MacroRank: Ranking Macro Placement Solutions Leveraging Translation Equivariancy

Yifan Chen[1], Jing Mai[1], Xiaohan Gao[1], Muhan Zhang[1], Yibo Lin[1,2*]
[1]Peking University, Beijing, China
[2]Beijing Advanced Innovation Center for Integrated Circuits
{chenyifan2019,magic3007,xiaohangao,muhan,yibolin}@pku.edu.cn

ABSTRACT

Modern large-scale designs make extensive use of heterogeneous macros, which can significantly affect routability. Predicting the final routing quality in the early macro placement stage can filter out poor solutions and speed up design closure. By observing that routing is correlated with the relative positions between instances, we propose MacroRank, a macro placement ranking framework leveraging translation equivariance and a Learning to Rank technique. The framework is able to learn the relative order of macro placement solutions and rank them based on routing quality metrics like wirelength, number of vias, and number of shorts. The experimental results show that compared with the most recent baseline, our framework can improve the Kendall rank correlation coefficient by 49.5% and the average performance of top-30 prediction by 8.1%, 2.3%, and 10.6% on wirelength, vias, and shorts, respectively.

ACM Reference Format:
Yifan Chen, Jing Mai, Xiaohan Gao, Muhan Zhang, Yibo Lin. 2023. MacroRank: Ranking Macro Placement Solutions Leveraging Translation Equivariancy . In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA. https://doi.org/10.1145/3566097.3567899

1 INTRODUCTION

Macro locations have a high impact on routing performance. Integrated circuits (IC) nowadays can consist of millions of standard cells and hundreds of macros. Macros are pre-designed heterogeneous blocks from memory modules or thirdparty IPs. As a macro can be tens or hundreds of times larger than standard cells with hundreds of pins for interconnection, its location can significantly impact the placement quality, especially on routability. A typical design flow, as shown in Figure 1, needs to iterate between macro placement, standard cell placement, and routing for routability optimization, slowing down the design iterations [1–7]. To speed up design closure, early prediction of routing performance at the macro placement stage is always in high demand.

Prior research for routability prediction is mainly performed at the cell placement stage with the known locations of both macros and cells. Most of these works adopt variations of convolutional neural networks (CNNs) to capture the correlation between the geometric distribution of the placement and routability metrics like wirelength, routing demands, and design rule violations [8–10].

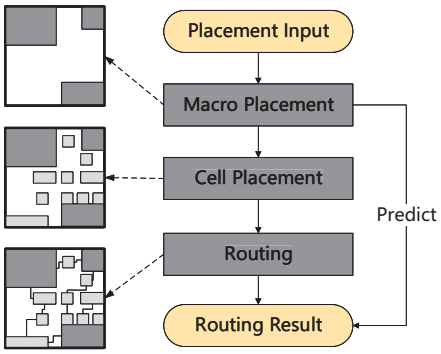


Figure 1: The design flow from placement to routing. The model aims to predict final routing performance in the early macro placement stage.

Routing performance prediction at the macro placement stage is much more difficult than that at cell placement, as only circuit netlists and macro locations are known. Previous attempts [11] unveil that only capturing the geometric distribution of macro placement to construct a regression model with CNN is not enough to learn the correlation to routing performance. To extract the interconnected information from circuit netlists, Mirhoseini et al [12] and Wang et al [13] integrate graph neural networks (GNNs) into reinforcement learning (RL) frameworks to optimize the locations of macros. As their models do not consider the domain-specific characteristics of macro placement, such as pin offset and sparsity of location information in the netlist, their frameworks need to invoke cell placement and routing hundreds of times to find solutions with acceptable quality, taking hours or even days.

To achieve accurate routing performance prediction at the macro placement stage, in this work, we propose MacroRank, a macro placement ranking framework, leveraging a translation equivariant hypergraph neural network (EHNN) and a Learning to Rank (LTR) technique. The EHNN can capture both interconnect and geometric information of macro placement, and the LTR technique takes into account the relative relationship of solution quality. Our model considers the domain-specific properties of macro placement such as pin offset and sparsity of location information in the netlist. The major contributions are summarized as follows.

- We propose MacroRank leveraging translation equivariance and a Learning to Rank technique that can rank macro placement solutions based on their routing quality.
- We propose a translation equivariant neural network, EHNN, which can well adapt to the task of extracting netlist and macro location information at the macro placement stage.
- We propose a Learning to Rank technique to learn the relative order of macro placement solutions, so as to accurately predict the top-30 solutions of widely-used routing metrics

*Corresponding author

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567899

such as wirelength, number of vias, and number of shorts [14].
- The experimental results on ISPD2015 benchmark show that compared with the most recent CNN-based model [11], our framework can improve the Kendall rank correlation coefficient by 49.5% and the average performance of top-30 prediction by 8.1%, 2.3%, and 10.6% on wirelength, vias, and shorts, respectively.

