TransPlace: Transferable Circuit Global Placement via Graph Neural Network

Yunbo Hou
School of Software and Microelectronics
Peking University
Beijing, China
yunboh@stu.pku.edu.cn

Haoran Ye
National Key Laboratory of General Artificial Intelligence
School of Intelligence Science and Technology
Peking University
Beijing, China
hrye@stu.pku.edu.cn

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

Siyuan Xu
Huawei Noah's Ark Lab
Shenzhen, China
xusiyuan520@huawei.com

Guojie Song*
National Key Laboratory of General Artificial Intelligence
School of Intelligence Science and Technology
Peking University
Beijing, China
gjsong@pku.edu.cn

Abstract

Global placement, a critical step in designing the physical layout of computer chips, is essential to optimize chip performance. Prior global placement methods optimize each circuit design individually from scratch. Their neglect of transferable knowledge limits its solution efficiency and chip performance as circuit complexity drastically increases. This study presents TransPlace, a global placement framework that learns to place millions of mixed-size cells in continuous space. TransPlace introduces i) Netlist Graph to efficiently model netlist topology, ii) Cell-flow and relative position encoding to learn SE(2)-invariant representation, iii) a tailored graph neural network architecture for informed parameterization of placement knowledge, and iv) a two-stage strategy for coarse-to-fine placement. Compared to state-of-the-art placement methods, TransPlace—trained on a few high-quality placements—can place unseen circuits with 1.2x speedup while reducing congestion by 30%, timing by 9%, and wirelength by 5%.

CCS Concepts

• **Hardware** → **Placement**; *Wire routing*; *Timing analysis.*

Keywords

EDA, circuit design, global placement, graph neural network

*Corresponding author.

KDD '25, August 3–7, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1245-6/25/08
https://doi.org/10.1145/3690624.3709185

ACM Reference Format:
Yunbo Hou, Haoran Ye, Yingxue Zhang, Siyuan Xu, and Guojie Song. 2025. TransPlace: Transferable Circuit Global Placement via Graph Neural Network. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3690624.3709185

1 Introduction

The development of integrated circuits (ICs) has propelled technological advancement from single chips to complex computing systems. Placement, a crucial stage in the design flow of Integrated Circuits (ICs), arranges electronic components within circuit layouts. As placement constructs the fundamental geometry from a topological netlist, it drives later design steps and guides prior stages, serving as a critical determinant of final chip performance.

The placement problem can be described as follows. Consider a circuit design represented by a hypergraph $H = (V, E)$, where V represents the set of cells (electronic units), and E represents the set of nets (hyperedges) between these cells. Placement determines cell positions $\boldsymbol{x}$ and $\boldsymbol{y}$ to minimize wirelength while avoiding cell overlap. This problem typically penalizes the density and mathematically formulates as [12, 22]

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad \sum_{e \in E} WL(e; \boldsymbol{x}, \boldsymbol{y}) + \lambda D(\boldsymbol{x}, \boldsymbol{y}). \tag{1}$$

Here, $WL(\cdot)$ is the wirelength cost function that evaluates the wirelength of any given net instance e, $D(\cdot, \cdot)$ is the density function to spread cells out in the layout, and λ is a weighting coefficient.

Despite decades of development, existing global placers tackle each placement instance from scratch using heuristics or expertly crafted algorithms [1, 8, 20, 22, 31, 38, 46]. These optimization-based and non-learnable global placers cannot leverage existing placement experience, thus limiting solution efficiency and chip performance. On the other hand, recent advances in transferable

placers [5, 27] focus on the floorplanning stage, employing reinforcement learning (RL) to train placement policies. However, these methods do not effectively scale to placements involving millions of mixed-size cells, and they have not been successfully applied to placement problems in continuous space.

This paper presents **TransPlace**, the first learning-based approach for global placement involving millions of mixed-size cells in the continuous space. We aim to leverage transferable placement knowledge to enhance both solution efficiency and placement quality. Learning to inductively place millions of cells presents significant challenges: (1) modeling large-scale and complex circuits into structured graphs; (2) extracting informed supervised signals from existing placements; (3) learning transferable knowledge from diverse heterogeneous circuit placements; and (4) adapting to circuit-specific constraints.

We address the above challenges through a series of techniques:

- We introduce **Netlist Graph** to represent the topology of a netlist. In this graph, cells and nets are represented as two distinct types of nodes, with pins serving as the connections between these nodes. The original circuit is hierarchically partitioned and coarsened, under optimal time complexity, which allows for an efficient and scalable encoding for circuits with millions of cells.
- We propose **Cell-flow**, a directed acyclic graph (DAG) in which each edge represents the relative positional relationship between cells. We extract cell-flows from well-placed circuits as demonstration data and train our model to learn SE(2)-invariant representations.
- We design **T**ransferable **P**lacement **G**raph **N**eural **N**etwork (**TPGNN**) to facilitate heterogeneous message passing within and across netlist graphs and cell-flows. TPGNN effectively parameterizes transferable placement knowledge by preserving full circuit topology and incorporating cell-flow inductive biases.
- We couple inductive placement with optimization-based **Circuit-adaptive Fine-tuning** that adapts to unique characteristics of unseen circuits, such as their terminal positions and core placement area. This two-stage strategy improves placement quality by integrating efficient inductive placement and fine-grained optimization-based placement.

We summarize our contributions as follows: (1) We introduce TransPlace, the first learning-based framework for large-scale global placement. (2) We propose a series of techniques—Netlist Graph modeling, Cell-flow, TPGNN, and a two-stage strategy—to enable and enhance large-scale inductive placement. (3) We comprehensively evaluate TransPlace across four standard benchmarks, demonstrating that TransPlace can surpass state-of-the-art global placers with a 1.2x speedup, 30% less congestion, 9% better timing, and 5% shorter wirelength.

2 TransPlace

TransPlace is schematically illustrated in Fig. 1, consisting of two stages: inductive placement and circuit-adaptive fine-tuning. For inductive placement, we introduce Netlist graph (§ 2.1) and cell-flow (§ 2.2) for efficient and topology-aware circuit modeling, SE(2)-invariant encoding and decoding that converts between relative and absolute cell positions (§ 2.3), and TPGNN to learn transferable placement knowledge (§ 2.4). Following inductive placement, we fine-tune the placement to adapt to circuit-specific characteristics and constraints (§ 2.5).

2.1 Netlist Graph

The circuit design can be represented as a hypergraph $H = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of cells (electronic units) and $\mathcal{E}$ is the set of nets (hyperedges). H stores the circuit topology produced in the preceding logic synthesis stage. As exemplified in Fig. 2(1), there are two types of electronic cells $\mathbb{V}$:

(1) **Terminals** $\mathbb{V}_T$ are big chunks of cells with fixed positions.
(2) **Movable cells** $\mathbb{V}_M$ are relatively smaller cells, and their positions are the decision variables of global placement.

The Netlist Graph (Fig. 2(2)), which serves as an input for inductive placement, is designed to fully retain the topological information. We formally define it as follows:

Definition 2.1 (Netlist Graph). $\mathcal{G} = \{\mathbb{V}, \mathbb{U}, \mathbb{P}, \boldsymbol{X}_{\mathbb{V}}, \boldsymbol{X}_{\mathbb{U}}, \boldsymbol{X}_{\mathbb{P}}, \boldsymbol{P}_T\}$, where $\mathbb{V} = \mathbb{V}_T \cup \mathbb{V}_M$ are electronic cells, nets $\mathbb{U}$ are the hyperedges connecting the cells, and pins $\mathbb{P} \subseteq \mathbb{V} \times \mathbb{U}$ represent the interactions among cells and nets. $\boldsymbol{X}$s are their feature matrices. $\boldsymbol{P}_T \in \mathbb{R}^{|\mathbb{V}_T| \times 2}$ stores the horizontal/vertical positions of terminals $\mathbb{V}_T$.

However, for some VLSIs with millions of cells and nets, directly processing the whole netlist graph exceeds memory limits and prevents effective learning. Therefore, we transform the original netlist graph $\mathcal{G}$ to **Hierarchical Netlist Graph** $\widetilde{\mathcal{G}} = f_{\text{HIER}}(\mathcal{G}) = (\mathcal{G}_R, \{\mathcal{G}_1, \cdots, \mathcal{G}_n\})$, where $\{\mathcal{G}_i | i = 1, \cdots, n\}$ are sub-netlist graphs of $\mathcal{G}$ (named **branch graphs**) and $\mathcal{G}_R$ is a transformation of $\mathcal{G}$ coarsened with the branch graphs (named **root graph**). Specifically, we first produce a partition result with KaHyPar [45]:

$$\begin{aligned} &\text{KaHyPar}(\mathcal{G}) = \{\mathbb{V}_i \subset \mathbb{V}_M | i = 1, \cdots, n\}, \\ &\text{where } \forall i \neq j, \mathbb{V}_i \cap \mathbb{V}_j = \emptyset. \end{aligned} \tag{2}$$

Note that the cells within each partition set are fully connected via nets. Otherwise, we further partition the branches. For every $\mathbb{V}_i \subset \mathbb{V}_M$, we extract **Sub-netlist Graph** and coarsen the original netlist graph; more algorithmic details are presented in Appendix A. The training and inference are performed on individual graphs in $\widetilde{\mathcal{G}}$, and the placement results of original $\mathcal{G}$ are reconstructed from $\widetilde{\mathcal{G}}$. In Appendix D.1, we demonstrate that the time complexity of Algorithm 1 is $O(|\mathbb{V}| + |\mathbb{U}| + |\mathbb{P}|)$, which is optimal.

2.2 Cell-flow

Learning (from) relative positional relationships instead of absolute spatial positions improves generalization across tasks [17, 23, 44, 55, 58]. However, the complexity of preserving all relative positions in global placement is $\sum_e degree_e$, where e denotes the hyperedges. For circuits with millions of nodes and hyperedges, such naive modeling would cause infeasible computational demands. In answer to this, we present cell-flow to efficiently model relative positional relationships in circuit designs.

