This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Neural Network Classifier-Based OPC with Imbalanced Training Data

Suhyeong Choi, Seongbo Shim, and Youngsoo Shin, *Fellow, IEEE*

***Abstract*— Machine learning-guided optical proximity correction, called ML-OPC in this paper, has recently been proposed to alleviate long runtime of model-based OPC. ML-OPC using regression methods has been presented but with limited prediction accuracy. We propose NNC-OPC, in which a neural network classifier serves as a mask bias model. A few techniques are applied to enhance basic NNC-OPC: parameterization of layout segments using polar Fourier transform signals, dimensionality reduction through weighted principal component analysis, and sampling of training layout segments. Training segments are typically imbalanced over the range of mask biases, which may cause large prediction error for segments that appear less frequently. This is resolved by three techniques: synthetic data generation, class reorganization, and an adaptive learning rate. Experiments with NNC-OPC with all techniques applied indicate that prediction error of mask bias and training time are reduced by 29% and 80%, respectively, compared to state-of-the-art ML-OPC with regression methods.**

***Index Terms*— Optical proximity correction (OPC), neural network classifier, polar Fourier transform**

I. Introduction

ACCURATE model-based OPC (MB-OPC) is based on iterative lithography simulation and mask correction. As feature sizes scale down, it requires more iterations, and each lithography simulation takes longer, because it has to satisfy smaller edge placement error (EPE) tolerance. Thus, MB-OPC in 20nm technology takes about 180 times longer than it does in 40nm technology [2].

To reduce OPC runtime, pattern matching-based OPC has been proposed [3], [4], [5], but it is only applied to regular pattern block such as SRAM and memory cell array due to the lack of pattern coverage [6]. As machine learning has been applied to a few design-for-manufacturability (DFM) applications (e.g. lithographic hotspot detection [7], [8], etch proximity correction [9], and DSA guide pattern verification [10]), it has also been applied to OPC resulting in the concept of ML-OPC. The goal of ML-OPC is to reduce OPC runtime without resorting to iterative lithography simulations while EPE and resist quality are not sacrificed.

ML-OPC consists of training and testing phases. In the training phase shown in Fig. 1(a), a segment of the layout (in the context of the surrounding patterns) is parameterized by a set of parameters x. The parameters are input to a mask bias model (F), which yields a predicted value of mask bias ($F(x)$) for the segment. The model F is optimized so that $F(x)$ approaches a target bias obtained by conventional MB-OPC. During subsequent testing shown in Fig. 1(b), a layout segment to be corrected is parameterized using the same set of parameters that were used in training, and the resulting model F predicts the mask bias for the segment. In the testing process, although F predicts a mask bias of each segment one by one, the dependencies between neighboring segments are taken into account because the model has been trained with mask biases given by MB-OPC that itself considers those dependencies. Any segment of the layout can be tested in this way, and not just one of the segments used in training.

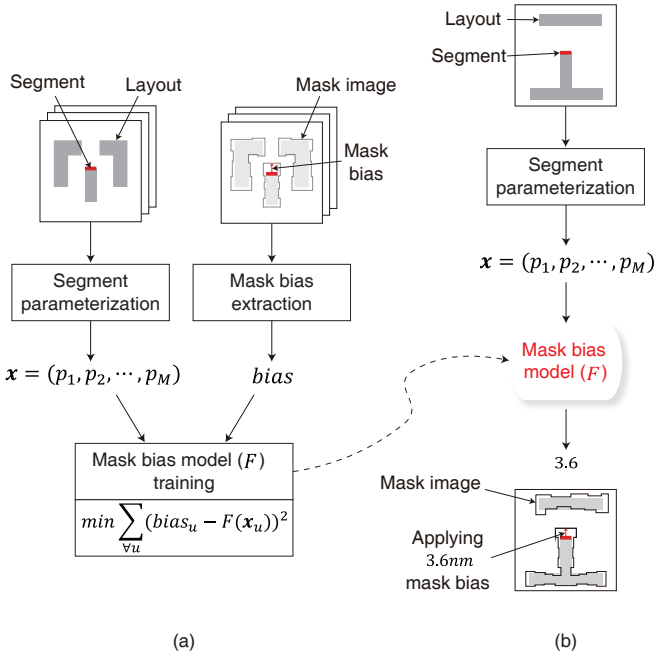

Fig. 1. ML-OPC: (a) training and (b) actual OPC.

The mask bias model F is a key component of ML-OPC, and can be based on regression or classification. Regression-based models [1], [11], [12], [13] are usually applied to edge-based OPC (see Fig. 2(a)), in which correction is performed by moving edge segments by the extent of a mask bias. This

Manuscript received May 16, 2017; revised September 20, 2017 and January 24, 2018. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A2A01008037). A preliminary version of this paper [1] was presented at SPIE Advanced Lithography, San Jose, CA, February 21-25, 2016.

