Automatic Layout Generation with Applications in Machine Learning Engine Evaluation

Haoyu Yang*, Wen Chen*, Piyush Pathak†, Frank Gennari†, Ya-Chieh Lai†, Bei Yu*
*The Chinese University of Hong Kong
†Cadence Design Systems, Inc.
Email: {hyyang,byu}@cse.cuhk.edu.hk

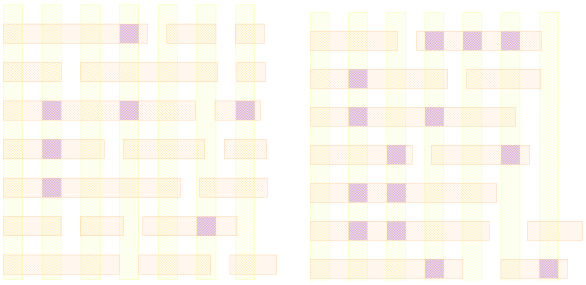
***Abstract*—Machine learning-based lithography hotspot detection has been deeply studied recently, from varies feature extraction techniques to efficient learning models. It has been observed that such machine learning-based frameworks are providing satisfactory metal layer hotspot prediction results on known public metal layer benchmarks. In this work, we seek to evaluate how these machine learning-based hotspot detectors generalize to complicated patterns. We first introduce a automatic layout generation tool that can synthesize varies layout patterns given a set of design rules. The tool currently supports both metal layer and via layer generation. As a case study, we conduct hotspot detection on the generated via layer layouts with representative machine learning-based hotspot detectors, which shows that continuous study on model robustness and generality is necessary to prototype and integrate the learning engines in DFM flows. The source code of the layout generation tool will be available at https://github.com/phdyang007/layout-generation.**

I. Introduction

The rapid development of machine learning techniques have brought promising alternatives to design for manufacturability problems, especially for lithography hotspot detection. Related researches range from varies feature extraction techniques [1]–[7] and efficient learning models [1], [8]–[13]. Most of these frameworks are presenting very high hotspot detection accuracy with reasonable false positive overhead. However, most of them are targeting at a public benchmark at legacy technology node from ICCAD2012 CAD Contest [14], with carefully designed and tuned parameters and machine learning engines.

In this paper, we seek to evaluate how these machine learning-based hotspot detectors generalize to complicated patterns. We first introduce an automatic layout generation tool that can synthesize various layout patterns given a set of design rules. Unlike the existing pattern generation solutions that seek to increase the pattern space within a constrained Euclidean distance [15], [16], the proposed automatic generation flow generates patterns for certain DRC constrained designs.

The tool supports both metal layer and via layer generation. In the configuration files, we can manually achieve certain layout properties including tip to tip distance, wire CD, layout density, via density and so on. These configurations enable a generation of diverse layout patterns under a given technology node, which can be applied to various DFM research and analysis including hotspot detection and OPC recipe development. An examples of via layers with different density can be found in Fig. 1, where the upper and the lower metal gratings are intentionally placed to assist via generation.



(a) Low via density (b) High via density

Fig. 1 Visualization of generated via patterns. Vias are randomly placed at the intersection region of given upper and lower metal shapes with a predetermined probability that controls via density.

As a case study, we conduct hotspot detection on the generated via layer layouts with representative machine learning-based hotspot detectors [1], [8], [10], which adopt smooth-boosting [1], shallow convolutional neural networks [8], [10], respectively. Although these hotspot detectors achieve promising results on ICCAD2012 CAD contest benchmarks, we show performance degradation occurs with complex dataset, which implies continuous study on model robustness and generality is necessary to prototype and integrate the learning engines in DFM flows. The contributions of this paper are listed below.

- We develop a layout generation toolkit with Python and KLayout [17] backbones, to support efficient unidirectional metal layer and via/contact generation.
- We feed the generated via layers into a commercial lithography simulator and construct a via layer hotspot benchmark suit for hotspot detector evaluation.
- Evaluation experiments show that representative hotspot detectors may fail to exhibit a satisfactory detection performance on either accuracy or false positive penalty.

