# RoutePlacer: An End-to-End Routability-Aware Placer with 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
haoran-ye@outlook.com

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

Placement is a critical and challenging step of modern chip design, with routability being an essential indicator of placement quality. Current routability-oriented placers typically apply an iterative two-stage approach, wherein the first stage generates a placement solution, and the second stage provides non-differentiable routing results to heuristically improve the solution quality. This method hinders jointly optimizing the routability aspect during placement. To address this problem, this work introduces **RoutePlacer**, an end-to-end routability-aware placement method. It trains **RouteGNN**, a customized graph neural network, to efficiently and accurately predict routability by capturing and fusing geometric and topological representations of placements. Well-trained RouteGNN then serves as a differentiable approximation of routability, enabling end-to-end gradient-based routability optimization. In addition, RouteGNN can improve two-stage placers as a plug-and-play alternative to external routers. Our experiments on DREAMPlace, an open-source AI4EDA platform, show that RoutePlacer can reduce Total Overflow by up to 16% while maintaining routed wirelength, compared to the state-of-the-art; integrating RouteGNN within two-stage placers leads to a 44% reduction in Total Overflow without compromising wirelength.



*Corresponding author.





*KDD '24, August 25–29, 2024, Barcelona, Spain.*





ACM ISBN 979-8-4007-0490-1/24/08.
https://doi.org/10.1145/3637528.3671895



## CCS CONCEPTS

• **Hardware** → **Placement**; **Wire routing**; *Software tools for EDA.*

## KEYWORDS

Placement, Routing, END-to-End model, graph neural network, EDA

**ACM Reference Format:**
Yunbo Hou, Haoran Ye, Yingxue Zhang, Siyuan Xu, and Guojie Song. 2024. RoutePlacer: An End-to-End Routability-Aware Placer with Graph Neural Network. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3637528.3671895


## 1 INTRODUCTION

The development of integrated circuits (ICs) has significantly advanced technology, progressing from individual chips to complex computing systems. Placement, among others, plays a crucial role in the intricate design flow of circuits. It constructs geometric positions of electronic components, such as memory components and logical gates, based on topological netlists. Placement can provide informative feedback for the preceding design stages and has a profound effect on downstream steps, as the positions of electronic components significantly influence the circuit performance.

Formally, the placement problem can be expressed as follows: Consider a circuit design represented by a hypergraph $H = (V, E)$, where $V$ represents the set of electronic units or cells, and $E$ represents the set of hyperedges or nets between these cells. The primary goal is to determine $\boldsymbol{x}$ and $\boldsymbol{y}$ to minimize wirelength while avoiding overlap between cells, where $\boldsymbol{x}$ and $\boldsymbol{y}$ denote cell positions.

The concept of routability is crucial when evaluating the placements of very-large-scale integrated (VLSI) circuits. Routability measures how effectively electrical signal pathways can be created on a chip. Imagine placement as deciding the locations of buildings

within a city. Routing involves designing a road network for the city. The aim is to link different sections of the city (which are cells), with roads (which are wires), while preventing excessive crowding (to prevent signal interference) and fitting within the available space (complying with the chip's physical and technological limitations, such as wire width and layer spacing). Overflow, which reflects the density of wires, is the key indicator of routability. A lower overflow value indicates better routability, meaning that the chip can more efficiently form pathways that adhere to design requirements.

State-of-the-art (SOTA) analytical placers treat the placement of cells as a nonlinear optimization problem [11, 16, 22]. Their goal is to minimize a differentiable objective function that includes the wirelength and overlap, using gradient-based optimization techniques. However, as circuits become more complex, simply focusing on this objective can lead to poor routability and routing detour failures. To address this, modern placement methods incorporate additional algorithmic modules and create an iterative two-stage process [32]: the first stage generates a placement solution, and the second stage provides non-differentiable routing results to heuristically perturb the solution. However, since these two stages are isolated and routability information is non-differentiable, these methods cannot optimize routability while generating analytical solutions, which limits the ability to jointly optimize the routability metric during placement.

To address this problem, we propose **RoutePlacer**, an end-to-end routability-aware placer that incorporates a differentiable congestion penalty into its objective function. We parameterize the congestion penalty with a graph neural network (GNN) and train it to accurately predict the congestion. The well-trained GNN can provide gradients of predicted congestion w.r.t. cell positions. This gradient information can be directly employed for gradient-based optimization, via forward and backward propagation, to minimize congestion when generating analytical placements.

During forward propagation, accurate congestion estimation is critical for reasonable penalty (gradient) assignment. To achieve this, we parameterize the congestion penalty with graph neural networks (GNNs), which have demonstrated exceptional performance in related tasks [21]. Preserving circuit information has been highlighted as a key factor in GNN performance [3, 6, 21]. Therefore, we convert the circuit design, hypergraph $H = (V, E)$, into **RouteGraph**, a heterogeneous graph that preserves two sources of information: topological information from hypergraph $H = (V, E)$ and geometrical information from cell location $(\boldsymbol{x}, \boldsymbol{y})$. The graph construction has a linear time complexity w.r.t. the scale of the circuit design, which ensures its efficiency. The RouteGraph contains three types of vertices: cells, nets, and grid cells. We use pins $\mathcal{P}$ that connect cells and nets to represent topology (named $topo-edges$). For the geometry representation, we divide the layout into grids, each representing a grid cell. Neighboring grid cells are linked by $geom-edges$. Finally, each cell is connected to the grid cell corresponding to its location, represented by $grid-edges$.

Then, we design **RouteGNN** to give accurate routability estimation conditional on RouteGraph. To collect and enrich topological and geometrical information, RouteGNN performs message-passing on $topo-edges$, $geom-edges$, and $grid-edges$ individually and fuse the message to update representations of cells, nets, and grid cells. Multiple layers of message-passing and fusing are stacked to capture the deep relationships between topology and geometry. We sum up the routability estimation for all cells as a congestion penalty.

In backward propagation, we compute the gradient of congestion penalty w.r.t. cell locations for position updates. We introduce **Differentiable Geometrical Feature Computation** for computing the gradient of cell features w.r.t. cell locations. Employing the chain rule, we can acquire the gradient of the congestion penalty w.r.t. cell locations. Cell positions are updated using the Nesterov Accelerated Gradient (NAG) optimizer [11].

Interleaving forward and backward propagation yields our end-to-end routability-aware RoutePlacer. In addition, RouteGNN can improve traditional two-stage methods in a plug-and-play manner. One can replace any external router with RouteGNN to leverage its congestion estimation for routability-aware placement.

We summarize our contributions as follows.

(1) We propose **RoutePlacer**, a routability-aware circuit placement method. It parameterizes a congestion penalty with GNN and integrates the differentiable penalty term into the optimization objective. It enables end-to-end routability optimization and improves two-stage placers in a plug-and-play manner.
(2) We introduce **RouteGNN** to learn accurate routability estimations conditional on **RouteGraph**, an efficient heterogeneous graph structure with topological and geometrical features.
(3) We present **Differentiable Geometrical Feature Computation** to enable gradient-based optimization. It calculates the gradient of cell features w.r.t. cell locations, preserving the complete gradient flow during backward propagation.
(4) We evaluate RoutePlacer on `DAC2012` and `ISPD2011` benchmarks, based on DREAMPlace, an open-source EDA framework. RoutePlacer achieves a 16% reduction in Total Overflow while maintaining routed wirelength compared to prior state-of-the-art (SOTA). Integrating RouteGNN within two-stage placers leads to a 44% reduction in Total Overflow without compromising wirelength. They show the SOTA performance and extensibility of RoutePlacer.

## 2 RELATED WORK

### 2.1 Placement

Prior placement methods can mainly be divided into four categories of methods based on their optimization strategies: **Meta-heuristic** methods [2, 29, 30], **Reinforcement Learning (RL)** methods [4, 5, 12, 14, 19, 20, 28], **partition-based** methods [1], **Quadratic Analytical** methods [9, 25, 27], and **Nonlinear Analytical** methods [8, 26, 31]. Meta-heuristic methods treat the placement as a step-wise optimization problem and solve it with a heuristic algorithm such as Simulated Annealing and Genetic Algorithm, which can theoretically reach an optimal solution. RL methods regard the placement problem as a "board game" and train an agent to place the cells one by one. However, these methods suffer from slow convergence, which restricts their usage only to the circuits with a small number of large-sized cells. 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.