Cell-flow (Fig. 2(3)) is a set of directed acyclic edges. It represents the relative positions of cells in a circuit for encoding (the

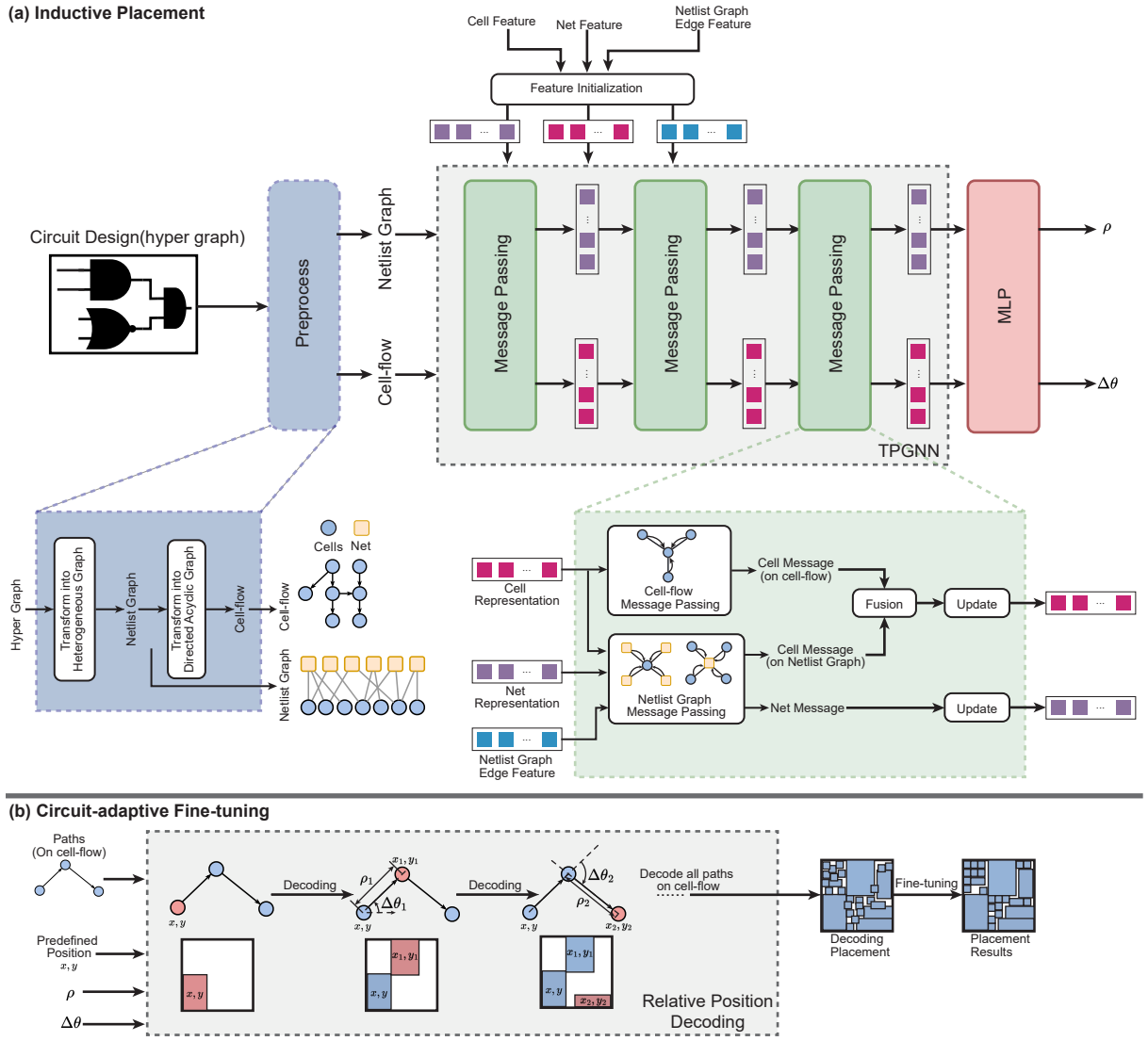

Figure 1: A schematic illustration of `TransPlace`. `TransPlace` contains two stages: Inductive Placement and Circuit-adaptive Fine-tuning. (a) Inductive Placement efficiently generates relative cell positions in one shot. It constructs the Cell-flow and Netlist Graph, applies message-passing on the two graphs to obtain cell and net representations, fuses and updates the hidden representations, and reads out relative position $\rho, \Delta\theta$. (b) Circuit-adaptive Fine-tuning decodes the relative positions within cell-flow into absolute positions and performs iterative gradient-based optimization.

demonstration data) and decoding (the inductive solution). We initialize a cell-flow from a netlist graph with a breadth-first search that starts from fixed cells and travels through nets. Meanwhile, we record the incoming flow edge for each cell using f_{INCOME}, and the net containing the flow edge using f_{NET}. This process is elaborated in Algorithm 2. Note that every connectivity branch contains at least one fixed cell (e.g. port), so cell-flow involves all cells. This is also guaranteed for sub-netlist graphs in § 2.1 because they are fully connected and contain a pseudo-terminal.

Definition 2.2 (Cell-flow). A set of directed edges $\mathbb{F} \subseteq \mathbb{V} \times \mathbb{V}$, where $\mathbb{V}$ are electronic cells in netlist graph $\mathcal{G}$. It is guaranteed that no loop will be found in $\mathbb{F}$.

As demonstrated in Appendix D.2, the time-consumption of generating cell-flow edges is $O(|\mathbb{V}| + |\mathbb{U}| + |\mathbb{P}|)$, and the cell-flow size is $|\mathbb{F}| = O(|\mathbb{P}|)$. So this procedure is optimal in terms of time and space complexity.

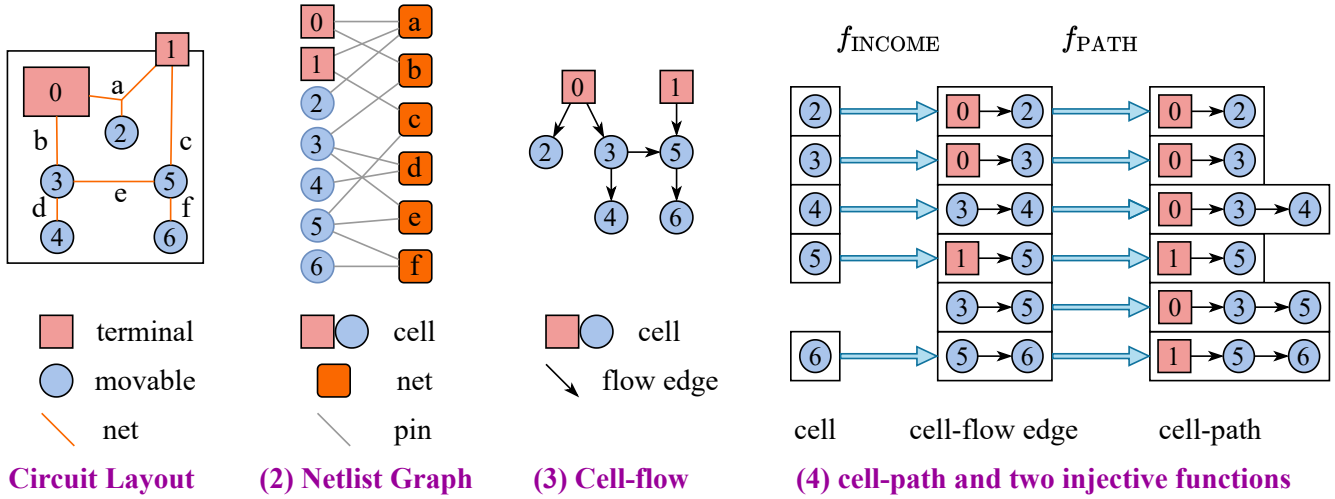

Figure 2: Overview of Netlist Graph, Cell-flow, cell-path, f_{INCOME}, and f_{PATH}.

2.3 SE(2)-invariant Encoding and Decoding

Based on the relative position relationship within cell-flow, we introduce an SE(2)-invariant encoding/decoding algorithm to convert between cell absolute/relative positions. This SE(2) invariance denotes that the representations remain unchanged under rotations and translations in the 2D plane, which allows our model to learn informed representation by preserving geometric consistency.

For the demonstration data (labels), we encode the absolute cell positions $\boldsymbol{P}$ into relative positions $\vec{\boldsymbol{P}}$ for training. This encoding process follows the cell-flow $\mathbb{F}$ within the placement and computes relative positions only for the flow edges $(v_i, v_j) \in \mathbb{F}$:

$$\vec{\boldsymbol{p}}_{i,j} = \boldsymbol{p}_j - \boldsymbol{p}_i, \qquad (v_i, v_j) \in \mathbb{F}, \tag{3}$$

where $\boldsymbol{p}_i$ is the absolute position of cell v_i. The vector $\vec{\boldsymbol{p}}_{i,j}$ of flow edge (v_i, v_j) is later transformed into diameter $\rho_{i,j}$ and angle $\theta_{i,j}$ in polar coordinate, and a corresponding deflection $\Delta\theta_{i,j}$ is calculated by subtracting the polar angle of v_i's cell-incoming-edge:

$$\rho_{i,j} = |\vec{\boldsymbol{p}}_{i,j}|_2 \qquad \theta_{i,j} = \arctan \vec{\boldsymbol{p}}_{i,j}, \tag{4}$$

$$\Delta\theta_{i,j} = \theta_{i,j} - \theta_{k,i}, \qquad \text{where } (v_k, v_i) = f_{\text{INCOME}}(v_i). \tag{5}$$

Overall, the distance ρ and the deflection $\Delta\theta$ are utilized as the SE(2)-invariant encoding of the cell positions $\boldsymbol{P}$:

$$(\boldsymbol{\rho}, \Delta\boldsymbol{\theta}) = f_{\text{ENCODE}}(\boldsymbol{P}_M; \mathcal{G}). \tag{6}$$

On the other hand, we need to convert the inductively generated relative positions into absolute ones. In this decoding process, given an input netlist graph $\mathcal{G}$ and a relative position encoding $(\boldsymbol{\rho}, \Delta\boldsymbol{\theta})$, we determine the absolute positions $\boldsymbol{P}_M$ of the movable cells by extracting the cell-paths (see Fig. 2(4)) from $\mathcal{G}$ as described in § 2.2. For a cell-path $(v_0, v_1, \cdots, v_{t-1}, v_t)_P$, which starts from a fixed cell v_0 with a fixed position $\boldsymbol{p}_0$, we can calculate the position vector $\vec{\boldsymbol{p}}_{t-1,t}$ of the ending cell-flow edge (v_{t-1}, v_t):

$$\rho_0 = |\boldsymbol{p}_0|_2 \qquad \theta_0 = \arctan \boldsymbol{p}_0 \qquad \theta_{t-1,t} = \theta_0 + \sum_{i=1,\cdots,t} \Delta\theta_{i-1,i} \tag{7}$$

$$\vec{\boldsymbol{p}}_{t-1,t} = [\rho_{t-1,t} \cos\theta_{t-1,t}, \rho_{t-1,t} \sin\theta_{t-1,t}]. \tag{8}$$

The absolute ending position of cell-flow edge (v_{t-1}, v_t) is the summation of the position vectors through the cell-path. As there might be multiple cell-flow edges ending at cell v_t, the decoded absolute position of cell v_t is averaged over edges:

$$\boldsymbol{p}_{t;(t-1,t)} = \boldsymbol{p}_0 + \sum_{i=1,\cdots,t} \vec{\boldsymbol{p}}_{i-1,i} \qquad \boldsymbol{p}_t = \underset{(v_i,v_t)\in\mathbb{F}}{\text{mean}} \boldsymbol{p}_{t;(i,t)}. \tag{9}$$

Overall, by stacking $\boldsymbol{p}_j$, $j \in \mathbb{V}_M$, we obtain the decoding results:

$$\boldsymbol{P}_M = f_{\text{DECODE}}(\boldsymbol{\rho}, \Delta\boldsymbol{\theta}; \mathcal{G}). \tag{10}$$

The time consumption of producing the position vectors for all edges in $\mathbb{F}$ is $\omega|\mathbb{F}|$, where ω is the average length of cell-paths. Since ω is typically small (see Appendix D.3), it can be treated as a constant. We also have $|\mathbb{F}| = O(|\mathbb{P}|)$ (Appendix D.2). As a result, the time required for relative position decoding is proportional to the scale of the netlist graph, which is optimal.

