Mitigating Distribution Shift for Congestion Optimization in Global Placement

Su Zheng[1,2], Lancheng Zou[1], Siting Liu[1], Yibo Lin[2], Bei Yu[1], Martin Wong[1]
[1]Chinese University of Hong Kong [2]Peking University

***Abstract*—The placement and routing (PnR) flow plays a critical role in physical design. Poor routing congestion is a possible problem causing severe routing detours, which can lead to deteriorated timing performance or even routing failure. Deep-learning-based congestion prediction model is designed to guide the global placement process in previous work. However, the distribution shift problem in this method limits its performance. In this paper, we mitigate the distribution shift problem with a look-ahead mechanism inspired by optical flow prediction and an invariant feature space learning technique. With the proposed method, we can achieve better congestion prediction performance and less-congested placement results.**

I. Introduction

Optimization of routing congestion is critical to placement [1]–[9]. To a large extent, it determines the routability of a placement solution. The widely-adopted idea of congestion optimization is to spread the cells in congested regions. Its key point lies in two perspectives: accurate congestion estimation and effective spreading strategy.

The literature has extensively investigated congestion optimization techniques in the above two perspectives [10]–[16]. Many prior studies integrate global routing into placement iterations and inflate cells according to the congestion map from global routing. Obtaining the congestion map from global routing is accurate but time-consuming.

To avoid the large overhead of invoking global routing, deep-learning-based approaches have been proposed to replace the time-consuming routing engines in congestion prediction. These methods predict the congestion hotspots according to placement features like the rectangular uniform wire density (RUDY) [17]. As the congestion prediction problem can be viewed as an image translation task in computer vision, various models have been investigated, such as fully-convolutional networks (FCNs) [18], generative adversarial networks (GANs) [19], etc. The grid-cells and nets in placement can be modeled with hypergraphs, and graph neural networks (GNNs) can achieve accurate congestion prediction [20]. Neural architecture search (NAS) enables automatic and flexible design of congestion prediction models [21]. These studies utilize the features extracted at the end of placement and build models to predict the congestion after routing. The models can be integrated into placement engines and guide the cell inflation. Recently, Liu et al. [22] further propose to directly integrate the congestion penalty from a deep neural network (DNN) into the objective of an analytical placer and leverage its gradient to guide congestion optimization [22].

Despite the existing efforts, we observe that there exists a distribution shift problem in most of the previous learning-based congestion prediction/optimization approaches. Modern analytical placement follows an iterative procedure to spread cells gradually. The distributions of cells vary significantly at different iterations. Fig. 1(a) and Fig. 1(b) plot the distributions of cells at the first iteration and the end of placement. Due to the wirelength minimization in the placement algorithm, the cells at the first iteration are concentrated in specific regions. At the end of placement, the cells are uniformly distributed because the algorithm puts more and more emphasis on density minimization as the placement progresses. Fig. 1(c) further shows the KL divergence between each iteration and the last iteration, which reveals the distribution shift of cell locations and placement features during the placement process. As a result, models trained on the features collected at the end of placement may not work well on features from intermediate iterations in practice.

To mitigate the distribution shift problem in congestion prediction and optimization, we propose LACO, a **L**ook-**A**head **C**ongestion **O**ptimization method. By predicting the future placement features, LACO mitigates the distribution shift on the inputs of the congestion prediction model. We exploit the novel cell flow to capture the motions of cells and improve the prediction performance. In addition, LACO also learns an invariant latent feature space in the DNN model, which can get less performance deterioration under distribution shift.

The major contributions of this paper are summarized as follows.

- We mitigate the distribution shift problem in deep-learning-based congestion optimization with a look-ahead mechanism that learns cell flow and placement feature prediction. The proposed multi-task learning scheme can provide less-shifted inputs for the congestion prediction model and achieve better performance than the model without the look-ahead mechanism.
- We further enhance the robustness of the proposed method by learning an invariant feature space. This mechanism reduces the distribution shift of the internal DNN feature maps under varying inputs, improving the performance of the model.
- Abundant ablation studies demonstrate the effectiveness of our method in improving congestion prediction and optimization. Our experiments show the importance of mitigating the distribution shift problem for deep-learning-based congestion optimization in global placement.

The rest of our article is organized as follows. Section II introduces the preliminaries. Section III shows the overview of the novel method. Section IV presents various experimental results that can prove the effectiveness of LACO. The conclusion is shown in Section V.

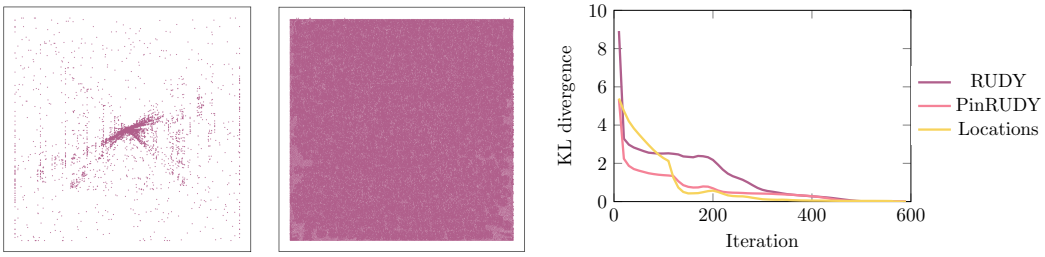
II. Preliminaries

A. Analytical Placement

Most state-of-the-art placement algorithms fall into the category of analytical placement [15], [23], [24], which typically consists of global placement (GP), legalization (LG), and detailed placement (DP). This paper focuses on GP, which is a time-consuming and dominant step. Assigning a 2D coordinate to each circuit cell, GP algorithms usually minimize the estimated wirelengths of the nets. To reduce the overlapping of cells, placement algorithms can utilize density cost functions to spread out the cells. For example, DREAMPlace [24] employs a weighted-average wirelength model $W_e(\boldsymbol{x}, \boldsymbol{y})$ and an electrostatics-based density model $D(\boldsymbol{x}, \boldsymbol{y})$ to optimize the cell locations $(\boldsymbol{x}, \boldsymbol{y})$. The optimization objective of DREAMPlace is defined as,

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

This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14209420).



