Congestion-aware Global Routing using Deep Convolutional Generative Adversarial Networks

Zhonghua Zhou*, Ziran Zhu†, Jianli Chen†, Yuzhe Ma‡, Bei Yu‡, Tsung-Yi Ho§, Guy Lemieux*, Andre Ivanov*
*Department of Electrical and Computer Engineering, The University of British Columbia
†Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong
§Department of Computer Science, National Tsing-Hua Uniersity

***Abstract*—The routing stage is one of the most time-consuming steps in System on Chip (SoC) physical design. For large designs, it can take days of effort to find a complete routing solution, and the result directly affects the circuit performance. In this paper, we present a routing strategy that decomposes global routing into three stages, with different objectives associated with each stage. This is in contrast to conventional approaches, which usually use a single global optimization objective for driving the entire process. Furthermore, we propose to use generative adversarial networks (GAN) to predict the congestion heatmap. This deep learning method has been used to successfully improve image recognition results. We adapt its use to global routing by converting data between the router and the image-based model. This model needs only placement and netlist information as input to make the forecast. Our GAN-based congestion estimator produces congestion heatmaps that show good fidelity with actual heatmaps produced by state-of-the-art global routers. Using this heatmap along with our modified routing flow, we achieve comparable global routing quality in terms of the total overflow and wirelength, but the runtime speedup on hard-to-route designs is significant.**

I. Introduction

As integrated circuit fabrication technologies evolve, design rules provided by manufacturers continue to increase in number and complexity in order to secure viable fabrications. These requirements are becoming increasingly difficult to meet, which in turn, makes physical design typically among the most time-consuming stages in electrical design automation (EDA) flows. To solve such challenges, predictive modeling during design flow and fast routing algorithms have garnered much attention from both academia and industry.

Global-routing-based congestion estimation is the most widely adopted predictive modeling algorithm in routers. In [1]–[5], simplified global routers are applied to quickly generate a rough congestion map, while probabilistic congestion estimation algorithms [6], [7] constitute alternative, faster approaches. These somewhat conventional approaches suffer from scalability limitations as advanced technology nodes bring new sets of design rules that are not considered by these latter algorithms. In recent years, machine learning techniques have been widely applied for finding accurate and fast routability estimations. In [8]–[11], simple supervised-learning algorithms are applied to make Design Rule Checking (DRC) violation prediction models. Xie *et al.* [12] proposed a congestion estimation algorithm using Fully Convolutional Networks (FCNs). However, the prediction outputs of these works cannot be encapsulated back into the global routing algorithms, as the DRC binary output (violation or not) do not correlate strongly with actual congestion levels. As a result, global routing algorithms gain no or only little benefit from these classification-like estimations.

Many contests [13]–[15] focused on global routing have been held in recent years, and various routing algorithms [5], [16], [17] have been proposed. Most of the routing algorithms follow a single or double stage approach. These iterative algorithms use similar strategies and aim to reduce congestion and wirelength simultaneously. As the number of iterations grows, such methods tend to inevitably end up in local optima [18] and thereby fail to yield globally acceptable/optimal solutions.

In this paper, our focus is on providing solutions to three problems: 1) We propose a new multi-objective global routing algorithm aimed at reducing the likelihood of converging on local optima during the iterative rip-up & reroute process; 2) We develop a scalable and accurate congestion estimation methodology which learns from physical properties of actual routed designs, thus learning from router decisions associated with these routed designs; and 3) We design a "bridge" algorithm to adapt the image-based data used by deep-learning generators with the placement-oriented and netlist-oriented data used by a global router. To the best of our knowledge, this is the first work that uses a Generative Adversarial Network (GAN) network for routability-driven global routing. To integrate GANs into the global routing process, we propose an image encoding solution, discussed in Section III, which interprets routing data as image patterns.

Our main contributions are summarized as follows:

- We present a multi-objective global routing algorithm improvement to NTHU-Route 2.0 [17], which decomposes the main routing stage into sub-phases with independent cost functions and objectives.
- We develop a content-encoder DCGAN (c-DCGAN) to predict accurate congestion before actual routing such that placements can be identified as routable or not at an early stage.
- We propose a data to image bi-directional translator which can be applied to any router allowing the direct use of our c-DCGAN regardless of the EDA tools and design formats.

The rest of this paper is organized as follows. Section II discusses the basic concepts. Section III explains the proposed frameworks of c-DCGAN and our multi-objective algorithm. Section IV presents experimental results, followed by conclusions in Section V.