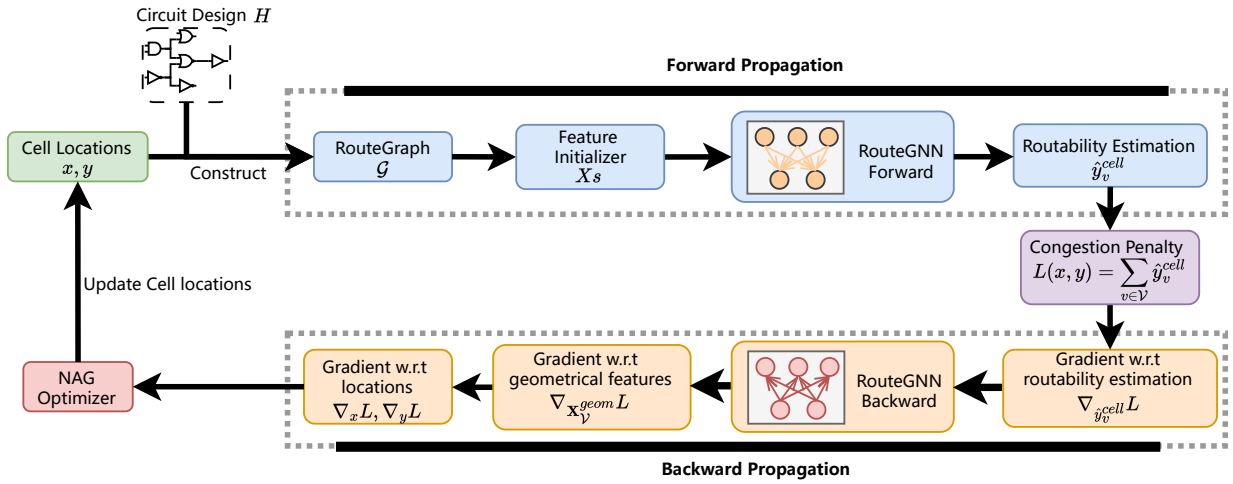



**Figure 1: Overview of RoutePlacer. Forward Propagation: We construct the RouteGraph and initialize features, which are then inputted into RouteGNN to obtain routability estimations. $Xs$ represents the features of cells, nets, grid cells, *topo-edges*, and *geom-edges*. Backward Propagation: We compute gradients of routability estimations w.r.t. cell locations via the proposed differentiable geometrical features and automatic differentiation tools. The gradient information is utilized for analytical routability optimization.**

Optimization methods are used to find solutions when the netlist and layout are sufficiently small.

The analytical methods are the mainstream choice for VLSI due to their efficiency and scalability. This methodological group formulates an objective function that includes wirelength and overlaps to optimize the positions of mixed-size cells. It can be further categorized into two types: quadratic and nonlinear methods. Quadratic methods [9, 25, 27] aim to minimize wirelength and address overlaps in an alternating manner, whereas nonlinear methods [8, 26, 31] employ a unified objective function that encompasses both wirelength and a parameterized overlap function. However, previous analytical placement approaches often overlook routability when optimizing the objective function, RoutePlacer introduces a differentiable congestion penalty into the objective function, enabling direct and synergistic routability optimization.

## 2.2 Routability Optimization

In modern placement, since traditional analytical placement methods struggle to guarantee satisfactory routability, two-stage routability optimization methods [7, 10, 15, 17] have been extensively developed as coarse-grained solutions. These methods typically comprise two phases: placement and routing. The placement phase employs a traditional placement approach, while the routing stage often uses heuristics and expert-designed routers to provide feedback for the placement phase to enhance routability. Specifically, two-stage methods expand all cell sizes to make more space for coarse-grained optimization. Therefore, we propose RoutePlacer, an end-to-end placer designed for fine-grained level optimization. RoutePlacer employs gradient-based optimization for cell position updates and can integrate with two-stage methods to achieve comprehensive optimization at both coarse-grained and fine-grained levels.

# 3 PROPOSED APPROACH

RoutePlacer is schematically illustrated in Fig. 1. We integrate a deep learning-based routability penalty into the objective function and calculate the congestion gradient to analytically optimize cell locations. Each optimization iteration consists of two steps: **Forward Propagation** and **Backward Propagation**. The forward propagation first constructs RouteGraph with its raw features. Then we input the RouteGraph and raw features into a well-trained RouteGNN to obtain routability estimation $\hat{y}_v^{cell}$ and sum up the estimation as routability penalty $L(\boldsymbol{x}, \boldsymbol{y}) = \sum_{v \in \mathcal{V}} \hat{y}_v^{cell}$. The backward propagation first calculates the gradient of congestion penalty over raw features and the gradient of raw features over cell locations. Based on chain rules, we can compute the gradient of routability penalty over cell locations. Finally, we apply the widely used gradient-based optimization method, NAG optimizer, to update cell locations and optimize routability. Note that the optimization targets cell locations rather than GNN parameters. The RouteGNN parameters, after training, are frozen during placement optimization.

## 3.1 Forward Propagation

*3.1.1 Circuit Design.* The circuit design is represented as a hypergraph that stores the topological information of the circuit produced in the logic synthesis stage. The circuit design is a hypergraph $H = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of electronic units (cells) and $\mathcal{E}$ represents the set of hyperedges (nets). We transform the hypergraph into a heterogeneous graph. To construct a netlist, we consider cells $\mathcal{V}$ and nets $\mathcal{E}$ as two types of vertices and link each net to cells interconnected by it. We call the constructed edges *topo-edges*, which stands for the topology between cells and nets.

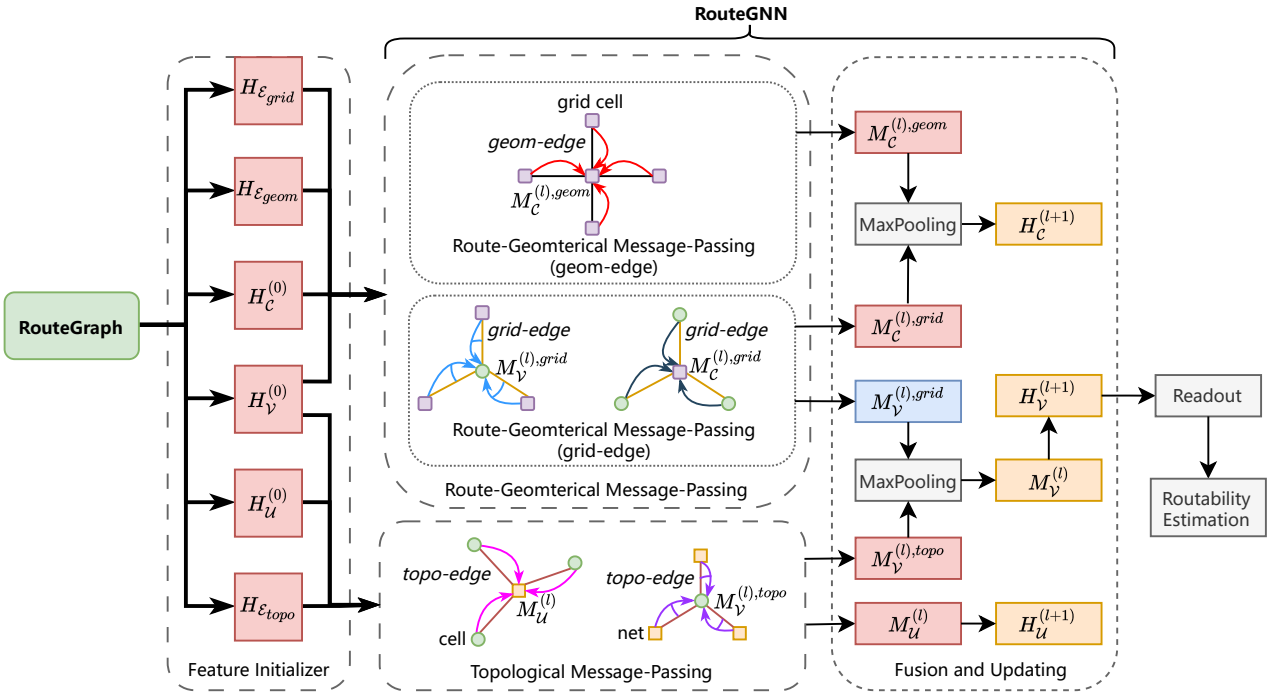



**Figure 2: Framework of RouteGNN. We initialize raw features and apply Route-Geometrical and Topological message-passing to gather topological and geometrical information. Then, we fuse and update the heterogeneous hidden representations to readout routability estimation.**

*3.1.2 RouteGraph.* In forward propagation, we require routability estimation using graph neural networks to calculate the congestion penalty. To achieve this, it is essential to design graphs that preserve circuit information. To better retain information within a single graph during placement, we must address two key challenges: effectiveness and efficiency.

Regarding effectiveness, in chip designs, the most informative sources are circuit design and cell locations. Handling these sources independently is suboptimal. Thus, we need to integrate them into a single heterogeneous graph.

Regarding efficiency, during the placement process, cells can be distributed in a very small area. Simply connecting geometrically adjacent cells can result in a time complexity of $O(||\mathcal{V}||^2)$, where $||\mathcal{V}||$ represents the number of cells. This level of complexity is unacceptable when dealing with circuits that comprise millions of cells. However, restricting geometrically adjacent links to a constant number may result in the omission of numerous edges, leading to an incomplete graph structure. Therefore, a proper trade-off is required to model geometrical information with minimal loss of information.

