Global Placement with Deep Learning-Enabled Explicit Routability Optimization

Siting Liu\textsuperscript{1}, Qi Sun\textsuperscript{1}, Peiyu Liao\textsuperscript{1}, Yibo Lin\textsuperscript{2}, Bei Yu\textsuperscript{1}
\textsuperscript{1}The Chinese University of Hong Kong \textsuperscript{2}Peking University
\{stliu,qsun,pyliao,byu\}@cse.cuhk.edu.hk, yibolin@pku.edu.cn

Abstract—Placement and routing (PnR) is the most time-consuming part of the physical design flow. Recognizing the routing performance ahead of time can assist designers and design tools to optimize placement results in advance. In this paper, we propose a fully convolutional network model to predict congestion hotspots and then incorporate this prediction model into a placement engine, DREAMPlace, to get a more route-friendly result. The experimental results on ISPD2015 benchmarks show that with the superior accuracy of the prediction model, our proposed approach can achieve up to 9.05\% reduction in congestion rate and 5.30\% reduction in routed wirelength compared with the state-of-the-art.

I. INTRODUCTION

Routability is one of the key concerns in modern placement for very-large-scale integrated (VLSI) circuits [1]. With the designs getting increasingly large and complicated, simply minimizing interconnect wirelength can no longer guarantee good routed wirelength [2]–[5], as poor routing congestion may lead to severe routing detour or even failure. Thus, routability optimization is a necessity for practical placement algorithms.

Modern routability-driven placement algorithms usually consist of a kernel placement solver and a routability estimator. The former minimizes for typical placement objectives, e.g., interconnect wirelength and overlap between cells, and the latter provides congestion feedback to guide the kernel solver. For example, most nonlinear placement algorithms take the feedback and inflate cells in congested regions to reduce the routing demands in these regions [6], [7]. Quadratic placement algorithms can also incorporate routability optimization in rough legalization [8], [9].

To enable fast and accurate routability estimation, deep learning is introduced for its high performance. Xie et al. propose a fully convolutional network (FCN) to predict number of total routing hotspots with features extracted from placement [10]. Yu et al. propose a generative adversarial network (GAN) to learn the correlation between placement algorithms can also incorporate routability optimization in rough legalization [8], [9].

However, most studies stop at modeling without really integrating the models to the state-of-the-art global placement engines for routability optimization. By developing a neural network model to accurately predict routing demands, we directly integrate the model into the placement objective for explicit routability optimization in this paper. The main contributions of this paper are listed as follows:

- We develop a FCN-based model to predict the routing congestion map given placement solutions.
- We integrate the model into the objective as a penalty term to explicitly optimize routability.
- Experimental results demonstrate that we can achieve more than 5\% improvement in routed wirelength compared with several state-of-the-art placers [6], [12].

The rest of our paper is organized as follows. Section II provides preliminaries including models and definitions. Section III presents detailed methods and the whole optimization flow. Section IV conducts several experiments to validate our methods, followed by a conclusion in Section V.
choose three features composing the input truth congestion hotspots information using Innovus global router and Through the inference of the pre-trained routability prediction model, three input features are extracted from the cell placement solution.

Fig. 1. Different from DREAMPlace, it involves the computation of tive function for routability. The overall computation flow is shown in A. Overall Flow

The routability prediction problem

The computation of the congestion penalty is shown in Fig. 3. Firstly, We add a machine learning-based congestion penalty into the objec-

• PinRUDY: RUDY:

• PinRUDY:

• MacroRegion:

with the model structure illustrated in Fig. 2, we can get an output map with size M × N, which contains congestion hotspots information. The routability prediction problem \( f_R \) can be formally expressed as,

\[
\begin{align*}
\text{MacroRegion}(i, j) &= \begin{cases} 
1 & (i, j) \text{ is in a macro cell,} \\
0 & \text{otherwise}
\end{cases} 
\tag{6}
\end{align*}
\]

With the model structure illustrated in Fig. 2, we can get an output map with size M × N, which contains congestion hotspots information. The routability prediction problem \( f_R \) can be formally expressed as,

\[
\begin{align*}
f_R : \mathbb{R}^{M \times N \times 3} &\rightarrow \mathbb{R}^{M \times N} 
\end{align*}
\]

We define the prediction error of \( f_R \) as mean square error and train the parameters with Adam optimizer [15].

