Machine Learning Framework for Early Routability Prediction with Artificial Netlist Generator

Daeyeon Kim[1], Hyunjeong Kwon[1], Sung-Yun Lee[1], Seungwon Kim[2], Mingyu Woo[3] and Seokhyeong Kang[1*]
[1]EE Department, POSTECH, Pohang, South Korea
[2]CSE and [3]ECE Department, UC San Diego, La Jolla, CA, USA
[*]shkang@postech.ac.kr

Abstract—**Recent routability research has exploited a machine learning (ML)-based modeling methodologies to consider various routability factors that are derived from placement solution. These factors are very related to the circuit characteristics (e.g., pin density, routing congestion, demand of routing resources, etc), and lack of circuit benchmarks in training can lead to poor predictability for 'unseen' circuit designs. In this paper, we propose a machine learning (ML) framework for early routability prediction modeling. The method includes a new *artificial netlist generator* (ANG) that generates an artificial gate-level netlist from the user-specified topology characteristics of synthetic circuit, even with real world circuit-like. In this framework, we exploit that ANG that supports obtaining ground truths for use in training ML-based model, the training dataset that have a wide range of topological characteristics provides strong ability to inference noisy, previous-unseen data. Compared to a design-*specific* training dataset [4] that is used for routability prediction modeling, we increase the test accuracy of binary classification ('pass' or 'fail') on timing, DRC and routability by 6.3%, 8.6% and 6.6%, and reduce the generalization error [12] by as much as 87% compared to design-specific training dataset [4].**

I. Introduction

Advanced-node VLSI design entails significantly complicated and tightened design-rules, so the required quality of result is increasingly difficult to meet. At the beginning of physical design process, designer needs to set up the numerous design parameters such as layout utilization, aspect ratio, the number of metal layers in the back-end-of-line (BEOL) stack, and place and route (P&R) clock period. On the other hand, the quality of layout design can be evaluated at the end of physical design process so the search for the best combination of design parameters is a time-consuming task. To reduce the turn-around time of physical design iterations, early prediction of routability (as measured by the number of design rule violations (DRVs) and timing-path violations after the routing stage) has become an essential procedure.

Many previous works proposed various methods for routability prediction using parameters that are derived from the circuit-level [1]–[3] or layout-level [4]–[7]. Those parameters are called as *routability factors*. Unfortunately, the effects of the numerous routability factors are difficult to assess because they are strongly related to each other and no exact model to define their relationship exists yet. Even P&R CAD tool is a kind of black-box, and this status complicates prediction of their behavior and results. Since these problems make the analytical approach difficult, so modeling approaches that use machine learning (ML) have been used in recent EDA research [8].

For accurate ML-based routability modeling, data aggregation from various circuits is very important since the routability is highly affected by circuit characteristics such as the demands of wirelength, routing congestion, pin density. In addition, a well-organized training dataset for ML decreases the sensitivity of model accuracy to perturbations of input [9], [10]. Many previous works that related to ML applications in EDA used only a few open-source circuit benchmarks or created new one by modifying an existing circuit. However, this approach is disadvantageous in terms of model scalability, as it requires designer effort and is difficult to obtain data on new circuit architectures.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (No. 2019R1A5A1027055).

In this paper, we propose an artificial netlist generator (ANG) that create an artificial gate-level netlist from the user-specified input parameters that represent the topological shape of circuit. This ANG enables to perform data preparation and exploration on routability factors related to circuit characteristics. We characterize the synthetic circuits according to the list of parameters that represent shape and properties, such as the size of circuit, interconnect complexity, timing delay. We demonstrate the data preparation and exploration to decrease the sensitivity to perturbation of circuit characteristics compared to design-specific training dataset [4]. Our main contributions are:

- We propose a new artificial netlist generator that can create a gate-level netlist from the user-specified input parameters. The input parameters characterize the topological characteristics of the synthetic circuit in terms of the number of instances, the number of primary I/Os, average net degree, average size of net bounding box (bbox) and average length of timing paths.
- We perform Rentian analysis [11] to show that the artificial netlist has a realistic topology compared to real-world synthetic circuits in terms of wirelength distribution, fanout distribution and Rent parameter.
- We propose a machine learning framework for early routability prediction. In this framework, we exploit the artificial netlist generator that supports obtaining ground-truths for use in training an ML-based routability prediction model. This framework can minimize wasted human effort when collecting synthetic circuit benchmarks, and also increase the reliability of the model.
- Compared to [4], we increased the test accuracy of timing, DRC and routability prediction by 6.3%, 8.6% and 6.6%, and reduce the generalization error [12] up to 87%.

II. Background

A. Related work

An early routability prediction tells us whether a placement solution with specific parameters (e.g., layout utilization, aspect ratio, BEOL stack, clock period, etc.) is routable or not in advance. Routability factors that imply whether a chip layout is over-designed are required to achieve accurate prediction [13]. Many previous works used the available features on the placement solution or the characteristics of the synthetic circuit as routability factors. This sub-section reviews routability modeling methods and routability factors in related works.

PDN-*pathfinding* [5] predict a near-optimal power delivery network (PDN) while considering routability and worst IR-drop when given layout utilization, VI density, and the number of instances. It uses IR drop and routability models to be able to predict a near-optimal PDN solution among all possible PDN candidates. The process uses an artificial design called mesh-like placement[1] to measure K_{th} [14] that represents the routability of a PDN solution. The authors use K_{th} parameter for routability modeling. [4] proposed an ML-based routability model that predicts whether a placement is routable. The authors extract features from the gcell grid, and those features indicate the local routing complexity. They acquire better predictability than

[1]This design is a placed benchmark where the nets are connected 'mesh-like'

using a congestion map. Yu-Hung et al. [7] propose a routability prediction model for macro placement optimization. They extract three types of feature maps that appear after macro placement; macro density map, pin density map, and connectivity map. The feature maps are converted to an image that is used as the input of the CNN model.

B. Limitations in related works

Various routability modeling parameters in previous works were extracted for several open-source circuit benchmarks and then used for model training. Since the routability factors are closely correlated with circuit characteristics, this data preparation approach has limitations in terms of scalability of model training. When a new circuit architecture is given as an input to the model, the routability factor obtained from the circuit is likely to be an outlier that has never seen during the training phase. In this case, we need to retrain the model using a number of data points related to the unseen circuit, but the data preparation method in previous works is a too time-consuming task and needs too much effort by designer. To the best of our knowledge, no previous work addresses a way to organize training dataset under consideration on this problem.