To address these challenges, we propose RouteGraph $\mathcal{G}$. For topology, we first transform the original hypergraph $H = (\mathcal{V}, \mathcal{E})$ into a heterogeneous graph with two types of vertices, including cells $\mathcal{V}$ and nets $\mathcal{E}$. Then we link the net to the cells that the corresponding hyperedge interconnects. The edge is called *topo-edges*, which stands for the topology between cells and nets.

To gather geometrical information, we begin by gridding the entire layout to create an $n \times m$ grid. Here, we set $n$ and $m$ to the numbers of routing grid cells in the horizontal and vertical directions as defined by the circuits. This grid serves as the basis for constructing a grid graph.

Within the grid graph, each node is referred to as a "grid cell" denoted as $C$, representing a specific grid in the layout. The edges, known as $\mathcal{E}_{geom}$ or *geom-edges*, signify the adjacent relationships between these grid cells. Additionally, we define *grid-edges* denoted as $\mathcal{E}_{grid}$ between cells and the corresponding grid cells within the grid graph. These *grid-edges* represent the precise positions of cells on the layout. Furthermore, *grid-edges* indirectly indicate the geometric adjacency relationships between cells.

We utilize *topo-edges*, *geom-edges*, and *grid-edges* to effectively integrate both topological and geometrical information into RouteGraph. As the complexity of constructing each type of edge is consistently linear, the overall complexity of constructing RouteGraph remains $O(||\mathcal{P}|| + ||\mathcal{V}|| + n + m)$ and is independent of cell distributions.

*3.1.3 Featurization.* RouteGraph comprises three types of vertices and three types of edges. In our model, *geom-edges* solely represent connections between grid cells, and as such, we do not assign features to *geom-edges*. We introduce $\boldsymbol{X}_{\mathcal{V}}$, $\boldsymbol{X}_{\mathcal{U}}$, $\boldsymbol{X}_{C}$, $\boldsymbol{X}_{\mathcal{E}_{topo}}$, and $\boldsymbol{X}_{\mathcal{E}_{grid}}$ to denote the features of cells $\mathcal{V}$, nets $\mathcal{U}$, grid cells $C$, topological edges $\mathcal{E}_{topo}$, and grid edges $\mathcal{E}_{grid}$, respectively. $\boldsymbol{X}_{\mathcal{V}}$ encompasses attributes of cells $\boldsymbol{X}_{\mathcal{V}}^{attr}$, such as size and connectivity

to nets, alongside the cells' geometric features $\boldsymbol{X}_{\mathcal{V}}^{geom}$, as detailed in Section 3.2.1. $\boldsymbol{X}_{\mathcal{U}}$ captures the span of nets as outlined in [6] and their connectivity to cells. $\boldsymbol{X}_{C}$ incorporates features like RUDY [18] and the central location of grid cells. $\boldsymbol{X}_{\mathcal{E}_{topo}}$ retains the intricate details of the interactions between cells and nets, employing the signal direction (input/output) as the feature for topological edges. Lastly, $\boldsymbol{X}_{\mathcal{E}_{grid}}$ records the distance between cell locations and the grid center to accurately represent their geometric adjacency.

*3.1.4 RouteGNN.* The framework of RouteGNN is shown in Fig. 2. RouteGNN takes a RouteGraph as input and maps raw features $\boldsymbol{X}_{\mathcal{V}}, \boldsymbol{X}_{\mathcal{U}}, \boldsymbol{X}_{C}, \boldsymbol{X}_{\mathcal{E}_{topo}}$ and $\boldsymbol{X}_{\mathcal{E}_{grid}}$ into hidden representations $\boldsymbol{H}_{\mathcal{V}}^{(0)}$, $\boldsymbol{H}_{\mathcal{U}}^{(0)}$, $\boldsymbol{H}_{C}^{(0)}$, $\boldsymbol{H}_{\mathcal{E}_{topo}}$, $\boldsymbol{H}_{\mathcal{E}_{grid}}$, via Multi-Layer Perceptrons (MLPs). Then, it generates deeper representations of cells $\mathcal{V}$, nets $\mathcal{U}$, and grid cells $C$ with $L$ layers of message-passing. Finally, the output cell representations are used for routability estimation after passing through readout layers.

In each layer, the topological information and the geometric information are collected through *Topological* and *Route-Geometrical* message-passing. The messages are then used to fuse and update the representations of cells $\mathcal{V}$, nets $\mathcal{U}$, and grid cells $C$.

To collect topological information, we interact the messages of cells $\mathcal{V}$ and nets $\mathcal{U}$ through *topo-edges* which preserve the topological connection between cells and nets. For layer $l$, we formulate the message passing as:

$$\boldsymbol{M}_{\mathcal{V}}^{(l),topo} = \Phi_{msg}^{\mathcal{U}\xrightarrow{\mathcal{E}_{topo}}\mathcal{V}}(\mathcal{U}, \mathcal{E}_{topo}, \boldsymbol{H}_{\mathcal{U}}^{(l)}, \boldsymbol{H}_{\mathcal{E}_{topo}}), \tag{1}$$

$$\boldsymbol{M}_{\mathcal{U}}^{(l)} = \Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{topo}}\mathcal{U}}(\mathcal{V}, \mathcal{E}_{topo}, \boldsymbol{H}_{\mathcal{V}}^{(l)}, \boldsymbol{H}_{\mathcal{E}_{topo}}), \tag{2}$$

where $\boldsymbol{M}_{\mathcal{V}}^{(l),topo}$ and $\boldsymbol{M}_{\mathcal{U}}^{(l)}$ denote the messages of cells $\mathcal{V}$ and nets $\mathcal{U}$ computed on *topo-edges* $\mathcal{E}_{topo}$; $\Phi_{msg}^{\mathcal{U}\xrightarrow{\mathcal{E}_{topo}}\mathcal{V}}$ is the message function which collects topological messages from nets $\mathcal{U}$ and sends them to cells $\mathcal{V}$ via *topo-edges* $\mathcal{E}_{topo}$. $\Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{topo}}\mathcal{U}}$ is similar. $\boldsymbol{H}_{\mathcal{V}}^{(l)}$ and $\boldsymbol{H}_{\mathcal{U}}^{(l)}$ denote the hidden representations of cells $\mathcal{V}$ and nets $\mathcal{U}$ of layer $l$. We design the message function $\Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{topo}}\mathcal{U}}$ as below to fuse the representations of surrounding cells $v|(v,u) \in \mathcal{E}_{topo}$ with *topo-edges* connecting them [13]:

$$\Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{topo}}\mathcal{U}}(\{(\mathbf{h}_{v}^{\mathcal{V}}, \mathbf{h}_{(v,u)}^{\mathcal{E}_{topo}})|(v,u) \in \mathcal{E}_{topo}\}) = \sum_{(v,u)\in\mathcal{E}_{topo}} (\boldsymbol{W}_{\mathcal{E}_{topo}\to\mathcal{U}}\mathbf{h}_{(u,v)}^{\mathcal{E}_{topo}}) \odot (\boldsymbol{W}_{\mathcal{V}\to\mathcal{U}}\mathbf{h}_{v}^{\mathcal{V}}), \tag{3}$$

where $\mathbf{h}_{v}^{\mathcal{V}}$ and $\mathbf{h}_{(u,v)}^{\mathcal{E}_{topo}}$ denote the hidden representation of the cell $v$ and the *topo-edge* $\mathcal{E}_{topo}$ connecting $v$ and $u$, respectively; $\boldsymbol{W}_{\mathcal{E}_{topo}\to\mathcal{U}}$ and $\boldsymbol{W}_{\mathcal{V}\to\mathcal{U}}$ are learnable weight matrices, and $\odot$ denotes the element-wise multiplication. Since the representations of *topo-edges* $\boldsymbol{H}_{\mathcal{E}_{topo}}$ have been collected, we design the message function $\Phi_{msg}^{\mathcal{U}\xrightarrow{\mathcal{E}_{topo}}\mathcal{V}}$ to speed up message-passing at the cost of minor loss of topological information. It is formally given as:

$$\Phi_{msg}^{\mathcal{U}\xrightarrow{\mathcal{E}_{topo}}\mathcal{V}}(\{\mathbf{h}_{u}^{\mathcal{U}}|(u,v) \in \mathcal{E}_{topo}\}) = \sum_{(u,v)\in\mathcal{E}_{topo}} (\boldsymbol{W}_{\mathcal{U}\to\mathcal{V}}\mathbf{h}_{v}^{\mathcal{U}}), \tag{4}$$

where $\mathbf{h}_{u}^{\mathcal{U}}$ is the hidden representation of net $u$ and $\boldsymbol{W}_{\mathcal{U}\to\mathcal{V}}$ is a learnable weight matrix.

Geometrical information is shared via interactions between geometrically adjacent cells in RouteGraph. Initially, cell messages are fused via *grid-edges*. These fused messages in grid cells are then exchanged to enable indirect interactions among adjacent cells through *geom-edges* $\mathcal{E}_{geom}$. Subsequently, grid cell messages are sent to cells via *grid-edges* $\mathcal{E}_{grid}$, enabling indirect interactions among geometrically adjacent cells.

