Global Placement with Deep Learning-Enabled Explicit Routability Optimization

Siting Liu[1], Qi Sun[1], Peiyu Liao[1], Yibo Lin[2], Bei Yu[1]
[1]The Chinese University of Hong Kong [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 FPGA placement and routing congestion [11].

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.

TABLE I Notations.

Notation	Description
$\mathbf{x}, \mathbf{y}$	Physical locations of cells in the layout
E	Set of nets
P	Set of pins
$W_e(\cdot,\cdot)$	Wirelength of net e
$D(\cdot,\cdot)$	Density penalty
$L(\cdot,\cdot)$	Routing congestion penalty
λ	Weight of the density penalty
γ	Parameter of the wirelength model for smoothness
η	Weight of the congestion penalty, starts with 8e-9

II. Preliminaries

A. DREAMPlace Algorithm

We use the notations defined in TABLE I if not specially mentioned. DREAMPlace [12] is a GPU-accelerated placement engine, implementing the eplace/RePlAce family as the kernel placement algorithm It uses weighted-average (WA) model to approximate wirelength

$$\tilde{W}_{e_x}(\mathbf{x},\mathbf{y}) = \frac{\sum_{i\in e} x_i e^{x_i/\gamma}}{\sum_{i\in e} e^{x_i/\gamma}} - \frac{\sum_{i\in e} x_i e^{-x_i/\gamma}}{\sum_{i\in e} e^{-x_i/\gamma}}, \tag{1}$$

$$W_e(\mathbf{x},\mathbf{y}) = \tilde{W}_{e_x}(\mathbf{x},\mathbf{y}) + \tilde{W}_{e_y}(\mathbf{x},\mathbf{y}), \tag{2}$$

DREAMPlace defines the optimization problem as,

$$\min_{\mathbf{x},\mathbf{y}} \sum_{e\in E} W_e(\mathbf{x},\mathbf{y}) + \lambda D(\mathbf{x},\mathbf{y}), \tag{3}$$

The key contribution of DREAMPlace is that it analogs an analytical placement problem to network training, which can get significant speedup. Currently DREAMPlace focuses wirelength minimization, while the support to routability optimization is limited.

B. RUDY

Rectangular uniform wire density (RUDY) [13] is a routing demand estimation technique. It provides a two-dimensional map representing the routing demand of different bins in the layout. To compute RUDY map, we first need to obtain the bounding box of a net,

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

where $x_e^l, y_e^l, x_e^h, y_e^h$ denote the left, bottom, right, and top edges of the bounding box of net e, p_e denotes the pins incident to the net, x_{p_e} and y_{p_e} denote the locations of the pins. Then, we can derive the equations for the overall RUDY computation,

$$w_e = x_e^h - x_e^l, \quad h^k = y_e^h - y_e^l, \tag{4a}$$

$$\mu_e = \begin{cases} 1 & x \in [x_e^l, x_e^h], y \in [y_e^l, y_e^h] \\ 0 & \text{otherwise,} \end{cases}, \tag{4b}$$

$$\text{RUDY}_e(\mathbf{x},\mathbf{y}) \propto \mu_e(\frac{1}{w_e} + \frac{1}{h_e}), \tag{4c}$$

$$\text{RUDY}(\mathbf{x},\mathbf{y}) = \sum_{e\in E} \text{RUDY}_e(\mathbf{x},\mathbf{y}), \tag{4d}$$

where RUDY_e is the routing demand for a single net e. Summing up the routing demand of all nets results in the overall RUDY map.

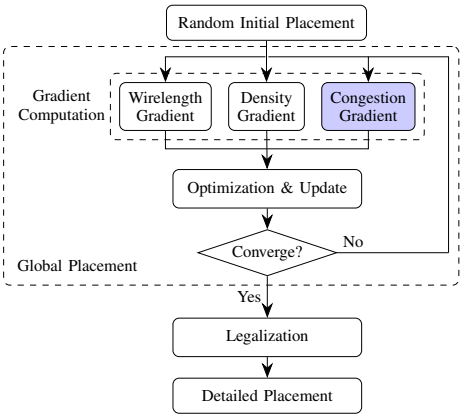


Fig. 1 The overall flow of our framework.