The authors are with the School of Electrical Engineering, KAIST, Daejeon 34101, Korea (e-mail: suhyeung93@kaist.ac.kr; sbshim@kaist.ac.kr; youngsoo@ee.kaist.ac.kr). S. Shim is also with Samsung Electronics, Hwasung 18448, Korea.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

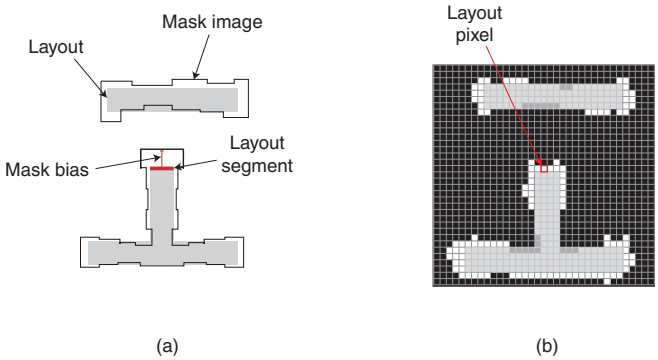

Fig. 2. Mask images of (a) edge-based and (b) pixel-based OPCs.

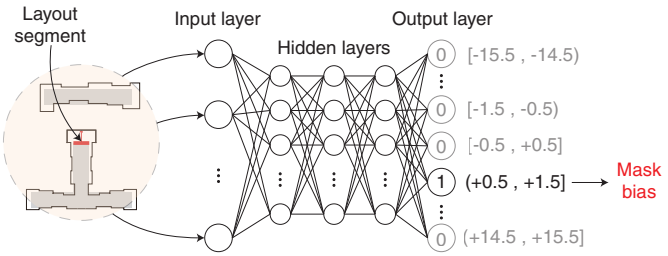

Fig. 3. NNC for mask bias prediction.

approach has shorter runtime, but its prediction accuracy is not satisfactory because conventional regression (either linear or logistic) cannot model the complicated trend of mask bias with sufficient accuracy; neural network regression easily overfits training segments due to its excessive flexibility [14]. The classification-based model [15], [16], [17] has been applied to pixel-based OPC (see Fig. 2(b)), in which a mask image is represented by an array of pixels. It has been shown to predict pixel values accurately (97% [17]), but only a few types of patterns have been investigated, due to the number of pixels required. To cover all the patterns in a modern VLSI layout, with thousands of types of patterns, would require some billions of pixels to be considered during model training, which is not realistic; the number of pixels can be supposed with a pixel and layout sizes of 1nm×1nm and some μm^2, respectively.

A. Neural Network Classifier-Based OPC (NNC-OPC)

We introduce a neural network classifier (NNC) as a mask bias model for edge-based OPC. As shown in Fig. 3, this NNC has multiple output nodes, each of which corresponds to a certain range of mask bias. The NNC maps an input segment to a single output node of value 1 while all the other output nodes have value 0; then the median of the corresponding range of mask bias becomes the predicted mask bias of the input segment.

We can classify training segments based on the amount of mask bias that they require for correction. If this classification is done on the basic of equal increments of bias, then some of the resulting classes will contain a lot of training segment, while some end up with many fewer; the class with a small number of member segments is called the 'small class', and the rest with a lot of segments is called the 'large class'. The goal of NNC training is to minimize the sum of errors in the predicted mask bias for all training segments, but this is likely to create an NNC which is more highly trained for segments in the large classes, while the small classes suffer increased prediction errors [18], [19], [20]. While this imbalance issue has been discussed only for binary classification in the field of DFM (e.g. lithographic hotspot detection) [21], we address this issue for multi-class application using the three techniques: (a) the generation of synthetic data to bulk up the small classes; (b) equalizing class size by breaking up some large classes and merging some of small classes; (c) reinforcing the training of small classes by introducing a new algorithm for training the NNC.

The accuracy of NNC-OPC and its runtime are affected by the way in which we prepare training segments and parameterize them. We use polar Fourier transform (PFT) signals as input parameters for our mask bias model; a concise set of PFT signals allows effective modeling of optical interference which affects the input segment, while popular parameters such as pattern densities and pixel values only model the geometry surrounding of the segment. We can further reduce the number of the parameters to increase training speed by principal component analysis (PCA); and it also improves accuracy by sorting out unnecessary parameters [8]. Another technique that we employ for runtime reduction is efficient sampling of training segments, which is popular in a few DFM applications [9], [22], [23]. The training segments are mapped to points in the parameter space, and clusters of points are replaced by a single representative point to avoid unnecessary computation.

B. Paper Organization

The remainder of this paper is organized as follows. In Section II, we show how layout segments are prepared for training as well as actual OPC. Specifically, parameterization using PFT signals, dimensionality reduction through weighted PCA, and the sampling of training segments are covered. In Section III, we address how an NNC is constructed by training and how it is used for actual OPC. The layout segments available for training may not be equally distributed over the range of mask biases that we want to predict, and we address this issue in Section IV. In Section V, we present the result of experiments designed to assess the proposed NNC-OPC, in terms of prediction accuracy and training time. Section VI concludes this paper.