II. Preliminaries

In this section, we discuss the concepts of Generative Adversarial Networks (GAN) and some elements of global routing for SoCs.

A. GAN

GANs [19] have shown good results in many fields that require data creation and predictions, but have not been studied for general applications in SoC physical design, particularly for routing congestion estimation. Our work here constitutes an exploration of the potential for GANs to alleviate the SoC global routing process. A GAN, as demonstrated in Fig. 1, trains two networks

978-1-7281-5758-0/19/$31.00

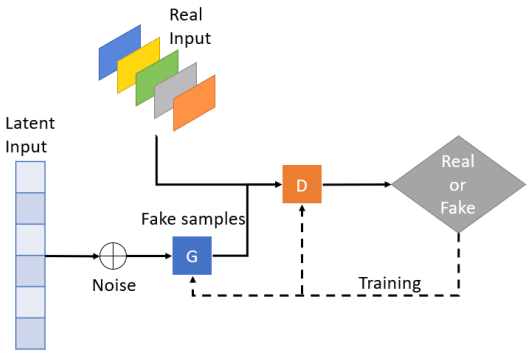


Fig. 1 General structure of GAN.

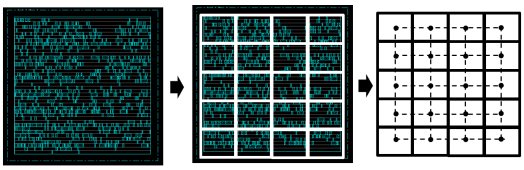


Fig. 2 From design to tiles to routing grid.

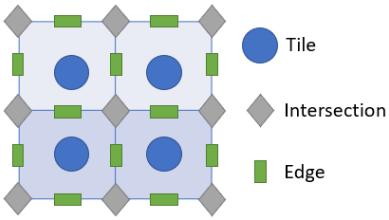


Fig. 3 A 2×2 routing grid.

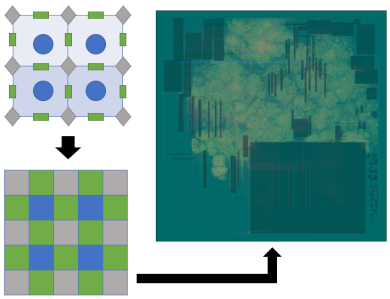


Fig. 4 Data-Image translation.

simultaneously: A **generator** model *Gen* and a **discriminator** model *D*. These two models compete with each other in such a way that *Gen* tries to generate samples which are as close to the real ones as possible, in order to fool *D*. Meanwhile, *D* classifies if a sample is from the real training dataset rather than a fake one produced by *Gen*, where the fake input is generated from random value called "noise". At its convergence, the **generator** is expected to produce outputs with a feature distribution of high similarity to the real dataset. Ultimately, the **generator** will be used to produce congestion estimates for our router.

B. Routing Grid

A routing grid is the logical graph representation of the physical SoC. In SoC routing methodologies, the grid graph G is partitioned into multiple uniform sub-regions, commonly referred to as vertices, as illustrated in Fig. 2. The tiles are crossed by one vertical and one horizontal gridline, along which routing channels can be created. Each vertex in G corresponds to a grid tile, and each edge in G between vertices represents the shared boundary of two tiles. The graph G used for global routing needs to capture the capacities of the routing regions. The *capacity* of an edge $e \in G$ between two vertices u and v is defined as the maximum number of available routing channels between the routing regions of u and v. The *usage* of an edge e defines the number of nets that have crossed this edge. If the total usage of an edge e is larger than its *capacity*, then the tile containing that edge is considered to be congested, and the amount of overhead is referred to as *overflow*, $of(e) = \text{usage}(e) - \text{capacity}(e)$.

III. Methodology

A. Data-Image Translator (DIT)

Among multiple existing machine learning techniques discussed in Section I, the work [12] uses circuit layout images generated by commercial tools as inputs to the learning model, which later outputs images as well. However, this approach leaves one unresolved problem, i.e., the relationship of routability information to the image characteristics. Therefore, here we developed a data to image translator (DIT) to relate circuit images to routing parameters.