(a) (b) (c)

Fig. 1 Illustration of the distribution shift problem. (a) The cells at the first iteration of the placement. (b) The cells at the 600-th iteration of the placement. (c) The curve of the KL divergence $KL(p_i \| p_{600})$, where p_i is the distribution of RUDYs, PinRUDYs, or cell locations at the i-th iteration. The data are obtained from DREAMPlace on the `des_perf_1` testcase in ISPD 2015 benchmark.

where E is the set of nets and λ is the density penalty weight. Analogizing the analytical placement problem to DNN training, DREAMPlace achieves a significant speedup with the help of the DNN training utilities.

B. Placement Features for Congestion Prediction

RUDY, PinRUDY, and MacroRegion are placement features that can be utilized to predict congestion hotspots [22]. RUDY estimates routing demand based on the nets. To obtain RUDY, we first get the bounding box of each net,

$$x_e^h = \max_{p_e} x_{p_e},\ x_e^l = \min_{p_e} x_{p_e},\ y_e^h = \max_{p_e} y_{p_e},\ y_e^l = \min_{p_e} y_{p_e}, \quad (2)$$

where p_e denotes a pin in the net e. The location of p_e is represented by (x_{p_e}, y_{p_e}).

In the region $x \in [x_e^l, x_e^h], y \in [y_e^l, y_e^h]$, we simply define the RUDY for net e as,

$$\mathbf{RUDY}_e(\boldsymbol{x}, \boldsymbol{y}) = \left(\frac{1}{x_e^h - x_e^l} + \frac{1}{y_e^h - y_e^l}\right). \quad (3)$$

If $x \notin [x_e^l, x_e^h]$ or $y \notin [y_e^l, y_e^h]$, we have $\mathbf{RUDY}_e(\boldsymbol{x}, \boldsymbol{y}) = 0$. Finally, RUDY can be defined as,

$$\mathbf{RUDY}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{e \in E} \mathbf{RUDY}_e(\boldsymbol{x}, \boldsymbol{y}). \quad (4)$$

PinRUDY is the pin density map inspired by RUDY. To compute the PinRUDY, we need to split the layout into a $N \times M$ grid and estimate the pin density of each grid-cell $b_{k,l}$. The PinRUDY of a pin is calculated with,

$$\mathbf{PinRUDY}_{p_e}(k,l) = \left(\frac{1}{x_e^h - x_e^l} + \frac{1}{y_e^h - y_e^l}\right), (x_{p_e}, y_{p_e}) \in b_{k,l}. \quad (5)$$

Finally, the PinRUDY of the grid-cell $b_{k,l}$ is defined as,

$$\mathbf{PinRUDY}(k,l) = \sum_{p_e \in b_{k,l}} \mathbf{PinRUDY}_{p_e}(k,l). \quad (6)$$

MacroRegion indicates whether a region is covered by a macro cell or not, which is defined as,

$$\mathbf{MacroRegion}(k,l) = \begin{cases} 1, & \text{if } b_{k,l} \text{ is in a macro cell,} \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

In practice, the RUDY map is also divided into $M \times N$ grid-cells. The RUDY of a grid-cell $b_{k,l}$ is calculated by summing up the RUDY values of the cells in it. RUDY, PinRUDY, and MacroRegion form the $3 \times M \times N$ inputs of the congestion prediction model in the deep-learning-based congestion optimization algorithm [22].

C. Deep-learning-based Congestion Optimization in Placement

The most recent strategy [22] for congestion optimization leverages a DNN to predict the congestion hotspots given the RUDY, PinRUDY, and MacroRegion at a placement iteration. It integrates the congestion penalty explicitly into the placement objective, which becomes,

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{e \in E} W_e(\boldsymbol{x}, \boldsymbol{y}) + \lambda D(\boldsymbol{x}, \boldsymbol{y}) + \eta L(\boldsymbol{x}, \boldsymbol{y}), \quad (8)$$

where $L(\boldsymbol{x}, \boldsymbol{y})$ is the routing congestion penalty obtained from the congestion prediction model. The weights λ and η control the trade-off between wirelength, density, and congestion.

III. Proposed Method

A. Overview

Equation (8) presents the objective of the previous deep-learning-based congestion optimization method. The proposed method also follows this optimization objective. Since the details of the wirelength and density models are discussed in existing global placement algorithms, we focus on the congestion penalty $L(\boldsymbol{x}, \boldsymbol{y})$ in this paper. In [22], the congestion penalty is obtained from an FCN, whose inputs are RUDY, PinRUDY, and MacroRegion. Thus, the congestion penalty at the i-th placement iteration $L_i(\boldsymbol{x}, \boldsymbol{y})$ can be defined as,

$$L_i(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{MN} \| f(\boldsymbol{r}_i, \boldsymbol{p}_i, \boldsymbol{m}_i) \|_2^2, \quad (9)$$

where $f(\cdot)$ is congestion hotspots estimated by the FCN-based congestion prediction model. The variables $\boldsymbol{r}_i$, $\boldsymbol{p}_i$, and $\boldsymbol{m}_i$ represent the RUDY, PinRUDY, and MacroRegion features at the i-th iteration, respectively. Note that the features are determined by $(\boldsymbol{x}, \boldsymbol{y})$.