The rest of the paper is organized as follows. Section II describes details of the layout generation flow. Section III presents a case study of state-of-the-art hotspot detector

evaluation, followed by conclusion in Section IV.

II. A Layout Generation Toolkit

In this section, we will introduce the details of our layout generation flow with Python and KLayout backbones. In current version, we support unidirectional metal layer and via/contact generation.

A. Metal Generation

The layout generation tool generates metal layers cell by cell where metal shape distribution and placement are controlled by a set of design constraints as listed below.

- `wire_cd` (w). The critical dimension of metal wire shapes, defined by wire width.
- `track_pitch` (p). The distance between adjacent wire tracks.
- `min_t2t` (t_1). The minimum line-end to line-end distance in each wire track.
- `max_t2t` (t_2). The maximum line-end to line-end distance in each wire track.
- `min_length` (l_1). The minimum length of a single wire shape along the wire tracks.
- `max_length` (l_2). The maximum length of a single wire shape along the wire tracks.
- `t2t_grid` (t_g). The unit size of line-end to line-end distance.
- `total_x` (x_t). The cell bounding box size in x direction.
- `total_y` (y_t). The cell bounding box size in y direction.

To make a better understanding of these concepts, we also visualize certain terminologies in Fig. 2. It can be seen that `min_t2t`, `max_t2t`, `min_length`, and `max_length` make key contributions to metal distribution and density.

Metal shapes are generated track-by-track and each track is filled as described in Algorithm 1. We draw shapes from an initial lower-left coordinate (x, y) (line 2), which will be increased until x reaches the total cell size constraint x_t during the generation procedure (line 3); the wire length is determined by a random number between l_1 and $\max(x_t - x, l_2)$ that ensures the shape length will never exceed the cell size (line 4); we then draw a rectangle defined by the lower-left and upper-right coordinates (lines 5 – 6); spacing between adjacent shapes on single track is generated by line-end to line-end constraints t_1 and t_2 (line 7) followed by the update of start coordinates (line 8). The entire cell is filled track-by-track that can be totally in parallel as different tracks are generated independently, as in Algorithm 2.

B. Via Generation

Vias, as conductive connections between different metal layers, are most likely to occur at regions where metals in adjacent layers are overlapped in the plane. The basic idea of via generation is creating candidate via locations with horizontal and vertical metal shapes that can be generated

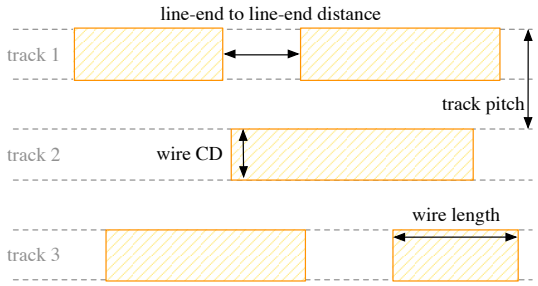


Fig. 2 Visualization of terminologies in metal generation constraints.

Algorithm 1 On Track Wire Generation

1: **function** `DrawWire`(w, t_1, t_2, t_g, l_1, l_2, x_t, x_o, y_o)
2: Initialize parameters $x, y \leftarrow 0$;
3: **while** $x < x_t$ **do**
4: $l \leftarrow \texttt{rand}(l_1, \max(l_2, x_t - x))$;
5: $x_{ll} \leftarrow x + x_o, y_{ll} \leftarrow y + y_o, x_{ur} \leftarrow x + l + x_o, y_{ur} \leftarrow y + w + y_o$;
6: Draw a rectangle defined by x_{ll}, y_{ll}, x_{ur} and y_{ur};
7: $t \leftarrow \texttt{rand}(t_1, \min(t_2, x - x_{ur}), t_g)$;
8: $x \leftarrow x + l + t$;
9: **end while**
10: **end function**

Algorithm 2 Wire Cell Generation

1: **function** `DrawWireCell`(w, p, t_1, t_2, t_g, l_1, l_2, x_t, y_t, x_o, y_o)
2: Initialize parameters $y \leftarrow 0$;
3: **while** $y < y_t$ **do**
4: `DrawWire`(w, t_1, t_2, t_g, l_1, l_2, x_t, x_o, y);
5: $y \leftarrow y_o + p + w$;
6: **end while**
7: **return** Cell;
8: **end function**

with Algorithm 1. Thus, we have additional via constraints besides those control wire shape generation.