Although different routing tools, either commercial or academic, take in various kind of design formats, such as "bookshelf" or "LEF/DEF", they essentially use the same graph structure discussed in Section II-B to store data. Therefore, we developed a translator which translates a logical routing grid to an image while preserving the design layout structure. As shown in Fig. 3, a 2×2 tile grid consists of three kinds of information: 4 tiles, 9 edge intersections, and 12 edges. We first ignore the tile width and height properties, which shrinks each tile into one single pixel. Following the same philosophy, edge length is ignored and each one of them corresponds to one pixel as well. The intersection contains no useful data, but is left in the image as gray "filler pixels" to preserve the spatial layout of the other information. With this mapping, the information surrounding each tile is drawn as a group of 2×2 pixels in Fig. 4 (drawn as one gray, two green, and one blue pixel). The actual data encoding used is described in Section III-B.

B. Data Encoding and Labeling

The proposed c-DCGAN provides guidance for the global router. Therefore, the final global routing solution is selected as the training target. For input features, our network only needs a minimum amount of netlist information: (1) pin density, (2) routing channel capacities, and (3) net density. The "pin density" is obtained after all pins' locations are identified. The routing channel capacities reflect edge capacities but are affected by the placement of macros. Net density is computed using Rectangular Uniform wire DensitY (RUDY) [20]. Visualizations of input features and output target are shown in Fig. 5.

Each image pixel in Fig. 4 has circuit information encoded into its RGB channels as follows. First, the red channel is to hold the ground truth of actual congestion values. This red channel is forced to zero for input images, while the estimated congestion appears here in the output data. In the 2×2 block of pixels, the green and blue channels are used to encode information about pin density, netlist density, and routing channel capacities as follows:

$image[2x][2y].g = image[2x][2y].b = 0$

$image[2x][2y+1].g = Vcapacity[x][y] + Vnetdensity[x][y]$

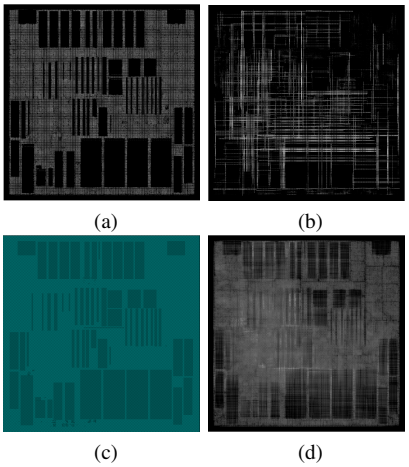


Fig. 5 Raw data for (a) Pin density; (b) Net density; (c) Routing channel capacity; and (d) Congestion heatmap.

$image[2x][2y+1].b = Vcapacity[x][y]$
$image[2x+1][2y].g = Hcapacity[x][y] + Hnetdensity[x][y]$
$image[2x+1][2y].b = Hcapacity[x][y]$
$image[2x+1][2y+1].g = Hnetdensity[x][y] + Vnetdensity[x][y]$
$image[2x+1][2y+1].b = pindensity[x][y]$

This colorization encoding is designed so no feature can directly interfere with the target congestion in the red channel. This also allows us to quickly distinguish and extract the congestion map from the final output image and export it back into the router.

C. Content-Aware Deep Convolutional GAN

Using the translator and encoding presented in Section III-A and Section III-B, we can generate design images that are independent of design tools and file formats. These images contain enough information for a network to perform congestion analysis. The prediction outputs of the network are also images that can be translated back and provided as input to the global routing process. In our case, *D* is only used during the model training process, and the final product used for congestion estimation is *G*.

D. Generator Design

The generator *Gen* learns the feature distribution of a dataset, which can be described using a mapping function:

$$Gen(z) : p_z \rightarrow p_g \tag{1}$$

where p_z is the feature distribution of an input data set z, which can be assumed to be a superposition of noise and latent data. p_g is the feature distribution of a real dataset. The generator's task is to maximize the log-likelihood that the discriminator flags fake samples as real, as is shown in Equation (2).

$$\max \mathbb{E}_Z[\log(D(Gen(z)))] \tag{2}$$

In our application, the generator receives, as input, 2D design placement & routing layout images. The output congestion prediction will be made by the *Gen* on top of the inputs.

Inspired by [21], [22], we developed a generator comprised of a stack of convolutional neural net layers, known as encoder, and a stack of transpose layers, known as decoder, as shown in Fig. 6. The encoder transforms the input into a latent space; the meaning of this space is created by the generative model, i.e. it does not correspond to anything of our own design. The decoder will then interprets and reconstructs an output from the latent space. In our specific application, the encoder extracts key features from input designs, and these features are used by the decoder perform the congestion estimation.