This paper presents a novel look-ahead mechanism, where a DNN predicts the placement features after K iterations, $\overline{\boldsymbol{r}}_{i+K}$, $\overline{\boldsymbol{p}}_{i+K}$, and $\overline{\boldsymbol{m}}_{i+K}$, given previous placement features. To enhance the performance of our look-ahead model, we estimate the novel cell flow $\overline{\boldsymbol{c}}_{i+K}$ to capture the motions of cells, forming a multi-task learning scheme that can improve the generalization ability.

Therefore, we estimate the congestion map according to the predicted placement features and cell flow, $\overline{\boldsymbol{r}}_{i+K}$, $\overline{\boldsymbol{p}}_{i+K}$, $\overline{\boldsymbol{m}}_{i+K}$, and $\overline{\boldsymbol{c}}_{i+K}$. The new congestion penalty becomes,

$$L_i(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{MN} \| f(\overline{\boldsymbol{r}}_{i+K}, \overline{\boldsymbol{p}}_{i+K}, \overline{\boldsymbol{m}}_{i+K}, \overline{\boldsymbol{c}}_{i+K}) \|_2^2, \quad (10)$$

We leverage a video prediction scheme for this task [25], [26]. The placement features at the i-th iteration, denoted by $[\boldsymbol{r}_i, \boldsymbol{p}_i, \boldsymbol{m}_i]$, can be regarded as a frame of video. The novel cell flow $\boldsymbol{c}_i$ can be regarded as the optical flow in the video that captures the motions of points. Since it has been proved that learning with optical flow can significantly improve the performance of video prediction [25], action prediction [27], and so on, we expect to improve placement feature prediction with the novel cell flow.

The information we need at the i-th iteration can be denoted by $\boldsymbol{X}_i = [\boldsymbol{r}_i, \boldsymbol{p}_i, \boldsymbol{m}_i, \boldsymbol{c}_i]$. Assuming we want to predict the data at the $i + K$ iteration, we can get C frames of inputs

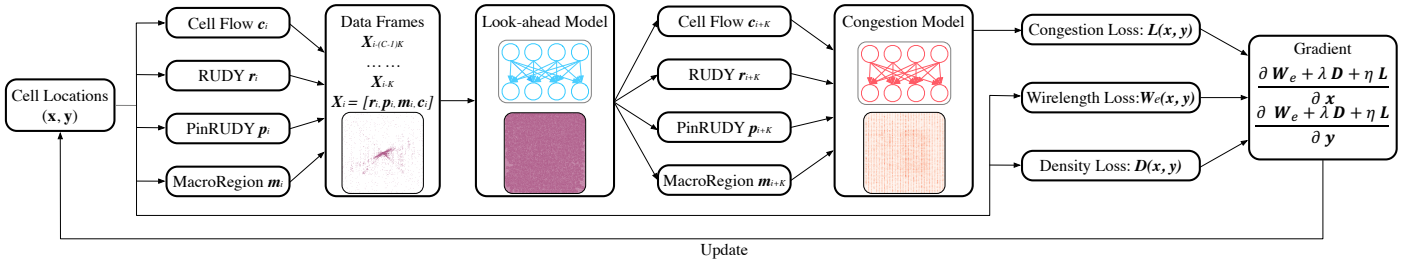


Fig. 2 Overview of LACO. The look-ahead model predicts future placement features. The cell flow captures the motions of cells. The predicted placement features and cell flow are leveraged to estimate the congestion hotspots, whose gradient can guide the update of the cell locations.

$\{\boldsymbol{X}_{i-(C-1)K}, \boldsymbol{X}_{i-(C-2)K}, \ldots, \boldsymbol{X}_i\}$, and predict $\overline{\boldsymbol{X}}_{i+K}$ with a DNN represented by,

$$\overline{\boldsymbol{X}}_{i+K} = g(\boldsymbol{X}_{i-(C-1)K}, \boldsymbol{X}_{i-(C-2)K}, \ldots, \boldsymbol{X}_i). \tag{11}$$

Finally, the novel congestion penalty can be summarized as,

$$L_i(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{MN} \| f \circ g(\boldsymbol{X}_{i-(C-1)K}, \boldsymbol{X}_{i-(C-2)K}, \ldots, \boldsymbol{X}_i) \|_2^2. \tag{12}$$

The gradient from the congestion penalty, $(\frac{\partial L_i(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{x}}, \frac{\partial L_i(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{y}})$ can be calculated with the chain rule and utilized to update the cell locations.

Since looking ahead cannot completely eliminate the distribution shift, we believe that learning an invariant latent feature space is beneficial for enhancing the robustness of placement feature prediction under varying inputs.

Therefore, we need to solve the following problems. (1) How to represent the motions of cells. (2) How to predict future placement features. (3) How to reduce the distribution variation of DNN feature maps. Problems (1) and (2) contribute to the less-shifted inputs for the congestion prediction model. Problem (3) is for the learning of invariant latent feature space.