- `via_x` (v_x). Via size along x direction.
- `via_y` (v_y). Via size along y direction.
- `density` (ρ). The probability of a via appearing at a candidate via location.
- `enclosure_x` (e_x). The minimum horizontal distance from a via to metal line-end.
- `enclosure_y` (e_y). The minimum horizontal distance from a via to metal line-end.
- `via_pitch_x` (v_{px}). The minimum center-to-center distance of two vias in the same tract in x direction.
- `via_pitch_x` (v_{py}). The minimum center-to-center distance of two vias in the same tract in y direction.

where `via_x` and `via_y` are directly satisfied by controlling `wire_cd`s of two metal layers, `density` contributes to a threshold whether we should place a via at a candidate

TABLE I Metal layer generation specs (μm).

cellname	wire_cd	track_pitch	min_t2t	max_t2t	min_length	max_length	t2t_grid	total_x	total_y
test1	0.016	0.032	0.012	0.2	0.044	0.1	0.005	100	100
test2	0.016	0.032	0.012	0.2	0.044	0.1	0.012	100	100
test3	0.016	0.032	0.012	0.4	0.044	0.5	0.005	100	100
test4	0.016	0.032	0.012	0.4	0.044	0.5	0.012	100	100
test5	0.016	0.032	0.012	0.6	0.044	0.5	0.036	100	100
test6	0.016	0.032	0.012	0.6	0.044	0.5	0.072	100	100

via location. Challenging problem comes with satisfying enclosure and pitch constraints.

We propose the concept of assist wire generation that generates on track wire shapes with line-ends shortened by the enclosure distance. As shown in Fig. 3, we generate via candidate locations by taking the intersection between the assist layer and M2 layer, instead of directly calculate the intersection between M1 and M2. With the help of assist wire, we can direclty filter out vias that violate enclosure constraints by checking the area of them. In real cases, enclosure constraints are applicable to both x and y directions, which can be solved by simply using assist wires in vertical tracks.

To deal with the pitch constraints, we create a via candidate matrix for each cell, with each entry representing a cross point between a horizontal and a vertical tracks. We put one at entries where there are metal shapes overlapping with each other and leave other entries zero. The via candidate matrix for the cell in Fig. 3 can then be written as follows.

$$\boldsymbol{M}_{vc} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

After removing enclosure violations, the matrix becomes

$$\boldsymbol{M}_{vc} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

Because the metal pitch is intentionally picked the same as via during metal layer generation, the problem of finding via pitch violations becomes finding $\begin{bmatrix} 1 & 1 \end{bmatrix}$ or $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ patterns in $\boldsymbol{M}_{vc}$. Finally, via pitch violations can be easily removed.

To summary, the via generation can be done with Algorithm 3, where we first generate metal layers that are to be connected by vias (lines 1–2); candidate via shapes are placed by taking the intersection of two metal layers along with the calculation of via candidate matrix (line 3); following steps are removing enclosure conflicts (line 4), `pitch_x` conflicts (lines 5–10) and `pitch_y` conflicts (lines 11–16).

C. Layout Generation Performance

We evalutate the performance of the proposed layout generation tool on an Intel Xeon 3.5GHz platform with single thread. We adopt the specs in TABLE I and TABLE II for metal and via layer generation, respectively, with each test cell has an area of $0.001mm^2$. The throughputs are

Algorithm 3 Via Generation

Require: M1 specs, M2 specs, v_x, v_y, e_x, e_y, v_{px}, v_{py}.
Ensure: Via cell.