The rest of the paper is organized as follows, Section 2 introduces the basic background and problem formulation; Section 3 explains the details of the proposed algorithm; Section 4 validates the algorithm with experimental results; Section 5 concludes the paper.

2 PRELIMINARIES

In this section, we review the background and the motivation.

2.1 Macro Placement and Routability Prediction

Macros differ from standard cells in that macros serve as large routing obstacles and request more reserved routing space in the periphery. Thus macro placement essentially affects the routability. In practice, macros are usually pre-placed manually and then the netlist with fixed macro positions is fed into standard cell placement. The target of macro placement is that, given a series of standard cells and *movable* macros, optimize the position for macros to achieve better routing results.

Thus, early estimation of routability has attracted much attention in the research community. The routability prediction aims to guide macro placement by predicting the routing performance of the design based on information at the macro placement stage. Precisely, the task is to find a model that is capable of estimating the routing performance for large-scale VLSI designs. We adopt three routing metrics from the ICCAD 2019 routing contest in this work [14], i.e., total wirelength of nets (*wirelength* for short), number of total vias (*vias* for short), and number of nets causing design rule violations (*shorts* for short).

2.2 Problem Formulation

In this work, we aim at ranking macro placement solutions based on their routing metrics, instead of caring about the absolute values of the metrics. Thus, the problem is defined to learn a ranking function $f(\cdot)$ which takes a netlist N and a macro placement solution x as inputs and generates a score s as the output such that

$$f(N, x_i) > f(N, x_j) \iff y_i > y_j. \tag{1}$$

Where x_i, x_j are two different solutions, and y_i, y_j are the real routing performance metric like wirelength. In other words, our task is to

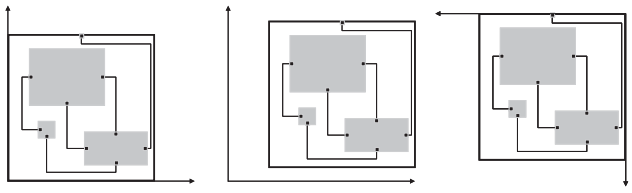
$$\max \sum_{i,j} \text{sign}\left(f(N, x_i) - f(N, x_j)\right) \text{sign}(y_i - y_j). \tag{2}$$

The accuracy can be evaluated by Kendall's Rank Correlation Coefficient defined as follows.

Definition 1 (Kendall's Rank Correlation Coefficient (Kendall's τ))**.** Let x and y be the model predicted values and the ground truth labels. (i, j) is a concordant pair iff $x_i > x_j \wedge y_i > y_j$ or $x_i < x_j \wedge y_i < y_j$; otherwise it is discordant. Assume there are n predicted values and n ground truth labels, let n_p be the number of concordant pairs and n_i be the number of discordant pairs. Then Kendall's τ is defined as

$$\tau = \frac{n_p - n_i}{\frac{1}{2}n(n-1)} \tag{3}$$

The higher the τ is, the better the prediction performance.



(a) Origin (b) Translation (c) Rotation

Figure 2: The rigid body transformation (or the selection of coordinate systems) of the whole layout will not affect the optimal solutions of placement and routing.

2.3 Equivariance

Applying GNN on circuits differs from conventional graph learning tasks in that placing on the 2D layout has strong spatial characteristics that the final routing results significantly rely on the relative spatial positions between standard cells and macros. As illustrated in Figure 2, supposing the routing resource is homogeneous on the layout, a *rigid transformation* like translation and rotation on the whole layout (including the instances, wires, and boundaries) will not affect the optimal solutions of placement and routing, which can be characterized by *E(2)-equivariance*.

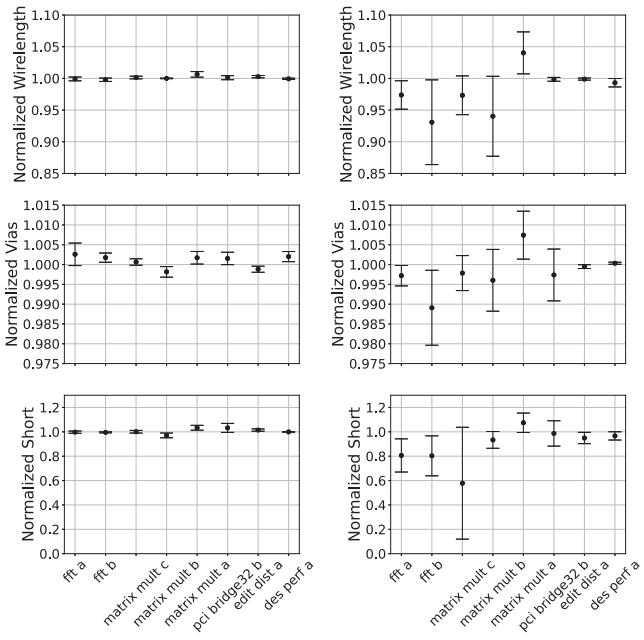
However, in practice, placement and routing algorithms can only find suboptimal solutions due to the NP-hardness of the problems, so the equivariancy may be lost. We investigate the translation and rotation equivariancy on open-source placement and routing tools (DREAMPlace [15] and CU.GR [16]) by translating the layout by $(0.5, 0)\times$, $(0, 0.5)\times$, $(0.5, 0.5)\times$, $(1, 1)\times$ die size, rotating by 180°, and flipping along the x-axis or the y-axis. We observe that translation equivariancy can be well maintained, while rotation equivariancy may not. Figure 3(a) indicates that the standard deviations of wirelength, vias, and shorts caused by different translations are less than 0.5 %, 0.3%, and 3.7%, respectively, but rotation and flipping have a very significant impact on the results, as shown in Figure 3(b).