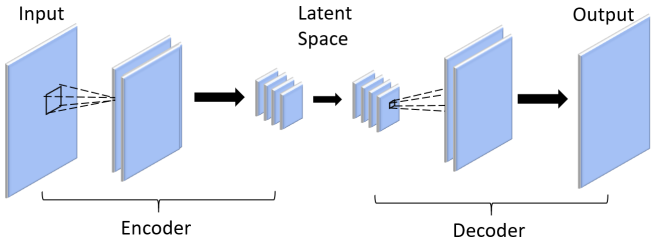


Fig. 6 Proposed generator model.

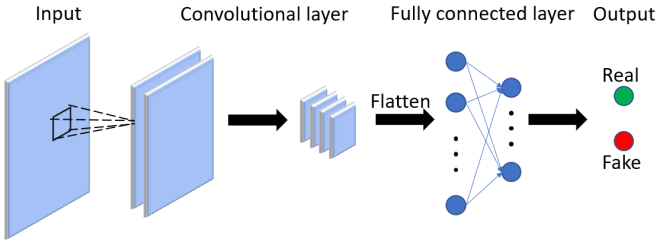


Fig. 7 Proposed discriminator model.

E. Discriminator Design

The discriminator *D* is constituted by CNNs with multiple convolution layers and one fully connected layer. The value function for *D* is shown in Equation (3):

$$\max_D V(D, Gen) = \mathbb{E}_x[logD(x)] + \mathbb{E}_z[\log(1 - D(Gen(z)))] \tag{3}$$

where x is the real dataset, and z is the superposition of noise and latent input for *Gen*. The objective of *D* is to maximize the value of $V(D, Gen)$, which is to maximize $\log D(x)$ and $D(Gen(z))$.

In this work, *D* classifies whether the congestion heatmap is the real output from the router or a fake one generated from *Gen*. The discriminator's structure is illustrated in Fig. 7. Batch normalization [23] and dropout [24] techniques are applied to each layer, for a faster convergence and prevent the model from over-fitting.

F. Model pre-process & Training

In our application and the details of pre-processing are listed in Algorithm 1.

In our case, circuits we wish to analyze contain up to approximately 400×400 tiles. For training and estimation purposes, we segment each design into multiple non-overlapping partitions of 32×32 tiles. Each partition is used to produce a 64×64 pixel image subsequently used as input for training. The tile size directly affects the prediction model size, memory requirements and, ultimately, prediction quality. Ideally the larger the input size, the better the model performance. However, in practice, open-source benchmarks are very limited in quantity. which restricted the growth of input size. Because if the size of input is enlarged, the number of inputs will be reduced, affecting the training convergence of the model. From our experiments, tile sizes of 32×32 yielded the best results for the trade-off. We plan to investigate these trade-offs further. The generator estimates a single output image for each input image. When estimating congestion for an entire circuit, we stitch together multiple output images back into the original circuit grid size.

Algorithm 1 Model Training Pre-processing

Require: Placed netlist.

```
pin_den, net_den, macro, congestion = Init_array();
while net ← nets.read() do
    net_den[net.position] = RUDY(net);
    while pin ← net.getNextPin() do
        pin_den[pin.position] += 1;
    end while
end while
patternRoute();
while edge ← grid.read() do
    congestion = edge.current_cap;
    macro = edge.max_cap;
end while
dataImageTranslator({pin_den, net_den, macro}) ;
Features: X = {pin_den, net_den, macro};
Target: Y = {congestion};
Resize: X, Y ∈ R^{W×H×1} to X_n, Y_n ∈ R^{64×64×1};
Combine: X_n, Y_n ∈ R^{64×64×1} to X_n, Y_n ∈ R^{64×64×3};
```

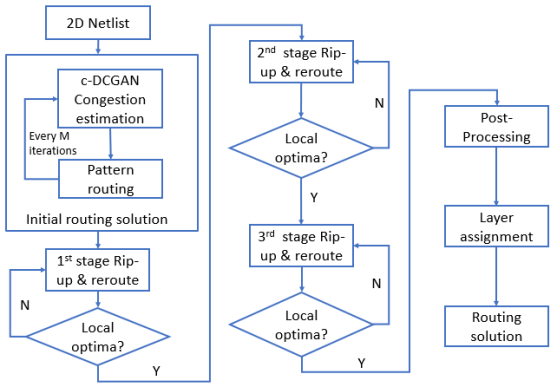


Fig. 8 Proposed multi-objective routing.