C. Our approach

An efficient method to obtain training data considering scalability and predictability of model is essential. Our approach is to use a method of artificial circuit generation that can consider the topological characteristic of circuit. If the artificial circuit has a realistic topology shape of circuit enough to use as training benchmark, then we can improve the prediction accuracy for real-world circuits with only the training data obtained from the artificial circuits. In addition, it is possible to create a large amount of training data without the need of a designer, and it is also relatively easy to generate retraining data for outliers.

Many random circuit-generation methods have been proposed for various purposes. [15] and [16] are benchmarks for evaluation of the sub-optimality of a gate-sizing algorithm. [14] is used to measure a routability factor K_{th}, but their shape such as 'mesh-like' or 'chain-like' are unrealistic, because their Rent parameter is always fixed as 0.5 and 0.2, respectively, whereas Rent parameters in real-world circuits range from 0.7 to 0.9. [17]–[19] were proposed for testing of FPGA architecture. Although the both critical-path delay and total wirelength after P&R are controllable, the focus of these methods is interest to test FPGA architecture and algorithms, not ASIC.

III. Machine Learning Framework for Early Routability Prediction Model

In this section, we introduce a machine learning framework (Fig. 1) for early routability prediction. The framework consists of training data generation, feature extraction and modeling.

A. Training data generation

We introduce a method to obtain numerous P&R tool results (they are used for training dataset) from various synthetic circuits without any use of real-world circuit designs. As we mentioned above, the training dataset covering various circuit characteristics can increase the predictive accuracy on unseen real-world circuit designs. In this process, we use the proposed ANG to obtain various synthetic circuits (details in Section IV). First, we use several parameters to characterize the topological characteristics of the synthetic circuit by the following parameters.

- ***Size of circuits:*** the number of instances (N_{inst}) and the total number of primary inputs and outputs (N_{prim})
- ***Interconnect complexity and demands of wirelength:*** average net degree (D_{avg}) and average size of net bounding box (B_{avg})
- ***Timing delay:*** average length of timing paths (T_{avg})

A gate-level netlist (synthetic circuit) consists of a list of standard-cell or primary I/O components and a list of nets that is a collection of

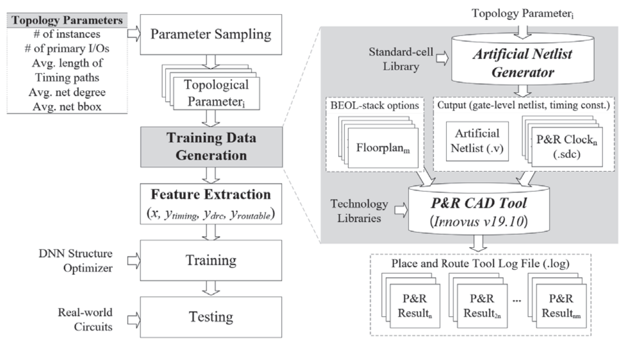


Fig. 1. Overall flow of the proposed machine learning framework for routability prediction model.

two or more interconnected components. Some attributes of the size, interconnection complexity, demands of wirelength and timing delay are observed in the topological shape of gate-level netlist. An artificial netlist with the specified characteristics can be obtained by taking the above parameters as input of ANG.

Next, we explore the design space exploration of the physical design to obtain P&R tool results that correspond to the artificial netlist. The following parameters represent the controllable design parameters that physical design engineers can tune during the early stage of physical design.

- ***Floorplan:*** layout utilization, aspect ratio, the number of metal layers, etc.
- ***Powerplan:*** stripe width, stripe spacing between VDD and VSS, VDD/VSS set-to-set pitch (for each metal layer), etc.
- ***Placement:*** clock period, min routing layer, max route layer, etc.

Under the limitations of computing resources and CAD tool licenses, we only explore the underlined parameters; (1) layout utilization, (2) aspect ratio, (3) the number of metal layers and (4) clock period. We use the default values for the others.

B. Routability factor extraction with ground-truth labeling

This step (Fig. 1) extracts the routability factors that are used as the input vector of routability prediction model that is obtained using a DNN. Each input vector is labelled 'pass' or 'fail' in terms of timing, DRC, and routability. We itemize the routability factors that are partially adapted from [4] and the output labels as:

Routability factor extraction

- **Circuit-related:** the number of combinational cells, the number of flipflops, the number of nets, average length of timing paths, average net degree, average size of net bbox.
- **Design parameter-related:** layout utilization, aspect ratio, the number of metal layers, P&R clock period.
- **Placement-related:** die area, pin density, total net bbox, initial worst negative slack (WNS), initial total negative slack (TNS), initial number of timing violations, horizontal track overflow, vertical track overflow.

Ground-truth labeling

- **Timing (y_{timing}):** *true*, if the number of violating timing paths equals to zero after post-routing.
- **DRC (y_{DRC}):** *true*, if the number of DRVs is less than $threshold$ after post-routing.[2]
- **Routability ($y_{routable}$):** *true*, if both y_{timing} and y_{DRC} are *true*.

Figure 2 illustrates the type and point of extracting routability factors from the P&R tool results. The number of DRVs and timing violations is used for ground-truth labeling.

[2] $threshold$ represents the fixable number of DRVs, we set it to 50

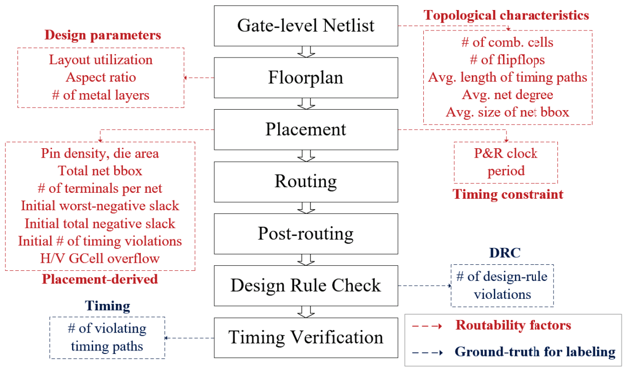


Fig. 2. Extracted input features with ground-truth labeling in generic P&R flow.