Note that we do not really translate or rotate the layouts; neither require the placement and routing algorithms to maintain rigid equivariancy. The target is to improve the layout representations learned by the GNN model by constraining the training under different coordinate systems for better generality. Considering the above observation, we develop the *EHNN* to better capture the transformation invariant knowledge and generalize our model to translation equivariant inputs. It can be easily modified to support rotation/flipping equivariance if the placement and routing algorithms maintain the rotation equivariancy as well.

2.4 Learning to Rank

In the macro placement stage, we want to pick up the macro placement solution with the best routing performance among possible candidates for each design. To select the best one, the relative relationship between them is noteworthy instead of the absolute value of each candidate. Therefore, a ranking model, rather than a regression model, is needed for this problem. Learning to Rank is an application of machine learning aiming to construct a ranking model to establish a binary relationship in a list of items. It was originally proposed in information retrieval and used to find the candidate closest to the query. In our task, each design is a query and macro placement solutions are the candidates. A well-known LTR method is called the pairwise method, which approximates the ranking problem to a binary classification problem aiming to distinguish which candidate is better in the chosen pair.

The pairwise method is often implemented with a scoring function $f(\cdot)$ which takes a single candidate X_i as input and outputs



(a) Translation (b) Rotation and flipping

Figure 3: The final routing results of ISPD 2015 benchmarks after applying different rigid transformations. Average values and standard deviations of each metric are normalized to that of the original layouts without any transformation.

the predicted score $s_i = f(X_i)$. For example, a classical pairwise method RankNet [17] adopts a probability model and defines the estimated probability of the candidate X_1 having higher quality than X_2 as

$$\text{Prob}(X_1 \succ X_2) = \frac{1}{1 + \exp\{f(X_2) - f(X_1)\}}. \tag{4}$$

Then many loss functions designed for classification tasks like cross-entropy can be used.

3 ALGORITHM

In this section, we will further detail our algorithm. There are three major parts: the data preparation in Section 3.2-Section 3.3, the proposed EHNN architecture in Section 3.4-Section 3.5 and the pairwise LTR loss in Section 3.6.

3.1 Overview

In data preparation, we first cluster the cells in the netlist, then convert it to a normal graph and extract initial node features. Our EHNN architecture, described in Figure 4, consists of three bottlenecks: (1) Hypergraph Convolutional Layers (HGCLs), (2) Equivariant Graph Convolutional Layers (EGCLs), and (3) Multi-layer Perceptron (MLP). The input to the network is the transformed netlist as the association matrix, the node features, the pin features and the macro position as coordinate embedding. The HGCLs take netlist, node feature, and pin feature as inputs, delivering the transformed node feature and pin feature. After that, the EGCLs take macro features and coordinate embedding as input and generates transformed macro features and coordinate embedding. At last, the macro feature generated by EGCLs is fed into the MLP to predict the target result. To further improve prediction performance, we propose a weighted pairwise LTR loss to train the EHNN. In the following sections, we will dive into the details.

3.2 Netlist Clustering

In most designs, the number of macros is much less than that of cells (see Table 1), which brings many difficulties in extracting useful information. To make it easier for the GNN model to learn macro-related information, and to reduce running time and memory usage, we cluster the cells in the netlist without changing the macros. This can be achieved by setting the weights of the macros to 0 in hMETIS [19] and applying the generated partition to the cells. Assume the shapes of the cells in the same cluster are $(w_1, h_1), ..., (w_n, h_n)$ and the target cell density constraint is γ (given at placement), then the shape of the cluster is set to

$$(w_c, h_c) = \left(\frac{\sum_{i=1}^n w_i}{\sum_{i=1}^n h_i} \sqrt{\frac{\sum_{i=1}^n w_i h_i}{\gamma}}, \frac{\sum_{i=1}^n h_i}{\sum_{i=1}^n w_i} \sqrt{\frac{\sum_{i=1}^n w_i h_i}{\gamma}} \right). \tag{5}$$

Since the cells, even after clustering, are much smaller than the macros, the pin offset of cells is not as important as that of macros. Thus, we simply set the pins located at the centers of clusters.