2.4 TPGNN

2.4.1 Inference. We illustrate the inference pipeline of TPGNN in Fig. 1. TPGNN takes a netlist graph and its cell-flow as inputs, then embeds raw features $\boldsymbol{X}_{\mathbb{V}}$, $\boldsymbol{X}_{\mathbb{U}}$, and $\boldsymbol{X}_{\mathbb{P}}$ into hidden representations $\boldsymbol{H}_{\mathbb{V}}^{(0)}$, $\boldsymbol{H}_{\mathbb{U}}^{(0)}$, $\boldsymbol{H}_{\mathbb{P}}$, via Multi-Layer Perceptrons (MLPs). Then, it generates deeper representations of cells $\mathbb{V}$, and nets $\mathbb{U}$ with L layers of message-passing. Finally, readout layers convert the output cell and net representations into relative cell positions.

In each layer, the topological information is collected through cell-flow and netlist graph message-passing. The messages are then used to fuse and update the representations of cells $\mathbb{V}$ and nets $\mathbb{U}$. For netlist graph message-passing, we interact the messages of cells $\mathbb{V}$ and nets $\mathbb{U}$ through graph edges which preserve the topological connection between cells and nets:

$$\boldsymbol{M}_{\mathbb{U}\to\mathbb{V}}^{(l)} = f_{\mathbb{U}\to\mathbb{V}}(\mathbb{U}, \mathbb{P}, \boldsymbol{H}_{\mathbb{U}}^{(l)}, \boldsymbol{H}_{\mathbb{P}}), \tag{11}$$

$$\boldsymbol{M}_{\mathbb{V}\to\mathbb{U}}^{(l)} = f_{\mathbb{V}\to\mathbb{U}}(\mathbb{V}, \mathbb{P}, \boldsymbol{H}_{\mathbb{V}}^{(l)}, \boldsymbol{H}_{\mathbb{P}}), \tag{12}$$

where l is the number of current layer, $\boldsymbol{M}_{\mathbb{V}}^{(l)}$ and $\boldsymbol{M}_{\mathbb{U}}^{(l)}$ denote the messages of cells $\mathbb{V}$ and nets $\mathbb{U}$ computed on netlist graphs. $f_{\mathbb{V}\to\mathbb{U}}$

is the message function that collects topological messages from nets $\mathbb{U}$ and sends them to cells $\mathbb{V}$ via netlist graph edges. $f_{\mathbb{U}\to\mathbb{V}}$ is similar. $\boldsymbol{H}_{\mathbb{V}}^{(l)}$ and $\boldsymbol{H}_{\mathbb{U}}^{(l)}$ denote the hidden representations of the cells $\mathbb{V}$ and nets $\mathbb{U}$ of layer l. Inspired by SchNet [26] we design the message function $f_{\mathbb{U}\to\mathbb{V}}$ to fuse representations of adjacent cells and edges. Similarly, influenced by CircuitGNN [18], we design $f_{\mathbb{V}\to\mathbb{U}}$ to utilize edge representations for computing weights during message passing, enabling cells to perceive geometric information. These functions can be expressed as follows:

$$f_{\mathbb{U}\to\mathbb{V}}(\{(\boldsymbol{h}_u^{\mathbb{U}}, \boldsymbol{h}_{(v,u)}^{\mathbb{P}}) | (v,u) \in \mathbb{P}\}) = \sum_{(v,u)\in\mathbb{P}} (\boldsymbol{W}_{\mathbb{P}\to\mathbb{V}} \boldsymbol{h}_{(v,u)}^{\mathbb{P}}) \odot \boldsymbol{W}_{\mathbb{U}\to\mathbb{V}} \boldsymbol{h}_u^{\mathbb{U}} \tag{13}$$

$$f_{\mathbb{V}\to\mathbb{U}}(\{(\boldsymbol{h}_v^{\mathbb{V}}, \boldsymbol{h}_{(v,u)}^{\mathbb{P}}) | (v,u) \in \mathbb{P}\}) = \sum_{(v,u)\in\mathbb{P}} (\boldsymbol{a}^\top \boldsymbol{h}_{(v,u)}^{\mathbb{P}}) \cdot (\boldsymbol{W}_{\mathbb{V}\to\mathbb{U}} \boldsymbol{h}_v^{\mathbb{V}}), \tag{14}$$

where $\boldsymbol{W}_{\mathbb{P}\to\mathbb{V}} \boldsymbol{h}_{(v,u)}^{\mathbb{P}}$, $\boldsymbol{W}_{\mathbb{U}\to\mathbb{V}} \boldsymbol{h}_u^{\mathbb{U}}$, and $\boldsymbol{W}_{\mathbb{V}\to\mathbb{U}} \boldsymbol{h}_v^{\mathbb{V}}$ are learnable weight matrices; $\boldsymbol{a}$ is a learnable vector; and $\odot$ refers to inner-product.

For Cell-Flow message-passing, we directly collect messages of cells $\mathbb{V}$ to obtain a fused message:

$$\boldsymbol{M}_{\mathbb{V}\to\mathbb{V}}^{(l)} = f_{\mathbb{V}\to\mathbb{V}}(\mathbb{V}, \mathbb{F}, \boldsymbol{H}_{\mathbb{V}}^{(l)}). \tag{15}$$

Here, $\boldsymbol{M}_{\mathbb{V}\to\mathbb{V}}^{(l)}$ denotes the messages of cells $\mathbb{V}$ computed on cell-flow and $\boldsymbol{H}_{\mathbb{V}}^{(l)}$ denotes the hidden representations of cells $\mathbb{V}$. $f_{\mathbb{V}\to\mathbb{V}}$ is the message function that transfers relative positional relationship messages from cells $\mathbb{V}$. We design $f_{\mathbb{V}\to\mathbb{V}}$ to speed up the message-passing as below:

$$f_{\mathbb{V}\to\mathbb{V}}(\{\boldsymbol{h}_v^{\mathbb{V}} | (v, v^*) \in \mathbb{F}\}) = \sum_{(v,v^*)\in\mathbb{F}} \boldsymbol{W}_{\mathbb{V}\to\mathbb{V}} \boldsymbol{h}_{v^*}^{\mathbb{V}}, \tag{16}$$

where $\boldsymbol{W}_{\mathbb{V}\to\mathbb{V}} \boldsymbol{h}_{v^*}^{\mathbb{V}}$ is a learnable weight matrix.

After computing the netlist graph and cell-flow messages for cells $\mathbb{V}$, we update the representations of cells $\mathbb{V}$ with the fused messages $\boldsymbol{M}_{\mathbb{V}}^{(l)}$:

$$\boldsymbol{M}_{\mathbb{V}}^{(l)} = MaxPooling(\boldsymbol{M}_{\mathbb{V}\to\mathbb{V}}^{(l)}, \boldsymbol{M}_{\mathbb{U}\to\mathbb{V}}^{(l)}). \tag{17}$$

At last, we combine the messages and hidden representations from layer l to generate the updated hidden representations for layer $l+1$:

$$\boldsymbol{H}_{\mathbb{V}}^{(l+1)} = f_{\text{update}}(\boldsymbol{H}_{\mathbb{V}}^{(l)}, \boldsymbol{M}_{\mathbb{V}}^{(l)}) \tag{18}$$

$$\boldsymbol{H}_{\mathbb{U}}^{(l+1)} = f_{\text{update}}(\boldsymbol{H}_{\mathbb{U}}^{(l)}, \boldsymbol{M}_{\mathbb{V}\to\mathbb{U}}^{(l)}) \tag{19}$$

Here, the update function $f_{\text{update}}(\boldsymbol{H}, \boldsymbol{M}) = \boldsymbol{H} + \boldsymbol{M}$.

After L iterations of message passing, we obtain the final representation of cells $\boldsymbol{H}_{\mathbb{V}} = \boldsymbol{H}_{\mathbb{V}}^{(L)}$ and nets $\boldsymbol{H}_{\mathbb{U}} = \boldsymbol{H}_{\mathbb{U}}^{(L)}$, and readout the relative cell positions, i.e., distance $\hat{\rho}$ and deflection $\Delta\hat{\theta}$ of cell flow edges $(v_i, v_j) \in \mathbb{F}$:

$$\boldsymbol{h}_{i,j}^{\rho} = \boldsymbol{h}_i^{\mathbb{V}} \oplus \boldsymbol{h}_t^{\mathbb{U}} \oplus \boldsymbol{h}_j^{\mathbb{V}}, \tag{20}$$

$$\boldsymbol{h}_{i,j}^{\theta} = \boldsymbol{h}_k^{\mathbb{V}} \oplus \boldsymbol{h}_s^{\mathbb{U}} \oplus \boldsymbol{h}_i^{\mathbb{V}} \oplus \boldsymbol{h}_t^{\mathbb{U}} \oplus \boldsymbol{h}_j^{\mathbb{V}}, \tag{21}$$

where $\oplus$ means concatenation, and $u_s = f_{\text{NET}}((v_k, v_i))$, $u_t = f_{\text{NET}}((v_i, v_j))$, $(v_k, v_i) = f_{\text{INCOME}}(v_i)$. Then we define:

$$\hat{\rho}_{i,j} = f_\rho(\boldsymbol{h}_{i,j}^{\rho}) = \exp(\beta + \alpha \cdot \text{Tanh}(\boldsymbol{a}_\rho^\top \cdot \boldsymbol{h}_{i,j}^{\rho} + b_\rho)) \tag{22}$$

$$\Delta\hat{\theta}_{i,j} = f_\theta(\boldsymbol{h}_{i,j}^{\theta}) = \pi \cdot \text{Tanh}(\boldsymbol{a}_\theta^\top \cdot \boldsymbol{h}_{i,j}^{\theta} + b_\theta), \tag{23}$$

where $\boldsymbol{a}_\rho$, $\boldsymbol{a}_\theta$, b_ρ, and b_θ are learnable weight vectors and scalars. α and β are hyperparameters used to control the range of ρ. In this paper, we set $\alpha = 15$ and $\beta = -2$ to flexibly adapt to $\rho \in (4e-8, 4e5)$.

2.4.2 Training. TPGNN is trained to imitate preplaced circuits generated by DREAMPlace [31], i.e., netlist graphs $\mathcal{G}$ with absolute positions for movable cells $\hat{\boldsymbol{P}}_M$. We encode the absolute positions into ground-truth relative ones as training labels. The training loss is determined by the difference between TPGNN outputs and ground truth relative positions using Smooth-L1 loss. We defer the full details to Appendix E.1.

2.5 Circuit-adaptive Fine-tuning

While inductively generating placement is efficient, it may neglect the unique characteristics of an unseen circuit, such as its terminal positions and core placement area. Therefore, after generating the inductive placements, we iteratively fine-tune them by minimizing wirelength [19] and density [22].