C. Routability prediction modeling

We use a modeling method that uses a DNN to predict routability early. The model is a binary classifier that predicts whether the placement solution is 'pass' or 'fail' for each label. To train the models, we use five-fold cross validation and early stopping (monitoring the validation loss) to reduce generalization error. For training, we use only the artificial data generated form the proposed training data generation flow. In contrast, we use the real-world circuit designs for testing. We evaluate the DNN models according to accuracy, precision, recall and F1 score for testing dataset. As a usage scenario of the models, when the prediction result of models ($y^{predict}_{timing}, y^{predict}_{DRC}, y^{predict}_{routable}$) are (*false*, *true*, *false*), physical designers can recognize #timing violations > 0 and therefore increased the clock period.

IV. Artificial Netlist Generator

In this section, we introduce an ANG that can create any parameterizing gate-level netlist while considering its topological shape. The overall flow (Fig. 3) consists of three steps: (A) graph initialization, (B) graph construction and (C) technology mapping. The inputs are topological parameters (N_{inst}, N_{prim}, D_{avg}, B_{avg}, and T_{avg}) and standard-cell library, and the outputs are a gate-level netlist (.v) and timing constraint (.sdc).

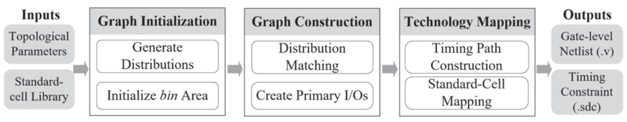


Fig. 3. Overall flow of proposed *artificial netlist generator*

A. Graph initialization

1) Generate distributions: In this step, the distributions of fanin ($Dist^{target}_{in}$), fanout ($Dist^{target}_{out}$) and size of net bbox ($Dist^{target}_{bbox}$) are generated based on D_{avg} and B_{avg}. The distributions of fanout and size of net bbox in real-world circuits resemble an exponential decay curve [11]. The parameters of the exponential curves of $Dist^{target}_{in}$ and $Dist^{target}_{out}$ are determined so that Eq. (1) is satisfied.

$$\begin{aligned} D_{avg} &= \sum\nolimits_x^{X_{in}} x \cdot Dist^{target}_{in}(x) = \sum\nolimits_x^{X_{out}} x \cdot Dist^{target}_{out}(x) \\ B_{avg} &= \sum\nolimits_x^{X_{bbox}} x \cdot Dist^{target}_{bbox}(x) \end{aligned} \quad (1)$$

where the domain of fanin, fanout and size of net bbox are given as X_{in}, X_{out} and X_{bbox}, respectively.

2) Initialize layout dimension: To be aware of bbox size when connecting vertices, we define a virtual layout dimension divided into 2-D *bin* grid. There are $X \cdot Y$ number of *bins* in layout dimension (X, Y), and each bin owns $\sqrt{N_{inst}}$ number of vertices. x-y coordinates of every vertex in $bin_{(x,y)}$ is (x, y) (Fig. 5).

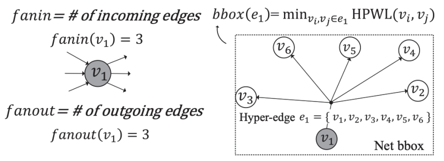

$fanin = \#\, of\, incoming\, edges$
$fanin(v_1) = 3$
$fanout = \#\, of\, outgoing\, edges$
$fanout(v_1) = 3$
$bbox(e_1) = \min_{v_i, v_j \in e_1} HPWL(v_i, v_j)$
Hyper-edge $e_1 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$
Net bbox

Fig. 4. Notations of fanin, fanout and net bbox in hyper-graph

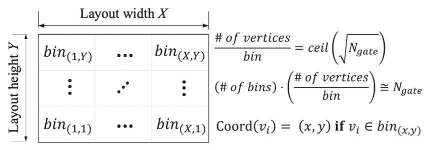

Layout width X
Layout height Y
$bin_{(1,Y)}$
$bin_{(X,Y)}$
$bin_{(1,1)}$
$bin_{(X,1)}$
$\frac{\#\, of\, vertices}{bin} = ceil\left(\sqrt{N_{gate}}\right)$
$(\#\, of\, bins) \cdot \left(\frac{\#\, of\, vertices}{bin}\right) \cong N_{gate}$
$\text{Coord}(v_i) = (x, y)$ if $v_i \in bin_{(x,y)}$

Fig. 5. Definition of layout dimension and *bin*

B. Graph construction

1) Distribution matching: The objective in this step is to complete the topological shape of the initialized graph considering the target distributions ($Dist^{target}_{in}$, $Dist^{target}_{out}$ and $Dist^{target}_{bbox}$). Our proposed algorithm creates an edge that minimizes the distribution error for each iteration and repeats until the number of edges becomes the target count. The distribution error of the graph G_t at iteration=t is defined as following Eq. (2).

$$\begin{aligned} Error(G_t) = &\sum\nolimits_x^{X_{in}} |Dist^{target}_{in}(x) - Dist^{G_t}_{in}(x)| \\ &+ \sum\nolimits_x^{X_{out}} |Dist^{target}_{out}(x) - Dist^{G_t}_{out}(x)| \\ &+ \sum\nolimits_x^{X_{bbox}} |Dist^{target}_{bbox}(x) - Dist^{G_t}_{bbox}(x)| \end{aligned} \quad (2)$$

where $Dist^{G_t}_{in}$ ($Dist^{G_t}_{out}$ or $Dist^{G_t}_{bbox}$) represents the fanin (fanout or bbox) distribution of the graph G_t. A gain means the amount of decrease in error when connecting a pair of source v_{out} and sink v_{in}, and it is calculated as:

$$Gain(v_{in}, v_{out}) = Gain_{in}(\alpha) + Gain_{out}(\beta) + Gain_{bbox}(\gamma_1, \gamma_2)$$
$$\text{where } \alpha = fanin(v_{in}),\ \beta = fanout(v_{out}),\ \gamma_1 = bbox(e^t_{out}),\ \text{and } \gamma_2 = bbox(e^{t+1}_{out}) \quad (3)$$