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,

$$\text{PinRUDY}_p(i,j) = \frac{1}{w_e} + \frac{1}{h_e}, \quad p \in e, p \in b_{ij}, \tag{5a}$$

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

  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,

$$f_{\text{R}} : \mathcal{X} \subset \mathbb{R}^{M \times N \times 3} \longrightarrow \mathcal{Y} \subset \mathbb{R}^{M \times N}. \tag{7}$$

We define the prediction error of f_{R} as mean square error and train the parameters with Adam optimizer [15].

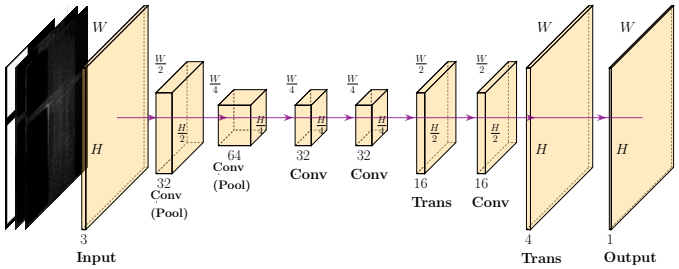


Fig. 2 Prediction model.

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.

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

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 $(\mathbf{x},\mathbf{y})$, 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 $\mathbf{M} \in \mathbb{R}^{M \times N \times 3}$. We feed this three-channel feature map $\mathbf{M}$ into our pre-trained model to generate a congestion map $f_{\text{R}}(\mathbf{M}) \in \mathbb{R}^{M \times N}$. The mean squared Frobenius norm is applied to compute the congestion penalty $L(\mathbf{x},\mathbf{y}) := \frac{1}{MN}\|f_{\text{R}}(\mathbf{M})\|_2^2$.

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 $\nabla_{\mathbf{A}}$ to represent taking a derivative with respect to a matrix $\mathbf{A}$ of size $M \times N$ as the gradient with respect to its *vectorized representation*, for simplicity.

$$\nabla_{\mathbf{A}} f := \nabla_{\text{vec}(\mathbf{A})} f, \tag{9a}$$

$$\text{vec}(\mathbf{A}) := \left[a_{1,1}, \cdots, a_{M,1}, \cdots, a_{1,N}, \cdots, a_{M,N}\right]^\top. \tag{9b}$$

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

$$\nabla_{f_{\text{R}}(\mathbf{M})} L = \frac{2}{MN} f_{\text{R}}(\mathbf{M}). \tag{10}$$

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: $\nabla_{f_{\text{R}}(\mathbf{M})} L$.
2) Back-propagate $\nabla_{f_{\text{R}}(\mathbf{M})} L$ through our pretrained neural network model, to obtain the gradient w.r.t. stacked features: $\nabla_{\mathbf{M}} L$.
3) Extract different channels of $\nabla_{\mathbf{M}} L$ 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,

$$\nabla_{\mathbf{x}} L = \mathbf{J}_{\mathbf{x}}(\mathbf{M})^\top \cdot \mathbf{J}_{\mathbf{M}}(f_{\text{R}}(\mathbf{M}))^\top \cdot \nabla_{f_{\text{R}}(\mathbf{M})} L. \tag{11}$$

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 $\nabla_{f_{\text{R}}(\mathbf{M})} L$ is explicitly calculated in Equation (10). The middle term $\mathbf{J}_{\mathbf{M}}(f_{\text{R}}(\mathbf{M}))$ represents the back-propagation transformation matrix, therefore we can calculate the gradient of loss w.r.t the RUDY features,

$$\nabla_{\mathbf{M}} L = \mathbf{J}_{\mathbf{M}}(f_{\text{R}}(\mathbf{M}))^\top \cdot \nabla_{f_{\text{R}}(\mathbf{M})} L. \tag{12}$$

Through the back-propagation of pre-trained neural network. As the input feature $\mathbf{M}$ contains three channels, the gradient w.r.t. $\mathbf{M}$ can also be