In Route-Geometrical Message-passing, we first collect messages of cells $\mathcal{V}$ to obtain a fused message of cells for grid cells $C$:

$$\boldsymbol{M}_{C}^{(l),grid} = \Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{grid}}C}(\mathcal{V}, \mathcal{E}_{grid}, \boldsymbol{H}_{\mathcal{V}}^{(l)}, \boldsymbol{H}_{\mathcal{E}_{grid}}). \tag{5}$$

Here, $\boldsymbol{M}_{C}^{(l),grid}$ denotes the messages of grid cells $C$ computed on *grid-edges*. $\Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{grid}}C}$ is the message function that transfers geometrical messages from cells $\mathcal{V}$ to grid cells $C$ via *grid-edges* $\mathcal{E}_{grid}$.

Since grid cells are designed to collect geometrical messages from cells located in the grid, we design $\Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{grid}}C}$ to speed up the message-passing as below:

$$\Phi_{msg}^{\mathcal{V}\xrightarrow{\mathcal{E}_{grid}}C}(\{\mathbf{h}_{v}^{\mathcal{V}}|(c,v) \in \mathcal{E}_{grid}\}) = \sum_{(c,v)\in\mathcal{E}_{grid}} (\boldsymbol{W}_{\mathcal{V}\to C}\mathbf{h}_{v}^{\mathcal{V}}), \tag{6}$$

where $\boldsymbol{W}_{\mathcal{V}\to C}$ is a learnable weight matrix.

To facilitate interaction between grid cells $C$ that have aggregated messages from cells $\mathcal{V}$ within each grid, we utilize *geom-edges* to connect these grid cells.

$$\boldsymbol{M}_{C}^{(l),geom} = \Phi_{msg}^{C\xrightarrow{\mathcal{E}_{geom}}C}(C, \mathcal{E}_{geom}, \boldsymbol{H}_{C}^{(l)}, \boldsymbol{H}_{\mathcal{E}_{geom}}), \tag{7}$$

where $\boldsymbol{M}_{C}^{(l),geom}$ is the message of grid cells $C$ computed on *geom-edges* and $\Phi_{msg}^{C\xrightarrow{\mathcal{E}_{geom}}C}$ is the message function which collects geometrical messages from grid cells $C$; $\boldsymbol{H}_{C}^{(l)}$ denotes the hidden representations of a grid cell. Since $geom-edges$ are designed for interactions between grid cells $C$, we design $\Phi_{msg}^{C\xrightarrow{\mathcal{E}_{geom}}C}$ to accelerate such interactions:

$$\Phi_{msg}^{C\xrightarrow{\mathcal{E}_{geom}}C}(\{\mathbf{h}_{c}^{C}|(c,c^{*}) \in \mathcal{E}_{geom}\}) = \sum_{(c,c^{*})\in\mathcal{E}_{geom}} (\boldsymbol{W}_{C\to C}\mathbf{h}_{c^{*}}^{C}), \tag{8}$$

where $\mathbf{h}_{c}^{C}$ is the representations of grid cells $C$, and $\boldsymbol{W}_{C\to C}$ is a learnable weigth matrix.

Having collected messages from geometrically adjacent grid cells, we send the grid cell messages $C$ to cells via *grid-edges*, enabling indirect interaction among these adjacent cells.

$$\boldsymbol{M}_{\mathcal{V}}^{(l),grid} = \Phi_{msg}^{C\xrightarrow{\mathcal{E}_{grid}}\mathcal{V}}(C, \mathcal{E}_{grid}, \boldsymbol{H}_{C}^{(l)}, \boldsymbol{H}_{\mathcal{E}_{grid}}), \tag{9}$$

Here, $\boldsymbol{M}_{\mathcal{V}}^{(l),grid}$ denotes the messages of cells $\mathcal{V}$ computed on *grid-edges*. $\Phi_{msg}^{C\xrightarrow{\mathcal{E}_{grid}}\mathcal{V}}$ is the message function that transfers messages from grid cells $C$ to cells $\mathcal{V}$ via *grid-edges* $\mathcal{E}_{grid}$.

To further enhance geometrical information interaction between the cell with other surrounding cells located in the grid, we use *grid-edges* $\mathcal{E}_{grid}$ representations to compute edge weights when convolving cells. The message function $\Phi_{msg}^{C\xrightarrow{\mathcal{E}_{grid}}\mathcal{V}}$ is given below:

$$\Phi_{msg}^{C\xrightarrow{\mathcal{E}_{grid}}\mathcal{V}}(\{(\mathbf{h}_{c}^{C}, \mathbf{h}_{(c,v)}^{\mathcal{E}_{grid}}) | (c,v) \in \mathcal{E}_{grid}\}) = \sum_{(c,v)\in\mathcal{E}_{grid}} (\alpha^{T}\mathbf{h}_{(c,v)}^{\mathcal{E}_{grid}}) \cdot (\boldsymbol{W}_{C\to\mathcal{V}}\mathbf{h}_{c}^{C}), \tag{10}$$

where $\mathbf{h}_{(c,v)}^{\mathcal{E}_{grid}}$ is the representations vector of *grid-edges* connecting $c, v$, $\alpha$ is a learnable weight vector, and $\boldsymbol{W}_{C\to\mathcal{V}}$ is a learnable weight matrix.

After computing topological and geometrical messages for cells $\mathcal{V}$ and grid cells $C$ with Topological and Route-Geometrical message-passing, we fuse them and update representations for cells $\mathcal{V}$ to obtain fused message $\boldsymbol{M}_{\mathcal{V}}^{(l)}$.

$$\boldsymbol{M}_{\mathcal{V}}^{(l)} = MaxPooling(\boldsymbol{M}_{\mathcal{V}}^{(l),grid}, \boldsymbol{M}_{\mathcal{V}}^{(l),geom}). \tag{11}$$

The process is similar to grid cells, which aggregates cell representations from their own grids and adjacent grids through *grid-edges* and *geom-edges*. Consequently, we fuse the messages $\boldsymbol{M}_{C}^{(l),grid}$ and $\boldsymbol{M}_{C}^{(l),geom}$ to obtain the fused message $\boldsymbol{M}_{C}^{(l)}$.

$$\boldsymbol{M}_{C}^{(l)} = MaxPooling(\boldsymbol{M}_{C}^{(l),grid}, \boldsymbol{M}_{C}^{(l),topo}). \tag{12}$$

To enhance hidden representations, we fuse messages and hidden representations of layer $l$ to update hidden representations of layer $l+1$.

$$\begin{aligned} &\boldsymbol{H}_{\mathcal{V}}^{(l+1)} = \Phi_{update}(\boldsymbol{H}_{\mathcal{V}}^{(l)}, \boldsymbol{M}_{\mathcal{V}}^{(l)}), \quad \boldsymbol{H}_{\mathcal{U}}^{(l+1)} = \Phi_{update}(\boldsymbol{H}_{\mathcal{U}}^{(l)}, \boldsymbol{M}_{\mathcal{U}}^{(l)}), \\ &\boldsymbol{H}_{C}^{(l+1)} = \Phi_{update}(\boldsymbol{H}_{C}^{(l)}, \boldsymbol{M}_{C}^{(l)}), \end{aligned} \tag{13}$$

where the update function $\Phi_{update}(\boldsymbol{H}, \boldsymbol{M}) = \boldsymbol{H} + Tanh(\boldsymbol{M})$

After $L$ iterations of message-passing, we read out the cell representations $\boldsymbol{H}_{\mathcal{V}}, \boldsymbol{H}_{\mathcal{U}}$ for routability estimation. For cell congestion prediction, we concatenate raw cell features with cell representations and pass them to an MLP:

$$\hat{y}_{v}^{cell} = \mathrm{MLP}(\boldsymbol{H}_{\mathcal{V}}^{(L)} \oplus \boldsymbol{X}_{\mathcal{V}}), \tag{14}$$

where $\oplus$ is the element-wise addition. Finally, congestion penalty is formulated as $L(\boldsymbol{x}, \boldsymbol{y}) = \sum_{v\in\mathcal{V}} \hat{y}_{v}^{cell}$.

## 3.2 Backward Propagation

Following the computation of the congestion penalty, the next step involves backward propagating it to obtain the gradient for gradient-based optimization. Therefore, we need to calculate the gradient of the objective function w.r.t. cell locations. This section presents our approach for deriving the gradient of the congestion penalty $L$ with respect to cell locations $\nabla_x L, \nabla_y L$. As depicted in Fig. 1 and guided by the chain rule, the derivatives $\nabla_x L, \nabla_y L$ are articulated as follows:

$$\nabla_x L = (J_x(\boldsymbol{X}_{\mathcal{V}}))^T \cdot \nabla_{\boldsymbol{X}_{\mathcal{V}}} L, \quad \nabla_y L = (J_y(\boldsymbol{X}_{\mathcal{V}}))^T \cdot \nabla_{\boldsymbol{X}_{\mathcal{V}}} L. \tag{15}$$

Here $J$ denotes a Jacobian matrix. $(J_x(\boldsymbol{X}_{\mathcal{V}}))$ and $\nabla_{\boldsymbol{X}_{\mathcal{V}}} L$ correspond to the backward process of differentiable features and **RouteGNN**. $\nabla_{\boldsymbol{X}_{\mathcal{V}}} L$ can be calculated by automatic differentiation toolkits, such as PyTorch, and $J_x(\boldsymbol{X}_{\mathcal{V}})$ is derived in Section 3.2.1. Finally, we apply the NAG optimizer to update cell locations using the derived gradients.