3.3 Initial Node Features

Before the graph learning process, we determine the initial feature vectors for each node $v_i \in \mathcal{V}$ based on its routability-related properties and the macros' positions on the layout. The instance sizes and the pin offsets are taken to estimate the routing resource requirement, and the reason we take these features is that modern placers usually consider these features in their routing congestion estimators. Thus, the initial feature of instances is composed of instance sizes and degrees, and the initial feature of pins is the pin offset. Apart from the routing requirement properties, we take the macro positions as features, and we also equip our network to extract the rigid transformation invariance knowledge from the spatial features (see Section 3.5). Meanwhile, we scale the die size to (1, 1) to normalize different designs.

3.4 Netlist Representation Learning

As mentioned above, the VLSI netlist is originally represented as a directed hypergraph with single-source multi-sink edges (nets). Usually, we ignore the edges' directivity in the placement stage, but the simplified undirected hypergraph is still inapplicable to many existing graph knowledge mining algorithms. Therefore, in recent years, finding appropriate graph models to transform the netlist from a hypergraph into a two-pin-edge graph has attracted much attention in the research community [20].

In our task, the macros are much larger than the cells, pin offset of macros attracts much more attention than that of cells. Therefore, the instances (cells and macros), pins, and nets are all modeled as nodes in the graph. Meanwhile, the relations between instances and pins are modeled as directed edges from macros to pins. Similarly, the relations between pins and nets are also modeled as directed edges from pins to nets. The message passing process between instances is composed of two stages, the forward stage and the backward stage. The forward stage passes messages from instances to pins and then to nets, and the backward stage does vice versa. An example is shown in Figure 5.

It can be noticed that one pin can only be connected to one instance and one net. Thus, when messages pass from instances to pins, we just concatenate instance feature to pin feature. Then a GATConv [21] is adopted to pass messages from pins to nets, delivering the edge feature. Similarly, in the backward stage, we concatenate the edge features to the pin features, use a linear layer to get the new pin features, and finally aggregate the messages of each instance by mean pooling. The whole process is implemented in an HGCL, which is shown in Figure 4.

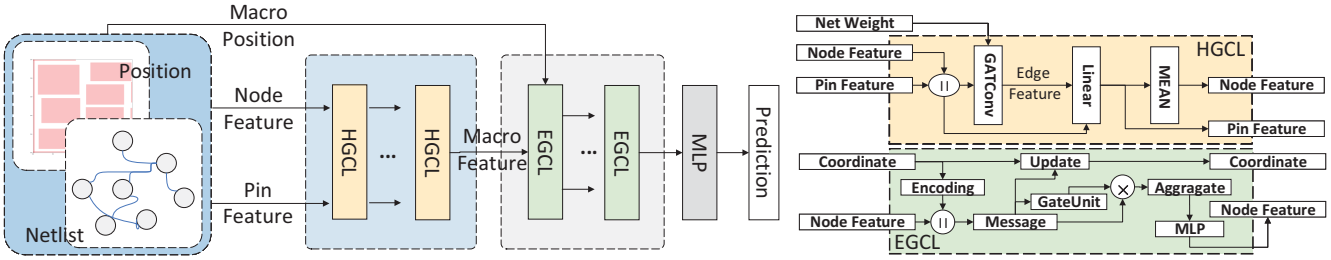


Figure 4: The EHNN architecture. The HGCLs take netlist, instance size, and pin offset as inputs, delivering the node feature. The EGCLs [18] treat the connection between macros as a complete graph and transforms macro features and macros' coordinate embedding. The last MLP takes the transformed macro feature generated by EGCLs as input to predict the target result.

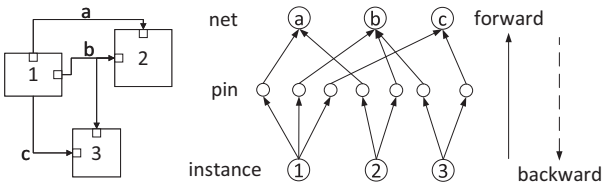


Figure 5: The transformation from the original netlist to the normal graph. The instance, pins, and nets are all treated as nodes in the graph.

3.5 Translation Equivariant Graph Embedding

A CNN-based model [11], taking three density maps of placed instances as input, has been proposed to predict routing design rule violations (DRVs). However, this way of modeling the position loses a lot of netlist information. Two significantly different designs can have the same density map as described above, resulting in completely different routing results.

Therefore, we want to explicitly model the macro position relationships between macros and consider the netlist by hypergraph neural network. But directly treating the location information as input features in the hypergraph neural network is not a good way because of the loss of cell location information and equivariance (as mentioned in Section 2.3). Thus, we use HGCL to capture netlist information and use EGCL [18] to capture location information.