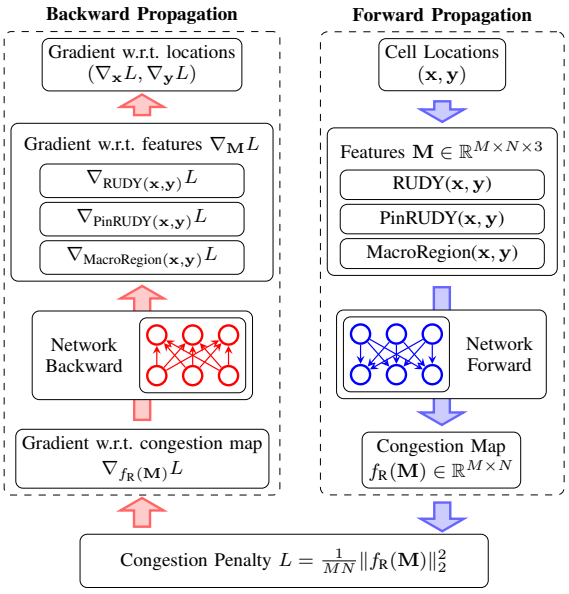


Fig. 3 The gradient-based optimization flow.

divided into three channels accordingly: $\nabla_{\text{RUDY}(\mathbf{x},\mathbf{y})} L$, $\nabla_{\text{PinRUDY}(\mathbf{x},\mathbf{y})} L$, and $\nabla_{\text{MacroRegion}(\mathbf{x},\mathbf{y})} L$.

We can rewrite the gradient w.r.t. cell location $\mathbf{x}$ as follows,

$$\begin{aligned} \nabla_{\mathbf{x}} L = &\mathbf{J}_{\mathbf{x}}(\text{RUDY}(\mathbf{x},\mathbf{y}))^{\top} \cdot \nabla_{\text{RUDY}(\mathbf{x},\mathbf{y})} L \\ &+ \mathbf{J}_{\mathbf{x}}(\text{PinRUDY}(\mathbf{x},\mathbf{y}))^{\top} \cdot \nabla_{\text{PinRUDY}(\mathbf{x},\mathbf{y})} L, \end{aligned} \tag{13}$$

Note that the macro regions remain fixed, which indicates that its Jacobian matrix is always a zero matrix, so we cancel out its gradient.

From Equation (4) of RUDY map, we can explicitly find the approximation of Jacobian matrices on the righthand side of Equation (13). Here we say *approximation* because the RUDY and PinRUDY functions are actually not differentiable everywhere, which forces us to replace gradients with subgradients. We take $\mathbf{J}_{\mathbf{x}}(\text{RUDY}(\mathbf{x},\mathbf{y}))$ as an example.

$$\mathbf{J}_{\mathbf{x}}(\text{RUDY}(\mathbf{x},\mathbf{y})) = \sum_e \mathbf{J}_{\mathbf{x}}(\text{RUDY}_e(\mathbf{x},\mathbf{y})), \tag{14}$$

where the entry at ith row and jth column of $\mathbf{J}_{\mathbf{x}}(\text{RUDY}_e(\mathbf{x},\mathbf{y}))$ is

$$\frac{\partial \text{vec}(\text{RUDY}_e(\mathbf{x},\mathbf{y}))_i}{\partial x_j} \propto -\mu_e(i',j') \cdot \frac{1}{w_e^2} \frac{\partial w_e}{\partial x_j}, \tag{15}$$

where $i' := i \bmod M, j' := \text{floor}(i/M)$ represents the corresponding bin location of ith entry of vectorized $\text{RUDY}_e(\mathbf{x},\mathbf{y})$ map. We cancel out h_e as it is not related to horizontal locations. Note that w_e is related to the maximal and the minimal, which introduces non-differentiability, so we use a subgradient for gradient descent. From Equation (15), we can only consider cells that have a pin connected to net e, otherwise the derivative $\partial w_e / \partial x_j$ is zero. Suppose that the jth cell at (x_j, y_j) has a pin at $(\tilde{x}_j, \tilde{y}_j)$ connected to net e, the derivative can be set to

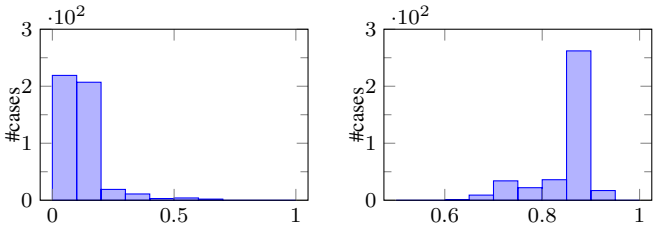
$$\frac{\partial w_e}{\partial x_j} = \delta_{jk} - \delta_{jl}, \tag{16}$$