Based on the c-DCGAN framework, the final objective function of the entire network becomes:

$$\min_G \max_D V(D,G) = \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))] \quad (4)$$

As just described, the purpose of the discriminator training is to maximize the probability of correctly labeling samples, while that of generator training is to minimize the log-likelihood of $1-D(G(z))$. Eventually, a generator becomes capable of generating output that the discriminator cannot distinguish from actual/real data; i.e, $D(G(z)) \rightarrow 1$. Such a generator is considered to have "converged" and is saved for prediction use.

G. Multi-objective Global Routing

The flowchart of our proposed algorithm is shown in Fig. 8. After the design netlist is read by the router, the first estimated congestion heatmap using our c-DCGAN (discussed in Section III-C) is obtained. After every *M* routing iterations in the pattern routing, c-DCGAN can be repeatedly called to generated a new congestion heatmap prediction, using the current incremental routing information. The congestion estimation directly affects the quality of the initial routing. In turn, this impacts our multi-objective rip-up & reroute optimization.

Our work decomposes the rip-up & reroute into three stages. Each with different cost functions, routing orders and routing algorithms with the aim of avoiding unnecessary searches and accelerating the overall routing process.

1) 1st Stage: The first stage of rip-up & reroute can be considered as a pre-processing stage. In this stage, the routing process is restricted to within the bounding box of the net. As a result, we can reduce the *overflow* without increasing *wirelength*. We define the cost function for calculating the price of crossing an edge as:

$$cost_1 = 1 + \frac{\rho}{1 + e^{-\varepsilon * \gamma}} + c_{via} \quad (5)$$

where ρ and ε are user defined parameters that determine the slope of the cost with respect to the congestion. They are empirically set as 0.8 and 2 in this work. Meanwhile, γ is the *overflow* value and c_{via} is the cost of a via, which is 1 if routing direction changed, 0 otherwise. When the *overflow* reduction is less than 2% in current iteration, the algorithm changes to the second stage.

2) 2nd Stage: A new cost function is applied in this 2nd stage:

$$cost_2 = 1 - e^{-\beta e^{-\gamma i}} + c_{history} \times c_{penalty} + c_{via} \quad (6)$$

where values of β and γ are user defined, i is the current rip-up & reroute iteration count, and $c_{history}$ has an initial value of 1 and is increased by 1 if that edge has overflow. The penalty $c_{penalty}$ is a user-defined scaling parameter. If the overflow falls under a threshold of 20 overflows, or has no improvement after 60 iterations, the third stage will be entered.

3) 3rd Stage: In the 3rd stage, congested areas become the focus of attention and are referred to "forbidden regions" [25]. The router is discouraged from using "forbidden regions" as these areas are already too congested and considering longer detours is likely to yield better final routing convergence. The cost function is:

$$cost_3 = 1 - e^{-\beta e^{-\gamma i}} + c_{via} + c_f \quad (7)$$

where c_f is related to the "forbidden regions" and is defined as:

$$c_f = \begin{cases} C_f \times (c_{demand}/c_{capacity}), & \text{in "forbidden areas";} \\ c_{history} \times c_{penalty} & \text{otherwise.} \end{cases} \quad (8)$$

Here C_f is a constant, which reflects the level of deterrence for the router to attempt routing a net through forbidden areas. The new $c_{penalty}$ is defined as follows:

$$c_{penalty} = \begin{cases} 1/(c_{capacity} - c_{demand}), & c_{capacity} > c_{demand}; \\ \xi + c_{demand}/c_{capacity}, & c_{capacity} < c_{demand}; \\ \xi, & c_{capacity} = c_{demand}. \end{cases} \quad (9)$$

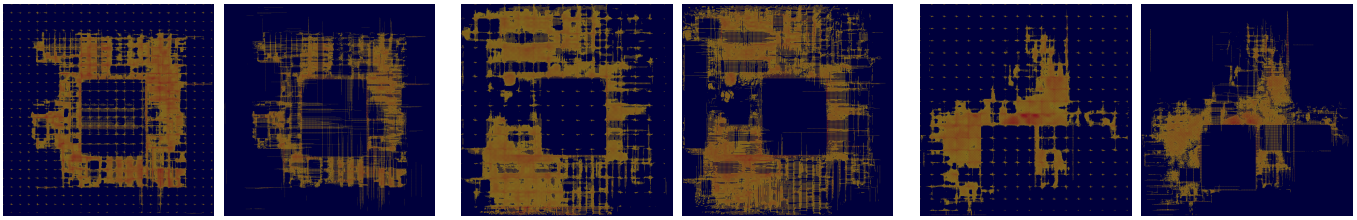
where ξ is a user defined parameter.