The wirelength objective is calculated given cell positions [19]:

$$\mathcal{L}_{\text{W}} = \mathcal{L}_{\text{W}}(x) + \mathcal{L}_{\text{W}}(y) \tag{24}$$

$$\mathcal{L}_{\text{W}}(x) = \sum_{u\in\mathbb{U}} \left(\frac{\sum_{(v,u)\in\mathbb{P}} x_v e^{\gamma x_v}}{\sum_{(v,u)\in\mathbb{P}} e^{\gamma x_v}} - \frac{\sum_{(v,u)\in\mathbb{P}} x_v e^{-\gamma x_v}}{\sum_{(v,u)\in\mathbb{P}} e^{-\gamma x_v}} \right), \tag{25}$$

where x_v denotes the x-coordinate of cell v and γ is a hyperparameter. $\mathcal{L}_{\text{W}}(y)$ is computed in a similar manner.

The unique solution of the electrostatic system is derived from [22]:

$$\begin{cases} \nabla \cdot \nabla\psi(x,y) = -\rho(x,y) \\ \hat{n} \cdot \nabla\psi(x,y) = 0, (x,y) \in \partial R \\ \iint_R \rho(x,y) = \iint_R \psi(x,y) = 0 \end{cases} \tag{26}$$

$$\mathcal{L}_{\text{D}} = \sum_{i\in\mathbb{V}} N_i(v) = \sum_{i\in\mathbb{V}} q_i \psi(v), \tag{27}$$

where $\rho(x,y)$ denotes the cell density in position (x,y) and $\psi(x,y)$ is the potential in position (x,y) [22].

Overall, the fine-tuning loss is given by Eq. (31), where λ_{D} weighs two loss terms. The fine-tuning parameters are updated after backward propagation and cell position updates, according to the rules given below.

$$\lambda_D = \lambda_D * \mu \tag{28}$$

$$\mu = \begin{cases} 1.05 * \max(0.999^{epochs}, 0.98) & \Delta HPWL < 0 \\ 1.05 * 1.05^{\frac{-\Delta HPWL}{350000}} & \Delta HPWL \geqslant 0 \end{cases} \tag{29}$$

$$\Delta HPWL = HPWL_{new} - HPWL_{old} \tag{30}$$

$$\mathcal{L}_{\text{fine-tune}} = \mathcal{L}_{\text{W}} + \lambda_{\text{D}} \mathcal{L}_{\text{D}}. \tag{31}$$

Here, HPWL is the half-perimeter wire length [31]. $HPWL_{new}$ and $HPWL_{old}$ represent the HPWL before and after this optimization step, respectively.

3 Experiments

This section comprehensively evaluates TransPlace and answers two key research questions (RQs):

(1) Can TransPlace transfer placement knowledge and enhance diverse unseen circuit designs?
(2) Can TransPlace improve multiple design objectives simultaneously [4, 59], even when primarily trained with specific focuses like congestion?

3.1 Knowledge Transfer across Diverse Designs

For the first RQ, our evaluations involve three benchmarks containing 37 circuits with diverse functionalities. We pick 5 large-scale circuits as training datasets to ensure sufficient transferable placement knowledge. To compare our model with other global placers, we use overflow and routed wirelength, measured after global routing, as our evaluation metrics. Although traditional placer usually takes half perimeter wirelength (HPWL) as an evaluation metric, it does not consider detours in the path, and wire congestion [2] may still cause routing failure even for solutions with better HPWL. Therefore, we evaluate placements after global routing to provide an accurate evaluation of routability quality. We conduct the following steps to compute overflow. We first divide the entire layout into grids, each with a predefined wire capacity denoted as RC. This limit represents the maximum number of wires allowed in each grid cell. Overflow $OF(i, j)$ occurs when the number of wires exceeds this limit in the grid cell (i, j). Total overflow can be defined as follows:

$$TOF = \sum_{i,j} OF(i, j). \tag{32}$$

We evaluate TransPlace against two classical placers: (1) NTUPlace [20], an analytical placer for mixed-size circuit designs, and (2) DREAMPlace [31], recognized for its efficient implementation of GPU acceleration in analytical placement. The global router NCTUgr [16] is used for the DAC2012 dataset [50], while FastRoute 4.0 [54] for the ISPD2015 and ISPD2019 datasets [9, 35]. Detailed placement is conducted using Abacus [49] and ABCDPlace [32] for final placement results. For NTUPlace, we set the number of maximum threads to 8. All the experiments are conducted using a single Nvidia RTX 3090 GPU and an Intel Platinum 8255C CPU.

The results on three datasets are shown in Table 1, Table 2, and Table 3. The evaluation encompasses the aggregate routed wirelength, the total overflow, and the total runtime including placement and fine-tuning steps which are short for RWL, OVFL, and RT. As shown by the results, compared to the state-of-the-art placer DREAMPlace, TransPlace can place a new circuit about 1.2x faster, with a 30% reduction in Total Overflow and 2% reduction on Wirelength (averaged over circuits in ISPD2015 and ISPD2019). A lower total overflow not only implies a reduced likelihood of wire congestion, but also enhances the potential for successful routability and design convergence. Notably, despite our training dataset comprising only five circuits from DAC2012, TransPlace exhibits adaptability across a wide array of circuits from three distinct benchmarks. This adaptability is important for meeting the ever-evolving demands of chip design. It demonstrates the potential of TransPlace to improve EDA workflows in practical, high-stakes environments.

3.2 Cross-Objective Optimization

TransPlace is trained on placements generated by DREAMPlace with routability-driven methods. To answer the second RQ, this section evaluates its performance on the ICCAD2015 dataset [24], which focuses on incremental timing-driven placement [29, 34]. This placement approach optimizes the positions of circuit elements to minimize interconnect delay in timing-critical paths. The metrics used for the evaluation are total negative slack (TNS), worst negative slack (WNS), and number of violation paths (NVP).

The calculation of TNS, WNS, and NVP is based on the timing graph, a directed acyclic graph defined by the circuits. In this graph, each node represents a pin in the circuit, and each edge indicates a directed pin-to-pin connection. We travel all nodes connected by edges in the graph, beginning at the source cell node. For each node, we calculate its actual arrival time; the difference between it and the required arrival time is termed "slack" [51]. The maximum slack across all nodes is known as the Worst Negative Slack (WNS), while the Total Negative Slack (TNS) is the sum of the slacks at all timing endpoints. A path with negative slack at any of its nodes is referred to as a violation path, and NVP denotes the total number of such paths.

We compare TransPlace with DREAMPlace 4.0 [29], DREAMPlace with timing-driven methods. In our Circuit-adaptive Fine-tuning approach, similar to DREAMPlace 4.0, we employ OpenTimer for evaluating TNS and WNS and adopt a momentum-based method for updating net weights used in the wirelength calculation. For the assessment of our timing metrics, which include TNS, WNS, and NVP, we utilize Cadence Innovus. Detailed placement is conducted using Abacus [49] and ABCDPlace [32] for final placement results.

Table 4 presents the complete comparison results on ICCAD2015 benchmark, where the best results are highlighted in bold and black. As shown by the results, compared to DREAMPlace, TransPlace can place a new circuit with a 9% reduction on TNS, 1% reduction on NVP, 8% reduction on WNS, and 1% reduction on rWL (averaged over circuits in the dataset). Although TransPlace uses routability-driven placement as the training labels, it shows improvement in timing metrics for unseen circuits. This capability evidences the model's proficiency in extending learned knowledge beyond the initial training focus. The ability to generalize from one objective to another ensures an adaptable and efficient design process, especially when dealing with complex circuits. TransPlace allows designers to improve multiple aspects simultaneously, showing promise to streamline the overall design process.

Fig. 3 visualizes the comparisons between DREAMPlace and TransPlace on superblue5. Fig. 3(a) reveals that our solution achieves more efficient congestion management, evidenced by a discernible reduction in red dots and fewer congestion peaks. Fig. 3(b) indicates our solution achieves much less vertical and horizontal overflow. Furthermore, Fig. 3(c) displays the distribution in timing path slack of TransPlace and DREAMPlace, which shows overall optimization for timing slack.

These findings emphasize the technical advances TransPlace provides and its applicability in real-world chip design scenarios. By effectively reducing congestion and optimizing timing path slack, TransPlace addresses the multifaceted challenges of designing

Table 1: Comparison results on ISPD2015.

Benchmark	Netlist	#cell	#nets	DREAMPlace			TransPlace		
				RT	OVFL↓	RWL($\times 10^6 um$)↓	RT	OVFL	RWL
ISPD2015	mgc_des_perf_1	113K	113K	**34.21**	41	**1.70**	37.25	**24**	1.81
	mgc_des_perf_a	109K	110K	143.58	13,123	**3.06**	**71.25**	**11,841**	3.22
	mgc_des_perf_b	113K	113K	107.58	13	**2.38**	**90.01**	**7**	2.59
	mgc_edit_dist_a	130K	131K	65.93	16,251	6.18	**61.53**	**13,738**	**6.15**
	mgc_fft_1	35K	33K	35.14	19	**0.65**	**11.05**	**5**	0.70
	mgc_fft_2	35K	33K	41.37	5	**0.65**	**26.18**	**2**	0.72
	mgc_fft_a	34K	32K	35.24	3,244	**0.99**	**11.64**	**2,604**	1.01
	mgc_matrix_mult_1	160K	159K	37.51	9	**3.01**	**21.96**	**5**	3.03
	mgc_matrix_mult_2	160K	159K	39.60	12	3.00	**20.54**	**0**	**2.98**
	mgc_matrix_mult_a	154K	154K	32.69	6,802	4.02	**18.54**	**4,477**	**3.91**
	mgc_pci_bridge32_a	30K	30K	157.47	3,376	**0.77**	**156.17**	**2,776**	0.77
	mgc_pci_bridge32_b	29K	29K	216.42	2,921	**1.01**	**147.57**	**2,419**	1.01
	mgc_superblue11_a	954K	936K	97.39	899	**44.04**	**85.69**	**561**	44.46
	mgc_superblue12	1293K	1293K	**70.33**	99,641	**40.86**	96.47	**67,690**	40.98
	mgc_superblue14	634K	620K	35.33	1,690	**29.54**	**33.46**	**1,283**	30.05
	mgc_superblue16_a	698K	697K	**82.64**	27,862	34.01	115.74	**26,894**	**33.59**
	mgc_superblue19	522K	512K	**27.23**	432	**20.47**	30.81	**110**	20.76
	Average ratio			1.50	1.31	0.98	1.00	1.00	1.00

Table 2: Comparison results on ISPD2019.