The EGCL [18] has been proved to be E(n)-equivariant. It takes node embedding $\boldsymbol{h}_i$, coordinate embedding $\boldsymbol{x}_i$, and edge information $\mathbf{e}_{ij}$ as inputs and outputs a transformation on $\boldsymbol{h}_i$, $\boldsymbol{x}_i$. The equations that define this layer have the following form,

$$\boldsymbol{m}_{ij} = \phi_e\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_j^{(l)}, \|\boldsymbol{x}_i^{(l)} - \boldsymbol{x}_j^{(l)}\|\right) \tag{6a}$$

$$\boldsymbol{m}_i = \sum_{j \in \boldsymbol{N}(i)} \boldsymbol{m}_{ij} \tag{6b}$$

$$\boldsymbol{h}_i^{(l+1)} = \phi_h\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{m}_i\right) \tag{6c}$$

$$\boldsymbol{x}_i^{(l+1)} = \boldsymbol{x}_i^{(l)} + \sum_{j \neq i} (\mathbf{x}_i^{(l)} - \boldsymbol{x}_j^{(l)}) \phi_x(\hat{\boldsymbol{m}}_{ij}) \tag{6d}$$

where $\mathcal{N}(i)$ denotes the neighbors of node i, and ϕ_e, ϕ_h, ϕ_x are non-linear mappings, and m_{ij} is the message passed from node i to node j.

As mentioned in Section 2.3, there may be only translation equivariance in placement/routing solvers. Thus, we replace the $\|\boldsymbol{x}_i^{(l)} - \boldsymbol{x}_j^{(l)}\|$ term in Eq. (6a) by position encoding $PE(\boldsymbol{x}_i^{(l)}, \boldsymbol{x}_j^{(l)}) = \left(d, (x_i^{(l)} - x_j^{(l)})/d, (y_i^{(l)} - y_j^{(l)})/d\right)$, where d denotes $\|\boldsymbol{x}_i^{(l)} - \boldsymbol{x}_j^{(l)}\|$ and $\boldsymbol{x}_i^{(l)} = (x_i^{(l)}, y_i^{(l)})$. Different from EGCL, we further propose a Fourier encoding inspired by NeRF [22], which is

$$FE(d) = (\sin \pi d, \sin 2\pi d + \pi/2, ..., \sin 2^i \pi d + \frac{\pi(i \mod 2)}{2}). \tag{7}$$

The function can map continuous input into a higher dimensional space to enable our model to more easily approximate a higher frequency function. Then the final position encoding is

$$PE = \left(FE(d), (x_i^{(l)} - x_j^{(l)})/d, (y_i^{(l)} - y_j^{(l)})/d\right). \tag{8}$$

Since all other parts in EGCL keep E(2)-equivariance, and the position coding keeps translation equivariance, EGCL has translation equivariance. Notice that HGCL obviously has translation equivalence, the translation equivariant of the whole network can be guaranteed.

Our input data only contains the location information of macros, which only account for a very small part of the nodes in the entire graph. As a result, if we directly put the whole graph into EGCL, there will be a lot of data missing in the graph, which is difficult to overcome. To consider the relative position relation between any two macros, we regard the relation between all macros as a complete graph. Note that if the number of macros is too large to build a complete graph, we can only connect K nearest macros to each macro. Meanwhile, since the position information of cells is unknown, only macro nodes are included in the input graph. Macro positions and the macro features extracted from the upstream hypergraph neural network are used as the input coordinate embedding and node embedding of the downstream EGCLs.

3.6 Pairwise Rank Loss

We use the EHNN model as the scoring function mentioned in Section 2.4. During the training stage, the dataset is partitioned by design. In each iteration, a pair of different solutions $\langle x_i, x_j \rangle$ of the same design is fed to the model and a pair of scores $\langle s_i = f(x_i), s_j = f(x_j) \rangle$ is obtained. Let $x_i \succ x_j$ denote the true label $\langle y_i, y_j \rangle$ of the pair $\langle x_i, y_j \rangle$ satisfies $y_i > y_j$. The model predicted probability of $x_i \succ x_j$ is defined in RankNet [17] as

$$P(x_i \succ x_j) = \text{Sigmoid}(s_i - s_j) = \frac{1}{1 + \exp\{-(s_i - s_j)\}}. \tag{9}$$

Then a binary cross-entropy function can be used as the loss function. LambdaRank [23] uses a weighted cross entropy loss,

$$L_{ij} = \log\{1 + \exp\{-(s_i - s_j)\}|\Delta Z_{ij}|, \tag{10}$$

where ΔZ_{ij} is the difference of the desired metric caused by swapping the ranks of the samples i, j. Different from DCG score used in LambdaRank, which is sensitive to the global bias of the labels, the softmax function is adopted as the metric in our work, that is

$$\Delta Z_{ij} = \frac{\exp(y_i)}{\sum_{p=1}^{n} \exp(y_p)} - \frac{\exp(y_j)}{\sum_{p=1}^{n} \exp(y_p)}. \tag{11}$$

Let D be the set of all designs, and $d \in D$ be the collection of solutions of the corresponding design. The final loss function is

$$Loss = \sum_{d \in D} \sum_{i \in d} \sum_{y_i > y_j, j \in d} L_{ij}. \quad (12)$$

4 EXPERIMENTAL RESULTS

In this section, we present the details of the experiments and the analysis of the results.