*3.2.1 Differentiable Geometrical Feature Computation.* Geometrical features play a significant role in congestion estimation [18, 21]. We aim to collect the gradient $J_x(\boldsymbol{X}_{v}^{geom})$ of geometrical features over cell locations for gradient-based congestion optimization. However, geometrical features are grid features rather than cells' raw features; they are non-differentiable w.r.t. the cell positions [3]. Therefore, we need to transform the grid feature into a one-dimensional vector $\boldsymbol{X}_{\mathcal{V}}^{geom}$ as cell raw features and ensure that the vector is differentiable w.r.t. cell locations.

We propose a **Differentiable Geometrical Feature Computation** for a soft assignment of grids. The closer the routing grid is to the cell, the more wiring demand the cell has on that grid. Therefore, for each cell, we consider the closest nine grids to the cell and calculate the normalized weighted sum of RUDY, wherein the reciprocal of the distance between a cell and a grid serves the weight. The above process can be formulated as below:

$$\boldsymbol{X}_{v}^{geom} = \sum_{(a,b)\in N_v} w_{v,(a,b)} RUDY(a,b), \tag{16}$$

$$w_{v,(a,b)} = \mathrm{softmax}(\frac{1}{dis(v,(a,b))}), \quad \mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}, \tag{17}$$

where $N_v$ denotes the closest nine grids to the cell $v$, and $dis(v,(a,b))$ represents the distance between the center of the grid $(a,b)$ and the location of cells $v$. Finally, the $(J_x(\boldsymbol{X}_{v}^{geom}))$ can be written as:

$$J_x(\boldsymbol{X}_{v}^{geom}) = \sum_{(a,b)\in N_v} \frac{\partial dis(v,(a,b))}{\partial x} \frac{\partial w_{v,(a,b)}}{\partial dis(v,(a,b))}, \tag{18}$$

where $\frac{\partial dis(v,(a,b))}{\partial x}$ and $\frac{\partial w_{v,(a,b)}}{\partial dis(v,(a,b))}$ can be easily calculated in a way similar to $J_y(\boldsymbol{X}_{v}^{geom})$.

# 4 EXPERIMENTS

Our experiments aim to address three research questions (RQs). (1) Effectiveness: Can RoutePlacer improve routability compared with analytical placement methods? (2) Efficiency: Can RoutePlacer efficiently handle two sources of information and give accurate routability estimation? (3) Extensibility: Can RoutePlacer improve traditional methods by replacing an external router with RouteGNN?

For the first RQ, we compare RoutePlacer against DREAMPlace using routability-related metrics. For the second RQ, we compare our runtime with NetlistGNN [21] on the collected placement and visualize the SSIM and NRMSE metrics to verify the reliability of predictions. For the third RQ, we incorporate a cell inflation method (detailed in Appendix B.4) into DREAMPlace and RoutePlacer, where DREAMPlace uses NCTUgr [24] but RoutePlacer uses RouteGNN to give feedback for cell inflation. Details of baselines and experimental settings are given in Appendix C. We conduct experiments on `ISPD2011` and `DAC2012` benchmarks, using NVIDIA RTX 3080 and Intel(R) Xeon(R) CPU E5-2699 with 31GB memory.

## 4.1 Evaluating Effectiveness

To evaluate the routability of placement solutions, we use NCTUgr to generate routing results and measure wirelength. We divide the entire layout into grids and assign each grid a wire limit, 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 $(i, j)$. Wirelength refers to the total length of all wires.

Furthermore, the wire limit is categorized into two dimensions: horizontal ($RC_h$) and vertical ($RC_v$). The number of horizontal and vertical wires in each grid cell must not surpass their respective limits. $OF_h(i, j)$ and $OF_v(i, j)$ represent the excess in the number of horizontal and vertical wires, respectively.

We compare DREAMPlace with RoutePlacer based on five metrics: total overflow (TOF), wirelength (WL), maximum overflow (MOF), horizontal congestion ratio (H-CR), and vertical congestion ratio (V-CR). They are defined as follows:

$$TOF = \sum_{i,j} OF(i, j), \qquad MOF = \max_{i,j} OF(i, j), \tag{19}$$

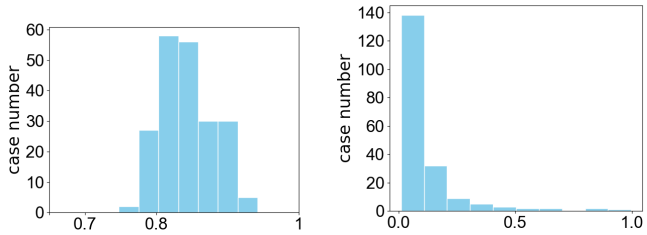
$$H-CR = \frac{\max_{i,j} OF_h(i, j)}{RC_h}, \qquad V-CR = \frac{\max_{i,j} OF_v(i, j)}{RC_v}. \tag{20}$$

The results on `ISPD2011` are shown in Table 1. We defer the results on `DAC2012` to Appendix D. Compared with DREAMPlace, RoutePlacer shows a 16% reduction in total overflow, 2.5% reduction in max overflow, 1% rise in H-CR, 6% reduction in V-CR, and maintains routed wirelength (averaged over benchmarks). The results indicate that RoutePlacer can optimize routability across various circuits.

## 4.2 Evaluating Efficiency

To train RouteGNN, we collect placements generated by DREAMPlace and obtain labels by NCTUgr. We employ MSE loss for training, where the labels are logarithmized to prevent an excessive output range. Further details can be found in Appendix C and B.5.

To evaluate the efficiency of our model, we measure the runtime of **RouteGNN** and **NetlistGNN**. The latter connects geometric neighbors to model geometrical relationships. Fig. 4 visualizes the runtime comparisons between NetlistGNN and RouteGNN. We observe that as the placement process advances, cell overlap diminishes and the cells become more evenly distributed. The inference time of geometric graph construction in NetlistGNN decreases along this process. In contrast, RouteGNN consistently maintains a short runtime, unaffected by the distribution of cells.




(a) SSIM statistics (b) NRMSE statistics

**Figure 3: Evaluations of RouteGNN. Experiments are conducted on a dataset comprising over 100 placements.**

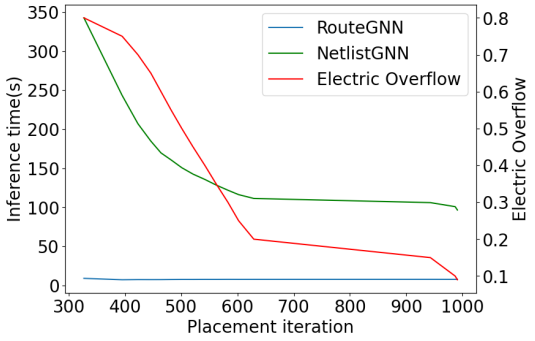



**Figure 4: Comparison of inference time on `superblue7`. We additionally plot the changes of electric overflow during placement [11].**

To evaluate the accuracy of our model, we employ two key metrics: NRMSE and SSIM. NRMSE quantifies the element-wise discrepancy between our prediction and the corresponding label at the cellular level, whereas SSIM, as referenced in [23], measures the structural similarity between our prediction and the ground truth at the grid level. Fig. 3 illustrates the distributions of the two statistics. In over 80% of all cases, RouteGNN gives predictions with high accuracy, with NRMSE values below 0.1 and SSIM values above 0.8.

## 4.3 Evaluating Extensibility

To verify the extensibility of RoutePlacer, we incorporate a cell inflation method (detailed in Appendix B.4). The cell inflation method requires routing results as inputs. We use NCTUgr to generate congestion maps for DREAMPlace, while using RouteGNN to generate routability estimations for RoutePlacer. The original routability estimations of RouteGNN are on the cell level. We average the routability estimations of all cells in each grid to formulate grid maps required by the cell inflation method. The transformation is formulated as follows:

$$\hat{y}^{grid}(i, j) = \frac{\sum_{v \in N_{i,j}} \hat{y}_v^{cell}}{||N_{i,j}||}, \tag{21}$$

where $N_{i,j}$ denotes the set of cells locate in grid $i, j$, and $||N_{i,j}||$ denotes the number of cells in the set.