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

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

$$\nabla_{\mathbf{x}} J = \sum_{e \in E} \nabla_{\mathbf{x}} W_e(\mathbf{x},\mathbf{y}) + \lambda \nabla_{\mathbf{x}} D(\mathbf{x},\mathbf{y}) + \eta \nabla_{\mathbf{x}} L(\mathbf{x},\mathbf{y}). \tag{17}$$

Our proposed placer utilizes gradients $(\nabla_{\mathbf{x}} J, \nabla_{\mathbf{y}} J)$ to adjust all movable cells and optimize the objective function.



(a) NRMS statistics. (b) SSIM statistics.

Fig. 4 Prediction model evaluation.

IV. Experimental Results

We conduct comprehensive evaluations to demonstrate the accuracy of the prediction model and the effectiveness of the proposed framework through two perspectives: congestion and routed wirelength.

A. Experimental Setup

The prediction model was developed in Python with PyTorch, and train on NVIDIA TITAN XP GPU. The congestion penalty operator is implemented in C++ and CUDA, and the incorporation uses Python. It is executed in the same environment with the proposed prediction model. Innovus version is v16.24 on a CentOS Linux server. NTUplace4dr [6] is executed on the CPU threads in the experiments.

B. Routability Prediction Model

We conduct experiments on ISPD 2015 benchmarks [16] with fence region constraints removed, as DREAMPlace does not support them yet. Fourteen of these benchmarks are used for training. For each design, we generate at least 100 different placements by DREAMPlace at different overflows and parameter settings. We then generate congestion information as the golden results using Cadence Innovus global router. Furthermore, this model is tested on the other four designs. We adopt the following two metrics, NRMS and SSIM [17], to evaluate the model.

NRMS indicates the element-wise difference between the prediction result and the ground truth and *SSIM* gives the structural similarity between them. Fig. 4 plots the evaluation of our proposed routability prediction network, where more than 90% of the test cases are in good performance with the metrics in [17] (*NRMS* < 0.2; *SSIM* > 0.8).

C. Routability-driven Placement

We still use the ISPD 2015 contest benchmarks to evaluate our congestion-driven placer With two state-of-the-art placers, DREAMPlace [12] and NTUplace4dr [6] as baselines. We again use Cadence Innovus global router to evaluate the routing performance.

We define the congestion rate as a metric to evaluate the performance of optimizing congestion hotspots.

Definition 1. *(Congestion Rate - CR) CR shows the average lack of routing resources after global routing. It is defined as,*

$$CR = \frac{\sum_{i=1}^{H} \sum_{j=1}^{W} Lk(i,j)}{HW}, \tag{18}$$

where Lk is the number of demand resources exceeds what is available.

CR reflects the level of congestion. We report Horizontal, **H-CR**, and Vertical, **V-CR** separately to show the effect in different directions. Also, we use the classic metric, Routed Wirelength, **WL**, to measure the routing performance. **WL** is estimated by Innovus global router.

TABLE II exhibits the experimental results on the ISPD 2015 benchmarks and reports comparison on **H-CR**, **V-CR**, **WL** and runtime, **RT**. As the results shows, the benchmark, which is marked as gray – `matrix_mult_c`, has a very poor performance using NTUplace4dr. By contrast, our proposed algorithm is more robust. Here we do

TABLE II Experiment results on ISPD 2015 benchmarks.