1: $C_1 \leftarrow$ Call `DrawWireCell` with M1 specs;
2: $C_2 \leftarrow$ Call `DrawWireCell` with M2 specs;
3: $\boldsymbol{M}_{vc} \leftarrow$ calculate via candidate matrix by taking C_1 `AND` C_2 and insert via shapes;
4: $\boldsymbol{M}_{vc} \leftarrow$ Detect enclosure conflicts and remove conflict vias;
5: **for** each $\boldsymbol{M}_{vc}(i,j)$ **do**
6: **if** $\boldsymbol{M}_{vc}(i,j) = 1$ and $\boldsymbol{M}_{vc}(i-1,j) = 1$ **then**
7: Delete via cooresponding to $\boldsymbol{M}_{vc}(i,j)$;
8: $\boldsymbol{M}_{vc}(i,j) \leftarrow 0$;
9: **end if**
10: **end for**
11: **for** each $\boldsymbol{M}_{vc}(i,j)$ **do**
12: **if** $\boldsymbol{M}_{vc}(i,j) = 1$ and $\boldsymbol{M}_{vc}(i,j-1) = 1$ **then**
13: Delete via cooresponding to $\boldsymbol{M}_{vc}(i,j)$;
14: $\boldsymbol{M}_{vc}(i,j) \leftarrow 0$;
15: **end if**
16: **end for**

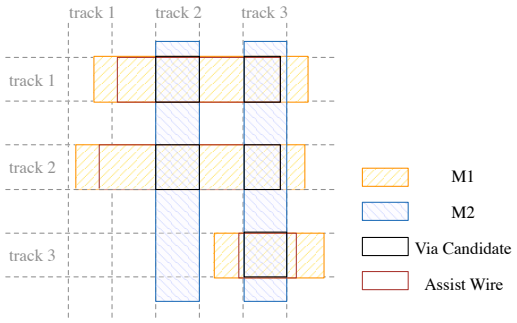


Fig. 3 Satisfying enclosure constraints with assist wires.

dipicted in Fig. 4 that corresponds to the area of metel/via cell generated per second.

III. Evaluation of Representative Hotspot Detectors

As a case study, we generate a group of via patterns with above via generation flow and obtain clip labels by feeding them into an industrial DRC and simulation tool. In this section, we will use the generated dataset to evaluate the performance of different loss functions on several represen-

TABLE II Via layer generation specs (μm).

cellname	via1_x	via1_y	m1_enc	m2_enc	min_via1_pitch_x	via_fraction	min_via1_pitch_y	total_x	total_y
test1	0.07	0.07	0.02	0.02	0.14	0.1	0.14	100	100
test2	0.07	0.07	0.02	0.02	0.14	0.2	0.14	100	100
test3	0.07	0.07	0.02	0.02	0.14	0.3	0.14	100	100
test4	0.07	0.07	0.02	0.02	0.14	0.4	0.14	100	100
test5	0.07	0.07	0.02	0.02	0.14	0.5	0.14	100	100
test6	0.07	0.07	0.02	0.02	0.14	0.6	0.14	100	100

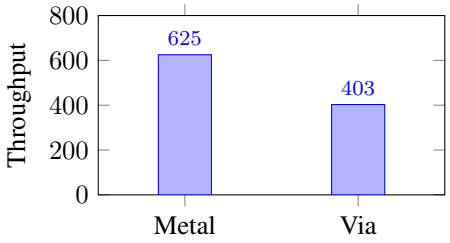


Fig. 4 Throughput (μm^2 / s) estimation of the layout generation tool.

TABLE III Neural Network Configuration.

Layer	Kernel Size	Stride	Output Node #
conv1-1	3	1	$12 \times 12 \times 16$
conv1-2	3	1	$12 \times 12 \times 16$
maxpooling1	2	2	$6 \times 6 \times 16$
conv2-1	3	1	$6 \times 6 \times 32$
conv2-2	3	1	$6 \times 6 \times 32$
maxpooling2	2	2	$3 \times 3 \times 32$
fc1	-	-	250
fc2	-	-	2

tative machine learning-based hotspot detectors [1], [8], [10]. We first introduce beneath ideas of these hotspot detectors.