**Table 1: Comparsion results on ISPD2011.**

| Netlist | #cell | #nets | TOF↓ | | MOF↓ | | H-CR↓ | | V-CR↓ | | WL($\times 10^6 um$)↓ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace |
| superblue1 | 848K | 823K | **72380** | 74694 | **16** | 20 | 0.22 | 0.22 | **0.19** | 0.20 | **12.90** | 12.91 |
| superblue2 | 1014K | 991K | **709172** | 895270 | 46 | **40** | 0.54 | **0.51** | **0.26** | 0.31 | **25.19** | 25.29 |
| superblue4 | 600K | 568K | **114232** | 119732 | **36** | 44 | **0.45** | 0.53 | **0.21** | 0.25 | **9.18** | 9.19 |
| superblue5 | 773K | 787K | **348976** | 363260 | **32** | 34 | 0.43 | 0.43 | 0.26 | **0.24** | **15.31** | 15.36 |
| superblue10 | 1129K | 1086K | **142986** | 251602 | 20 | 20 | 0.29 | **0.28** | **0.15** | 0.16 | **24.10** | 24.33 |
| superblue12 | 1293K | 1293K | **2112070** | 2201282 | **104** | 112 | **1.13** | 1.22 | 0.59 | **0.55** | **15.47** | 15.56 |
| superblue15 | 1124K | 1080K | **115962** | 116112 | 16 | 16 | **0.25** | 0.26 | 0.13 | 0.13 | **14.51** | 14.52 |
| superblue18 | 484K | 469K | **100336** | 104162 | 16 | 16 | **0.25** | 0.27 | 0.19 | **0.18** | **8.65** | 8.67 |
| Average ratio | | | 1.00 | 1.15 | 1.00 | 1.06 | 1.00 | 1.04 | 1.00 | 1.04 | 1.00 | 1.01 |

**Table 2: Comparsion results on ISPD2011. RoutePlacer and DREAMPlace additionally implement cell inflation methods.**

| Netlist | #cell | #nets | TOF↓ | | MOF↓ | | H-CR↓ | | V-CR↓ | | WL($\times 10^6 um$)↓ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace |
| superblue1 | 848K | 823K | **4612** | 5340 | 36 | **16** | **0.14** | 0.19 | 0.32 | **0.19** | 13.72 | **12.97** |
| superblue2 | 1014K | 991K | **45338** | 64464 | **12** | 16 | 0.19 | **0.14** | **0.14** | 0.18 | 26.23 | **25.40** |
| superblue4 | 600K | 568K | **6632** | 7242 | 8 | 8 | 0.18 | **0.13** | 0.13 | 0.13 | 9.79 | **9.35** |
| superblue5 | 773K | 787K | **36726** | 108106 | **18** | 26 | **0.23** | 0.35 | 0.19 | 0.19 | 17.09 | **16.47** |
| superblue10 | 1129K | 1086K | **44116** | 45158 | 12 | 12 | **0.20** | 0.21 | 0.13 | 0.13 | 24.91 | **24.66** |
| superblue12 | 1293K | 1293K | **26296542** | 28714180 | **184** | 456 | 1.39 | **1.34** | **1.12** | 1.83 | **39.67** | 41.89 |
| superblue15 | 1124K | 1080K | **15904** | 41430 | 8 | 8 | **0.18** | 0.19 | 0.09 | 0.09 | **15.07** | 15.30 |
| superblue18 | 484K | 469K | 91388 | **22412** | 18 | **16** | 0.27 | **0.20** | 0.19 | 0.16 | **8.98** | 9.21 |
| Average ratio | | | 1.00 | 1.45 | 1.00 | 1.20 | 1.00 | 1.02 | 1.00 | 1.04 | 1.00 | 0.99 |

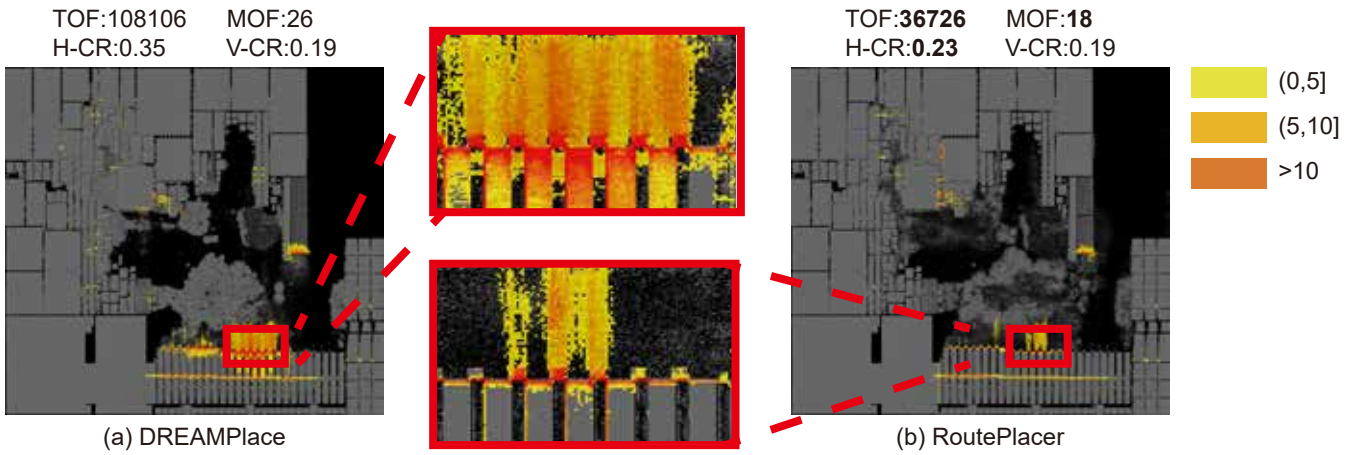



**Figure 5: Visualization of placement solutions and their overflow on superblue5. The legend on the right shows the numerical range of overflow corresponding to each color. (a) A placement generated by DREAMPlace. (b) A placement generated by RoutePlacer. We implement cell inflation for both methods.**

The results on ISPD2011 and DAC2012 are shown in Table 2 and Appendix D, respectively. We report the same five metrics described above. It is demonstrated that RoutePlacer shows a 44% reduction in total overflow, a 1% rise in H-CR, an 8% reduction in V-CR, and only a 1.5% rise in routed wirelength (averaged over the benchmark). It strongly indicates that RoutePlacer can be extended to traditional two-stage pipelines. Incorporating RouteGNN into two-stage methods can yield improved routability-aware placers.

We visualize the generated placements and their overflow in Fig. 5. The cell is denoted by the grey area, and overflow visualization employs a heatmap-like method, where the intensity of the red color corresponds directly to the overflow level. The more intense the red, the higher the overflow. In Fig. 6, we present the detailed distributions of overflow. Specifically, we conduct histogram analyses of overflow $OF(i, j)$, using overflow values as the x-axis and the count of matrix elements for each value as the y-axis. These results demonstrate that RoutePlacer achieves a placement with reduced overflow compared to DREAMPlace, indicating enhanced routability.

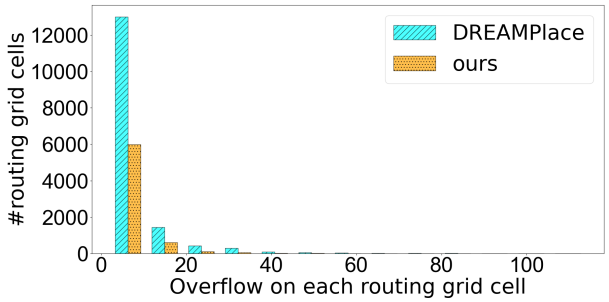



**Figure 6: Overflow distributions of DREAMPlace and RoutePlacer on superblue5. Both of them implement cell inflation.**

## 5 CONCLUSION

This work introduces RoutePlacer, an end-to-end routability-aware placer. It enables analytical routability optimization by training RouteGNN to be a differentiable approximation of routability. Experimental results demonstrate the state-of-the-art performance of RoutePlacer in reducing overflow. RoutePlacer lays a broad foundation for future works. We plan to improve the accuracy of RouteGNN by capturing the shifts in cell positions. It is also promising to include differentiable approximations of other critical metrics, such as timing and power.