II. Preparation of Layout Segments

A. Parameterization Using PFT Signals

The polar Fourier transform (PFT) is widely used in modeling optical diffraction and interference [24]. The real part of the PFT basis function is given by

$$\Psi_{nm}(r,\varphi) = J_n(r)\cos(m\varphi), \tag{1}$$

where the n-th Bessel function (J_n) and $\cos(m\varphi)$ are respectively its radial and angular components; n is the number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

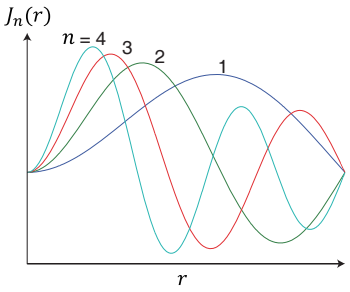

Fig. 4. Bessel functions with zero boundary condition.

critical points in the radial direction as shown in Fig. 4, and m is the number of periods in the direction of angular φ. PFT basis function becomes more complicated as n and m increase as shown in Fig. 5(a). PFT basis functions are orthogonal to one another due to the orthogonality of the Bessel and cosine functions.

Using PFT basis functions, the PFT signal (ϕ_k) of a layout segment can be expressed as follows:

$$\phi_k = \sum_{\forall x,y} \Psi_k(x,y) L(x,y), \tag{2}$$

where Ψ_k is the k-th PFT basis function in Cartesian coordinates (nm is replaced by k for a sake of simplicity), which has its origin at the center of the layout segment as shown in Fig. 5(b). The segment (and its surroundings) is represented by an array of pixels, which we call the layout image (L). A pixel in this image is 1 if it is within a polygon of the layout, or 0 otherwise. Note that an ambit of PFT basis function is close to optical diameter, so that influence of surrounding patterns on a segment of interest is all taken into account; optical diameter of 45nm technology, for instance, is around 2μm [25].

In commercial OPC tools, PFT signals are commonly used to determine the printability of a pattern on a wafer by approximating the variation in light intensity over layout segments [26], [27]. The intensity of light (I) at the center of a layout segment can be expressed as follows:

$$I = \sum_{\forall k} c_k \phi_k^2, \tag{3}$$

where c_k is the contribution of the k-th PFT basis function. PFT basis functions and corresponding c_ks are predetermined once lithographic condition is given. Critical dimensions (CDs) of some sample patterns on a wafer are measured from SEM image after actual lithography process, and c_ks are calibrated to yield simulated CDs as close as possible to measured ones [28], [29]. At this moment, some candidates of PFT basis functions with both n and m increasing from 0 are taken into account. Final PFT basis functions are then picked in descending order of corresponding c_ks until target accuracies of CDs are satisfied for the sample patterns [30]; at last, around 20 PFT signals are determined.

There are two reasons why we have chosen PFT signals as representative parameters of a layout segment. First, they are readily available from commercial OPC tools, which therefore can serve as a platform to build our proposed NNC-OPC. Second, a layout segment can be modeled by fewer parameters

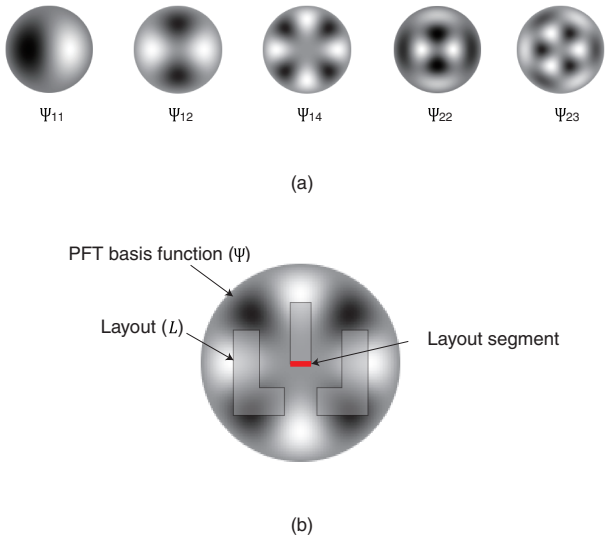

Fig. 5. (a) PFT basis functions and (b) obtaining PFT signal.

by using PFTs instead of pattern densities or pixel values, which allows the NNC to be trained more quickly [1].