where e^t_{out} is a hyper-edge for which the source is v_{out}, and e^{t+1}_{out} represents $e^t_{out} \cup \{v_{in}\}$. $Gain_{in}(\alpha)$ denotes the gain when an arbitrary vertex v_{in} that meets $\alpha = fanin(v_{in})$ is selected as a sink. $Gain_{out}(\beta)$ is a gain when an arbitrary vertex v_{out} that meets $\beta = fanout(v_{out})$ is selected as a source. If v_{in} and v_{out} are connected, then $fanin(v_{in})$ increases to $\alpha + 1$, $fanout(v_{out})$ increases to $\beta + 1$. $Gain_{bbox}(\gamma_1, \gamma_2)$ means a gain where an arbitrary hyper-edge e^t_{out} that satisfies $\gamma_1 = bbox(e^t_{out})$ changes to $\gamma_2 = bbox(e^{t+1}_{out})$ after connection. $Gain(v_{in}, v_{out})$ is the same as $\Delta Error = Error(G_t) - Error(G_{t+1})$ where G_{t+1} is the graph after connecting v_{in} and v_{out}.

The objective of the proposed distribution matching algorithm (Algo. 1) is to minimize the distribution error of the graph. After the edge length ω is sampled considering the net bbox distribution (Line 1), the sink and source vertex are searched for each *bin* pair for which the distance satisfies ω (Lines 3-8). If the maximum gain is not $-\infty$, then v^{max}_{in} and v^{max}_{out} are connected (Line 9). We find the list of sink and source vertices to maximize *Gain* from the *bin* pair (Lines 12-15). For each source and sink vertex, calculate the gain (Lines 17-22) and return the maximum gain with the two vertices (Line 24).

2) Primary I/Os generation: All vertices are located somewhere between primary I/Os, and the primary I/Os are located at both either end of the graph, where the data signals start and end. To finds appropriate vertices that will be connected to the primary I/Os, we consider the topological order for each vertex. The gain value by the connection is considered.

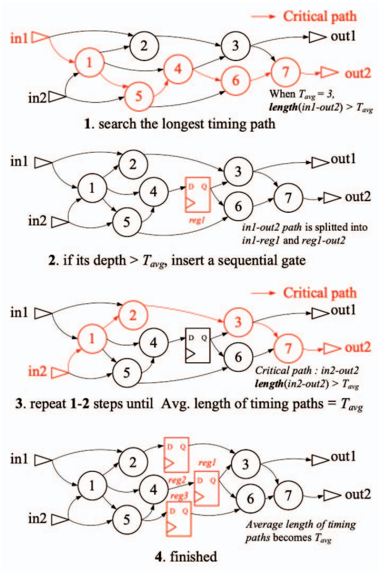
Algorithm 1: Distribution Matching

```
objective: Minimize the error of distributions $Error(G)$
while Sample edge length $\omega$ considering $Dist_{bbox}^{target}$ do
    Initialize $Gain_{max} \leftarrow -\infty$ ;
    foreach $bin_{(x_1,y_1)}$, $bin_{(x_2,y_2)}$ where $|x_1 - x_2| + |y_1 - y_2| = \omega$ do
        $(Gain, v_{in}, v_{out}) \leftarrow$ GetMaxGain($bin_{(x_1,y_1)}$,$bin_{(x_2,y_2)}$) ;
        if $Gain > Gain_{max}$ then
            Update $(Gain_{max}, v_{in}^{max}, v_{out}^{max}) \leftarrow (Gain, v_{in}, v_{out})$ ;
        end
    end
    Connect $v_{in}^{max}$ and $v_{out}^{max}$ if $Gain_{max} \neq -\infty$ ;
end
GetMaxGain ($bin_{(x_1,y_1)}$, $bin_{(x_2,y_2)}$)
    Get $\alpha_{max} \leftarrow \arg\max_{\alpha} Gain_{in}(\alpha)$ ;
    Get $\beta_{max} \leftarrow \arg\max_{\beta} Gain_{out}(\beta)$ ;
    $V_{sink} \leftarrow bin_{(x_1,y_1)}.fanin(\alpha_{max})$ ;
    /* list of vertices where $\forall v_i \in bin_{(x_1,y_1)}, fanin(v_i) = \alpha_{max}$ */
    $V_{source} \leftarrow bin_{(x_2,y_2)}.fanout(\beta_{max})$ ;
    /* list of vertices where $\forall v_i \in bin_{(x_2,y_2)}, fanout(v_i) = \beta_{max}$ */
    foreach $(v_{in}, v_{out}) \in V_{sink} \times V_{source}$ do
        $e_{out}^{t+1} \leftarrow e_{out}^{t} \cup \{v_{in}\}$ ;
        $\gamma_1 \leftarrow bbox(e_{out}^{t}), \gamma_2 \leftarrow bbox(e_{out}^{t+1})$ ;
        $Gain(v_{in}, v_{out}) \leftarrow$
          $Gain_{in}(\alpha_{max}) + Gain_{out}(\beta_{max}) + Gain_{bbox}(\gamma_1, \gamma_2)$ ;
        if $Gain(v_{in}, v_{out}) > Gain_{max}$ then
            $(Gain_{max}, v_{in}^{max}, v_{out}^{max}) \leftarrow$
              $(Gain(v_{in}, v_{out}), v_{in}, v_{out})$ ;
        end
    end
    return $(Gain_{max}, v_{in}^{max}, v_{out}^{max})$
```

C. Technology mapping

1) Timing path construction : The gate-level netlist is broken down into timing paths (in2out, in2reg, reg2reg and reg2out) that are checked at timing verification to verify whether any violating path exists for given P&R clock period. We focus on the average length of timing paths among the topological characteristic of gate-level netlist, because this length determines the delay in signal propagation. In this step, the sequential cells (flipflops) are inserted to split the long timing paths to match the desired average length T_{avg}. After the critical timing path is found using topological sort, a flipflop is inserted in the timing paths if its length is longer than T_{avg}. If the average length of timing paths in the graph is greater than T_{avg}, then the timing path construction must be repeated (Fig. 6).The average length of timing paths determines the searching space of P&R clock periods, and we empirically set the searching space to 50%-200% of $C_{delay} \cdot T_{avg}$ (C_{delay} represents the average cell delay).