Benchmark	Netlist	#cell	#nets	DREAMPlace			TransPlace		
				RT	OVFL↓	RWL($\times 10^6 um$)↓	RT	OVFL	RWL
ISPD2019	ispd19_test1	9K	3K	9.89	0	0.09	**9.56**	0	**0.09**
	ispd19_test2	73K	72K	**40.75**	0	**2.72**	42.61	0	2.72
	ispd19_test3	8K	9K	**15.69**	0	0.12	18.25	0	**0.12**
	ispd19_test4	151K	152K	32.88	9299	**4.68**	**23.26**	**9159**	4.69
	ispd19_test6	181K	180K	53.27	0	**6.92**	**44.41**	0	6.98
	ispd19_test7	362K	359K	55.30	0	**13.66**	**46.95**	0	14.39
	ispd19_test8	543K	538K	58.26	0	**20.59**	**49.32**	0	20.94
	ispd19_test9	903K	895K	**62.85**	0	**32.75**	70.65	0	33.47
	ispd19_test10	903K	895K	68.83	366	**33.21**	**54.91**	**143**	35.46
	Average ratio			1.06	2.43	0.98	1.00	1.00	1.00

Table 3: Comparison results on DAC2012

Benchmark	Netlist	#cell	#nets	NTUPlace			DREAMPlace			TransPlace		
				RT	OVFL↓	RWL($\times 10^6 um$)↓	RT	OVFL	RWL	RT	OVFL	RWL
DAC2012	superblue2	1014K	991K	6028	115624	23.65	337.17	54220	24.37	**158.80**	**33808**	**23.66**
	superblue3	920K	898K	5443	212946	16.78	**174.25**	7574	**15.43**	202.70	**6432**	15.56
	superblue6	1014K	1007K	5374	143118	16.23	248.75	4092	15.78	**219.97**	**2090**	**15.32**
	superblue7	1365K	1340K	8406	4780	20.94	183.39	5182	20.15	**148.69**	**4794**	**20.15**
	superblue9	847K	834K	6039	41942	12.01	128.59	3700	12.40	**115.70**	**2644**	**12.12**
	superblue11	955K	936K	5621	18342	15.45	**167.64**	16606	17.44	197.41	**2856**	**16.14**
	superblue12	1293K	1293K	27751	580406	17.67	80.30	21804508	36.59	**59.36**	**2777526**	**17.73**
	superblue14	635K	620K	4368	10510	10.71	143.81	19284	**10.41**	**99.38**	**13552**	10.56
	superblue16	699K	697K	5089	16618	11.60	137.78	10148	11.03	**96.38**	**9422**	**10.87**
	superblue19	523K	512K	3749	33546	7.48	**73.89**	7422	7.29	83.17	**3478**	7.40
	Average ratio			83.57	14.06	1.02	1.24	2.55	1.12	1.00	1.00	1.00

Table 4: Comparison results on ICCAD2015.

netlist_name	#cell	#nets	DREAMPlace				TransPlace			
			WNS↓	TNS↓	NVP↓	rWL↓	WNS	TNS	NVP	rWL
superblue1	1216K	1215K	**-16.98**	-27864.90	**7922**	**99590980**	-20.10	**-25575.90**	**7971**	**100961600**
superblue3	1219K	1224K	-17.48	-12899.00	4412	98362130	**-14.86**	**-11608.30**	**3700**	**97555240**
superblue4	802K	802K	-20.81	**-24490.30**	**5967**	**67779710**	**-16.41**	-30316.90	8493	74526630
superblue5	1091K	1100K	-25.31	-20993.40	8094	107397500	**-25.17**	**-12956.90**	**4708**	**99634140**
superblue7	1938K	1933K	**-16.40**	**-13619.30**	**7683**	**125085700**	**-14.68**	**-13105.80**	8415	126574700
superblue10	1888K	1898K	**-29.40**	**-103000.00**	**12193**	217113500	-33.38	-105000.00	13122	**202172100**
superblue16	986K	999K	**-14.50**	-25465.30	11591	**91522550**	**-13.65**	**-21145.00**	**10747**	93100710
superblue18	772K	771K	-17.50	**-11082.30**	**3336**	49204750	**-13.93**	-13300.60	5663	**48455550**
Average ratio			1.08	1.09	1.01	1.01	1.00	1.00	1.00	1.00

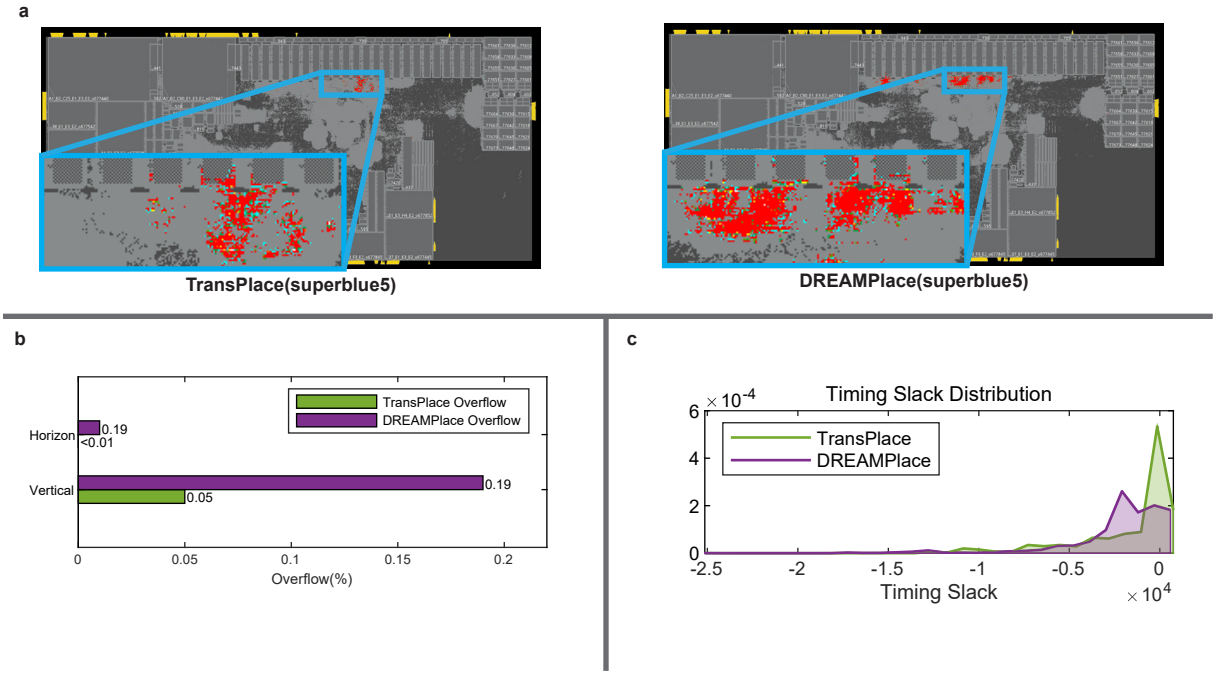

Figure 3: Visualizations of comparisons on superblue5. (a) Congestion visualization of the placement generated by DREAMPlace and TransPlace. The grey parts denote the cells, and the red dots indicate that there is congestion. Density of red dots indicates the level of congestion. (b) Overall comparison of vertical and horizontal overflow (lower overflow implies better performance). (c) Distribution of timing path slack for placements. A higher proportion of smaller slack indicates better performance.

contemporary complex circuits. This capability to simultaneously optimize multiple design objectives highlights TransPlace's potential to streamline the chip design process.

4 Related Work

Global Placement. Global placement methods have been extensively developed since the 1960s. The proposed methods are broadly classified into four categories: meta-heuristics [1, 30, 38, 57], partition-based methods [8], RL-based methods [5, 6, 10, 27, 43, 48], and analytical methods [11–13, 22, 31]. Meta-heuristics treat placement as a step-wise optimization problem and solve it with heuristic algorithms such as Simulated Annealing and Genetic Algorithm. Partition-based methods iteratively divide the chip's netlist and layout into smaller sub-netlists and sub-layouts, based on the cost function of the cut edges. Optimization methods are used to find solutions when the netlist and layout are sufficiently small. RL-based methods regard the placement problem as a Markov Decision Process (MDP) and train a policy to sequentially place the cells. All of them suffer from low convergence rates, which restricts their usage only to the circuits with a small number of large-sized cells.

Compared to them, analytical methods can optimize modern circuits with millions of cells. These methods optimize the positions of mixed-size cells by formulating the problem as an analytical function of cell coordinates. Analytical methods have gained popularity recently due to their compatibility with deep learning toolkits, which enables GPU-accelerated placements [42]. However, they approach each placement instance independently from scratch, unable to utilize previous placement experiences. Prior to our work, the complexity and scale of global placement hindered the application of deep learning techniques for learning-based enhancement.

Graph Neural Networks (GNNs) for Placement. GNNs demonstrate state-of-the-art performance in predictive and generative tasks involving graph data [53]. Since integrated circuits (ICs) can naturally be represented as graphs, there has been a growing adoption of GNNs in IC design, particularly in the critical placement phase [3]. GNNs are applied to various tasks in the EDA workflow, including behavioral and logic design [39, 52], logic synthesis [25, 60], partitioning and floorplanning [5, 14, 15, 21, 27, 28, 36, 37, 47], routing congestion prediction [7, 18], timing analysis [40, 41], etc. Among them, most relevant to our work is the application of GNNs for the floorplanning stage. However, they still resort to optimization-based methods for global placement due to their limited scalability. Prior to this work, the application of GNNs to inductive global placement remains limited due to significant technical challenges (§ 1).

5 Conclusion

This work introduces `TransPlace`, the first learning-based approach for global placement in integrated circuit design. We develop a series of techniques to address the technical challenges of learning large-scale cell placement. We demonstrate that `TransPlace` can effectively transfer versatile placement knowledge through evaluations on four benchmarks that feature diverse circuits and design objectives.

`TransPlace` shows promise in improving integrated circuit design by learning parameterized knowledge, which results in higher-quality circuits and accelerates design cycles. `TransPlace` can serve as an initial setup for placers, streamlining the optimization process for customized objectives and providing warm starts for optimization. Additionally, for designs requiring multiple logic optimizations, `TransPlace` can efficiently provide placement solutions, enabling early evaluation and issue resolution, thereby speeding up design convergence.

6 ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. 62276006).

References

[1] Vidal-Obiols Alex, Cortadella Jordi, Petit Jordi, Galceran-Oms Marc, and Martorell Ferran. 2021. Multilevel Dataflow-Driven Macro Placement Guided by RTL Structure and Analytical Methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 12 (Dec. 2021), 2542–2555.

[2] Charles J. Alpert, Zhuo Li, Michael D. Moffitt, Gi-Joon Nam, Jarrod A. Roy, and Gustavo Tellez. 2010. What makes a design difficult to route. Paper presented at the 10th International Symposium on Physical Design, San Francisco, California, USA, 14–16 March 2010.

[3] Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu. 2023. Graph neural networks: A powerful and versatile tool for advancing design, reliability, and security of ICs. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. Association for Computing Machinery, New York, NY, USA, 83–90.

[4] M.B. Anand, H. Shibata, and M Kakumu. 1998. Multiobjective optimization of VLSI interconnect parameters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17 (1998), 1252–1261.

[5] Mirhoseini Azalia and Anna Goldie. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (Jun. 2021), 207–212.

[6] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Lin Xie, and Jinkyoo Park. 2024. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. *arXiv preprint arXiv:2306.17100* (2024). https://github.com/ai4co/rl4co.

[7] Wang Bowen, Shen Guibao, Li Dong, Hao Jianye, Liu Wulong, Huang Yu, Wu Hongzhong, Lin Yibo, Chen Guangyong, and Heng Pheng Ann. 2022. LHNN: lattice hypergraph neural network for VLSI congestion prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC '22)*. Association for Computing Machinery, New York, NY, USA, 1297–1302.