B. Dimension Reduction by Weighted PCA (WPCA)

Even though the use of PFT signal significantly reduced the number of parameters, a further reduction can improve prediction accuracy and reduce training times [10]. PCA identifies orthogonal axes, along which the variance of the data is maximized, and projects the data on to the new axes; we call the axes principal components. Where the row vector $t_u \in \mathbb{R}^M$ is u-th item of training data in a space of M parameters, first principal component (PC_1) and a scalar obtained by projecting the row vector on to PC_1 are respectively defined as follows:

$$\mathrm{PC}_1 = \underset{||\mathrm{PC}||=1}{\arg\max} \sum_{\forall u} (t'_{u,1})^2, \tag{4}$$

and

$$t'_{u,1} = t_u \cdot \mathrm{PC}_1. \tag{5}$$

By inputting Equation (5) into Equation (4), we can find PC_1, and subsequently obtain the scalar. PC_2 which holds the second largest variance is obtained in the same manner. After subtracting $(\mathrm{PC}_1)^T$ from t_us, the component of the resulting data is found, under the constraint that it is orthogonal to PC_1. Further orthogonal components PC_3, PC_4, and so on are then found. Eventually, the variances of new principal components become very small, and so these components are redundant when some data are isolated from others because component values are very similar for all; thus, they are discarded reducing the dimensionality of the data-set.

PCA needs to be modified to deal with parameters with different levels of importance [31]. For example, we found that the variance of the 20-th PFT signals of some example training segments was larger than that of the 5-th PFT signals as shown in Fig. 6; while the 5-th PFT signals of the training segments are mostly in the range of [−0.1, 0.1], their 20-th PFT signals reside in the range of [−0.3, 0.3]. Therefore, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

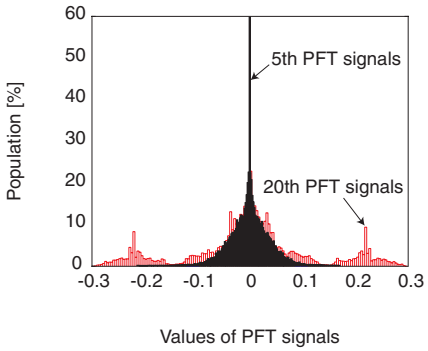

Fig. 6. Population distributions of two PFT signals.

20-th PFT signal makes a major contribution to PC_1 while the 5-th PFT signal is possibly discarded. However, this is not desirable because contribution of the 5-th PFT basis function to light intensity is about 14 times larger than that of the 20-th PFT basis function. In order to take different contributions of PFT basis functions into account, we therefore introduce a weighted PCA or WPCA. Each PFT signal ϕ_k is multiplied by its own weight:

$$\phi'_k = \sqrt{c_k}\phi_k, \tag{6}$$

where $\sqrt{c_k}$ is the contribution of k-th PFT signal in light amplitude unit. We can then apply the standard PCA to the weighted PFT signals. Since the inputs to a standard NNC should be in the range $[-1, 1]$ [32], [33], the PCs are normalized by using the maximum value of all PCs.

C. Sampling of Training Layout Segments

Layouts supplied for training may contain some segments with similar surrounding patterns. For instance, the layout of a memory circuit includes many repeated patterns which produce very similar PFT signals and PCs. Training an NNC with many of similar segments wastes training time, and it also distorts the training process, so that the mask bias is accurately predicted for those segments but not others [9].

We address this problem with a systematic method of sampling layout segments used in training, shown diagrammatically in Fig. 7. Each segment is mapped to a point in an M-dimensional space, where M is the number of PCs produced by WPCA. A data point in this space is then selected at random, and all the other data points within some Cartesian radius are discarded because they are assumed to be substituted by the selected point. This process is repeated until no data points remain. The NNC is then trained using only the representative points, and the prediction accuracy for the whole training segments is recorded. This process is repeated with different radii, and the radius that produce the highest accuracy is used to sample the layout segments.

There are other methods that may serve our purpose, e.g. k-means clustering and ordering points to identify the clustering structure (OPTICS) [36], which is a popular density-based clustering method. However, the coverage of representative points, which affects accuracy, is controllable only in our sampling method, which is why we propose a new method.

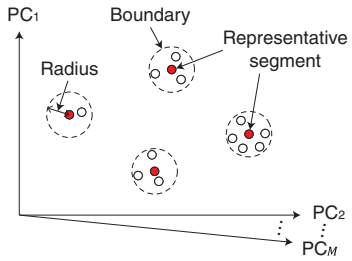

Fig. 7. Sampling of representative training layout segments.

Comparision of three sampling methods are presented in Section V-C.

III. NNC-Based Mask Bias Model

A. NNC Construction

Our NNC consists of neurons (corresponding to nodes in Fig. 3) and synapses (corresponding to edges). Input nodes receive M parameters extracted from the layout segment to be corrected. Their values are propagated to every node in the first hidden layer through weighted edges, which multiply the values of the propagated signal. The weighted signals are received by each hidden node and summed; if the total is larger than a threshold, the hidden node outputs 1, but otherwise 0. Thresholding can be replaced by a sigmoid function $f(s)$, which outputs a real number between 0 and 1:

$$f(s) = \frac{1}{1+e^{-(s-\theta)}}, \tag{7}$$

where s is the sum of the weighted signals, and θ is the threshold. Each node in the first hidden layer then propagates its output value to every node in the next layer, and this process is repeated until the output layer is reached.