4) Post-Processing: After three stages of routing, some edges become less congested, but the $c_{history}$ term increases and may become a predominant term in the cost function. As a result, these edges are blocked from being valid routing candidates again. In order to solve this issue, we remove the $c_{history}$ term from the cost function such that the router can re-evaluate the grid and most edges are released for reuse after multiple optimization iterations.

IV. Experimental Results

A. Congestion estimation quality

The baseline router on which we implemented our algorithms is NTHU-Route 2.0 [17]. We compare the results of our work with other available academic routers, NTHU-Route 2.0, NTUgr2 [25], and NCTU-gr 2.0 [16]. We implemented the c-DCGAN for congestion heatmap estimation using `Keras` with `TensorFlow` backend. We trained using 4619 placement instances, each using a



(a) a3 predicted (b) a3 real (c) a5 predicted (d) a5 real (e) b3 predicted (f) b3 real

Fig. 9 Predicted congestion heatmaps versus actual congestion heatmaps.

TABLE I Congestion Estimation Quality Metrics

Design	PCC	mean ‖normalized-error‖	Std. deviation of normalized-error
bigblue1 (b1)	0.9544	0.0327	0.0672
newblue1 (n1)	0.9333	0.0250	0.0600
newblue6 (n6)	0.9288	0.0047	0.0463
adaptec1 (a1)	0.9270	0.0447	0.0751
newblue2 (n2)	0.9266	0.0124	0.0468
bigblue4 (b4)	0.9157	0.0142	0.0566
adaptec5 (a5)	0.9129	0.0014	0.0468
newblue4 (n4)	0.9040	0.0318	0.0609
bigblue2 (b2)	0.9038	0.0028	0.0456
adaptec3 (a3)	0.9026	0.0202	0.0519
newblue5 (n5)	0.8996	0.0201	0.0496
bigblue3 (b3)	0.8843	0.0110	0.0428
adaptec4 (a4)	0.8644	0.0108	0.0377
adaptec2 (a2)	0.8453	0.0268	0.0620
newblue3 (n3)	0.1808	0.0227	0.0632

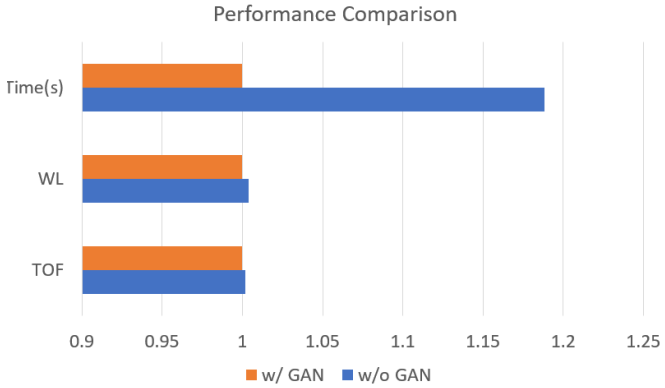


Fig. 10 Global routing improvement using c-DCGAN estimator.

64×64 image grid with 3 image channels. The model is trained on a single Nvidia 1080 Ti GPU. The router algorithm is implemented in C++ and tested on our server with Intel 2.2GHz CPUs. We trained the model for 300 epochs, which took approximately 8 hours. The dataset was randomly split into two groups, 80% for training and 20% for evaluation. After the training, the model is saved onto the disk, and loaded into the router to make predictions.

Visual correlation is shown in Fig. 9. For a more quantitative comparison, we introduce the use of the Pearson Correlation Coefficient (PCC) which ranges from -1 (absence of correlation) to 1.0 (perfect correlation). This value is computed by comparing the congestion of each edge (estimated vs actual).

The accuracy of congestion prediction is shown in TABLE I. The second column shows the PPC. The third column of the table shows the mean absolute normalized-error in congestion is at most 4.47%, while the fourth column shows the standard deviation in the normalized-error. Note that "n3" has an unusually low PCC score because "n3" is a synthetic design purposely created to challenge routers. The high PPC score in all other circuits shows that our algorithm strongly predicts congestion for all legal designs in the benchmark suite.

B. Routing quality

To show the c-DCGAN is useful, we first used the original estimator used in NTHU-Route 2.0 in our modified router across the entire benchmark suite. Then, we used the c-DCGAN estimator and show normalized total results in Fig. 10. There is a small improvement in both total wirelength and total TOF, but a large improvement in total routing runtime. The start point of the runtime counting is the beginning of the first stage of rip-up & reroute, and the endpoint is the end of the post-processing. The 18.8% total runtime improvement is from a reduced effort by the global router, not from a faster estimate computation.