## 6 ACKNOWLEDGMENTS

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

## REFERENCES

[1] Breuer Melvin A. 1977. A class of min-cut placement algorithms. In *Proceedings of the 14th Design Automation Conference (DAC '77)*. IEEE, Dakar, 284–290. https://doi.org/10.1145/62882.62896

[2] 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. https://doi.org/10.1109/TCAD.2020.3047724

[3] Ghose Amur, Zhang Vincent, Zhang Yingxue, Li Dong, Liu Wulong, and Coates Mark. 2021. Generalizable Cross-Graph Embedding for GNN-based Congestion Prediction. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD '21)*. IEEE, Munich, Germany, 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643446

[4] Mirhoseini Azalia and Anna Goldie. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (Jun. 2021), 207–212. https://doi.org/10.1038/s41586-021-03544-w

[5] Federico Berto, Chuanbo Hua, Junyoung Park, Minsu Kim, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Joungho Kim, and Jinkyoo Park. 2023. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. arXiv:2306.17100

[6] 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. https://doi.org/10.1145/3489517.3530675

[7] Huang Chau-Chin, Lee Hsin-Ying, Lin Bo-Qiao, Yang Sheng-Wei, Chang Chin-Hao, Chen Szu-To, Chang Yao-Wen, Chen Tung-Chieh, and Bustany Ismail. 2018. NTUplace4dr: A Detailed-Routing-Driven Placer for Mixed-Size Circuit Designs With Technology and Region Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 3 (Mar. 2018), 669–681. https://doi.org/10.1109/TCAD.2017.2712665

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

[9] Cheng Chung-Kuan, Ho Chia-Tung, and Holtz Chester. 2022. Net Separation-Oriented Printed Circuit Board Placement via Margin Maximization. In *2022 27th Asia and South Pacific Design Automation Conference (ASPDAC '22)*. IEEE, Taipei, Taiwan, 288–293. https://doi.org/10.1109/ASP-DAC52403.2022.9712480

[10] Roy Jarrod, Viswanathan Natarajan, Nam Gi-Joon, Alpert Charles J., and Markov Igor. 2009. CRISP: Congestion reduction by iterated spreading during placement. In *2009 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '09)*. IEEE, San Jose, CA, USA, 357–362. https://doi.org/10.1145/1687399.1687467

[11] 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

[12] 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.

[13] 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

[14] 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

[15] Hsu Meng-Kai, Chen Yi-Fang, Huang Chau-Chin, Chou Sheng, Lin Tzu-Hen, Chen Tung-Chieh, and Chang Yao-Wen. 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, 12 (Dec. 2014), 1914–1927. https://doi.org/10.1109/TCAD.2014.2360453

[16] Kim Myung-Chul, Lee Dong-Jin, and Markov Igor. 2013. SimPL: an algorithm for placing VLSI circuits. *Commun. ACM* 56, 6 (Jun. 2013), 105–113. https://doi.org/10.1145/2461256.2461279

[17] Kim Myung-Chul, Hu Jin, Lee Dong-Jin, and Markov Igor. 2011. A SimPLR method for routability-driven placement. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD '11)*. IEEE, San Jose, California, USA, 67–73. https://doi.org/10.1109/ICCAD.2011.6105307

[18] Spindler Peter and Johannes Frank. 2007. Fast and Accurate Routing Demand Estimation for Efficient Routability-driven Placement. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*. EDA Consortium, San Jose, CA, USA, 1–6. https://doi.org/10.1109/DATE.2007.364463

[19] 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., online, 16508–16519. https://doi.org/10.48550/arXiv.2111.00234

[20] 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

[21] Yang Shuwen, Yang Zhihao, Li Dong, Zhang Yingxue, Zhang Zhanguang, Song Guojie, and Hao Jianye. 2022. Versatile Multi-stage Graph Neural Network for Circuit Representation. In *Advances in Neural Information Processing Systems (NeurIPS '22)*. Curran Associates, Inc., New Orleans, Louisiana, USA, Article 1477, 11 pages.

[22] "Chen Tung-Chieh, Hsu Tien-Chang, Jiang Zhe-Wei, and Chang Yao-Wen". "2005". "NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs". In *"Proceedings of the 2005 International Symposium on Physical Design" ("ISPD '05")*. "Association for Computing Machinery", "New York, NY, USA", "236–238". https://doi.org/10.1145/1055137.1055188

[23] Zhou Wang, Bovik Alan Conrad, Sheikh Hamid, and Simoncelli Eero. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (Apr. 2004), 600–612. https://doi.org/10.1109/TIP.2003.819861

[24] Liu Wen-Hao, Kao Wei-Chun, Li Yih-Lang, and Chao Kai-Yuan. 2013. NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing. *IEEE Transactions on computer-aided design of integrated circuits and systems* 32, 5 (May 2013), 709–722. https://doi.org/10.1109/TCAD.2012.2235124

[25] Li Wuxi, Dhar Shounak, and Pan David Z. 2016. UTPlaceF: A Routability-Driven FPGA Placer With Physical and Congestion Aware Packing. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '16)*. IEEE, Austin, TX, USA, 869–882. https://doi.org/10.1145/2966986.2980083

[26] Li Wuxi, Lin Yibo, and Pan David Z. 2019. elfPlace: Electrostatics-based Placement for Large-Scale Heterogeneous FPGAs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '19)*. IEEE, Westminster, CO, USA, 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942075

[27] He Xu, Huang Tao, Xiao Linfu, Tian Haitong, Cui Guxin, and Young Evangeline. 2011. Ripple: An effective routability-driven placer by iterative cell movement. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '11)*. IEEE, San Jose, CA, USA, 74–79. https://doi.org/10.1109/ICCAD.2011.6105308

[28] Lai Yao, Mu Yao, and Luo Ping. 2022. MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning. In *Advances in Neural Information Processing Systems (NeurIPS '22)*. Curran Associates, Inc., New Orleans, Louisiana, USA, 24019–24030. https://doi.org/10.48550/arXiv.2211.13382

[29] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. Large Language Models as Hyper-Heuristics for Combinatorial Optimization. arXiv:2402.01145

[30] Liu Yen-Chun, Chen Tung-Chieh, Chang Yao-Wen, and Kuo Sy-Yen. 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 (ASPDAC '19)*. Association for Computing Machinery, New York, NY, USA, 557–562. https://doi.org/10.1145/3287624.3287677

[31] Lin Yibo, Dhar Shounak, Li Wuxi, Ren Haoxing, Khailany Brucek, and Pan David Z. 2019. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *2019 56th ACM/IEEE Design Automation Conference (DAC '19)*. IEEE, Las Vegas, NV, USA, 1–6. https://doi.org/10.1145/3316781.3317803

[32] Lin Yibo, Pan David Z., Ren Haoxing, and Khailany Brucek. 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 '20)*. IEEE, Shanghai, China, 1–4. https://doi.org/10.1109/CSTIC49141.2020.9282573

# A NOTATION

| Notation | Description |
|---|---|
| $\boldsymbol{x}$,$\boldsymbol{y}$ | Physical locations of cells in the layout |
| $\mathcal{G}$ | RouteGraph |
| $\mathcal{V}$ | Set of cells |
| $\mathcal{U}$ | Set of nets |
| $\mathcal{C}$ | Set of grid cells |
| *topo-edge* | Set of topo-edges which denotes the topology between cells and nets |
| *geom-edge* | Set of geom-edges which signify the adjacent relationships between these grid cells. |
| *grid-edge* | Set of grid-edges which are denoted as edges between cells and the corresponding grid cells within the grid graph |
| $D(\cdot,\cdot)$ | Density Penalty |
| $L(\cdot,\cdot)$ | Routing Congestion Penalty |
| $\lambda_D$ | Weight of density penalty |
| $\gamma$ | Parameter of wirelength model for smoothness |
| $\eta$ | Weight of congestion penalty |

# B DETAILS OF OUR METHOD

## B.1 Loss

Our loss function for transductive placement optimization can be expressed as:

$$Loss = WL(x,y) + \lambda D(x,y) + \eta L(x,y), \tag{22}$$

where $WL(x,y)$ denotes wirelength function, $D(x,y)$ represents density function and $L(x,y)$ is routing congestion as illustrated in Section 3.

## B.2 Wirelength Objective

We follow [11] to implement the wirelength function:

$$WL(x,y) = \left(\frac{\sum_{v\in u} x_v \exp\left(\frac{x_v}{\gamma}\right)}{\sum_{v\in u}\exp\left(\frac{x_v}{\gamma}\right)} - \frac{\sum_{v\in u} x_v \exp\left(-\frac{x_v}{\gamma}\right)}{\sum_{v\in u}\exp\left(-\frac{x_v}{\gamma}\right)} + \right. \tag{23}$$

$$\left. \frac{\sum_{v\in u} x_v \exp\left(\frac{y_v}{\gamma}\right)}{\sum_{v\in u}\exp\left(\frac{y_v}{\gamma}\right)} - \frac{\sum_{v\in u} x_v \exp\left(-\frac{y_v}{\gamma}\right)}{\sum_{v\in u}\exp\left(-\frac{y_v}{\gamma}\right)}\right), \tag{24}$$

where $x_v$ and $y_v$ denote the coordinates of cell $v$, and $\gamma$ is a hyperparameter of the wirelength model for smoothness.