Each node in the output layer is assigned to a range of mask bias (e.g. $[-0.5, +0.5]$) as illustrated in Fig. 3. These ranges are obtained by running MB-OPC on the same training segments, and then dividing the whole range of the result into intervals of equal length. The number of intervals determines the number of output nodes which affects training time. Our NNC has 31 output nodes spanning mask bias values between -15.5nm and $+15.5$nm with a 1nm increment. If a mask bias of a training layout segment is 0.3nm, for instance, the output node with $[-0.5, +0.5]$ as its interval is trained to output 1 while 0 outputted by the other nodes.

B. NNC Training and Actual OPC

The edge weights and threshold values in our NNC are optimized to minimize prediction errors over all the training layout segments. The error[1] in the predicted mask bias of a layout segment is expressed as follows:

$$err = \sum_{\forall i}(O_i - Y_i)^2, \tag{8}$$

[1]The choise of error function affects accuracy and training time [35], [**?**]. We choose square error function due to its best accuracy over other functions, such as cross-entropy.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

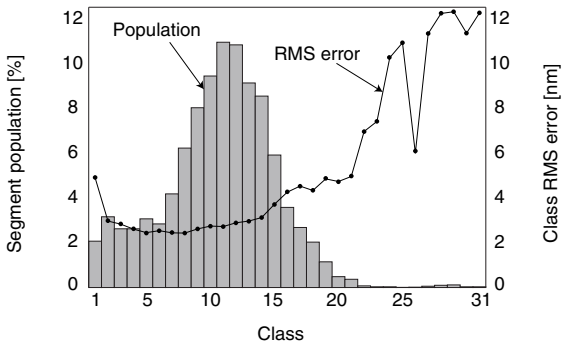

Fig. 8. Population of each class vs its RMS error.

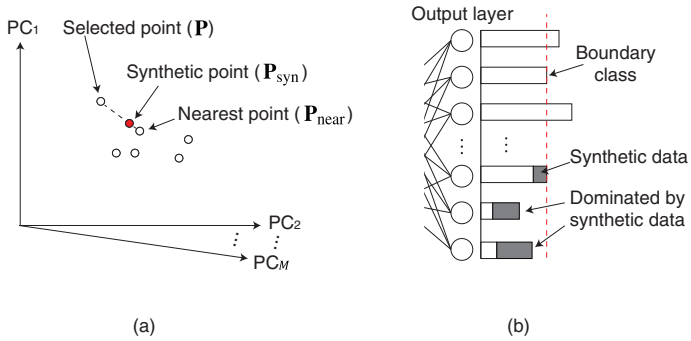

Fig. 9. (a) Synthetic data generation and (b) population histogram of training data after synthetic data generation.

where O_i is the value of the i-th output node, and Y_i is its target value obtained from MB-OPC. Our goal is then to minimize the sum of Equation (8) over all segments. This could be done by gradient descent method [37], but it requires too many evaluations of NNC for all the training segments to obtain the derivative of the total error.

We use stochastic gradient descent (SGD) instead, in which the NNC only needs to be evaluated for a randomly selected training segment; its weights and thresholds are adjusted to reduce the error in its prediction. We repeat this for other training segments, and if every training segment is used once, first epoch ends. Second, third, and so on epochs then carry over until the sum of the errors is no longer reducing or an epoch count is exceeded [38]. The corrected weight (w) and threshold (θ) by a segment are determined using the standard gradient descent method:

$$w_{ij} := w_{ij} - \eta \frac{\partial err}{\partial w_{ij}} \tag{9}$$

and

$$\theta_j := \theta_j - \eta \frac{\partial err}{\partial \theta_j}, \tag{10}$$

where w_{ij} denotes the weight on the edge connecting the i-th and j-th nodes, and θ_j is the threshold of the j-th node. The coefficient η is the learning rate; if it is too small, an excessive number of iterations will be required, and if it is too large, divergence is likely due to excessive changes in weights and thresholds.

The optimized NNC is used as our mask bias model for actual OPC, which we call NNC-OPC. The parameters of each layout segment to be corrected are submitted to the NNC, which outputs a value between 0 and 1 for each node in its output layer. The output node with the largest value is identified, and the predicted mask bias is median of its assigned mask bias range.

IV. Dealing with Imbalanced Training Data

Each training layout segment is mapped to one output node of the NNC depending on its predicted mask bias. We will call a set of training layout segments that are mapped to the same output node a class. Fig. 8 shows the percentage of segments in each class (the first y axis), using our NNC with 31 output nodes. Some classes are well populated, and we would expect that the NNC is trained to classify layout segment in their classes well; thus, corresponding output nodes carry smaller error. The opposite is true for classes that are not well populated, which are called small classes. This presumption is by the root-mean-square (RMS) error in each class as shown in Fig. 8 (the second y axis).

An easy way to reduce the RMS error of small classes with fewer members would be to replicate their members, but this inherently causes overfitting [39]. We propose three techniques to address this problem, namely synthetic data generation, class reorganization, and an adaptive learning rate.