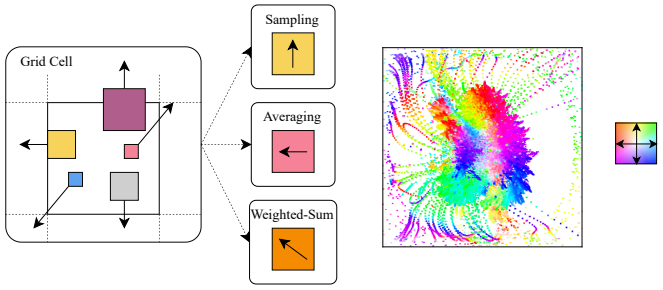
The overview of our algorithm is presented in Fig. 2. At each placement iteration, we generate the RUDY, PinRUDY, and MacroRegion features as [22]. In addition, we downsample the cell information with a quasi-voxelization process and obtain the cell flows, which will be discussed in Section III-B to solve problem (1). To predict future placement features, C frames of data are inputted to a look-ahead DNN model described in Section III-C, solving problem (2). In Section III-D, we will introduce our technique to learn an invariant latent feature space in the look-ahead model, solving problem (3). Moreover, a congestion prediction DNN estimates the congestion hotspots according to the predicted RUDY, PinRUDY, MacroRegion, and cell flow. The gradients from the congestion penalty $L(\boldsymbol{x}, \boldsymbol{y})$ can be utilized to update the cell locations $(\boldsymbol{x}, \boldsymbol{y})$, which will be discussed in Section III-E.

B. Cell Flow

To predict the motions of cells, we need to represent the cells for the motion estimation model. As shown in Fig. 1, the cells can be viewed as points distributed within a certain range. Thus, point cloud is a suitable representation of the cells. Each point has a 2D coordinate representing its location and an additional feature vector containing its information, such as the size and type.

Inspired by the optical flow in computer vision, we leverage the novel cell flow to represent the motions of cells in the point cloud. Assuming that a cell j has a location $(x_{i,j}, y_{i,j})$ at the i-th iteration of global placement, the cell flow of this cell between the i-th and $(i-K)$-th iteration is defined as $\boldsymbol{c}'_i(x_{i,j}, y_{i,j}) = (x_{i,j} - x_{i-K,j}, y_{i,j} - y_{i-K,j})$.

An intuitive way to look ahead is to predict the motion of every cell during the placement. However, a circuit may contain more than one million cells, which imposes a heavy burden not only on the runtime but also on the training of the prediction model. Thus, we downsample the cell flow with a quasi-voxelization process to reduce computation.



(a) (b)

Fig. 3 Illustration of quasi-voxelization and cell flow. (a) The motions of cells and the downsampled cell flow under different quasi-voxelization scheme. (b) The cell flow at the 150-th iteration under the weighted-sum scheme. Different colors represent different directions.

Inspired by point cloud applications [28] in computer vision, we design three quasi-voxelization methods to downsample the cell flows. To get a consistent feature size with RUDY, we also divide the layout to an $M \times N$ grid in cell flow computation. For the grid-cell $b_{k,l}$, we use $\boldsymbol{c}_i(k,l)$ to denote the downsampled cell flow, which can be computed with one of the following schemes:

1) Sampling: It uses the cell flow of the largest cell. Representing the size of cell j with s_j, we can define the cell flow as,
$$\boldsymbol{c}_i(k,l) = s_{\hat{j}} \boldsymbol{c}'_i(x_{\hat{j}}, y_{\hat{j}}), \quad \hat{j} = \arg\max_j s_j, (x_{i,j}, y_{i,j}) \in b_{k,l}. \tag{13}$$
2) Averaging: It uses the average cell flow of the cells in $b_{k,l}$,
$$\boldsymbol{c}_i(k,l) = \frac{1}{N_{k,l}} \sum_{(x_{i,j}, y_{i,j}) \in b_{k,l}} \boldsymbol{c}'_i(x_{i,j}, y_{i,j}), \tag{14}$$
where $N_{k,l}$ is the number of cells in the grid-cell $b_{k,l}$.
3) Weighted-sum: It uses the weighted sum of the cell flows,
$$\boldsymbol{c}_i(k,l) = \sum_{(x_{i,j}, y_{i,j}) \in b_{k,l}} \frac{s_j}{N_{k,l}} \times \boldsymbol{c}'_i(x_{i,j}, y_{i,j}). \tag{15}$$

Fig. 3(a) shows the difference between the schemes. In weighted-sum, we consider the motions and sizes of all cells, which preserves more information than others. Thus, weighted-sum is the default scheme. After quasi-voxelization, we can get an $2 \times M \times N$ downsampled cell flow, which aggregates the horizontal and vertical motions. As an example, Fig. 3(b) presents the cell flow at the 150 iteration on `des_perf_1`, where the motions are represented by colors.

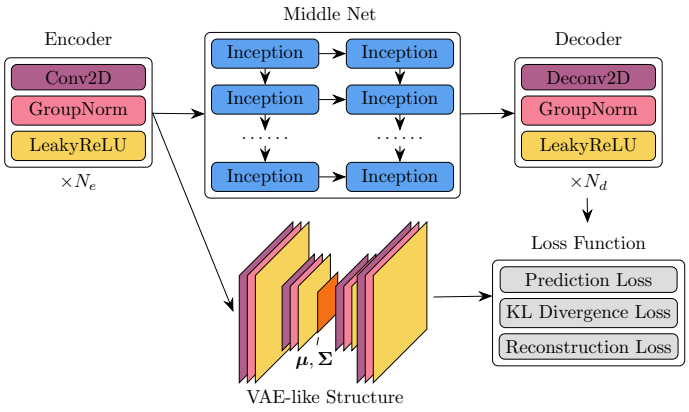


Fig. 4 Overview of the look-ahead model, which consists of a video prediction network inspired by [26] and a VAE-like structure.