A. Feature Tensor Extraction and Batch Biased Learning (BBL) [8]

Layout hotspot detection, originally, can be regarded as an image classification problem. However, the input size is much larger than images in traditional classification tasks because a large area of layout context is required to determine whether a clip contains hotspot or not. [8] tackles the problem with a layout compression technique in frequency domain that dramatically reduces input size as well as preserve a large fraction of original layout information. The compression is conducted following four steps including (1) layout slicing based on critical dimension (2) DCT on sub clip blocks (3) flatten DCT coefficients in zig-zag form and (4) discarding high frequency components and constructing feature tensor. Additionally, the compressed data can be easily learned with shallow neural networks. As a case study, we will use the architecture specified in TABLE III in all the deep learning related experiments.

In the procedure of training a hotspot detector, neuron weights are updated towards pattern labels. [8] takes advantage of the fact that any non-hotspot instances with predicted non-hotspot probability over 50% will be correctly classified. Therefore, it is not necessary to guide the neural networks to learn a high confidence prediction. Accordingly, a batch biased learning is proposed to introduce label penalty on non-hotspot patterns based on their training loss on current step which in turn controls the training label penalty.

B. Smooth Boosting [1]

[1] provides a legacy machine learning solution for hotspot detection which includes feature engineering and learning engine development. In the feature extraction stage, [1] improves traditional concentric circle area sampling (CCAS) [3] with mutual information, which considers that only partial of sampled circles have strong contributions to final feature vectors and labels. Circle selection problem can be formulated as follows.

$$\mathfrak{I}_{n_c}^{*} = \arg\min_{\mathfrak{I}_{n_c} \subseteq \mathfrak{I}} \sum_{i \in \mathfrak{I}_{n_c}} I(C_i; Y), \tag{3a}$$

$$\text{s.t. } |i-j| > d, \forall i, j \in \mathfrak{I}_{n_c}, \tag{3b}$$

where $\mathfrak{I}_{n_c}^{*}$ includes selected circle indices, $\mathfrak{I} = \{i | 1 < i < r_{\max}, i \in \mathbb{N}\}$ contains indices of all circle candidates and

$$I(C_i; Y) = \sum_{c_i \in C_i} \sum_{y \in Y} p(c_i, y) \log \frac{p(c_i, y)}{p(c_i)p(y)}, \tag{4}$$

which calculates the mutual information of i_{th} circle and labels. Here C_is and Y are random variables defined in circle index space and label space.

Smooth Boosting [18], as an ensemble learning engine with good noise tolerance, is chosen as the preferred model here for hotspot detection tasks, where the weight of each weak classifier is updated during training such that each weak classifier satisfies Equation (5).

$$\frac{1}{2} \sum_{j=1}^{n} M_t(j) \left| h_t\left(\mathbf{x}_j\right) - y_j \right| \leq \frac{1}{2} - \gamma, \tag{5}$$

where $M_t(j)$s are weight terms, h_ts are hypothesis function of each weak classifier and γ is a hyper-parameter defined in [18].

In the inference stage, all weak learners will work together to predict the label of input clips with their weighted output being the final classification results, as in Equation (6).

$$f = \text{sign}\left(\frac{1}{T} \sum_{t=1}^{T} h_t \right). \tag{6}$$

C. LithoROC [10]

Receiver operating characteristic (ROC) has be widely used as a machine learning model evaluation metric. Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$ are the i-th data sample in the feature space and its true class label, we can divide the dataset into a positive sample set $D_+ = \{(x_i^+, +1\}_{i=1}^{N^+}$ and a negative sample set $D_- = \{(x_i^-, -1\}_{i=1}^{N^-}$, where N_+ and N_- denote the number of positive and negative samples respectively, and $N = N_+ + N_-$. Let $f(x)$ denote the prediction model.