[8] Melvin A. Breuer. 1977. A Class of Min-Cut Placement Algorithms. Paper presented at the Proceedings of the 14th Design Automation Conference, January 1977.

[9] Ismail Bustany, David Chinnery, Joseph Shinnerl, and Vladimir Yutsis. 2015. ISPD 2015 Benchmarks with Fence Regions and Routing Blockages for Detailed-Routing-Driven Placement. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 157–164.

[10] Fu-Chieh Chang, Yu-Wei Tseng, Ya-Wen Yu, and Ssu-Rui Lee. 2022. Flexible Multiple-Objective Reinforcement Learning for Chip Placement. https://doi.org/10.48550/ARXIV.2204.06407

[11] Gengjie Chen, Chak-Wa Pui, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Evangeline F. Y. Young, and Bei Yu. 2018. RippleFPGA: Routability-Driven Simultaneous Packing and Placement for Modern FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 10 (2018), 2022–2035. https://doi.org/10.1109/TCAD.2017.2778058

[12] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. 2008. NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27 (2008), 1228–1240. https://doi.org/10.1109/TCAD.2008.923063

[13] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. 2019. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 9 (2019), 1717–1730. https://doi.org/10.1109/TCAD.2018.2859220

[14] Ruoyu Cheng, Xianglong Lyu, Yang Li, Junjie Ye, Jianye Hao, and Junchi Yan. 2022. The policy-gradient placement and generative routing neural networks for chip design. *Advances in Neural Information Processing Systems* 35 (2022), 26350–26362.

[15] Ruoyu Cheng and Junchi Yan. 2021. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems* 34 (2021), 16508–16519.

[16] Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li. 2012. NCTU-GR: Efficient Simulated Evolution-Based Rerouting and Congestion-Relaxed Layer Assignment on 3-D Global Routing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20 (2012), 459–472. https://doi.org/10.1109/TVLSI.2010.2102780

[17] weitao du, He Zhang, Yuanqi Du, Qi Meng, Wei Chen, Tie-Yan Liu, Nanning Zheng, and Bin Shao. 2022. SE(3) Equivariant Graph Neural Networks with Complete Local Frames. In *ICML 2022*, Vol. 162. Microsoft, PMLR, Honolulu, Hawaii, USA, 5583–5608.

[18] Yunbo Hou, Haoran Ye, Yingxue Zhang, Siyuan Xu, and Guojie Song. 2024. RoutePlacer: An End-to-End Routability-Aware Placer with Graph Neural Network. arXiv:2406.02651 [cs.LG] https://arxiv.org/abs/2406.02651

[19] Meng-Kai Hsu, Yao-Wen Chang, and Valeriy Balabanov. 2011. TSV-aware analytical placement for 3D IC designs. Paper presented at the 48th ACM/EDAC/IEEE Design Automation Conference, San Diego, CA, USA, 5–9 June 2011.

[20] Meng-Kai Hsu, Yi-Fang Chen, Chau-Chin Huang, Sheng Chou, Tzu-Hen Lin, Tung-Chieh Chen, and Yao-Wen Chang. 2014. NTUplace4h: A Novel Routability-Driven Placement Algorithm for Hierarchical Mixed-Size Circuit Designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33 (2014), 1914–1927. https://doi.org/10.1109/TCAD.2014.2360453

[21] Zixuan Jiang, Ebrahim Songhori, Shen Wang, Anna Goldie, Azalia Mirhoseini, Joe Jiang, Young-Joon Lee, and David Z Pan. 2021. Delving into macro placement with reinforcement learning. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. IEEE, IEEE, New York, NY, USA, 1–3.

[22] Lu Jingwei, Chen Pengwen, Chang Chin-Chih, Sha Lu, Huang Dennis Jen-Hsin, Teng Chin-Chi, and Cheng Chung-Kuan. 2015. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. *ACM*

Transactions on Design Automation of Electronic Systems 20, 2 (Mar. 2015), 1–34. https://doi.org/10.1145/2699873

[23] Haeyeon Kim, Minsu Kim, Federico Berto, Joungho Kim, and Jinkyoo Park. 2023. Devformer: A symmetric transformer for context-aware device placement. In *International Conference on Machine Learning*. PMLR, JMLR.org, Honolulu, Hawaii, USA, 16541–16566.

[24] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. 2015. ICCAD-2015 CAD Contest in Incremental Timing-driven Placement and Benchmark Suite. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (Austin, TX, USA) *(ICCAD '15)*. IEEE Press, New York, NY, USA, 921–926.

[25] Robert Kirby, Saad Godil, Rajarshi Roy, and Bryan Catanzaro. 2019. CongestionNet: Routing congestion prediction using deep graph neural networks. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, IEEE, New York, NY, USA, 217–222.

[26] Schütt Kristof, Sauceda Huziel E., Kindermans P.-J, Tkatchenko Alexandre, and Müller Klaus-Robert. 2018. SchNet – A deep learning architecture for molecules and materials. *The Journal of Chemical Physics* 148, 24 (Jun. 2018), 241722. https://doi.org/10.1063/1.5019779

[27] Yao Lai, Jinxin Liu, Zhentao Tang, Bin Wang, Jianye Hao, and Ping Luo. 2023. ChiPFormer: transferable chip placement via offline decision transformer. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*. JMLR.org, Honolulu, Hawaii, USA, 18346–18364. https://doi.org/10.5555/3618408.3619165

[28] Yao Lai, Yao Mu, and Ping Luo. 2022. Maskplace: Fast chip placement via reinforced visual representation learning. *Advances in Neural Information Processing Systems* 35 (2022), 24019–24030.

[29] Peiyu Liao, Siting Liu, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. 2022. DREAMPlace 4.0: Timing-driven Global Placement with Momentum-based Net Weighting. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. European Design and Automation Association, Leuven, BEL, 939–944. https://doi.org/10.23919/DATE54114.2022.9774725

[30] Jai-Ming Lin, You-Lun Deng, Ya-Chu Yang, Jia-Jian Chen, and Po-Chen Lu. 2021. Dataflow-Aware Macro Placement Based on Simulated Evolution Algorithm for Mixed-Size Designs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 5 (2021), 973–984. https://doi.org/10.1109/TVLSI.2021.3057921

[31] Yibo Lin, Zixuan Jiang, Jiaqi Gu, Wuxi Li, Shounak Dhar, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2021. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 4 (2021), 748–761. https://doi.org/10.1109/TCAD.2020.3003843

[32] Yibo Lin, Wuxi Li, Jiaqi Gu, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2020. ABCDPlace: Accelerated Batch-Based Concurrent Detailed Placement on Multithreaded CPUs and GPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39 (2020), 5083–5096.

[33] Yibo Lin, David Z. Pan, Haoxing Ren, and Brucek Khailany. 2020. DREAMPlace 2.0: Open-Source GPU-Accelerated Global and Detailed Placement for Large-Scale VLSI Designs. In *2020 China Semiconductor Technology International Conference (CSTIC)*. IEEE, New York, NY, USA, 1–4.

[34] Zhifeng Lin, Yanyue Xie, Gang Qian, Jianli Chen, Sifei Wang, Jun Yu, and Yao-Wen Chang. 2021. Timing-Driven Placement for FPGAs with Heterogeneous Architectures and Clock Constraints. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE Press, New York, NY, USA, 1564–1569.

[35] Wen-Hao Liu, Stefanus Mantik, Wing-Kai Chow, Yixiao Ding, Amin Farshidi, and Gracieli Posser. 2019. ISPD 2019 Initial Detailed Routing Contest and Benchmark with Advanced Routing Rules. Paper presented at the Proceedings of the 2019 Symposium on International Symposium on Physical Design, San Francisco, CA, USA, 14–17 April 2019.

[36] 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 *Proceedings of the 59th ACM/IEEE Design Automation Conference*. Association for Computing Machinery, San Francisco, California, 1303–1308.

[37] Yiting Liu, Ziyi Ju, Zhengming Li, Mingzhi Dong, Hai Zhou, Jia Wang, Fan Yang, Xuan Zeng, and Li Shang. 2022. Graphplanner: Floorplanning with graph neural network. *ACM Transactions on Design Automation of Electronic Systems* 28, 2 (2022), 1–24.

[38] Yen-Chun Liu, Tung-Chieh Chen, Yao-Wen Chang, and Sy-Yen Kuo. 2019. MDP-Trees: Multi-Domain Macro Placement for Ultra Large-Scale Mixed-Size Designs. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '19)*. Association for Computing Machinery, New York, NY, USA, 557–562. https://doi.org/10.1145/3287624.3287677

[39] Daniela Sánchez Lopera and Wolfgang Ecker. 2022. Applying gnns to timing estimation at rtl. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. Association for Computing Machinery, New York, NY, USA, 1–8.

[40] Yi-Chen Lu, Siddhartha Nath, Vishal Khandelwal, and Sung Kyu Lim. 2021. Doomed run prediction in physical design by exploiting sequential flow and graph learning. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, IEEE, New York, NY, USA, 1–9.

[41] Yi-Chen Lu, Siddhartha Nath, Vishal Khandelwal, and Sung Kyu Lim. 2021. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, IEEE, New York, NY, USA, 733–738.

[42] Yibai Meng, Wuxi Li, Yibo Lin, and David Z. Pan. 2022. elfPlace: Electrostatics-Based Placement for Large-Scale Heterogeneous FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 1 (2022), 155–168. https://doi.org/10.1109/TCAD.2021.3053191

[43] Cheng Ruoyu and Yan Junchi. 2021. On joint learning for solving placement and routing in chip design. In *Advances in Neural Information Processing Systems (NeurIPS '21)*. Curran Associates, Inc., New York, NY, USA, 16508–16519. https://doi.org/10.48550/arXiv.2111.00234

[44] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. 2021. E (n) equivariant graph neural networks. In *International conference on machine learning*. PMLR, Proceedings of Machine Learning Research, Virtual Event, 9323–9332.

[45] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. 2023. High-Quality Hypergraph Partitioning. *ACM J. Exp. Algorithmics* 27, Article 1.9 (feb 2023), 39 pages. https://doi.org/10.1145/3529090

[46] C. Sechen and A. Sangiovanni-Vincentelli. 1985. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits* 20 (1985), 510–522. https://doi.org/10.1109/JSSC.1985.1052337

[47] Yunqi Shi, Ke Xue, Song Lei, and Chao Qian. 2024. Macro placement by wire-mask-guided black-box optimization. *Advances in Neural Information Processing Systems* 36 (2024), 19 pages.

[48] Yunqi Shi, Ke Xue, Lei Song, and Chao Qian. 2023. Macro Placement by Wire-Mask-Guided Black-Box Optimization. In *Advances in Neural Information Processing Systems (NeurIPS '23)*. Curran Associates, Inc., New Orleans, USA, 6825–6843. https://doi.org/10.48550/arXiv.2306.16844

[49] Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes. 2008. Abacus: Fast Legalization of Standard Cell Circuits with Minimal Movement. Paper presented at the Proceedings of the 2008 International Symposium on Physical Design, Portland, Oregon, USA, 13–16 April 2008.