Routing performance of all designs is shown in TABLE II. TOF is the *total overflow* number, WL is the *wirelength* of the final routing solution, and T is the runtime for the complete global routing process (in seconds). Note that NTHU-Route 2.0 did not finish routing "n3" within a 24-hour time-frame.

Compared with NTHU-Route 2.0, our version successfully routes three more circuits (a2, n5, and b3) while also reducing TOF for the remaining unroutable circuits. In addition, the hardest-to-route circuit n3) completes at least 8 times faster at the lowest TOF among all routers. Overall, our approach produces excellent TOF results, beating NTUgr2 in 3 of 5 unroutable circuits and beating NCTU-gr in 4 of 5 unroutable circuits. The Total TOF is reduced by 21% comparing with NCTU-gr 2.0, while our TOF is 0.22% higher than NTUGR2. The total *wirelength* is reduced by 7%, and 1% compared to NTUgr2 and NCTU-gr 2.0, respectively.

We integrated our approach into NTHU-Route because it is open source. We can't verify whether our congestion estimation will also benefit NTUgr2 or NCTR-gr because they are not open source.

More importantly, our total runtime is 11.97× and 3.44× faster than NTUgr2 and NCTU-gr 2.0, respectively. Excluding n3, our total runtime is 1.8× faster than NTHU-Route 2.0. Most of the runtime speedup comes from the efficiency of the multi-stage routing algorithm. As shown earlier, about 19% of this speedup comes from the improved congestion estimation model.

V. Conclusions

We proposed a new routing algorithm along with a c-DCGAN framework which significantly improves run-time while maintaining competitive results in routing quality. The multi-objective global routing algorithm is able to successfully route all of the same designs as the top prior global routers. In addition, we are able to

TABLE II Comparison with State-of-the-art Global Routers

	NTUgr2 [25]			NCTU-gr [16]			NTHU-Route 2.0 [17]				Ours			
	TOF	WL	T(s)	TOF	WL	T(s)	TOF	WL	T(s)	T Ratio	TOF	WL	T(s)	T Ratio
a1	0	5.60	177.2	0	5.44	102.53	0	5.36	207.11	**1.00**	0	5.38	207.37	**1.00**
a3	0	13.41	157.7	0	13.11	154	0	13.15	225.84	**0.86**	0	13.10	263.89	1.00
a4	0	12.29	59.2	0	12.19	63.6	0	12.17	56.11	**0.73**	0	12.23	77.29	1.00
a5	0	16.03	520.4	0	15.95	381.71	0	15.53	549.98	**0.90**	0	15.64	611.39	1.00
b1	0	5.85	428.4	0	5.97	204.32	0	5.57	406.78	1.44	0	5.60	283.41	**1.00**
n2	0	7.66	27.6	0	7.59	35.73	0	7.59	30.82	1.01	0	7.59	30.65	**1.00**
n6	0	18.55	487.3	0	18.27	238.72	0	17.69	968.39	2.20	0	17.67	439.83	**1.00**
a2	0	5.36	39.8	0	5.27	36.02	2	5.23	93.4	1.82	0	5.24	51.23	**1.00**
n5	0	23.90	1,220	0	23.46	281.47	18	23.14	721.52	1.81	0	23.21	399.4	**1.00**
b3	0	13.47	206.0	0	13.17	99.4	32	13.07	307.45	2.43	0	13.10	126.38	**1.00**
b2	2	9.42	6,617	4	9.10	171.35	84	9.00	400.29	2.12	8	9.01	189.17	**1.00**
n1	38	4.87	14,339	108	4.70	120.39	144	4.60	483.1	3.45	18	4.63	140.05	**1.00**
n4	148	13.55	16,327	172	13.00	158.86	242	12.88	1,033	2.30	172	12.90	449.65	**1.00**
b4	212	23.96	4,479	512	23.17	277.64	266	22.78	2,146	2.31	160	22.74	930.7	**1.00**
n3	31,136	17.96	36,326	37,182	10.80	21,053	—	—	>24 hrs	—	31,050	10.70	2,604	—
Total	31,536	191.90	81,411	37,978	181.19	23,379	788	167.78	>86,400		31,606	178.74	6,804	
Ratio	1.01	1.07	11.97	1.21	1.02	3.44	—	—	—	1.82	**1.00**	**1.00**	**1.00**	