Benchmark	NTUplace4dr [6]				DREAMPlace [12]				Ours			
	H-CR	V-CR	WL (e+06 um)	RT (s)	H-CR	V-CR	WL (e+06 um)	RT (s)	H-CR	V-CR	WL (e+06 um)	RT (s)
des_perf_1	0.101	0.038	1.32	331	0.143	0.129	1.23	10.868	0.153	0.126	1.23	44.07
des_perf_a	0.022	0.038	2.25	345	0.015	0.021	2.05	12.834	0.020	0.028	1.91	44.51
des_perf_b	0.001	0.002	1.75	349	0.005	0.010	1.71	11.829	0.004	0.010	1.71	45.99
fft_1	0.125	0.093	0.52	79	0.106	0.063	0.45	7.656	0.101	0.061	0.45	43.66
fft_2	0.821	0.002	0.53	113	0.664	0.006	0.44	7.636	0.665	0.006	0.44	48.05
fft_a	0.116	0.015	0.82	111	0.248	0.016	1.08	7.317	0.191	0.015	0.97	42.78
fft_b	0.211	0.067	1.05	101	0.177	0.026	1.21	7.942	0.142	0.047	1.12	46.81
matrix_mult_1	0.156	0.057	2.57	297	0.165	0.340	2.19	13.69	0.168	0.334	2.19	52.9
matrix_mult_2	0.210	0.073	2.41	344	0.253	0.238	2.28	13.69	0.251	0.242	2.28	52.25
matrix_mult_a	0.017	0.028	3.65	374	0.145	0.113	5.45	15.30	0.020	0.024	3.49	47.91
matrix_mult_b	0.032	0.035	3.67	307	0.044	0.025	4.51	14.91	0.020	0.028	3.47	50.5
matrix_mult_c	48.956	29.719	126.71	2674	0.089	0.017	4.87	14.38	0.029	0.016	3.42	48.41
pci_bridge32_a	0.110	0.056	0.54	121	0.192	0.098	0.52	7.33	0.076	0.036	0.43	44.64
pci_bridge32_b	0.001	0.004	0.77	95	0.001	0.005	0.83	8.08	0.002	0.008	0.65	43.72
superblue12	0.034	0.495	46.70	10813	0.125	0.374	38.1	96.18	0.131	0.379	36.46	547.8
superblue14	0.064	0.056	29.50	7010	0.041	0.051	26.1	54.26	0.055	0.081	25.28	168.35
superblue16_a	0.186	0.031	33.40	7068	0.090	0.013	28.2	54.92	0.164	0.028	28.70	170.21
superblue19	0.022	0.089	20.50	7890	0.033	0.093	17.0	42.23	0.039	0.091	16.70	97.84
Average	0.131	**0.069**	8.94	2102.82	0.141	0.091	7.68	**22.28**	**0.124**	0.087	**7.27**	91.13

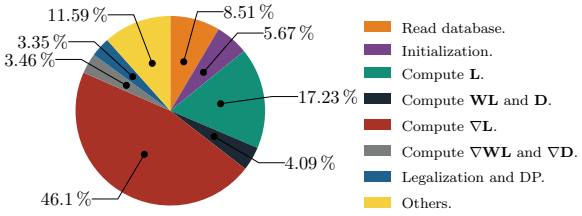


Fig. 5 The runtime breakdown on ISPD2015 mgc_des_perf_1.

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 $\nabla \mathbf{L}$, 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 $\mathbf{L}$, 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 have 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.

References

[1] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in VLSI placement research," *Proceedings of the IEEE*, 2015.

[2] B. Hu and M. Marek-Sadowska, "Fine granularity clustering-based placement," *IEEE TCAD*, april 2004.

[3] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang, "NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs," in *Proc. ISPD*, 2005.

[4] M.-C. Kim, D.-J. Lee, and I. L. Markov, "SimPL: An effective placement algorithm," *IEEE TCAD*, 2012.

[5] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics-based placement using fast fourier transform and nesterov's method," *ACM TODAES*, 2015.

[6] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, Y.-W. Chang, T.-C. Chen, and I. Bustany, "NTUplace4dr: a detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE TCAD*, 2017.

[7] C. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, 2019.

[8] M. Kim, J. Hu, D. Lee, and I. L. Markov, "A simplr method for routability-driven placement," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011.

[9] X. He, T. Huang, W.-K. Chow, J. Kuang, K.-C. Lam, W. Cai, and E. F. Y. Young, "Ripple 2.0: High quality routability-driven placement via global router integration," in *Proc. DAC*, 2013.

[10] 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 *Proc. ICCAD*, 2018.

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

[12] 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 *Proc. DAC*, 2019.

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

[14] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu, "DRC hotspot prediction at sub-10nm process nodes using customized convolutional network," in *Proc. ISPD*, 2020.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[16] 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.

[17] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, and D. Z. Pan, "High-definition routing congestion prediction for large-scale FPGAs," in *Proc. ASPDAC*, 2020.