2) Standard-cell mapping: This is a process of mapping the vertex to a standard-cell that has an appropriate driving strength that corresponds to fanin and fanout. We use two types of look-up tables (LUTs); i.e., cell-type and frequency-of-use. The cell-type LUT gives the list of available cell types considering driving strength when fanin and fanout are given as input. When the frequency-of-use LUT receives the list of available cell types, the LUT returns a specific cell type under the frequency of use per cell. Sometimes a cell with weak driving strength may have a large fanout; in this case, the source connected to the large fanout is replaced with a driver cell.



Critical path
in1
in2
out1
out2
When $T_{avg} = 3$, length(in1-out2) > T_{avg}
1. search the longest timing path
reg1
in1-out2 path is splitted into in1-reg1 and reg1-out2
2. if its depth > T_{avg}, insert a sequential gate
Critical path : in2-out2
length(in2-out2) > T_{avg}
3. repeat 1-2 steps until Avg. length of timing paths = T_{avg}
reg1
reg2
reg3
Average length of timing paths becomes T_{avg}
4. finished

Fig. 6. Procedure of timing path construction.

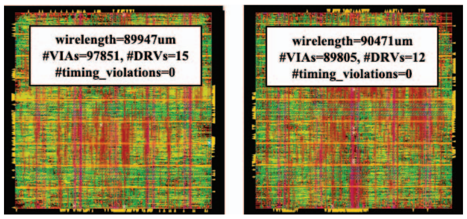
V. Experimental Results

We implemented artificial netlist generator in *C++* programming language and used the *Intel 22nm* technology library. We performed P&R using *Cadence Innovus Implementation System v19.10* [20]. We implemented the DNNs by using *Python Tensorflow* with *Keras* [21]. To generate distributions (Eq. 1), we used a curve-fitting method in the *SciPy* library [22].

A. Evaluation for artificial netlist generator

Design of experiments. We generated artificial netlists that have topological parameters that were extracted from real-world circuit designs [23], and then compared the characteristic parameters of (a) *real* circuits to those of (b) *fake* circuits generated by the proposed *artificial netlist generator* (Table I). All characteristic parameters are measured after placement that had layout utilization 0.5, aspect ratio 1.0 and top metal layer set to m8. To compare the similarity of topological shape between *real* and *fake*, we used the Rent parameter, topological parameter, fanout distribution and net bbox distribution. We computed the placement-based Rent parameters using RentCon [24]. B_{avg} is normalized by bin_size (Fig. 5). All parameters can be extracted after virtual P&R of commercial synthesis tool.

Comparison of topological similarity. The characteristic parameters of *real* and *fake* circuits agreed well (Table I). The average error of Rent parameter was ~5% and topological parameter was ~7%. Sometimes overhead during timing path construction or the insertion of drivers causes the differences in the shapes between the two circuits.



(a) *pci_bridge32* (b) *fake_pci_bridge32*

Fig. 7. Quality of results comparison after post-routing between (a) *pci_bridge32* and (b) *fake_pci_bridge32*.

The topological characteristics of the two circuits are similar so that the layout results after post-routing are also similar. Figure 7 shows the post-layouts of (a) *pci_bridge32* and (b) *fake_pci_bridge32* (the design parameters in both designs were set to be the same). We can observe that the wirelength, # of vias, # of DRVs and # of timing violations in both cases are close. Therefore, artificial circuits can be used as the training benchmarks that can substitute real circuits because the trends in the outputs (labels) for measuring routability are similar. We conclude that the *artificial netlist generator* enables the exploration of topological characteristics to obtain P&R circuit benchmarks.

B. Evaluation for the proposed machine learning framework

Design of experiments. The model training process optimizes the hyper-parameters to minimize the loss using the training data. A