maintain total overflow (TOF) and wirelength (WL) results while making run-time by up to 12× faster. The deep-learning congestion estimation framework generates accurate congestion heatmaps from design layout information, i.e, pins, nets and macros. The quality of this estimation helps the router achieve some of its run-time performance gains. Overall, we believe this is the first time neural networks have been integrated into the algorithmic flow of global routers. As future work, we would like to continue to improve our estimation quality, routing quality, as well as investigate the precise way in which our router is able to achieve improved run-time while maintaining quality – we believe it has a better ability to escape local optima using the improved congestion information.

REFERENCES

[1] M.-K. Hsu, S. Chou, T.-H. Lin, and Y.-W. Chang, "Routability-driven analytical placement for mixed-size circuit designs," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov 2011, pp. 80–84.

[2] X. He, T. Huang, L. Xiao, H. Tian, G. Cui, and E. F. Y. Young, "Ripple: An effective routability-driven placer by iterative cell movement," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov 2011, pp. 74–79.

[3] M.-C. Kim, J. Hu, D.-J. Lee, and I. L. Markov, "A SimPLR method for routability-driven placement," in *IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 67–73.

[4] W. Liu, Y. Li, and C. Koh, "A fast maze-free routing congestion estimator with hybrid unilateral monotonic routing," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov 2012, pp. 713–719.

[5] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *Asia and South Pacific Design Automation Conference*, 2009, pp. 576–581.

[6] J. Lou, S. Krishnamoorthy, and H. S. Sheng, "Estimating routing congestion using probabilistic analysis," in *International Symposium on Physical Design*, 2001, pp. 112–117.

[7] J. Westra, C. Bartels, and P. Groeneveld, "Probabilistic congestion prediction," in *International Symposium on Physical Design*, 2004, pp. 204–209.

[8] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *International Symposium on Physical Design*, 2017, pp. 15–21.

[9] A. F. Tabrizi, N. K. Darav, S. Xu, L. Rakai, I. Bustany, A. Kennings, and L. Behjat, "A machine learning framework to identify detailed routing short violations from a placed netlist," in *Design Automation Conference*, 2018, pp. 48:1–48:6.

[10] L. Chen, C. Huang, Y. Chang, and H. Chen, "A learning-based methodology for routability prediction in placement," in *International Symposium on VLSI Design, Automation and Test*, Apr 2018, pp. 1–4.

[11] Z. Zhou, S. Chahal, T. Ho, and A. Ivanov, "Supervised-learning congestion predictor for routability-driven global routing," in *International Symposium on VLSI Design, Automation and Test*. IEEE, 2019, pp. 1–4.

[12] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and Nvidia, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *International Conference on Computer-Aided Design*, 2018, pp. 80:1–80:8.

[13] (2008) Ispd 2008 global routing contest. [Online]. Available: http://www.ispd.cc/contests/08/ispd08rc.html

[14] V. Yutsis, I. S. Bustany, D. Chinnery, J. R. Shinnerl, and W.-H. Liu, "ISPD 2014 benchmarks with sub-45nm technology rules for detailed-routing-driven placement," in *International Symposium on Physical Design*, 2014, pp. 161–168.

[15] 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 *International Symposium on Physical Design*, 2015, pp. 157–164.

[16] W. Liu, W. Kao, Y. Li, and K. Chao, "Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 709–722, May 2013.

[17] Y. Chang, Y. Lee, and T. Wang, "Nthu-route 2.0: A fast and stable global router," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov 2008, pp. 338–343.

[18] M. M. Ozdal and M. D. F. Wong, "Archer: A history-based global routing algorithm," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 4, pp. 528–540, April 2009.

[19] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Neural Information Processing Systems Vol. 2*, 2014, pp. 2672–2680.

[20] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Design, Automation Test in Europe*, April 2007, pp. 1–6.

[21] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Y. Young, "Gan-opc: Mask optimization with lithography-guided generative adversarial nets," in *Design Automation Conference*, 2018, pp. 131:1–131:6.

[22] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Artificial Neural Networks - Volume Part I*, 2011, pp. 52–59.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning Vol. 37*, 2015, pp. 448–456.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[25] H. Chen, C. Hsu, and Y. Chang, "High-performance global routing with fast overflow reduction," in *Asia and South Pacific Design Automation Conference*, Jan 2009, pp. 582–587.