## B.3 Density Objective

The unique solution of the electrostatic system (mentioned in density objective) is derived from [11]:

$$D(x,y) = \sum_{i\in v_{(x,y)}} N_i(x,y) = \sum_{i\in v_{(x,y)}} q_i\psi_i(x,y), \tag{25}$$

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

The parameters $\lambda_D$ are updated after backward propagation and cell position updates, according to the rules given below where hyperparameters follow DREAMPlace default settings.

$$\begin{aligned} \lambda_D &= \lambda_D * \mu, \\ \mu &= \begin{cases} 1.05 * \max\left(0.999^{epochs}, 0.98\right) & \Delta HPWL < 0 \\ 1.05 * 1.05^{\frac{-\Delta HPWL}{350000}} & \Delta HPWL \geqslant 0 \end{cases}, \\ \Delta HPWL &= HPWL_{new} - HPWL_{old}. \end{aligned}$$

## B.4 Cell Inflation

Given a congestion map *congestion* generated by the router, a cell inflation method expands cell size according to the rules given below when electric overflow is less than 0.2. First, for each cell, we compute the maximum congestion among grids that overlap with the cell. Then, based on the maximum congestion, we scale the cell size proportionally. The whole process is formulated as:

$$congestion_{exp} = (congestion)^{exponent}, \tag{27}$$

$$increment = \max_{(\boldsymbol{x},\boldsymbol{y})\in N_v} congestion_{exp}(\boldsymbol{x},\boldsymbol{y}), \tag{28}$$

$$ratio = \sqrt{increment}, \tag{29}$$

$$size_v^{width} = size_v^{width} * ratio, \tag{30}$$

$$size_v^{height} = size_v^{height} * ratio, \tag{31}$$

where *exponent* is the hyperparameters, $N_v$ denotes the set of grids that overlap with the cell $v$, $size_v^{width}$ and $size_v^{height}$ denote the height and width of the cell $v$.

## B.5 Training Placement Collection

To ensure diversity in the training dataset of placement, we use electric overflow [11] as a key metric and collect placement at various electric overflow levels. This strategy allows RouteGNN to learn a richer representation of routability. Electric overflow ranging from 0 to 1 reflects the overlap level between cells. We begin collecting placements when the electric overflow first drops to 0.8; subsequently, each time the electric overflow decreases by 0.05, we collect a placement again.

**Table 3: Comparsion results on DAC2012**

| Netlist | #cell | #nets | TOF↓ | | MOF↓ | | H-CR↓ | | V-CR↓ | | WL($\times 10^6 um$)↓ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace |
| superblue2 | 1014K | 991K | 1153398 | **1152128** | 56 | **54** | 0.56 | **0.52** | **0.27** | 0.32 | 22.83 | **22.82** |
| superblue3 | 920K | 898K | **236218** | 240382 | 64 | **52** | 0.64 | **0.52** | **0.23** | 0.27 | 14.29 | **14.25** |
| superblue6 | 1014K | 1007K | **132370** | 205802 | 52 | **48** | 0.51 | **0.50** | **0.22** | 0.24 | **14.33** | 14.37 |
| superblue7 | 1365K | 1340K | **21198** | 22050 | **16** | 20 | 0.20 | **0.19** | **0.15** | 0.18 | 19.21 | **19.19** |
| superblue11 | 955K | 936K | **46704** | 79720 | 16 | 16 | 0.22 | 0.22 | **0.12** | 0.13 | **14.36** | 14.42 |
| superblue12 | 1293K | 1293K | 2112070 | **1478142** | 104 | **90** | 0.80 | **0.71** | 0.45 | **0.41** | 15.47 | **15.02** |
| superblue14 | 635K | 620K | **20482** | 23490 | 24 | **20** | **0.30** | 0.25 | 0.13 | **0.12** | 9.89 | **9.88** |
| superblue16 | 699K | 697K | **23776** | 24110 | 24 | 24 | **0.26** | 0.28 | **0.18** | 0.19 | 10.47 | **10.46** |
| superblue19 | 523K | 512K | **61098** | 82312 | **32** | 40 | 0.34 | 0.40 | **0.18** | 0.22 | **6.72** | 6.81 |
| Average ratio | | | 1.00 | 1.17 | 1.00 | 0.99 | 1.00 | 0.96 | 1.00 | 1.09 | 1.00 | 1.00 |

**Table 4: Comparsion results on DAC2012. RoutePlacer and DREAMPlace additionally implement cell inflation methods.**

| Netlist | #cell | #nets | TOF↓ | | MOF↓ | | H-CR↓ | | V-CR↓ | | WL($\times 10^6 um$)↓ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace | RoutePlacer | DREAMPlace |
| superblue2 | 1014K | 991K | 55690 | **43400** | **14** | 16 | 0.17 | **0.14** | **0.14** | 0.15 | 22.97 | **22.78** |
| superblue3 | 920K | 898K | **7002** | 11064 | **8** | 14 | 0.13 | 0.13 | **0.11** | 0.14 | 15.45 | **14.65** |
| superblue6 | 1014K | 1007K | **4880** | 5104 | 8 | 8 | **0.13** | 0.14 | **0.11** | 0.12 | 14.80 | **14.72** |
| superblue7 | 1365K | 1340K | 17106 | **12518** | **12** | 16 | 0.17 | **0.15** | **0.13** | 0.15 | 19.65 | **19.19** |
| superblue11 | 955K | 936K | **10140** | 19348 | **6** | 8 | 0.12 | 0.12 | **0.08** | 0.11 | 17.13 | **14.51** |
| superblue12 | 1293K | 1293K | **3722464** | 4184918 | **114** | 116 | **0.87** | 0.88 | 0.52 | 0.52 | **20.39** | 21.73 |
| superblue14 | 635K | 620K | 21702 | **8724** | 22 | **8** | 0.26 | **0.13** | 0.13 | **0.12** | **9.89** | **9.92** |
| superblue16 | 699K | 697K | **14048** | 16810 | **14** | 18 | **0.18** | 0.22 | **0.13** | 0.17 | 10.59 | **10.45** |
| superblue19 | 523K | 512K | **5278** | 22104 | 14 | 14 | **0.12** | 0.16 | 0.13 | 0.13 | 6.92 | **6.82** |
| Average ratio | | | 1.00 | 1.43 | 1.00 | 1.15 | 1.00 | 0.95 | 1.00 | 1.13 | 1.00 | 0.98 |

**Table 5: Hyperparameters of our method on DAC2012**

| Netlist | Num adjust | $\eta$ | *exponent* |
|---|---|---|---|
| superblue2 | 5 | 3.00e-7 | 3.5 |
| superblue3 | 6 | 1.00e-1 | 1.5 |
| superblue6 | 6 | 1.00e+1 | 2.5 |
| superblue7 | 5 | 3.00e-2 | 2.5 |
| superblue11 | 4 | 1.00e+0 | 2.5 |
| superblue12 | 4 | 1.00e+1 | 3.5 |
| superblue14 | 4 | 3.00e-2 | 2.5 |
| superblue16 | 5 | 1.00e-3 | 2.5 |
| superblue19 | 4 | 1.00e+1 | 1.5 |

**Table 6: Hyperparameters of our method on ISPD2011**

| Netlist | Num adjust | $\eta$ | *exponent* |
|---|---|---|---|
| superblue1 | 3 | 3.00e-2 | 2.5 |
| superblue2 | 5 | 3.00e-3 | 2.5 |
| superblue4 | 3 | 1.00e+1 | 3.5 |
| superblue5 | 5 | 1.00e+0 | 2.5 |
| superblue10 | 4 | 1.00e+1 | 2.5 |
| superblue12 | 4 | 1.00e+1 | 2.5 |
| superblue15 | 5 | 1.00e+1 | 2.5 |
| superblue18 | 5 | 3.00e-3 | 1.5 |

## C BASELINE SETTINGS

For RouteGNN, we set hidden layer dimensions of *cell*, *net*, *grid cell*, $topo-edges$, $geom-edges$, $grid-edges$ ($F_{\mathcal{V}}$, $F_{\mathcal{U}}$,$F_C$,$F_{\mathcal{E}_{topo}}$,$F_{\mathcal{E}_{geom}}$, ,$F_{\mathcal{E}_{grid}}$) = (32, 64, 16, 8, 4, 4) and message-passing layers $L = 2$. We set the grid cell size equal to the routing grid cell size on each circuit. During training, we use Adam optimizer with learning rate $\gamma = 0.0002$, learning rate decay $\Delta\gamma = 0.02$, and weight decay $\eta = 0.0002$; we set training epoch $e = 100$ and use MSE loss for training. To avoid excessive output range, we logarithmize both label and model output. For NetlistGNN, we use its default model settings.

Both DREAMPlace and RoutePlacer use `NAG Optimizer` [11] to ensure a fair comparison. Our training parameters are given in Table 5 and 6. "Num adjust" denotes the maximum number of adjustments for cell inflation. $\eta$ and *exponent* refer to the congestion penalty and cell inflation hyperparameter, respectively, as detailed in Section 3.2.1 and Appendix B.4. The adjustment strategy for $\gamma$ and $\lambda_D$, adopted by both DREAMPlace and our method, follows Lu et al. [31]. For DREAMPlace and our method, the applied hyperparameters adhere to the default setting, if not mentioned in the table. Our source code is available at https://github.com/sorarain/RoutePlacer.

## D EXTENDED RESULTS ON DAC2012

We evaluate the effectiveness and extensibility of RoutePlacer on DAC2012, with results presented in Table 3 and Table 4.

## E OTHER MATERIALS

Due to space limitations, the other parts of the Appendix are in the full version of this paper[1].

[1]https://arxiv.org/abs/2406.02651