TABLE I
COMPARISON OF CHARACTERISTIC PARAMETERS BETWEEN *real* AND *fake* CIRCUITS

Design	Syn. clk [ns]	Rent param.	N_{inst}	N_{prim}	D_{avg}	B_{avg}	T_{avg}	Fanout distribution						Net bbox distribution					
								1	2	3	4	5	6≤	0	1	2	3	4	5≤
(a) characteristic parameters of *real* circuits																			
aes_cipher_top	0.6	0.592	11919	391	3.2	0.90	10.3	0.548	0.168	0.103	0.073	0.046	0.062	0.467	0.340	0.131	0.040	0.011	0.011
des3	0.5	0.560	44533	298	3.1	0.41	4.9	0.536	0.180	0.138	0.060	0.035	0.051	0.758	0.202	0.028	0.007	0.004	0.001
jpeg_encoder	0.7	0.509	38025	47	3.1	0.40	7.8	0.512	0.330	0.080	0.031	0.005	0.042	0.846	0.101	0.026	0.012	0.006	0.009
keccak (sha3)	1.0	0.730	33323	585	2.9	1.74	22.5	0.568	0.190	0.128	0.060	0.031	0.023	0.588	0.165	0.071	0.044	0.031	0.101
ldpc_decoder_802_3an	0.6	0.771	63626	4100	2.9	1.76	11.0	0.667	0.159	0.072	0.027	0.020	0.055	0.546	0.148	0.067	0.049	0.045	0.145
mpeg2_top	0.7	0.549	12608	230	3.4	0.50	5.9	0.619	0.267	0.041	0.015	0.006	0.052	0.816	0.104	0.030	0.025	0.017	0.008
pci_bridge32	0.5	0.561	11354	269	3.3	0.57	8.8	0.616	0.256	0.038	0.027	0.011	0.052	0.759	0.151	0.039	0.023	0.011	0.017
point_scalar_mult	0.6	0.609	43572	932	3.4	0.61	11.4	0.423	0.359	0.105	0.049	0.007	0.057	0.778	0.119	0.036	0.025	0.018	0.024
spi_top	0.55	0.611	1831	116	3.2	0.89	8.8	0.616	0.151	0.139	0.023	0.007	0.064	0.565	0.284	0.067	0.047	0.016	0.021
tv80s	0.8	0.663	4298	46	3.3	1.03	19.7	0.554	0.190	0.104	0.042	0.025	0.085	0.523	0.272	0.090	0.047	0.030	0.038
usb_phy	0.25	0.460	389	33	2.9	0.72	5.2	0.571	0.208	0.095	0.077	0.015	0.034	0.500	0.414	0.059	0.017	0.005	0.005
usbf_top	0.5	0.605	8457	249	3.2	0.69	9.1	0.593	0.176	0.110	0.042	0.024	0.055	0.692	0.194	0.055	0.021	0.013	0.025
vga_enh_top	0.4	0.651	49871	198	3.5	0.56	11.9	0.593	0.343	0.009	0.004	0.002	0.049	0.841	0.099	0.016	0.007	0.003	0.034
wb_conmax_top	0.6	0.653	17180	2546	3.1	0.98	13.3	0.710	0.096	0.064	0.048	0.032	0.047	0.693	0.134	0.059	0.037	0.027	0.050
wb_size_bridge	0.3	0.703	278	182	2.8	0.89	8.2	0.640	0.223	0.047	0.040	0.007	0.043	0.454	0.365	0.137	0.038	0.011	0.000
(b) characteristic parameters of *fake* circuits																			
fake_aes_cipher_top	-	0.652	13096	391	3.2	0.72	9.9	0.586	0.146	0.090	0.063	0.040	0.051	0.583	0.284	0.084	0.028	0.010	0.011
fake_des3	-	0.541	46220	298	3.0	0.38	5.7	0.593	0.156	0.120	0.052	0.030	0.049	0.776	0.189	0.025	0.005	0.002	0.003
fake_jpeg_encoder	-	0.531	36821	47	3.1	0.39	8.0	0.522	0.322	0.078	0.030	0.005	0.043	0.792	0.164	0.029	0.008	0.003	0.004
fake_keccak	-	0.703	35552	585	2.9	0.76	25.0	0.612	0.170	0.115	0.054	0.028	0.021	0.666	0.195	0.063	0.030	0.017	0.029
fake_ldpc_decoder_802_3an	-	0.742	67255	4100	3.1	1.40	9.5	0.679	0.139	0.068	0.056	0.018	0.040	0.575	0.176	0.078	0.046	0.030	0.095
fake_mpeg2_top	-	0.528	12908	230	3.5	0.47	8.0	0.667	0.223	0.035	0.013	0.005	0.057	0.790	0.140	0.035	0.018	0.010	0.007
fake_pci_bridge32	-	0.565	11232	269	3.4	0.54	7.5	0.654	0.218	0.032	0.023	0.009	0.064	0.766	0.154	0.033	0.019	0.012	0.016
fake_point_scalar_mult	-	0.606	46325	932	3.4	0.61	12.9	0.499	0.304	0.089	0.042	0.006	0.060	0.717	0.181	0.048	0.023	0.012	0.019
fake_spi_top	-	0.601	1963	116	3.4	0.83	9.5	0.653	0.126	0.115	0.019	0.006	0.081	0.589	0.282	0.071	0.019	0.011	0.028
fake_tv80s	-	0.653	4407	46	3.3	0.87	19.5	0.580	0.179	0.098	0.039	0.024	0.080	0.582	0.245	0.091	0.041	0.019	0.022
fake_usb_phy	-	0.547	398	33	3.0	0.82	5.9	0.613	0.171	0.078	0.065	0.030	0.043	0.448	0.427	0.089	0.010	0.022	0.004
fake_usbf_top	-	0.602	8650	249	3.2	0.63	10.0	0.635	0.151	0.094	0.036	0.020	0.064	0.708	0.195	0.045	0.019	0.012	0.021
fake_vga_enh_top	-	0.570	54896	198	3.3	0.46	12.7	0.676	0.267	0.007	0.003	0.001	0.046	0.852	0.088	0.018	0.012	0.008	0.022
fake_wb_conmax_top	-	0.686	21221	2546	3.8	1.00	13.7	0.731	0.065	0.043	0.033	0.021	0.107	0.652	0.156	0.069	0.053	0.022	0.050
fake_wb_size_bridge	-	0.667	332	182	3.1	0.84	4.8	0.611	0.063	0.060	0.196	0.036	0.034	0.363	0.492	0.133	0.009	0.002	0.001

huge difference between the characteristic of training and testing data can lead to poor predictability, since the deep learning optimizer determines whether the training should continue under consideration on the predictability of unseen data through the validation data sampled from training dataset. The problem is the generalization error. In this experiment, we show that our machine learning framework can reduce the generalization error and increase predicability without any need for real-world data. We compared the evaluation metrics of (a) $Model_{prev}$ trained using design-*specific* dataset in previous work [4] and (b) $Model_{ours}$ trained using design-*various* dataset generated from our framework. $Model_{prev}$ and $Model_{ours}$ shared the same training flow, optimizer and DNN structure. The only difference between the two training datasets is the number of training circuits. We describe the details of training and testing datasets used in the experiment as follows.

- **Training data:** We used two different training datasets according to the used gate-level netlists (Table II); (a) design-*specific* dataset that is used in [4], uses only few circuit benchmarks. The *aes_x1, aes_x2* and *aes_x3* circuits were created by instantiating and stitching from one to three *aes_cipher_top* designs; (b) design-*various* dataset was generated using the proposed ML framework; we randomly sampled 40 artificial netlists from the range of topological parameters as described in table. They share the design space of layout utilization, aspect ratio, the number of metal layers.
- **Testing data:** The testing dataset is created by using 15 *Open-Cores* circuits (Table Ia). The design space is the same as in Table II, and the range of P&R clock periods is $\{1.0, 1.5, 2.0, 2.5\}\times$ synthesis clock period.[3] The number of samples for each circuit is 360, so the total number of samples is 5400.

For each output labels (y_{timing}, y_{DRC} and $y_{routable}$), we evaluate the accuracy, loss, recall, precision and F1 score for the testing dataset over 20 different hidden layers (Table III) and six optimizers.[4] The total number of trained models was 120 for each training dataset.