[50] Natarajan Viswanathan, Charles Alpert, Cliff Sze, Zhuo Li, and Yaoguang Wei. 2012. ISPD 2019 Initial Detailed Routing Contest and Benchmark with Advanced Routing Rules. Paper presented at the Proceedings of the 49th Annual Design Automation Conference, San Francisco, CA, USA, 3–7 June 2012.

[51] J. Vygen. 2006. Slack in static timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 9 (2006), 1876–1885.

[52] Nan Wu, Yuan Xie, and Cong Hao. 2021. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. Association for Computing Machinery, New York, NY, USA, 39–44.

[53] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.

[54] Yue Xu, Yanheng Zhang, and Chris Chu. 2009. FastRoute 4.0: Global router with efficient via minimization. Paper presented at the 2009 Asia and South Pacific Design Automation Conference, Yokohama, 19–22 January 2009.

[55] Shuwen Yang, Ziyao Li, Guojie Song, and Lingsheng Cai. 2021. Deep Molecular Representation Learning via Fusing Physical and Chemical Information. Paper presented at the Advances in Neural Information Processing Systems, 28 November – 9 December 2022.

[56] Shuwen Yang, Zhihao Yang, Dong Li, Yingxue Zhang, Zhanguang Zhang, Guojie Song, and Jianye HAO. 2022. Versatile Multi-stage Graph Neural Network for Circuit Representation. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). Curran Associates, Inc., New Orleans, Louisiana, USA, Article 1477, 11 pages.

[57] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *Advances in Neural Information Processing Systems*. https://github.com/ai4co/reevo.

[58] Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. 2023. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. Paper presented at the Advances in Neural Information Processing Systems, New Orleans, USA, 10–16 December 2023.

[59] Ricardo Salem Zebulum, Marco Aurélio Pacheco, and Marley Vellasco. 2015. A multi-objective optimisation methodology applied to the synthesis of low-power operational amplifiers. Paper presented at the 13th International Conference in Microelectronics and Packaging, Singapore, 7–9 December 2011.

[60] Xinyi Zhou, Junjie Ye, Chak-Wa Pui, Kun Shao, Guangliang Zhang, Bin Wang, Jianye Hao, Guangyong Chen, and Pheng Ann Heng. 2022. Heterogeneous graph neural network-based imitation learning for gate sizing acceleration. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. Association for Computing Machinery, New York, NY, USA, 1–9.

A Sub-netlist Graph Construction

The construction of sub-netlist graph follows Algorithm 1.

Algorithm 1 Extracting sub-netlist graphs and coarsening original netlist graph.

Input: netlist graph $\mathcal{G} = \{\mathbb{V}, \mathbb{U}, \mathbb{P}, \boldsymbol{X}_{\mathbb{V}}, \boldsymbol{X}_{\mathbb{U}}, \boldsymbol{X}_{\mathbb{P}}, \boldsymbol{P}_T, \boldsymbol{P}_P\}$, partition sets $\{\mathbb{V}_i\}$
Initialize $\mathbb{U}_i = \emptyset, \mathbb{P}_i = \emptyset \quad \forall i = 1, \cdots, n$
Initialize f_{BELONG} with $\{\mathbb{V}_i\}$
for all $(v, u) \in \mathbb{P}$ with $f_{\text{BELONG}}(v) \neq 0$ **do**
  Add u into $\mathbb{U}_{f_{\text{BELONG}}(v)}$
  Add (v, u) into $\mathbb{P}_{f_{\text{BELONG}}(v)}$
end for
Initialize pseudo cell set $\hat{\mathbb{V}}_M = \emptyset$
for all $i \in \{1, \cdots, n\}$ **do**
  Initialize feature matrices:

$$\boldsymbol{X}_{\mathbb{V},i} = \boldsymbol{X}_{\mathbb{V}}[\mathbb{V}_i, :], \boldsymbol{X}_{\mathbb{U},i} = \boldsymbol{X}_{\mathbb{U}}[\mathbb{U}_i, :], \boldsymbol{X}_{\mathbb{P},i} = \boldsymbol{X}_{\mathbb{P}}[\mathbb{P}_i, :] \tag{33}$$

  Find the biggest cell $v^* \in \mathbb{V}_i$
  Initialize terminal and movable:

$$\mathbb{V}_{T,i} = \{v^*\}, \mathbb{V}_{P,i} = \emptyset, \mathbb{V}_{M,i} = \mathbb{V}_i / v^* \tag{34}$$

  Initialize cell positions $\boldsymbol{P}_{T,i} = [[0, 0]], \boldsymbol{P}_{P,i} = []$
  Construct i-th sub-netlist graph:

$$\mathcal{G}_i = \{\mathbb{V}_{T,i} \cup \mathbb{V}_{M,i}, \mathbb{U}_i, \mathbb{P}_i, \boldsymbol{X}_{\mathbb{V},i}, \boldsymbol{X}_{\mathbb{U},i}, \boldsymbol{X}_{\mathbb{P},i}, \boldsymbol{P}_{T,i}, \boldsymbol{P}_{P,i}\} \tag{35}$$

  Add **Pseudo Cell** $\hat{v}_i$ into $\hat{\mathbb{V}}_M$
end for
Set $\mathbb{V}'_M = \mathbb{V}_M - \bigcup_{i=1,\cdots,n} \mathbb{V}_i + \hat{\mathbb{V}}_M$
Set $\mathbb{U}' = \{u \in \mathbb{U} | \exists v \in \mathbb{V}'_M, (v, u) \in \mathbb{P}\}$
Initialize **Pseudo Pin** set:

$$\hat{\mathbb{P}} = \{(\hat{v}_i, u) | \exists v \in \mathbb{V}_i, (v, u) \in \mathbb{P} \wedge u \in \mathbb{U}'\} \tag{36}$$

Set $\mathbb{P}' = \{(v, u) \in \mathbb{P} | v \in \mathbb{V}'_M\} \cup \hat{\mathbb{P}}$
Calculate $\boldsymbol{X}'_{\mathbb{V}}, \boldsymbol{X}'_{\mathbb{U}}, \boldsymbol{X}'_{\mathbb{P}}$ according to Appendix C
Root graph $\mathcal{G}_R$:

$$\mathcal{G}_R = \{\mathbb{V}_T \cup \mathbb{V}_P \cup \mathbb{V}'_M, \mathbb{U}', \mathbb{P}', \boldsymbol{X}'_{\mathbb{V}}, \boldsymbol{X}'_{\mathbb{U}}, \boldsymbol{X}'_{\mathbb{P}}, \boldsymbol{P}_T, \boldsymbol{P}_P\} \tag{37}$$

Output: sub-netlist graphs $\{\mathcal{G}_i\}$ and coarsened netlist graph $\mathcal{G}_R$

B Cell-flow Construction

The construction of cell-flow follows Algorithm 2.

Algorithm 2 The algorithm of constructing cell-flow from a netlist graph.

Input: netlist graph $\mathcal{G} = \{\mathbb{V}, \mathbb{U}, \mathbb{P}, \boldsymbol{X}_{\mathbb{V}}, \boldsymbol{X}_{\mathbb{U}}, \boldsymbol{X}_{\mathbb{P}}, \boldsymbol{P}_T, \boldsymbol{P}_P\}$
Initialize flow edges $\mathbb{F} = \emptyset$
Initialize cell-incoming-edge function $f_{\text{INCOME}} \in \mathbb{V} \rightarrow \mathbb{F}$
Initialize edge-net function $f_{\text{NET}} \in \mathbb{F} \rightarrow \mathbb{U}$
Initialize $\hat{\mathbb{V}} = \mathbb{V}_T \cup \mathbb{V}_P$ and $\hat{\mathbb{U}} = \mathbb{U}$
repeat
  Pop a cell v from $\hat{\mathbb{V}}$
  for all u with $(v, u) \in \mathbb{P} \wedge u \in \hat{\mathbb{U}}$ **do**
    Remove u from $\hat{\mathbb{U}}$
    for all $\overline{v}$ with $(\overline{v}, u) \in \mathbb{P} \wedge \overline{v} \in \mathbb{V}_M$ **do**
      Add $\overline{v}$ to $\hat{\mathbb{V}}$
      Add $(v, \overline{v})$ to $\mathbb{F}$
      Set $f_{\text{NET}}((v, \overline{v})) = u$
      if $f_{\text{INCOME}}(\overline{v}) =$ NULL **then**
        Set $f_{\text{INCOME}}(\overline{v}) = (v, \overline{v})$
      end if
    end for
  end for
until $\hat{\mathbb{V}}$ is empty
Output: cell-flow $\mathbb{F}$, function f_{INCOME} and f_{NET}

C Featurization of pseudo cells and pins

Although most of $\boldsymbol{X}'_{\mathbb{V}}, \boldsymbol{X}'_{\mathbb{U}}, \boldsymbol{X}'_{\mathbb{P}}$ can be inherited from original netlist graph $\mathcal{G}$, we need to generate features for pseudo cells $\hat{\mathbb{V}}_M$ and pseudo pins $\hat{\mathbb{P}}$. Here, we regard a pseudo cell $\hat{v}_i$ as normal cell with width and height $(\sqrt{5S_i}, \sqrt{5S_i})$, where $S_i = \sum_{v_j \in \mathbb{V}_i} w_j h_j$ is the summation of cell areas inside i-th partition $\mathbb{V}_i$. Pseudo pins are regarded as normal pins connected among nets and pseudo cells. Then we follow the cell/pin featurization in [56] to generate the features for pseudo cells and pins, and concatenate them to $\boldsymbol{X}'_{\mathbb{V}}, \boldsymbol{X}'_{\mathbb{P}}$.

D Optimality Demonstration

D.1 Optimality of Sub-netlist Graph Generation

The time complexity of Algorithm 1 is composed of three parts:

(1) Initialization of all $\mathbb{U}_i$ and $\mathbb{P}_i$ takes $O(|\mathbb{P}|)$.
(2) Constructing sub-netlist graphs takes $\sum_i (|\mathbb{V}_i| + |\mathbb{U}_i| + |\mathbb{P}_i|) = O(|\mathbb{V}| + \eta|\mathbb{U}| + |\mathbb{P}|)$, with overlap ratio $\eta = \sum_i |\mathbb{U}_i| / |\mathbb{U}|$.
(3) Note that we set $f_{\text{BELONG}}(\hat{v}_i) = 0$, so we can identify $v \in \mathbb{V}'_M$ and $v \in \mathbb{V}_i$ in $O(1)$ time. $u \in \mathbb{U}'$ can also be identified if we label the nets when calculating $\mathbb{U}'$. Therefore, constructing root graph $\mathcal{G}_R$ takes $O(|\mathbb{V}| + |\mathbb{U}| + |\mathbb{P}|)$ in total.

Time complexity of sub-netlist graph generation is $O(|\mathbb{V}| + \eta|\mathbb{U}| + |\mathbb{P}|)$. With the help of placement prototype tool KaHyPar[45], it can approximate $O(|\mathbb{V}| + |\mathbb{U}| + |\mathbb{P}|)$ because the overlap among $\{\mathbb{U}_i | i = 1, \cdots, n\}$ is minor ($\eta \rightarrow 1$). Table 6 shows the overlap ratio η in major datasets.