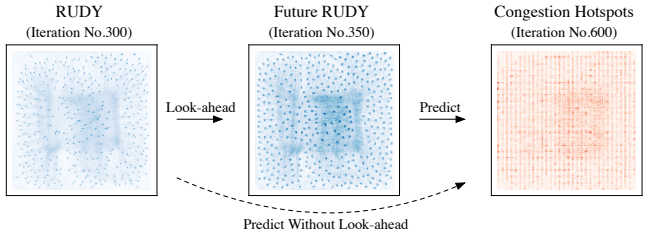


Fig. 5 Visual illustration of the look-ahead mechanism. We only show the RUDY maps and congestion hotspots for simplification. The dashed line represents the flow without a look-ahead mechanism.

C. Look-ahead Model

The task of predicting future placement features is similar to the video prediction task in computer vision, where the model takes several frames as the input and outputs future frames. Although there exist complex models for video tasks [29], SimVP [26] proves that convolutional neural network (CNN) can achieve state-of-the-art video prediction performance with a simple structure and a small number of parameters, which reduces the runtime of the prediction model. Therefore, the proposed look-ahead model is adapted from SimVP.

As shown in Fig. 4, the look-ahead model contains the encoder, middle net, and decoder. The encoder consists of multiple blocks of convolution, group normalization, and leaky ReLU layers. The decoder is similar to the encoder but replaces the convolution layers with deconvolution layers. The middle net includes multiple Inception modules designed in [26], each of which consists of a bottleneck convolution layer with 1×1 kernel followed by parallel group convolution layers. The branch to a VAE-like structure aims to learn an invariant latent feature space, which will be introduced in Section III-D. The loss of the look-ahead model includes three functions. The prediction loss is $\|\overline{\boldsymbol{X}}_{i+K} - \boldsymbol{X}_{i+K}\|_2^2$. The KL divergence loss and reconstruction loss are from the VAE-like structure. To reduce the runtime, the look-ahead model uses a lower resolution and interpolates its outputs to match the input size of the congetsion prediction model. Inspired by residual neural networks, we input $\boldsymbol{X}_i$ to the congestion model as a shortcut to capture more information. These techniques speed up the look-ahead prediction without performance degradation and make it easier to train the DNN.

Fig. 5 shows a simple illustration of the look-ahead mechanism. By looking ahead to a future iteration, we can obtain a RUDY map that better indicates the final placement features.

D. Learning Invariant Feature Space

Inspired by variational autoencoder (VAE) [30], we map the DNN feature map from the encoder in our look-ahead model to a standard Gaussian distribution with several convolution layers and reconstruct the DNN feature map with deconvolution layers. The encoded features of the VAE-like structure are decoupled to a mean vector $\boldsymbol{\mu}$ and a variance matrix $\boldsymbol{\Sigma}$. To learn a standard Gaussian latent space, we minimize the KL divergence between the encoded features and a standard Gaussian distribution,

$$KL(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})) = \frac{1}{2}\left(-\log(|\boldsymbol{\Sigma}|) - N_k + tr(\boldsymbol{\Sigma}) + \boldsymbol{\mu}^T\boldsymbol{\mu}\right), \tag{16}$$

where N_k is the dimensionality of the distribution. We also minimize the reconstruction loss, which is defined as the mean squared error (MSE) between the input and output of the VAE-like structure. In the training of our look-ahead model, the loss function is the weighted sum of the prediction loss, KL divergence loss, and reconstruction loss. After training, the VAE-like structure can be ignored in congestion prediction. Thus, it does not incur runtime overhead.

E. Congestion Optimization in Global Placement

To verify the effectiveness of our method, we follow the previous deep-learning-based congestion optimization algorithm [22] to design our congestion prediction model, which consists of five convolution and two deconvolution layers. The gradient of the congestion penalty can be propagated from the congestion prediction to the cell locations. The gradients $\nabla_{\overline{\boldsymbol{X}}_{i+K}} L_i(\boldsymbol{x}, \boldsymbol{y})$ and $\nabla_{\boldsymbol{X}_i} \overline{\boldsymbol{X}}_{i+K}$ are computed by the auto-gradient feature of DNN toolboxes. To update the cell locations, $\nabla_{\boldsymbol{x}} \boldsymbol{X}_i$ can be computed by summing up the following gradients:

1) $\nabla_{\boldsymbol{x}} \boldsymbol{r}_i$. According to the definition of RUDY in Equation (4), it can be computed with,

$$\nabla_{\boldsymbol{x}} \boldsymbol{r}_i = \sum_{e \in E} \nabla_{\boldsymbol{x}} \mathbf{RUDY}_e(\boldsymbol{x}, \boldsymbol{y}), \tag{17a}$$

$$\nabla_{x_{i,j}} \mathbf{RUDY}_e(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} -\frac{1}{(x_e^h - x_e^l)^2}, & \text{in } [x_e^l, x_e^h] \times [y_e^l, y_e^h], \\ 0, & \text{in other locations.} \end{cases} \tag{17b}$$

2) $\nabla_{\boldsymbol{x}} \boldsymbol{p}_i$ can be derived like $\nabla_{\boldsymbol{x}} \boldsymbol{r}_i$ because RUDY and PinRUDY have similar definitions.
3) $\nabla_{\boldsymbol{x}} \boldsymbol{m}_i = 0$ because the macro cells are fixed.
4) $\nabla_{\boldsymbol{x}} \boldsymbol{c}_i$ differs in three quasi-voxelization schemes. In the sampling scheme (Equation (13)), the gradient is $s_{\hat{j}}$ for a selected cell and 0 for others. For the points in the grid-cell $b_{k,l}$, the gradient is $\frac{1}{N_{k,l}}$ in the averaging scheme (Equation (14)), while the gradient is $\frac{s_j}{N_{k,l}}$ in the weighted-sum scheme (Equation (15)).