Based on the equivalence of area under the ROC curve(AUC) and Wilcoxon-Mann-Whitney (WMW) statistic test of ranks, pairwise convex surrogate loss $\Phi(f(x_i^+) - f(x_j^-))$ is applied to make the AUC differentiable. The original AUC function is defined as following:

$$AUC = \frac{1}{N_+ N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} I(f(x_i^+) - f(x_j^-)), \tag{7}$$

where $I((f(x_i^+) - f(x_j^-)))$ is the discontinuous indicator function. The differentiable form is shown below to work as the loss function.

$$L_\Phi(f) = \frac{1}{N_+ N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} \Phi(f(x_i^+) - f(x_j^-)). \tag{8}$$

Then maximizing the AUC score is equivalent to minimizing the loss function $L_\Phi(f)$. Let $z = f(x_i^+) - f(x_j^-)$, [10] recommend four surrogate loss functions for differentiable AUC approximation: the pairwise squared loss (PSL) [19], the pairwise hinge loss (PHL) [20], the pairwise logistic loss (PLL) [21], and the R loss function [22] as shown below.

$$\Phi_{PSL}(z) = (1 - z)^2, \tag{9}$$

$$\Phi_{PHL}(z) = \max(1 - z, 0), \tag{10}$$

$$\Phi_{PLL}(z) = \log(1 + \exp(-\beta z)), \tag{11}$$

$$\Phi_R(z) = \begin{cases} (-(z - \gamma))^p, & \text{if } z > \gamma, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

It should also be noted that in R loss function we have $0 < \gamma < 1$ and $p > 1$. Additionally, we heuristically define two new cubic loss functions (Pairwise Cubic Loss Function1 and Pairwise Cubic Loss Function2) and test their performances and compare with previous ones.

$$\Phi_{PCL1}(z) = \max(8 - (1 + z)^3, 0), \tag{13}$$

$$\Phi_{PCL2}(z) = \max((1 - z)^3, 0). \tag{14}$$

Fig. 5 illustrates the comparison of all surrogate loss functions. Note that there is a sharp increase in penalty when z goes to -1 in our cubic functions. The purpose is to apply more penalty when false positive and false negative simultaneously appear. As we can see from the curve of PCL2, the penalty for z is highly to the original ones when $z \in [0, 1]$. Furthermore, we can also clearly distinguish larger penalty when any mis-prediction occurs (for curve region $[-1, 0]$), which is expected to result in slightly better performance.

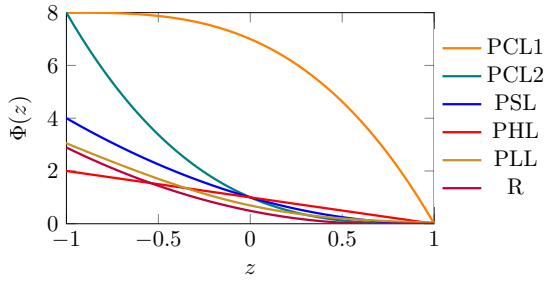
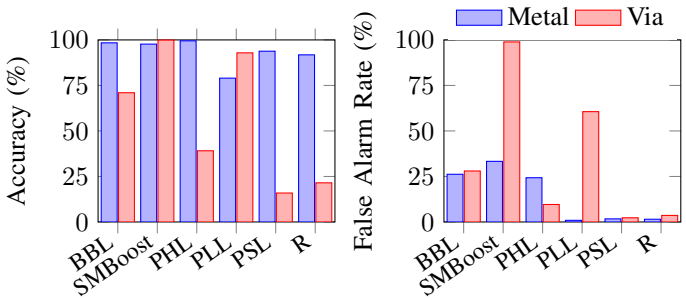


Fig. 5 Visualization of different approximations of AUC.



(a) Accuracy (b) False Alarm

Fig. 6 Evaluation of machine learning-based hotspot detectors.