Generalization error analysis. Generalization error is the degree to which one fails to accurately predict outcomes for unseen data. To fit hyper-parameters that minimize this generalization error, modern ML techniques estimate the accuracy of unseen testing data by considering the loss or accuracy of the validation dataset that is an independent set separated from the training dataset; i.e., the interval between test and validation accuracy (or loss) can be regarded as a measure of whether the training dataset is similar to the distribution of real-world data. We measured the difference between validation loss and test loss, that is generalization error, for 120 training scenarios that corresponded to the hidden layer and optimizer that were used. Figure 8 illustrates the min-max and average values of the evaluation metric scores for 120 training scenarios. The average score of $Model_{ours}$ is higher than $Model_{prev}$; $\sim$8.6% in test accuracy and $\sim$0.077 in F1 score. From the range of min-max values, we can observe that the results of $Model_{ours}$ are more stable over 120 training scenarios. Table IV shows the validation and test loss of $Model_{prev}$ and $Model_{ours}$. The average interval between test_loss and val_loss were 0.321$\sim$0.427 for $Model_{ours}$, but 2.759$\sim$3.289 for $Model_{prev}$. The design-*various* dataset leads to achieve smaller generalization error $\sim$87% than design-*specific* dataset [4]. As a conclusion, the stability and robustness of $Model_{ours}$ is better than $Model_{prev}$ when it processes unseen circuit design. In addition, our framework requires less human effort when searching a best neural network design and optimizer.

TABLE II
DESIGN OF EXPERIMENTS TO OBTAIN TWO DIFFERENT TRAINING DATASETS

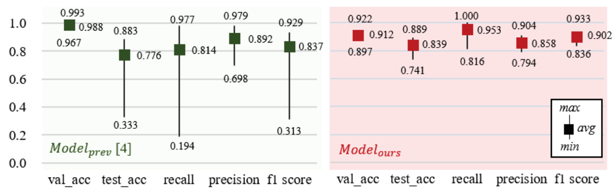
	(a) design-*specific* dataset [4]			(b) design-*various* dataset (ours)
	aes_x1	*aes_x2*	*aes_x3*	*artificial netlists*
Syn. Clk [ns]	0.6			-
N_{inst}	11919	24161	35604	$\{5k, 10k, \cdots, 60k\}$
N_{prim}	391	780	1169	$\{30, 110, \cdots, 460\}$
D_{avg}	2.2	2.2	2.2	$\{2.0, 2.2, \cdots, 4.0\}$
B_{avg}	0.9	0.7	0.6	$\{0.4, 0.8, \cdots, 2.0\}$
T_{avg}	10.3	10.6	10.4	$\{5.0, 10.0, \cdots, 30.0\}$
Layout util.	$\{50\%, 54\%, \cdots, 90\%\}$			
Aspect ratio	$\{0.5, 1.0, 2.0\}$			
Top metal layer	$\{m6, m7, m8\}$			
P&R Clk [ns]	$\{0.8, 1.1, 1.4, 1.7\}$			$0.1 \times \{0.5, 1.0, 1.5, 2.0\} \times T_{avg}$

TABLE III
THE LIST OF DNN STRUCTURES

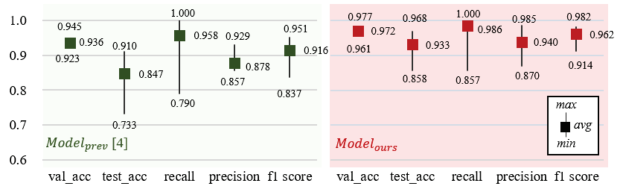
Index	Shape of hidden layer	Index	Shape of hidden layer
1	(304, 432, 448)	11	(32, 64, 240, 352, 384)
2	(96, 240, 480)	12	(64, 96, 192, 320, 432)
3	(96, 272, 448)	13	(32, 64, 240, 352, 384)
4	(80, 192, 320)	14	(96, 112, 192, 208, 256)
5	(320, 384, 448)	15	(48, 112, 256, 432, 464)
6	(112, 128, 160, 160)	16	(96, 160, 272, 368, 400, 496)
7	(32, 48, 64, 448)	17	(48, 48, 176, 192, 272, 496)
8	(208, 272, 320, 368)	18	(176, 256, 416, 448, 464, 496)
9	(32, 80, 240, 352)	19	(80, 320, 352, 448, 480, 480)
10	(128, 176, 288, 320)	20	(48, 64, 80, 112, 160, 304)

[3]The minimum clock period that has no timing violation was used when synthesis.

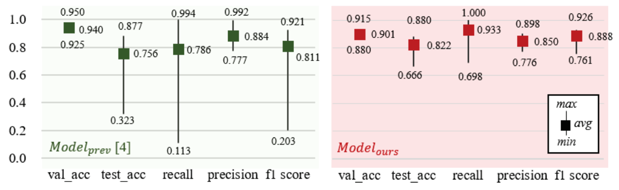
[4](optimizer, learning rate): (SGD, 0.01), (RMSprop, 0.001), (Adagrad, 0.01), (Adam, 0.001), (Adamax, 0.002), (Nadam, 0.002).



(a) y_{timing}



(b) y_{DRC}



(c) $y_{routable}$

Fig. 8. Comparison of evaluation metric scores between $Model_{prev}$ and $Model_{ours}$. Each model has 120 scenarios according to the used hidden layer shape and optimizer.

TABLE IV
COMPARISON OF VALIDATION AND TEST LOSS

	$Model_{prev}$				$Model_{ours}$			
	avg	*min*	*max*	*stdev*	*avg*	*min*	*max*	*stdev*
	(a) y_{timing}							
val_loss	0.052	0.036	0.111	0.013	0.204	0.179	0.244	0.010
test_loss	3.341	0.594	20.715	3.562	0.631	0.366	2.495	0.292
	(a) y_{DRC}							
val_loss	0.120	0.109	0.141	0.006	0.076	0.062	0.098	0.008
test_loss	2.984	0.531	10.638	1.959	0.397	0.140	2.111	0.325
	(a) $y_{routable}$							
val_loss	0.136	0.113	0.196	0.015	0.230	0.205	0.269	0.012
test_loss	2.895	0.676	11.536	1.969	0.560	0.332	1.732	0.195

Data visualization. We used t-SNE [25] to illustrates the distributions (Fig. 9) of the three datasets. We focus on the overlap between training and testing dataset. $Model_{ours}$ created using our framework is distributed over the area and overlaps the area of testing dataset. The diversity of features in the design-*various* dataset that is generated using our framework covers the wider range than the design-*specific* dataset [4].