The gradient $\nabla_{\boldsymbol{y}} \boldsymbol{X}_i$ can be derived similarly. Finally, the gradients from the congestion penalty $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{y})$ and $\nabla_{\boldsymbol{y}} L(\boldsymbol{x}, \boldsymbol{y})$ can be computed with the chain rule.

IV. Experiments

A. Experimental Setup

All DNN models in this paper are developed in Python with PyTorch and trained on NVIDIA TITAN Xp GPU. We implement the operators for RUDY, PinRUDY, MacroRegion, and cell flow with C++ and CUDA, which can be integrated into PyTorch programs. We conduct experiments on ISPD 2015 benchmark [31] following the settings of [22]. To provide a fair comparison, we integrate all methods in our experiments to the recently improved DREAMPlace. LACO uses 4 frames of data as the inputs, then predicts the placement features and cell flow after 50 iterations. It uses 512×512 grids for congestion prediction and a 64×64 resolution for the look-ahead model. We

generate 100 placement solutions for each design using DREAMPlace with different parameters. The ground truths of congestion hotspots are obtained from Cadence Innovus v17.1 based on the placement solutions. We use an end-to-end model trained on the first 8 designs in TABLE I to evaluate LACO on other designs. For the test on the first 8 designs, we randomly select 800 placement solutions from other designs as the training set.

B. Comparison with Previous Methods

We evaluate the results of placement with information obtained from Innovus. The worst congestion score (WCS) and wirelength (WL) after global routing are used as the metrics. WCS is defined as the maximum ratio of overflow routing tracks ($\#OF$) to available tracks ($\#AV$),

$$\text{WCS}_H = \max_{x,y} \frac{\#\text{OF}_H}{\#\text{AV}_H}, \quad \text{WCS}_V = \max_{x,y} \frac{\#\text{OF}_V}{\#\text{AV}_V}, \tag{18}$$

where WCS_H and WCS_V refer to the WCS on the horizontal and vertical directions, respectively.

TABLE I compares DREAMPlace [24], DREAM-Cong [22], and our method LACO. DREAM-Cong brings a little improvement in WCS. Due to better congestion estimation, LACO achieves 8.0% and 6.4% improvement in WCS_H and WCS_V, respectively, compared to the baseline. Its WL is comparable to DREAMPlace and better than DREAM-Cong.

C. Ablation Study

In this section, we show the effectiveness of LACO with a series of ablation studies. We apply the proposed mechanisms to the original DREAM-Cong one by one to show their effects. The following schemes are compared:

1) DREAM-Cong: It uses congestion prediction only.
2) Look-ahead-only: It estimates the congestion with the predicted placement features $[\overline{\boldsymbol{r}}_{i+K}, \overline{\boldsymbol{p}}_{i+K}, \overline{\boldsymbol{m}}_{i+K}]$.
3) Cell-flow: It estimates the congestion with the predicted placement features and cell flows $[\overline{\boldsymbol{r}}_{i+K}, \overline{\boldsymbol{p}}_{i+K}, \overline{\boldsymbol{m}}_{i+K}, \overline{\boldsymbol{c}}_{i+K}]$.
4) Cell-flow+KL: In addition to Cell-flow, it uses the VAE-like structure to learn an invariant latent feature space. It is the final version of LACO that is used in Section IV-B.

To compare these schemes on congestion prediction, we use the NRMS and SSIM metrics that are used in [32]. Normalized root mean-square error (NRMS) is defined as,

$$\text{NRMS}(\overline{\boldsymbol{Y}}, \boldsymbol{Y}) = \frac{\|\overline{\boldsymbol{Y}} - \boldsymbol{Y}\|_2}{(Y_{\max} - Y_{\min})\sqrt{N_Y}}, \tag{19}$$

where $\boldsymbol{Y}$ is the ground truth, $\overline{\boldsymbol{Y}}$ is the predicted congestion, and N_Y is number of grid-cells. Structural similarity (SSIM) measures the similarity of two images in terms of statistics. In congestion prediction, the similarity can be defined as,

$$\text{SSIM}(\overline{\boldsymbol{Y}}, \boldsymbol{Y}) = \frac{(2\mu_{\boldsymbol{Y}}\mu_{\overline{\boldsymbol{Y}}} + C_1)(2\sigma_{\boldsymbol{Y},\overline{\boldsymbol{Y}}} + C_2)}{(\mu_{\boldsymbol{Y}}^2 + \mu_{\overline{\boldsymbol{Y}}}^2 + C_1)(\sigma_{\boldsymbol{Y}}^2 + \sigma_{\overline{\boldsymbol{Y}}}^2 + C_2)}, \tag{20}$$

where $\mu_{\boldsymbol{Y}}$ and $\mu_{\overline{\boldsymbol{Y}}}$ are the mean values, $\sigma_{\boldsymbol{Y}}^2$ and $\sigma_{\overline{\boldsymbol{Y}}}^2$ are the variances. The correlation coefficient between the ground truth and congestion prediction is $\sigma_{\boldsymbol{Y},\overline{\boldsymbol{Y}}}$. C_1 and C_2 are two constants.

We use the first 8 designs to train these models and compare their performance on congestion prediction. Fig. 6 compares the schemes on NRMS and SSIM. Look-ahead-only improves the NRMS and SSIM significantly. Cell-flow and Cell-flow+KL bring further improvement. Finally, Cell-flow+KL achieves 34.8% and 28.7% improvement in NRMS and SSIM, respectively. This experiment shows the effectiveness of LACO on congestion prediction. A better estimation of congestion hotspots can provide an more accurate congestion panelty on the cell locations.