III. ALGORITHMS

A. Overall Flow

We add a machine learning-based congestion penalty into the objective function for routability. The overall computation flow is shown in Fig. 1. Different from DREAMPlace, it involves the computation of congestion gradient to explicitly optimize cell placement.

The computation of the congestion penalty is shown in Fig. 3. Firstly, three input features are extracted from the cell placement solution. Through the inference of the pre-trained routability prediction model, we get the predicted congestion map. Finally, we take mean squared Frobenius norm of this congestion map as the congestion penalty.

B. Routability Prediction Model

There are many previous works on network-based routability evaluation [14], [11], [10]. In our proposed model, we obtain the ground truth congestion hotspots information using Innovus global router and choose three features composing the input \( M \times N \times 3 \) feature map from the cell placement solution.

• RUDY: The RUDY map defined in Equation (4).

• PinRUDY: We further define PinRUDY as the pin density map using \( \text{RUDY}_e, p \in e \) as the weight each pin \( p \) incident to net \( e \). Suppose we divide the layout into \( M \times N \) bins, we can compute the PinRUDY for each bin \( b_{ij} \) as follows,

\[
\text{PinRUDY}_p(i,j) = \frac{1}{n_e} + \frac{1}{n_p}, \quad p \in e, p \in b_{ij} 
\tag{5a}
\]

\[
\text{PinRUDY}(i,j) = \sum_{p \in b_{ij}} \text{PinRUDY}_p(i,j), i \in [1,M], j \in [1,N]. 
\tag{5b}
\]

where \( p \) denotes the pins covered by bin \( b_{ij} \), \( e \) is the net that this pin \( p \) is incident to.

• MacroRegion: We also adopt an \( M \times N \) macro region map to indicate the covered region of macro cells. For each bin \( (i,j) \), the macro region map can be computed as,

\[
\text{MacroRegion}(i,j) = \begin{cases} 
1 & (i, j) \text{ is in a macro cell,} \\
0 & \text{otherwise}
\end{cases} 
\tag{6}
\]

With the model structure illustrated in Fig. 2, we can get an output map with size \( M \times N \), which contains congestion hotspots information. The routability prediction problem \( f_R \) can be formally expressed as,

\[
\begin{align*}
f_R : \mathbb{R}^{M \times N \times 3} &\rightarrow \mathbb{R}^{M \times N} 
\end{align*}
\]

We define the prediction error of \( f_R \) as mean square error and train the parameters with Adam optimizer [15].

C. Routability-Driven Placement

To incorporate routing information into DREAMPlace, we add a new penalty term into our objective function and formulate the new optimization problem as follows.

\[
\min_{x,y} \sum_{e \in E} W_e(x,y) + \lambda D(x,y) + \eta L(x,y), 
\tag{8}
\]

The computation flow, shown in Fig. 3, describes how the gradients of this congestion penalty with respect to cell locations are computed. Given the cell locations \( (x, y) \), we extract input features with the functions defined in Equations (4) (5) (6) and then stack these three feature maps into a three-channel feature map \( M \in \mathbb{R}^{M \times N \times 3} \). We feed this three-channel feature map \( M \) into our pre-trained model to generate a congestion map \( f_R(M) \in \mathbb{R}^{M \times N} \). The mean squared Frobenius norm is applied to compute the congestion penalty \( L(x,y) := \frac{1}{MN} \| f_R(M) \|_2^2 \).