4.1 Experimental Settings

4.1.1 Dataset. 12 circuits from ISPD 2015 benchmark [24] containing macros are selected to generate the training data and test data. Statistics for these circuits can be found in Table 1. The fixed macros are freed to be movable and placed together with cells by the open-source placer DREAMPlace [15]. After convergence, we perturb the macros in the macro legalization stage to generate about 300 layouts for each circuit with different macro positions. Then the placer is restarted to accomplish cell placement. The placement results are routed by the open-source global router CU.GR [16] to obtain the global routing metrics (wirelength, vias, and shorts) as labels. We use hMETIS [19] to cluster the cells in the netlist to reduce the training and running time. To verify the generalization performance of the model, the circuits in the dataset are divided into 2 groups, each containing six designs. When group 1 is used for training, group 2 is used for testing and vice versa.

Table 1: Dataset statistics.

Group	Design Name	Macros	Macro Coverage	Instances	Nets	Macro Placements
1	des perf a	4	50%	108666	110281	300
	fft a	6	65%	33641	32088	300
	matrix mult a	10	67%	154460	154284	296
	matrix mult c	10	67%	151247	151612	296
	superblue14	336	48%	633661	619697	299
	superblue19	280	60%	521805	511606	298
2	edit dist a	6	29%	129993	131134	300
	fft b	11	69%	33646	32088	300
	matrix mult b	10	67%	151247	151612	294
	pci bridge32 b	8	47%	29283	29417	299
	superblue11 a	1443	59%	954445	935613	284
	superblue16 a	419	48%	698367	680450	299

4.1.2 Configuration. The prediction model is implemented with Pytorch Geometric [25]. The CNN baseline is based on a pre-trained VGG11 [26] presented by Pytorch with the classifier part replaced by an MLP regressor and fine tuned with the method proposed in [11]. The GNN baseline is composed of 3 HGCLs with a hidden dimension of 16, and the coordinates are used as input instance features. The hidden dimension of the EHNN model is set to 16, and the number of HGCL and EGCL layers are 2 and 4 respectively. MacroRank uses the same model with EHNN but is trained with pairwise rank loss while the other 3 models are trained with mean absolute error (MAE) loss. Those models are all trained by Adam optimizer with a learning rate of 10^{-3}. The training process of each model lasts 400 epochs. Note that although MacroRank needs to sample paired data in each iteration, the number of iterations in one epoch is set to be the same as that in other models. We evaluate each of the models after the last training epoch. The training time of CNN, GNN, EHNN, and MacroRank are 1h, 2.2h, 4.5h, and 6.6h respectively. Our code is available in https://github.com/PKU-IDEA/MacroRank.

4.2 Performance and Comparison

We test CNN, GNN, EHNN, and MacroRank on both 2 groups, where MacroRank denotes EHNN trained with pairwise rank loss. The experimental results are listed in Table 2 and Table 3.

Table 2: Result of routing performance prediction with different models. The lower the MRE, the higher Kendall's τ, and the better the prediction.

	Metric	Mean Relative Error (MRE)			Kendall's τ		
Labels	Model\Group	Group 1	Group 2	Average	Group 1	Group 2	Average
Wirelength	CNN [11]	**0.336**	0.232	**0.283**	-0.015	0.234	0.109
	GNN [27]	0.801	0.522	0.662	0.243	0.227	0.235
	EHNN	0.798	**0.175**	0.486	0.224	0.029	0.127
	MacroRank†	-	-	-	**0.417**	**0.344**	**0.381**
Vias	CNN [11]	0.832	0.678	0.755	-0.009	-0.014	-0.011
	GNN [27]	0.425	0.603	0.514	0.269	0.163	0.216
	EHNN	**0.192**	**0.371**	**0.281**	0.205	-0.166	0.0195
	MacroRank†	-	-	-	**0.365**	**0.240**	**0.302**
Shorts	CNN [11]	6.740	2.292	4.516	0.297	0.299	0.298
	GNN [27]	7.612	2.040	4.826	0.278	0.198	0.238
	EHNN	**2.889**	**2.001**	**2.445**	0.211	0.208	0.209
	MacroRank†	-	-	-	**0.297**	**0.305**	**0.301**

† MacroRank = EHNN + LTR

4.2.1 Comparison of Relative Error and Correlation Coefficients. Since MacroRank does not predict the true label directly, we only analyze the differences between the three regression models (CNN, GNN, and EHNN) based on the MRE. First, GNN performs the worst among the three models, which indicates that it is not a good way to simply take the coordinates as input node features in GNN. On the opposite, EHNN dominates GNN in all groups and outperforms CNN [11] in almost all groups except the first group of wirelength prediction. It confirms the effectiveness of leveraging equivariance in our GNN model to model position information. Besides, the contrast between EHNN and CNN [11] also demonstrates that introducing interconnection information can better help the model to make predictions on the quality of macro placement solutions.

