conference on Neural Information Processing Systems (NIPS'22)*. 32897–32912.
[24] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.* 23, 1 (2022), 5232–5270.
[25] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *the Web Conference (WWW'21)*. 1785–1797.
[26] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Conference on Neural Information Processing Systems (NIPS'15)*. 2692–2700.
[27] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. In *Conference on Neural Information Processing Systems (NIPS'17)*. 5055–5065.
[28] Yiting Liu, Ziyi Ju, Zhengming Li, Mingzhi Dong, Hai Zhou, Jia Wang, Fan Yang, Xuan Zeng, and Li Shang. 2022. Floorplanning with graph attention. In *ACM/IEEE Design Automation Conference (DAC'22)*. 1303–1308.
[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Workshop*.
[30] MCNC Benchmark. 2007. University of Michigan. Retrieved from http://vlsicad.eecs.umich.edu/BK/MCNCbench
[31] GSRC Benchmark. 2007. University of Michigan. Retrieved from http://vlsicad.eecs.umich.edu/BK/GSRCbench
[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

Received 17 October 2023; revised 6 March 2024; accepted 14 March 2024