To successfully proceed through the gradient-based optimization, we are required to compute the gradients of loss function \( L \) with respect to cell locations. Note that gradients are only defined on vector fields, therefore we use the notation \( \nabla_A \) to represent taking a derivative with respect to a matrix \( A \) of size \( M \times N \) as the gradient with respect to its vectorized representation, for simplicity.

\[
\nabla_A f := \nabla_{\text{vec}(A)} f, 
\tag{9a}
\]

\[
\text{vec}(A) := [a_{1,1}, \cdots, a_{M,1}, \cdots, a_{1,N}, \cdots, a_{M,N}]^T. 
\tag{9b}
\]

Here \( a_{i,j} \) represents the entry at \( i \)th row and \( j \)th column. With the definition (9), the gradient with respect to \( f_R(M) \) can be computed as

\[
\nabla_{f_R(M)} L = \frac{2}{MN} f_R(M). 
\tag{10}
\]

Now we consider full steps of gradient computation with chain rule. Illustrated in Fig. 3, the gradient propagation have three consecutive parts.

1) Compute gradient w.r.t. congestion map: \( \nabla_{f_R(M)} L \).
2) Back-propagate \( \nabla_{f_R(M)} L \) through our pretrained neural network model, to obtain the gradient w.r.t. stacked features: \( \nabla_M L \).
3) Extract different channels of \( \nabla_M L \) as gradient w.r.t. three different maps, compute gradient w.r.t. cell locations respectively and finally sum them together.

The chain rule in matrix calculus indicates the following formulation,

\[
\nabla_M L = J_M(f_R(M))^\top \cdot \nabla_{f_R(M)} L. 
\tag{11}
\]

Here \( J \) denotes a Jacobian matrix. To enable the full steps of gradient computation, we are going to complete the multiplication on the righthand side of Equation (11) step-by-step.

In fact, the rightmost term \( \nabla_{f_R(M)} L \) is explicitly calculated in Equation (10). The middle term \( J_M(f_R(M)) \) represents the back-propagation transformation matrix, therefore we can calculate the gradient of loss w.r.t the RUDY features,

\[
\nabla_M L = J_M(f_R(M))^\top \cdot \nabla_{f_R(M)} L. 
\tag{12}
\]

Through the back-propagation of pre-trained neural network. As the input feature \( M \) contains three channels, the gradient w.r.t. \( M \) can also be
Here we say approximation movable cells and optimize the objective function. We take gradients with subgradients. We take are actually not differentiable everywhere, which forces us to replace this net respectively. The notation has a pin at so we use a subgradient for gradient descent. From Equation (15), we to the maximal and the minimal, which introduces non-differentiability, the derivative can be set to

\[
\partial \eta \left( \tilde{x}_t \right) = \delta_{\tilde{x}_t} - \delta_{j_t},
\]

where \( k := \text{argmax}_{(\tilde{x}_i, \tilde{y}_i) \in \tilde{e}_t} \tilde{x}_t \) and \( l := \text{argmin}_{(\tilde{x}_i, \tilde{y}_i) \in \tilde{e}_t} \tilde{x}_t \) are the indices of pins that have maximal and minimal horizontal coordinate in this net respectively. The notation \( \delta_{\tilde{x}_t} \) is the Knödel delta function.

Similarly to calculate \( J_\theta(\text{PinRUDY}(x, y)) \). Back to Equation (13), we propagate the gradient and calculate \( \nabla_x L \). Let \( J(x, y) \) be the objective function that is required to be minimized in Equation (8).

\[
\nabla_x J = \sum_{e \in E} \nabla e W_e(x, y) + \lambda \nabla x D(x, y) + \eta \nabla x L(x, y).
\]
not count this fail benchmark when calculating the average metrics of NTUplace4dr. The results show that we can get 9.05% reduction in total congestion rate and 5.30% reduction in routed wirelength compared with DREAMPlace, and 18.68% reduction compared with NTUplace4dr [6].

Fig. 5 plots the runtime breakdown of our proposed method on mgc_des_perf_1. The most-time-consuming part is the computation of congestion penalty gradients $\nabla L$, which takes 46.10% of the total runtime. Since it is a non-differentiability problem, it needs to be computed discretely instead of using a fast mathematical implementation. Furthermore, it takes 17.23% of the total runtime to compute congestion penalty $L$, which includes features extraction, inference, and $l_2$-norm computation. Actually, the $l_2$-norm operator in PyTorch takes 13.6% of the total time, which can be further optimized in the future. Even though the average runtime of our proposed approach is 4× slower than DREAMPlace, it is nearly 23× faster than NTUplace4dr due to the native support for GPU acceleration.