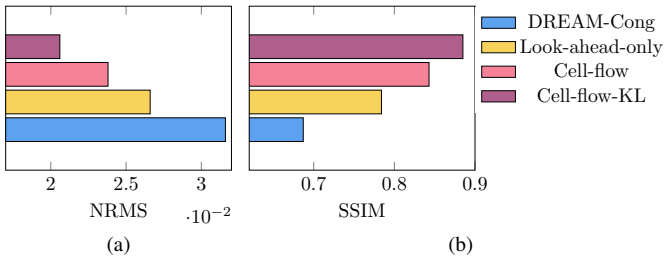


Fig. 6 Scheme comparison on (a) NRMS ↓; (b) SSIM ↑.

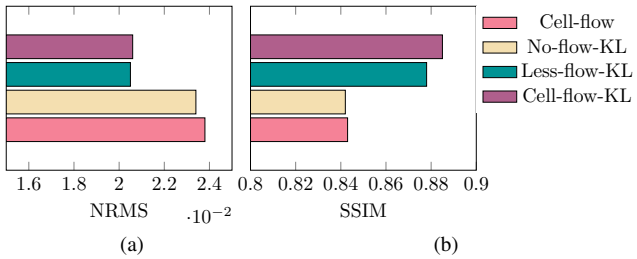


Fig. 7 Scheme comparison on (a) NRMS ↓; (b) SSIM ↑.

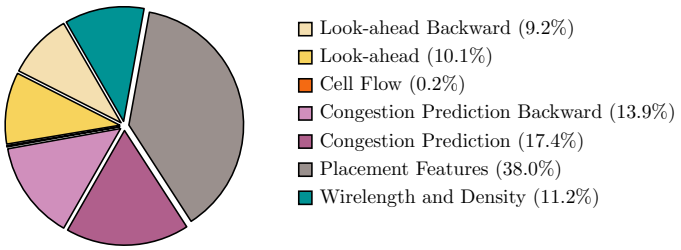


Fig. 8 Runtime breakdown of LACO, based on DREAMPlace.

To show the effects of cell flow and invariant latent feature space, we also compare the following schemes with Cell-flow and Cell-flow+KL:

1) No-flow-KL: It removes everything about cell flow in Cell-flow+KL.
2) Less-flow-KL: It keeps most features of Cell-flow+KL, but do not input the cell flow into the congestion prediction model.

As shown in Fig. 7, Less-flow-KL has comparable performance with Cell-flow+KL but has slightly worse SSIM. However, totally removing the cell flow leads to obvious deterioration on both NRMS and SSIM. Cell-flow does not employ the VAE-like structure and also has worse performance than Cell-flow+KL.

We test the three quasi-voxelization schemes in congestion prediction. The average scheme has the worst performance, obtaining 28.8% larger NRMS than the weighted-sum scheme. The sampling scheme has 2.1% larger NRMS than weighted-sum. Thus, we use the weighted-sum scheme by default.

D. Runtime

Fig. 8 shows the average runtime breakdown of LACO on the testcases. The novel look-ahead mechanism does not induce much runtime overhead. Most time is spent on placement feature gathering and congestion prediction, which is not brought by the look-ahead model. Since computing the cell flow only needs to consider the cells, it consumes much less time than placement feature gathering, which needs to traverse all nets.

V. Conclusion

TABLE I Comparison Between LACO and Previous Methods on ISPD 2015 Benchmark

Benchmark	#Cells	#Nets	DREAMPlace			DREAM-Cong			LACO		
			WCS_H	WCS_V	$WL(10^5\mu m)$	WCS_H	WCS_V	$WL(10^5\mu m)$	WCS_H	WCS_V	$WL(10^5\mu m)$
des_perf_1	113k	113k	0.47	0.40	13.88	0.47	0.40	13.82	0.40	0.40	13.87
des_perf_a	109k	110k	2.25	1.67	22.21	1.89	1.60	22.33	1.69	1.30	22.27
des_perf_b	113k	113k	0.07	0.27	16.70	0.13	0.27	16.71	0.07	0.20	16.57
edit_dist_a	130k	131k	4.05	4.14	53.54	4.30	4.07	53.62	3.50	3.14	53.40
fft_1	35k	33k	0.59	0.40	4.96	0.43	0.47	4.95	0.46	0.40	4.91
fft_2	35k	33k	0.40	0.78	5.86	0.36	0.67	5.88	0.27	0.61	5.84
fft_a	34k	32k	0.55	0.56	10.56	0.83	0.77	10.53	0.50	0.56	10.52
fft_b	34k	32k	3.50	2.33	12.13	3.50	2.67	12.16	3.33	2.33	12.12
matrix_mult_1	160k	159k	0.71	0.53	25.85	0.88	0.58	28.95	0.68	0.44	25.83
matrix_mult_2	160k	159k	0.65	0.42	25.71	0.78	0.84	29.99	0.61	0.45	25.71
matrix_mult_a	154k	154k	0.47	0.40	36.99	0.44	0.37	37.02	0.47	0.37	36.78
matrix_mult_b	151k	152k	8.69	2.65	35.08	8.69	2.65	35.29	8.69	2.65	35.07
matrix_mult_c	151k	152k	0.53	0.40	35.42	0.50	0.27	35.97	0.47	0.30	35.42
pci_bridge32_a	30k	30k	2.06	0.84	6.12	1.83	0.87	6.14	1.89	0.95	6.13
pci_bridge32_b	29k	29k	0.03	0.23	9.77	0.14	0.31	10.57	0.10	0.20	9.65
superblue11_a	954k	936k	1.10	25.00	392.78	1.15	23.00	396.98	1.10	25.00	392.93
superblue12	1293k	1293k	3.00	3.00	414.10	2.73	2.57	414.12	2.45	2.57	413.95
superblue14	634k	620k	1.10	4.17	277.32	1.06	4.67	277.69	1.00	3.50	277.97
superblue16_a	698k	697k	0.91	10.75	309.04	1.00	10.00	310.17	1.00	9.75	309.03
superblue19	522k	512k	1.70	3.67	201.34	1.30	4.33	202.36	1.57	3.50	201.27
Average	-	-	1.64	3.13	95.47	1.62	3.07	96.22	**1.51**	**2.93**	**95.46**
Ratio	-	-	1.00	1.00	1.00	0.99	0.98	1.01	**0.92**	**0.94**	**1.00**