D. Experiments

We implement these hotspot detection solutions with Python and Tensorflow 1.9. All experiments are conducted on a Intel platform with Titan Xp graphic unit. For the experiments on BBL and SMBoost, we adopt the same parameter settings as in [8] and [1] respectively. In the experiments on LithoRoC, we explore the performances of [8] with surrogate loss function suggested by [10] to seek for the increasing and stabilizing the hotspot detection accuracy and minimizing false alarms. In our experiment, we adopt the original unified data from [8]. The detector is evaluated with $\gamma = 1\text{e} - 3$, $\alpha = 0.65$, $m = 32$ and $k = 2000$. We set $\beta = 3$ in PLL loss function and $\gamma = 0.7$, $p = 2$ in R loss function. In each iteration, $\frac{m}{2}$ hotspot and $\frac{m}{2}$ non-hotspot training instances are randomly selected to perform weight updating. Learning rate decays after every k iterations. During optimization, we randomly select one between-class pair to calculate z. After each iteration, a new batch is generated, so all between-class pairs have equal probability to be fetched for optimization. In the end, we use the last saved model to test prediction performance, which is measured following previous works [14].

Definition 1 (Accuracy (Acc)). *The ratio between the number of correctly predicted hotspot clips and the number of all real hotspot clips.*

Definition 2 (False Alarm Rate (FA)). *The ratio between the number of correctly predict non-hotspot clips and the number of all non-hotspot clips.*

TABLE IV CNN-based hotspot detectors behave unstably on challenging datasets.

ID	PSL		PHL		PLL		R		PCL1		PCL2		BBL	
	Acc (%)	FA (%)	Acc (%)	FA (%)	Acc (%)	FA (%)	Acc (%)	FA (%)	Acc (%)	FA (%)	Acc (%)	FA (%)	Acc (%)	FA (%)
1	10.7	0.96	94.17	60.95	87.87	43.79	10.82	1.92	81.33	36.36	24.14	2.82	58.74	24.50
2	17.48	2.5	79.31	38.28	53.26	16.90	30.56	7.81	83.59	39.37	86.09	44.69	62.78	23.43
3	17.6	3.26	93.34	59.54	33.17	6.59	17.95	2.11	40.07	8.77	85.25	40.90	60.64	22.92
4	20.81	4.23	38.76	7.94	28.18	4.99	30.92	7.11	92.39	51.34	72.53	27.66	65.40	31.37
5	18.67	3.07	17.36	1.66	64.09	22.86	15.57	1.86	85.49	39.88	46.25	11.46	58.26	24.71
Ave	17.05	2.80	64.59	33.67	53.31	19.03	21.16	4.16	76.57	35.14	62.85	25.51	61.16	25.39
Var	14.39	1.45	1204.02	780.35	586.94	246.03	83.02	9.13	433.51	249.92	727.40	330.40	8.78	11.74

To explore the robustness of the network, we perform five parallel experiments and differentiate them with IDs, as shown in the first column of TABLE IV. Adjacent columns display detailed results with headers "PSL", "PHL", "PLL" and "R", which correspond to the AUC approximation function defined from Equation (9) to Equation (12) respectively. Columns "Acc" and "FA" denote hotspot detection accuracy and false alarm in the percentage form respectively. In the last two rows, we compute the average and variance after Bessel's Correction for each column to estimate their performance and stability. It can be seen that although these training solutions behave reasonably good on some designs (see blue bars of Fig. 6), they all fail on challenging datasets. Furthermore, some of the existing solutions rely highly on the initial status of the neural networks and hence yield unstable performance, as listed in TABLE IV.

IV. Conclusion

In this paper, we propose a test layout generation toolkit built upon KLayout backbone. The toolkit supports both contact/via and uni-directional metal layer generation with certain design rule configurations. The tool can also be easily extended to other purposes including cell processing, complicated test pattern generation (e.g. Hilbert space patterns) and layout transformation. As a case study, we generate a set of DRC-clean via layout clips to evaluate the robustness and stability of state-of-the-art machine learning-based hotspot detectors. Here we introduce three recent hotspot detectors that adopt batch biased learning, smooth boosting and AUC optimization respectively, which all exhibit good performance on public metal layer design but unstable on the generated via designs. The study shows continuous research are necessary to increase the robustness and stability of machine learning models and hence prototype such frameworks into real back-end design flow.