A. Synthetic Data Generation

We can increase the size of small classes by adding synthetic data to them. The small classes are defined by both size and RMS error of class. If the RMS error of a class is larger than the average RMS error of all classes, then that class is considered to be small class, because it is inadequately trained. All classes which are less populated than the largest 'small class' are also tagged as small classes. The remaining classes are all regarded as large classes, and the class of the smallest population shall be the boundary class, i.e. its population is considered to be the minimum population necessary for adequate training.

Synthetic data is generated for a small class as follows: All the segments in the class are mapped to points in PC space (see small circles in Fig. 9(a)); we randomly pick a point ($\mathbf{P}$) and then find its nearest neighbor ($\mathbf{P}_{\text{near}}$) using Euclidean distance as a measure; a synthetic data point ($\mathbf{P}_{\text{syn}}$) is then generated at a random location along the line connecting $\mathbf{P}$ and $\mathbf{P}_{\text{near}}$. This process can be repeated until the class becomes as large as the boundary class; however, classes dominated by synthetic data can reduce the effectiveness of the overall training process so that RMS error of large classes increases. Therefore, we limit the proportion of synthetic data that we add to a class. The proportion is empirically determined (this will be demonstrated in Section V-D).

B. Class Reorganization

We choose the classes to be reorganized by comparing their size to the average. If $\bar{P}$ is the average size of a class as given

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

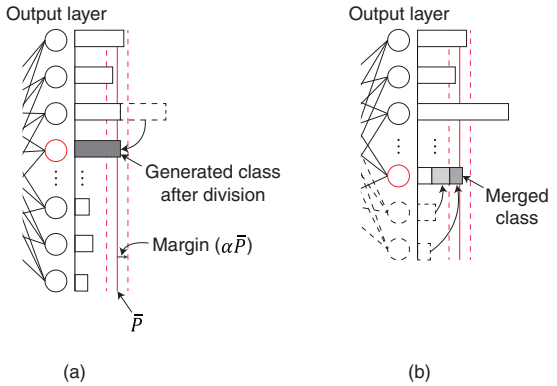

Fig. 10. Class reorganization: (a) dividing a large class and (b) merging small classes.

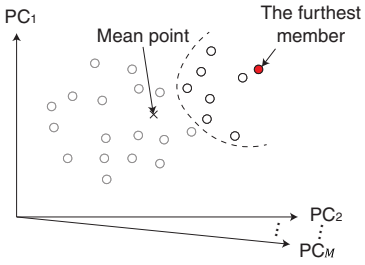

Fig. 11. Collecting members of large class to generate new class.

by

$$\bar{P} = \frac{\#\,Training\ data}{\#\,Classes}, \tag{11}$$

then classes larger than $(1+\alpha)\bar{P}$ are considered for splitting (Fig. 10(a)), and classes smaller than $(1-\alpha)\bar{P}$ are considered for combination (Fig. 10(b)); α is a margin. Large classes are evenly divided into small classes, which must be sufficiently large to fall into the central band of classes around the average; otherwise they would become candidates for recombination. Since the minimum number of classes that can be created by division is 2, we see

$$\frac{(1+\alpha)\bar{P}}{2} > (1-\alpha)\bar{P}; \tag{12}$$

obviously the $\bar{P}$s are canceled out, and the rest can be rearranged easily enough to yield $\alpha > 1/3$. We do not want to make α much larger than that, in order to keep the number of output nodes in our NNC as low as possible. After the number of members in a new class is determined, each new class is split out from a large class as follows. The mean point of all members of a class is first obtained in PC space as illustrated in Fig. 11, and the furthest member from the mean point is identified. We then collect members in ascending order of distance from the furthest member to make the first partition, equivalently a new class. Another partition is obtained by iterating these after excluding already collected members; and the last partition consists of the left members.

Fig. 10(b) shows how classes which are smaller than $(1-\alpha)\bar{P}$ are merged to create classes in the central band, with sizes between $(1-\alpha)\bar{P}$ and $(1+\alpha)\bar{P}$. Our procedure is to merge the smallest class with the largest. This creates a new class

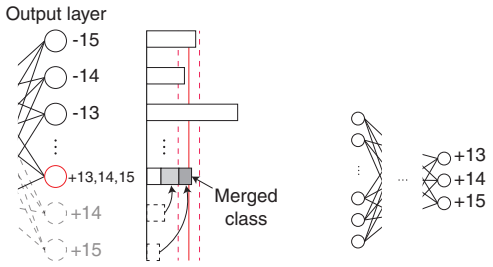

Fig. 12. (a) Main NNC after class reorganization and (b) sub-NNC for merged class.

that cannot be larger than $(1+\alpha)\bar{P}$ provided that α is at least $1/3$, because $2(1-\alpha)\bar{P} < (1+\alpha)\bar{P}$. The new class is a subject for further combination if the class can still be smaller than $(1+\alpha)\bar{P}$ after another combination; we do not stop merger when the new class becomes just larger than $(1-\alpha)\bar{P}$ because we can further reduce the number of classes (or output nodes). This process of merging the smallest and largest classes is repeated until no classes smaller than $(1-\alpha)\bar{P}$ remain or, as may happen, a single class smaller than $(1-\alpha)\bar{P}$ remains. In this case, the last small class is merged with the smallest class in the central band, even though this may produce a class somewhat larger than $(1+\alpha)\bar{P}$.