Although EHNN achieves lower MRE, it performs quite poorly on Kendall's τ. The reason mainly lies in what we discussed in Section 2.4, using regression loss like MAE or MSE cannot directly reflect the quality of relative order prediction. Figure 6 visualizes the relationship between real wirelength and the predicted one by different models, which also confirms our conclusion. MacroRank achieves the best Kendall's τ on all the groups, more precisely, 49.5% better than CNN [11]. A visualization of the outputs of MacroRank is shown in Figure 6, which shows that the output score of MacroRank has a much better positive correlation with the true label.

4.2.2 Comparison of Top-30 prediction. Finally, we estimate the QoR of the top-30 solutions selected by those models by averaging the real routing performance metrics (wirelength, vias, and shorts) of the top-30 solutions selected by each model on every design from the test set (about 300 samples per design, see Table 1). The average of all the samples in the test set is chosen as the baseline (Mean in Table 3). The result is shown in Table 3. MacroRank outperforms all other models and improves wirelength, vias, and shorts by 8.1%, 2.3%, and 10.6% over the CNN [11], 15.3%, 1.6%, and 53.8% over the GNN [27], 10.0%, 2.3%, and 36.7% over the EHNN. The comparison shows that our model is practical and can effectively guide users to choose good solutions from possible candidates.

5 CONCLUSION

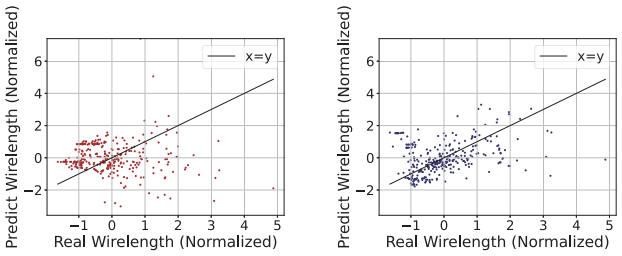
In this paper, we propose MacroRank leveraging translation equivariance and a Learning to Rank technique that can rank macro placement solutions based on their routing quality. The proposed model is able to accurately predict the relative order of the quality of macro placement solutions at the early macro placement stage, to guide our macro placement process. The experimental results on ISPD2015 benchmark show that compared with the most recent CNN-based model [11], our framework can improve the Kendall rank correlation coefficient by 49.5% and the average performance

Table 3: Average performance of the top-30 solutions selected by different models from the test set (about 300 samples per design, see Table 1). The final average is calculated by dividing each item by the corresponding Mean*. The bold ones are the best, and the brown ones are the second best.

Metrics	Wirelength ($2 \times 10^5 \mu m$)					Vias ($\times 10^5$)					Shorts ($\times 10^5$)				
Design\Model	Mean*	CNN [11]	GNN [27]	EHNN	MacroRank†	Mean*	CNN [11]	GNN [27]	EHNN	MacroRank†	Mean*	CNN [11]	GNN [27]	EHNN	MacroRank†
des perf a	17.516	19.553	20.406	**15.142**	17.805	5.095	5.194	5.238	**4.928**	4.997	7.794	6.386	9.261	**3.458**	3.933
fft a	5.845	**5.118**	5.805	6.166	5.526	1.534	1.528	1.566	**1.520**	1.543	2.125	**0.312**	2.265	3.188	3.030
matrix mult a	28.681	30.737	35.765	**23.905**	25.288	6.467	6.548	6.748	**6.285**	6.321	9.102	16.836	16.062	2.105	**1.311**
matrix mult c	27.816	24.595	28.586	26.901	**22.813**	6.284	6.351	6.241	6.263	**6.075**	11.223	**2.626**	13.957	7.905	2.665
superblue14	346.573	354.187	336.416	352.569	**293.825**	39.848	39.503	38.800	40.583	**38.702**	12.002	8.771	9.063	8.937	**7.850**
superblue19	248.976	253.602	226.386	241.326	**209.756**	30.616	30.833	29.773	31.219	**29.813**	5.569	5.192	4.907	**4.657**	5.220
edit dist a	26.814	28.613	26.300	29.314	**25.711**	7.207	7.116	7.053	7.432	**7.024**	6.794	7.587	**5.341**	10.619	7.154
fft b	6.799	5.975	**5.686**	7.529	6.422	1.550	1.564	1.579	**1.523**	1.580	1.944	**0.212**	2.925	2.756	0.675
matrix mult b	28.664	23.561	30.515	26.733	**23.211**	6.327	6.358	6.374	6.417	**6.132**	12.121	**3.803**	6.577	5.749	4.448
pci bridge32 b	6.515	**6.078**	6.300	7.124	6.107	1.500	1.506	1.498	1.512	**1.493**	1.049	**0.813**	0.974	1.335	1.068
superblue11 a	553.656	538.208	515.019	513.904	**444.291**	54.625	54.645	55.284	**54.300**	55.742	11.801	13.973	**9.105**	14.035	10.612
superblue16 a	394.954	**297.142**	429.541	297.175	297.831	41.702	41.486	**37.582**	43.372	37.616	24.695	13.891	18.970	11.507	**8.456**
Average	1.000	0.951	1.015	0.968	**0.880**	1.000	1.003	0.996	1.003	**0.980**	1.000	0.731	1.017	0.904	**0.661**