Acknowledgments

This work is supported in part by Cadence Design Systems, Inc. and The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

References

[1] H. Zhang, B. Yu, and E. F. Y. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *Proc. ICCAD*, 2016, pp. 47:1–47:8.

[2] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting multi-layer layout hotspots with adaptive squish patterns," in *Proc. ASPDAC*, 2019, pp. 299–304.

[3] T. Matsunawa, J.-R. Gao, B. Yu, and D. Z. Pan, "A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction," in *Proc. SPIE*, vol. 9427, 2015.

[4] F. Yang, S. Sinha, C. C. Chiang, X. Zeng, and D. Zhou, "Improved tangent space based distance metric for lithographic hotspot classification," *IEEE TCAD*, vol. 36, no. 9, pp. 1545–1556, 2017.

[5] Y. Tomioka, T. Matsunawa, C. Kodama, and S. Nojima, "Lithography hotspot detection by two-stage cascade classifier using histogram of oriented light propagation," in *Proc. ASPDAC*, 2017, pp. 81–86.

[6] H. Geng, H. Yang, B. Yu, X. Li, and X. Zeng, "Sparse VLSI layout feature extraction: A dictionary learning approach," in *Proc. ISVLSI*, 2018, pp. 488–493.

[7] H. Yang, Y. Lin, B. Yu, and E. F. Y. Young, "Lithography hotspot detection: From shallow to deep learning," in *Proc. SOCC*, 2017, pp. 233–238.

[8] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," *IEEE TCAD*, vol. 38, no. 6, pp. 1175–1187, 2019.

[9] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *Proc. DAC*, 2019, pp. 146:1–146:6.

[10] W. Ye, Y. Lin, M. Li, Q. Liu, and D. Z. Pan, "LithoROC: lithography hotspot detection with explicit ROC optimization," in *Proc. ASPDAC*, 2019, pp. 292–298.

[11] D. Ding, B. Yu, J. Ghosh, and D. Z. Pan, "EPIC: Efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation," in *Proc. ASPDAC*, 2012, pp. 263–270.

[12] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: a deep learning approach," *JM3*, vol. 16, no. 3, p. 033504, 2017.

[13] M. Shin and J.-H. Lee, "Accurate lithography hotspot detection using deep convolutional neural networks," *JM3*, vol. 15, no. 4, p. 043507, 2016.

[14] A. J. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in *Proc. ICCAD*, 2012, pp. 349–350.

[15] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "DeePattern: Layout pattern generation with transforming convolutional auto-encoder," in *Proc. DAC*, 2019, pp. 148:1–148:6.

[16] S. Shim and Y. Shin, "Topology-oriented pattern extraction and classification for synthesizing lithography test patterns," *JM3*, vol. 14, no. 1, pp. 013 503–013 503, 2015.

[17] "KLAYOUT," https://www.klayout.de.

[18] R. A. Servedio, "Smooth boosting and learning with malicious noise," *Journal of Machine Learning Research*, vol. 4, no. Sep, pp. 633–648, 2003.

[19] W. Gao, R. Jin, S. Zhu, and Z.-H. Zhou, "One-pass auc optimization," in *Proc. ICML*, 2013, pp. 906–914.

[20] H. Steck, "Hinge rank loss and the area under the roc curve," in *European Conference on Machine Learning*. Springer, 2007, pp. 347–358.

[21] C. Rudin and R. E. Schapire, "Margin-based ranking and an equivalence between adaboost and rankboost," *Journal of Machine Learning Research*, vol. 10, no. Oct, pp. 2193–2232, 2009.

[22] L. Yan, R. H. Dodier, M. Mozer, and R. H. Wolniewicz, "Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic," in *Proc. ICML*, 2003, pp. 848–855.