D.2 Optimality of Cell-flow Generation

Algorithm 2 elaborates on how to generate cell-flow $\mathbb{F}$ from a netlist graph $\mathcal{G}$. Besides the initialization steps which cost $O(|\mathbb{V}| + |\mathbb{U}|)$, the bottleneck of its time consumption is the loop execution. For every $(v, u) \in \mathbb{P}$ and net u untraveled, we connect v to almost every neighbor through u and label u as a traveled net. Because each net is traveled at most once, the loop body executes for no more than $\sum_{u \in \mathbb{U}} |(v, u) \in \mathbb{P}| = |\mathbb{P}|$ times. Every statement in the loop body takes $O(1)$, so the time complexity of cell-flow generation is $O(|\mathbb{V}| + |\mathbb{U}| + |\mathbb{P}|)$. Also note that the number of cell-flow edges equals to the execution times of loop body, i.e. $|\mathbb{F}| = O(|\mathbb{P}|)$.

Table 5: Runtime comparison between DREAMPlace and TransPlace on ISPD 2015

Netlist	DREAMPlace	Ours(Inductive Placement)	Ours(Fine-tune)	Ours(Total)
mgc_des_perf_1	34.32	1.42	35.83	37.25
mgc_des_perf_a	143.58	1.17	70.08	71.25
mgc_des_perf_b	107.58	1.42	88.59	90.01
mgc_edit_dist_a	65.93	1.52	60.01	61.53
mgc_fft_1	35.14	0.44	10.61	11.05
mgc_fft_2	41.37	0.35	25.83	26.18
mgc_fft_a	35.24	0.29	11.35	11.64
mgc_matrix_mult_1	37.51	1.92	20.04	21.96
mgc_matrix_mult_2	39.60	1.89	18.65	20.54
mgc_matrix_mult_a	32.69	1.65	16.89	18.54
mgc_pci_bridge32_a	157.47	0.38	155.79	156.17
mgc_pci_bridge32_b	216.42	0.26	147.31	147.57
mgc_superblue11_a	97.39	11.40	74.29	85.69
mgc_superblue12	70.33	15.43	81.04	96.47
mgc_superblue14	35.33	7.11	26.35	33.46
mgc_superblue16_a	82.64	6.66	109.08	115.74
mgc_superblue19	27.23	5.84	24.97	30.81
Average Ratio	1.49	0.07	0.93	1.00

Table 6: Overlap Ratio η and Average Cell-path Length ω in `ISPD2015`.

Netlist	# of cells	# of nets	η	ω
mgc_fft_1	35K	33K	1.1262	3.1798
mgc_fft_2	35K	33K	1.1257	3.1941
mgc_fft_a	34K	32K	1.0369	2.6055
mgc_fft_b	34K	32K	1.0399	2.5855
mgc_des_perf_1	113K	113K	1.1079	15.7122
mgc_matrix_mult_1	155K	158K	1.1177	5.4420
mgc_matrix_mult_2	155K	158K	1.1178	5.4057
mgc_matrix_mult_a	149K	154K	1.0370	3.6518
mgc_superblue12	1293K	1293K	1.1509	20.8853
mgc_superblue14	634K	620K	1.1305	4.4139
mgc_superblue19	522K	512K	1.0969	4.4376

D.3 Optimality of SE(2)-invariant Encoding and Decoding

The time complexities of encoding and decoding are $O(|\mathbb{F}|) = O(|\mathbb{P}|)$ and $O(\omega|\mathbb{F}|) = O(\omega|\mathbb{P}|)$, respectively, where ω is the average length of cell-paths. The concrete values of ω in major datasets are listed in Table 6. For most datasets, ω is small, so it can be regarded as a constant.

D.4 Time Complexity of Fine-tuning

In every epoch, the time complexities of calculating Wire-length, Density and Anchor objectives are $O(|\mathbb{P}|)$, $O(|\mathbb{V}|\log|\mathbb{V}|)$ [22], and $O(|\mathbb{V}|)$, respectively. So the whole fine-tuning has the complexity of $O(t_f(|\mathbb{P}| + |\mathbb{V}|\log|\mathbb{V}|))$.

E Experimental Settings

E.1 TPGNN Training

TPGNN is trained to imitate preplaced circuits generated by DREAMPlace [31]. These circuits consist of netlist graphs $\mathcal{G}$ with absolute positions for movable cells $\hat{\boldsymbol{P}}_M$. We encode these absolute positions into ground-truth relative positions, which serve as training labels.

The hidden layer dimensions of *cell*, *net*, and *pin* are set to $D_{\mathbb{V}} = 64$, $D_{\mathbb{U}} = 64$, and $D_{\mathbb{P}} = 8$, respectively. We set message passing layers $L = 3$ and loss weights $\lambda_\theta = 1.0, \lambda_{\mathrm{D}} = 8e-6, \lambda_{\mathrm{A}} = 1e-2$.

When optimizing **TPGNN** with `Adam Optimizer`, we use learning rate $\Lambda_t = 5e-5$, learning rate decay $\Delta\Lambda_t = 1e-3$, weight decay $\eta_t = 5e-4$, and training epoch $\tau_t = 500$. The training loss is determined by the difference between TPGNN outputs and ground truth relative positions:

$$\mathcal{L}_{\text{train}} = \text{Smooth-L1}(\boldsymbol{\rho}, \hat{\boldsymbol{\rho}}) + \lambda_\theta \cdot \text{Smooth-L1}(\Delta\boldsymbol{\theta}, \Delta\hat{\boldsymbol{\theta}}), \tag{38}$$

where Smooth-L1 is a robust L1 loss function, and λ_θ weighs two loss terms, which is set to 0.1. Our source code is available at https://github.com/sorarain/TransPlace.

E.2 Optimization Settings

We use default baseline settings. Both DREAMPlace and circuit-adaptive fine-tuning use `NAG Optimizer` [22] for a fair comparison, adopting the γ and λ_{D} adjustment strategy from Lu et al. [22]. Routability optimization via cell inflation follows Lin et al. [33] with default hyperparameters. Timing optimization differs from DREAMPlace [29], evaluating timing metrics and updating net weights every 15 iterations after placement overflow reaches $st_{overflow} = 0.5$ or $st_{iteration} = 500$. For `DAC2012`, `ISPD2015`, and `ISPD2019`, the rudy map guides cell inflation. Our fine-tuning parameters are given in Table 7, Table 8, Table 9, and Table 4; "Iteration" is the maximum iteration limit, a variable setting up an early stop strategy [22, 31].

Table 7: Hyperparameters of circuit-adaptive fine-tuning on ISPD2015.

Netlist	Learning_rate	Density_weight	Iteration	RePlAce_UPPER_PCOF	Theta	Delta_x	Delta_y
mgc_des_perf_1	1e-1	8e-5	400	1.05	120	43.46	0
mgc_des_perf_a	1e-2	8e-4	650	1.07	0	87.89	87.89
mgc_des_perf_b	1e-2	8e-4	600	1.07	180	-58.59	0
mgc_edit_dist_a	1e-2	8e-4	650	1.07	120	-78.12	78.12
mgc_fft_1	1e-2	8e-3	170	1.10	90	0	-25.78
mgc_fft_2	3e-1	8e-3	150	1.10	90	-33.40	-33.40
mgc_fft_a	3e-1	8e-3	150	1.10	270	-156.25	78.12
mgc_matrix_mult_1	1e-1	8e-5	350	1.07	0	107.42	-107.42
mgc_matrix_mult_2	1e-2	8e-4	320	1.07	0	-108.28	0
mgc_matrix_mult_a	1e-1	8e-4	300	1.07	270	-585.94	0
mgc_pci_bridge32_a	1e-2	8e-5	750	1.05	180	39.06	-78.12
mgc_pci_bridge32_b	1e-2	8e-5	1000	1.05	90	78.12	-156.25
mgc_superblue11_a	1e-2	8e-4	650	1.05	240	-786.80	684.67
mgc_superblue12	1e-2	8e-9	1000	1.06	0	-276.80	201.62
mgc_superblue14	1e-2	8e-4	350	1.06	0	-253.05	-368.09
mgc_superblue16_a	1e-2	8e-9	2000	1.06	0	-144.81	-157.15
mgc_superblue19	1e-1	8e-5	400	1.05	270	-199.38	403.24

Table 8: Hyperparameters of circuit-adaptive fine-tuning on DAC2012.

Netlist	Learning_rate	Density_weight	Iteration	RePlAce_UPPER_PCOF	Theta	Delta_x	Delta_y
superblue2	1e-2	3e-5	2000	1.05	270	-240.16	0
superblue3	1e-2	8e-9	2000	1.06	120	-498.95	-323.44
superblue6	1e-2	8e-7	2000	1.05	240	0	0
superblue7	3e-4	1e-5	2000	1.05	270	-155.33	0
superblue9	3e-3	3e-6	2000	1.05	270	-132.71	0
superblue11	1e-2	1e-3	2000	1.05	90	0	0
superblue12	3e-2	8e-4	600	1.05	90	-138.40	0
superblue14	1e-3	3e-5	1000	1.05	90	0	0
superblue16	4e-2	6e-5	1250	1.05	270	0	-157.15
superblue19	1e-2	8e-5	2000	1.05	270	-99.69	0

Table 9: Hyperparameters of circuit-adaptive fine-tuning on ISPD2019.

Netlist	Learning_rate	Density_weight	Iteration	Theta	Delta_x	Delta_y
ispd19_test1	2e-3	5e-9	550	90	0	-28.13
ispd19_test2	3e-4	1e-7	600	90	0	-113.91
ispd19_test3	2e-3	5e-9	550	180	-27.01	0
ispd19_test4	3e-4	1e-3	550	180	-156.25	-151.37
ispd19_test6	1e-2	8e-5	1000	0	-173.83	-149.53
ispd19_test7	1e-2	8e-5	1000	0	-217.29	-186.80
ispd19_test8	1e-2	8e-5	1000	0	-344.73	-224.06
ispd19_test9	1e-2	1e-7	700	0	-384.38	-300.23
ispd19_test10	1e-2	1e-5	550	180	0	0

F Detailed Runtime Analysis

Here we detailed analysis of the runtime of baselines and TransPlace.
We compare DREAMPlace with our method on ISPD2015 in Tab 5.

Table 10: Hyperparameters of circuit-adaptive fine-tuning on ICCAD2015.

Netlist	Learning_rate	Density_weight	Iteration	RePlAce_UPPER_PCOF	Theta	Delta_x	Delta_y
superblue1	1e-02	8e-03	1,000	1.06	72	-219.38	80.38
superblue3	1e-02	8e-05	1,000	1.05	0	0	0
superblue4	1e-02	8e-05	1,000	1.05	300	581.99	0
superblue5	1e-02	8e-05	1,000	1.05	0	0	0
superblue7	1e-02	8e-05	1,000	1.06	144	-310.66	278
superblue10	1e-02	8e-05	1,000	1.05	120	424.84	302.08
superblue16	1e-02	8e-05	1,000	1.05	144	-144.81	78.57
superblue18	1e-02	8e-05	1,000	1.05	0	0	0