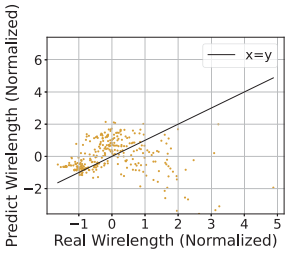
† MacroRank = EHNN + LTR

* Mean denotes the average of all the samples in the test set.

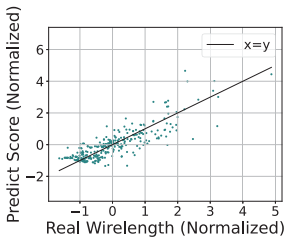


(a) CNN [11]

(b) GNN [27]



(c) EHNN



(d) MacroRank=EHNN+LTR

Figure 6: Visualizations of normalized outputs predicted by different models. All above are tested on mgc_edit_dist_a. Notice that MacroRank predicts the order preserving score rather than directly regressing on the wirelength.

of top-30 prediction by 8.1%, 2.3%, and 10.6% on wirelength, vias, and shorts, respectively.

ACKNOWLEDGE

This project is supported in part by the National Key Research and Development Program of China (No. 2021ZD0114702).

REFERENCES

[1] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE TCAD*, vol. 27, no. 7, pp. 1228–1240, 2008.

[2] J. Lu, H. Zhuang, P. Chen, H. Chang, C.-C. Chang, Y.-C. Wong, L. Sha, D. Huang, Y. Luo, C.-C. Teng *et al.*, "ePlace-MS: Electrostatics-based placement for mixed-size circuits," *IEEE TCAD*, vol. 34, no. 5, pp. 685–698, 2015.

[3] J. Cong and M. Xie, "A robust mixed-size legalization and detailed placement algorithm," *IEEE TCADICS*, vol. 27, no. 8, pp. 1349–1362, 2008.

[4] H.-C. Chen, Y.-L. Chuang, Y.-W. Chang, and Y.-C. Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs," in *ICCAD*. IEEE, 2008, pp. 218–223.

[5] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and T.-Y. Liu, "Mp-trees: A packing-based macro placement algorithm for modern mixed-size designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 9, pp. 1621–1634, 2008.

[6] Y.-F. Chen, C.-C. Huang, C.-H. Chiou, Y.-W. Chang, and C.-J. Wang, "Routability-driven blockage-aware macro placement," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[7] C.-H. Chiou, C.-H. Chang, S.-T. Chen, and Y.-W. Chang, "Circular-contour-based obstacle-aware macro placement," in *ASP-DAC*. IEEE, 2016, pp. 172–177.

[8] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[9] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[10] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu, "Design rule violation prediction at sub-10nm process nodes using customized convolutional networks," *TCADICS*, 2021.

[11] Y.-H. Huang, Z. Xie, G.-Q. Fang, T.-C. Yu, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routability-driven macro placement with embedded cnn-based prediction model," in *DATE*. IEEE, 2019, pp. 180–185.

[12] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[13] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," in *NeurIPS*, 2020.

[14] S. Dolgov, A. Volkov, L. Wang, and B. Xu, "2019 cad contest: Lef/def based global routing," in *ICCAD*, 2019, pp. 1–4.

[15] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *DAC*, 2019, pp. 1–6.

[16] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young, "Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model," in *DAC*, 2020.

[17] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML*, 2005, pp. 89–96.

[18] V. G. Satorras, E. Hoogeboom, and M. Welling, "E(n) equivariant graph neural networks," *CoRR*, vol. abs/2102.09844, 2021.

[19] S. Kim, E. Shragowitz, G. Karypis, and R. B. Lin, "Interleaving of gate sizing and constructive placement for predictable performance," in *VLSI-DAT 2007. International Symposium on*, 2007.

[20] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognition*, vol. 110, p. 107637, 2021.

[21] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" *CoRR*, vol. abs/2105.14491, 2021.

[22] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*. Springer, 2020, pp. 405–421.

[23] C. Burges, R. Ragno, and Q. Le, "Learning to rank with nonsmooth cost functions," *Advances in neural information processing systems*, vol. 19, 2006.

[24] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proc. ISPD*, 2015, pp. 157–164.

[25] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[27] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," *CoRR*, vol. abs/1809.09401, 2018.