Equalizing the sizes of classes, using the process which we have just described, is effective in producing a more balanced and accurate NNC. However, the classes and ranges of mask bias in new NNC are no longer one-to-one correspondence. Classes which have been splitted to reduce their size pose no problem: each new class has the same range of mask biases as the original class, and all that happens is that more than one output node now indicate the same range as shown in Fig. 12(a). However, classes which have been combined will now correspond to several ranges of output bias (see the red-colored node in Fig. 12(a)). We create a sub-NNC for each merged class, which has the same inputs as the main NNC, but its outputs are the original classes that make up the merged class as illustrated in Fig. 12(b). Each sub-NNC is trained using the data that belongs to corresponding merged class; if sub-NNC confronts imbalanced training again, the generation of synthetic data can be applied.

C. Adaptive Learning Rate

An NNC which is trained using SGD is likely to have poor prediction accuracy for small classes because the weights w and thresholds θ of the NNC are optimized relatively fewer times for them compared to large classes. We introduce an adaptive learning rate η' to address this issue. A high learning rate is required to be applied when Equations (9) and (10) are used to train a small class, which effectively correct w and θ. The adaptive learning rate is given by

$$\eta_l' = \sqrt[R]{\frac{P_{\max}}{P_l}}\eta, \tag{13}$$

where $P_{\max}$ and P_l are respectively the size of the largest and the l-th classes; and η_l' indicates the new learning rate for the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I
COMPARISON OF NNR-OPC AND NNC-OPC IN PREDICTION ACCURACY

Layout	RMS error (nm)			
	NNR [1]	NNC (sampling, WPCA)	NNC (no sampling, WPCA)	NNC (no sampling, no WPCA)
Training	3.1	2.5	2.1	2.1
Layout 1	3.5	2.4	2.8	3.5
Layout 2	3.6	2.8	3.2	4.5
Layout 3	3.3	1.9	2.2	2.1
Layout 4	3.8	3.1	3.5	5.1
Layout 5	3.8	3.1	3.5	5.1
Layout 6	3.2	1.9	2.1	2.1
Layout 7	3.3	1.9	2.1	2.0
Layout 8	3.5	2.5	2.8	3.6
Layout 9	3.7	2.8	3.2	4.4
Layout 10	3.7	2.9	3.3	4.7
Average	3.5	2.5	2.9	3.7

l-th class. Relaxation factor R is adopted to improve convergence by reducing overshooting near the local minimum. The value of R, which is of course greater than 1, is determined empirically, and this will be discussed in Section V-F.

Note that adaptive learning rate is different from replicating members of small classes. For instance, 1.2 times of enhancement in learning for a small class cannot be implemented by simple replication while adaptive learning rate can do; the number of small class members can be 1.2 times by replicating some of the members, but in fact only learning of replicated members is enhanced by two times, yet that of the others is not. We demonstrate this in Section V-F.

V. EXPERIMENTS

NNC-OPC is implemented using a commercial OPC package [27] for PFT signal extraction, MATLAB for both WPCA implementation and training segments sampling, and Python for NNC construction. We prepare 50,000 layout segments for NNC training and 960,845 segments from 10 different layouts with similar area for testing or actual OPC; all layouts are metal 1 layers in 20nm technology and the target lithography process is 1.2NA ArF immersion lithography with annular illumination. Each layout segment is parameterized by 10 PFT signals, which are then reduced to 5 PCs. Our NNC has 5 nodes (corresponding to 5 PCs) in its input layer, 12 nodes in each of three hidden layers, and 31 nodes in its output layer.

A. Assessment of NNC-OPC

We compare our NNC-OPC with the most recent ML-OPC based on neural network regression (NNR-OPC) [1], which outperforms [11] and [13] both in accuracy and training time. This NNR-OPC requires 10 nodes in its input layer to accommodate 10 PFT signals, 10 nodes in each of its three hidden layers, and a single output node. It is trained using the same 50,000 layout segments that we prepare for our NNC-OPC. Conventional SGD-guided training is used for both NNR- and NNC-OPCs.

Table I shows the RMS error of the predicted mask bias for the segments in the 10 test layouts, as well as those in the training layout. Our NNC-OPC with sampling and WPCA

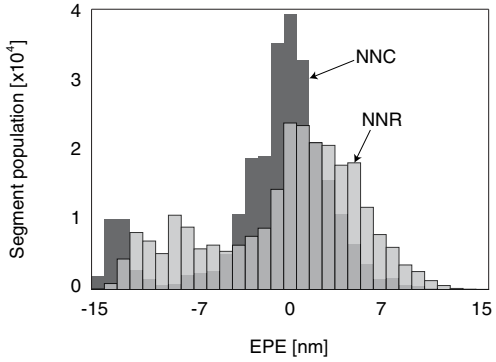

Fig. 13. EPE distributions of NNR- and NNC-OPCs (for layout 1).