V. CONCLUSION

In this paper, we develop a routability-driven placer based on a deep learning model. With fully convolutional networks, we can achieve efficient and relatively accurate routing congestion modeling at the placement stage. We further integrate the model to the placement objective for explicitly guiding the cell movement. The evaluation on modified ISPD2015 benchmarks has shown that the network can achieve rather accurate prediction. Eventually, we can achieve up to 9.05% reduction in the congestion rate and 5.30% reduction in routed wirelength compared with DREAMPlace and NTUplace4dr.

ACKNOWLEDGMENT

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

### Table II Experiment results on ISPD 2015 benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>NTUplace4dr [6]</th>
<th>DREAMPlace [12]</th>
<th>Ours (e+06 um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>des_perf_1</td>
<td>0.101 0.038</td>
<td>0.143 0.129</td>
<td>0.153 0.126</td>
</tr>
<tr>
<td>des_perf_a</td>
<td>0.022 0.038</td>
<td>0.015 0.021</td>
<td>0.020 0.028</td>
</tr>
<tr>
<td>des_perf_b</td>
<td>0.001 0.002</td>
<td>0.005 0.015</td>
<td>0.040 0.010</td>
</tr>
<tr>
<td>fft_1</td>
<td>0.125 0.093</td>
<td>0.106 0.063</td>
<td>0.101 0.061</td>
</tr>
<tr>
<td>fft_2</td>
<td>0.821 0.002</td>
<td>0.664 0.006</td>
<td>0.665 0.006</td>
</tr>
<tr>
<td>fft_a</td>
<td>0.116 0.015</td>
<td>0.248 0.016</td>
<td>0.191 0.015</td>
</tr>
<tr>
<td>fft_b</td>
<td>0.211 0.067</td>
<td>0.177 0.026</td>
<td>0.142 0.047</td>
</tr>
<tr>
<td>matrix_mult_1</td>
<td>0.156 0.057</td>
<td>0.165 0.340</td>
<td>0.168 0.334</td>
</tr>
<tr>
<td>matrix_mult_2</td>
<td>0.210 0.073</td>
<td>0.228 0.128</td>
<td>0.225 0.124</td>
</tr>
<tr>
<td>matrix_mult_3</td>
<td>0.017 0.028</td>
<td>0.145 0.113</td>
<td>0.030 0.024</td>
</tr>
<tr>
<td>matrix_mult_b</td>
<td>0.032 0.035</td>
<td>0.044 0.025</td>
<td>0.020 0.028</td>
</tr>
<tr>
<td>matrix_mult_c</td>
<td>48.956 29.719</td>
<td>0.089 0.017</td>
<td>0.029 0.016</td>
</tr>
<tr>
<td>pci_bridge13_a</td>
<td>0.110 0.056</td>
<td>0.192 0.098</td>
<td>0.076 0.036</td>
</tr>
<tr>
<td>pci_bridge13_b</td>
<td>0.001 0.004</td>
<td>0.001 0.005</td>
<td>0.002 0.008</td>
</tr>
<tr>
<td>superblue12</td>
<td>0.034 0.495</td>
<td>0.125 0.374</td>
<td>0.131 0.379</td>
</tr>
<tr>
<td>superblue14</td>
<td>0.064 0.056</td>
<td>0.041 0.051</td>
<td>0.055 0.081</td>
</tr>
<tr>
<td>superblue16_a</td>
<td>0.186 0.031</td>
<td>0.090 0.013</td>
<td>0.164 0.028</td>
</tr>
<tr>
<td>superblue16_b</td>
<td>0.022 0.089</td>
<td>0.033 0.093</td>
<td>0.039 0.091</td>
</tr>
</tbody>
</table>

Average    | 0.131 0.069     | 0.141 0.091     | 0.124 0.087    |

### REFERENCES