In this paper, we propose a look-ahead congestion optimization method to mitigate the distribution shift problem for deep-learning-based congestion optimization in global placement. By predicting the future placement features, we achieve a significant improvement in congestion prediction. With the quasi-voxelization and cell flow prediction mechanism, we enhance the performance by capturing the motions of cells. The VAE-like structure also contributes to the congestion prediction with less distribution shift. Integrating the proposed mechanisms to global placement contributes to a maximum of 34.8% improvement in congestion prediction and 8.0% improvement in congestion optimization than previous methods.

REFERENCES

[1] T.-C. Chen *et al.*, "NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs," in *Proc. ISPD*, 2005, pp. 236–238.

[2] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proc. ISPD*, 2005, pp. 185–192.

[3] A. B. Kahng and Q. Wang, "A faster implementation of APlace," in *Proc. ISPD*, 2006, pp. 218–220.

[4] Y. Wei *et al.*, "GLARE: Global and local wiring aware routability evaluation," in *Proc. DAC*, 2012, pp. 768–773.

[5] T. Lin *et al.*, "POLAR: placement based on novel rough legalization and refinement," in *Proc. ICCAD*, 2013, pp. 357–362.

[6] X. He *et al.*, "Ripple: A robust and effective routability-driven placer," *IEEE TCAD*, vol. 32, no. 10, pp. 1546–1556, 2013.

[7] M.-K. Hsu *et al.*, "NTUplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE TCAD*, vol. 33, no. 12, pp. 1914–1927, 2014.

[8] F.-K. Sun and Y.-W. Chang, "BiG: A bivariate gradient-based wirelength model for analytical circuit placement," in *Proc. DAC*, 2019.

[9] H. Szentimrey *et al.*, "Machine learning for congestion management and routability prediction within fpga placement," *ACM TODAES*, vol. 25, no. 5, 2020.

[10] B. Hu and M. Marek-Sadowska, "Fine granularity clustering-based placement," *IEEE TCAD*, vol. 23, no. 4, pp. 527–536, april 2004.

[11] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proc. ASPDAC*, 2007, pp. 135–140.

[12] T. Taghavi *et al.*, "New placement prediction and mitigation techniques for local routing congestion," in *Proc. ICCAD*, 2010, pp. 621–624.

[13] W.-H. Liu, C.-K. Koh, and Y.-L. Li, "Optimization of placement solutions for routability," in *Proc. DAC*, 2013, pp. 153:1–153:9.

[14] J. Lu *et al.*, "ePlace-MS: Electrostatics-based placement for mixed-size circuits," *IEEE TCAD*, vol. 34, no. 5, pp. 685–698, 2015.

[15] C.-C. Huang *et al.*, "NTUplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE TCAD*, vol. 37, no. 3, pp. 669–681, 2018.

[16] J.-M. Lin *et al.*, "Routability-driven global placer target on removing global and local congestion for vlsi designs," in *Proc. ICCAD*, 2021.

[17] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proc. DATE*, 2007.

[18] Z. Xie *et al.*, "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. ICCAD*, 2018, pp. 80:1–80:8.

[19] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proc. DAC*, 2019.

[20] B. Wang *et al.*, "LHNN: Lattice hypergraph neural network for VLSI congestion prediction," in *Proc. DAC*, 2022, pp. 1297–1302.

[21] C.-C. Chang *et al.*, "Automatic routability predictor development using neural architecture search," in *Proc. ICCAD*, 2021.

[22] S. Liu *et al.*, "Global placement with deep learning-enabled explicit routability optimization," in *Proc. DATE*, 2021.

[23] C. Cheng *et al.*, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, vol. 38, no. 9, pp. 1717–1730, 2019.

[24] Y. Lin *et al.*, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *Proc. DAC*, 2019.

[25] X. Bei, Y. Yang, and S. Soatto, "Learning semantic-aware dynamics for video prediction," in *Proc. CVPR*, 2021, pp. 902–912.

[26] Z. Gao *et al.*, "SimVP: Simpler yet better video prediction," in *Proc. CVPR*, 2022, pp. 3170–3180.

[27] S. Sun *et al.*, "Optical flow guided feature: A fast and robust motion representation for video action recognition," in *Proc. CVPR*, 2018, pp. 1390–1399.

[28] Z. Liu *et al.*, "Point-voxel CNN for efficient 3D deep learning," *Proc. NeurIPS*, vol. 32, 2019.

[29] Y. Wang *et al.*, "PredRNN: A recurrent neural network for spatiotemporal predictive learning," *IEEE TPAMI*, 2022.

[30] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. ICLR*, 2014.

[31] I. S. Bustany *et al.*, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proc. ISPD*, 2015, pp. 157–164.

[32] M. B. Alawieh *et al.*, "High-definition routing congestion prediction for large-scale FPGAs," in *Proc. ASPDAC*, 2020, pp. 26–31.