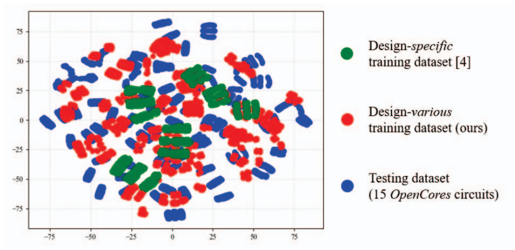


Fig. 9. Visualized data distributions of design-*specific* dataset [4] (green), design-*various* (red) and testing dataset (blue) using t-SNE [25].

VI. CONCLUSION

The proposed ML framework can automatically generate a training dataset for early routability prediction while representing the distribution of real-world data. It requires less human effort to collect P&R data samples. The new *artificial netlist generator* enables control of the topology shape of synthetic circuit by the user-specified input parameters, topological characteristics can be explored during training data preparation. We prove that the artificial netlist has a realistic topology compared to real-world synthetic circuits. The size and interconnect complexity of synthetic circuit benchmark can be extended, so various samples for training can be obtained in preparation for new emerging circuit designs. Compared to the design-specific training dataset [4], we achieve up to 9% in test accuracy, and up to 87% reduction of generalization error for the output labels (y_{timing}, y_{DRC} and $y_{routable}$). We expect that this framework can be applied to various ML-based applications in physical design. For example, other usage scenarios can be considered: (i) evaluation of circuit benchmarks for routability-driven placement algorithm, (ii) collecting image clips of hotspots or routing congestion for model construction, and (iii) pin accessibility-aware cell layout optimization, etc.

ACKNOWLEDGMENT

The authors gratefully acknowledge extremely valuable scientific discussions with Andrew B. Kahng (UC San Diego), Kwangok Jeong (Samsung Electronics) and Taigon Song (Kyungpook National University).

REFERENCES

[1] X. Yang, E. Bozorgzadeh, M. Sarrafzadeh, "Wirelength estimation based on rent exponents of partitioning and placement", *Proc. SLIP*, 2001, pp. 25-31.
[2] L. Hagen, A. B. Kahng, F. J. Kurdahi and C. Ramachandran, "On the intrinsic Rent parameter and spectra-based partitioning methodologies", *Proc. EURO-DAC*, 1992, pp. 202-208.
[3] T. Taghavi, F. Dabiri, A. Nahapetian, M. Sarrafzadeh, "Tutorial on congestion prediction," *Proc. SLIP*, 2007, pp. 15-24.
[4] W. J. Chan, Y. Du, A. B. Kahng, S. Nath and K. Samadi, "BEOL stack-aware routability prediction from placement using data mining techniques", *Proc. ICCD*, 2016, pp. 41-48.
[5] A. B. Kahng, S. Kang, S. Kim, K. Samadi and B. Xu, "Power Delivery Pathfinding for Emerging Die-to-Wafer Integration Technology", *Proc. DATE*, 2019, pp. 842-847.
[6] P. Spindler and F. M. Johannes, "Fast and Accurate Routing Demand Estimation for Efficient Routability-driven Placement", *Proc. DATE*, 2007, pp. 1-6.
[7] Y. Huang, Z. Xie, G. Fang, T. Yu, H. Ren, Y. Chen and J. Hu, "Routability-Driven Macro Placement with Embedded CNN-Based Prediction Model", *Proc. DATE*, 2019, pp. 180-185.
[8] A. B. Kahng, "Machine Learning Applications in Physical Design: Recent Results and Directions", *Proc. ISPD*, 2014, pp. 68-73.
[9] K. Liu, H. Yang, Y. Ma, B. Tan, B. Yu, Evangeline F. Y. Young, R. Karri, and S. Garg, "Adversarial Perturbation Attacks on ML-based CAD: A Case Study on CNN-based Lithographic Hotspot Detection", *ACM Trans. on DAES*, 2020, pp. 31.
[10] K. Liu, B. Tan, R. Karri and S. Garg, "Poisoning the (Data) Well in ML-Based CAD: A Case Study of Hiding Lithographic Hotspots", *Proc. DATE*, 2020, pp. 306-309.
[11] P. Zarkesh-Ha, J. A. Davis and J. D. Meindl, "Prediction of net-length distribution for global interconnects in a heterogeneous system-on-a-chip", *IEEE Trans. on VLSI*, 2000, pp. 649-659.
[12] C. Nadeau and Y. Bengio, "Inference for the Generalization Error", *Proc. NeurIPS*, 2000, pp. 307-313.
[13] A. B. Kahng, S. Mantik and D. Stroobandt, "Toward accurate models of achievable routing", *IEEE Trans. on CAD*, 2001, pp. 648-659.
[14] A. Kahng, A. B. Kahng, H. Lee and J. Li, "PROBE: A Placement, Routing, Back-End-of-Line Measurement Utility", *IEEE Trans. on CAD*, 2018, pp. 1459-1472.
[15] A. B. Kahng and S. Kang. 2012. "Construction of realistic gate sizing benchmarks with known optimal solutions", *Proc. ISPD*, 2012, pp. 153–160.
[16] P. Gupta, A. B. Kahng, A. Kasibhatla and P. Sharma, "Eyecharts: Constructive benchmarking of gate sizing heuristics", *Proc. DAC*, 2010, pp. 597-602.
[17] D. Stroobandt, P. Verplaetse and J. van Campenhout, "Generating Synthetic Benchmark Circuits for Evaluating CAD Tools", *IEEE Trans. on CAD*, 2000, pp. 1011-1022.
[18] M. D. Hutton, J. Rose, and D. Corneil, "Automatic generation of synthetic sequential benchmark circuits", *IEEE Trans. on CAD*, 2002, pp. 928-940.
[19] P. D. Kundarewich and J. Rose, "Synthetic circuit generation using clustering and iteration", *IEEE Trans. on CAD*, 2004, pp. 869-887.
[20] Cadence Innovus User Guide, https://www.cadence.com.
[21] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, http://tensorflow.org.
[22] *SciPy: Open Source Scientific Tools*, http://www.scipy.org.
[23] OpenCores: Open Source IP-Cores, http://www.opencores.org.
[24] K. Jeong, A. B. Kahng and H. Yao, "Rent Parameter Evaluation Using Different Methods", http://vlsicad.ucsd.edu/WLD/RentCon.pdf.
[25] L. Maaten and G. Hinton, "Visualizing data using t-SNE", *Journal of Machine Learning Research*, 2008.