TABLE II
COMPARISON OF NNR-OPC AND NNC-OPC IN TRAINING TIME

	NNR [1]	NNC (sampling, WPCA)	NNC (no sampling, WPCA)	NNC (no sampling, no WPCA)
Training time (min)	101	20	120	176

produces lower errors than NNR-OPC for all the layouts. Over all the test layouts, NNC-OPC has an average error of 2.5nm, while the error for NNR-OPC is 3.5nm. Note that, NNR-OPC yields larger error for test layouts (3.5nm) than for training layout (3.1nm) due to its overfitting, while NNC-OPC achieves the same level of errors (2.5nm). RMS error is almost linearly translated to EPE. Average EPE of NNR- and NNC-OPCs are 3.9nm and 2.6nm, respectively, and as shown in Fig. 13, the EPE distribution of NNC-OPC is more centered compared to that of NNR-OPC.

Training time is compared in Table II. NNC-OPC takes significantly less time (20min) compared to NNR-OPC (101min). This can be explained by the fact that NNC-OPC uses 5 parameters while NNR-OPC uses 10, and the number of data points that is actually used in training is also smaller in NNC-OPC because of the sampling of training layout segments (Section II-C).

B. Impact of WPCA and Sampling of Training Layout Segments

WPCA reduceds 10 PFT signals into 5 PCs. The sampling techniques described in Section II-C reduces 50,000 training layout segments to 11,983. The results in the last two columns of Tables I and II are designed to identify the contributions of these processes.

As shown in Table II, WPCA reduces training time by 32% (from 176 to 120) due to smaller number of input parameters, so the smaller number of input nodes. WPCA reduces the average RMS error from 3.7nm to 2.9nm; this is because WPCA yields only a handful of critical parameters that help distinguish layout segments more clearly, so classification by NNC with WPCA becomes easier.

Applying the sampling allows further reduction of runtime by 57% ($\frac{120-20}{176}$) because the number of training segments is reduced to only 24%. Similar training segments which are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

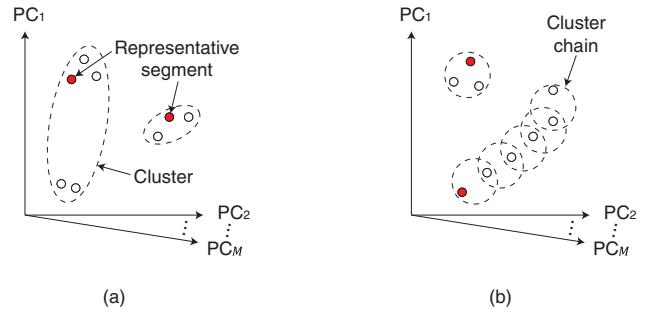

Fig. 14. Sampling with (a) *k*-means clustering and (b) OPTICS.

TABLE III
COMPARISON OF SAMPLING METHODS

	Proposed	*k*-means	OPTICS
Sampling time (min)	0.3	313	0.5
# Sampled training data	11,983	10,000	6,526
Training time (min)	20	16	13
RMS error (nm)	2.5	2.6	2.8

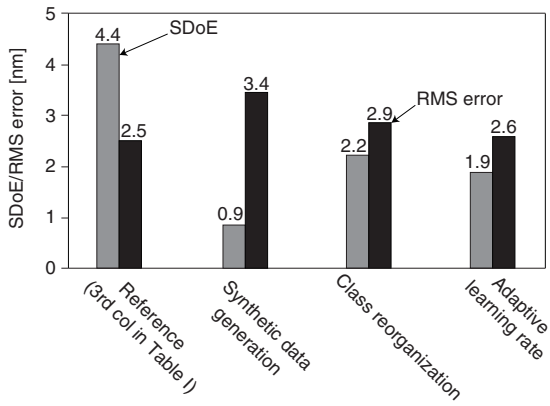

Fig. 15. SDoE and RMS error comparisons between four NNC-OPCs.

frequently learned so with low error are eliminated by the sampling. Thus, RMS error of training segments increases by 0.4nm rather than decreases. However, this helps covering more diverse segments avoiding overfitting the similar segments, which yields further reduction of RMS error for test layouts by 0.4nm.

C. Comparison with Other Sampling Methods

We implement two other popular clustering-based sampling methods: *k*-means clustering and OPTICS. The same 50,000 layout segments, which are represented by 5 PCs, are used for sampling, and the same NNC in Section V-A is used for training.

In *k*-means clustering, data points are grouped into *k* number of clusters, while sum of within-cluster variances is minimized. The member that is closest to the mean point of all members in a cluster becomes the representative of that cluster. In this method, some data points that are far apart may be grouped together as illustrated in Fig. 14(a), so that the number of clusters is kept to *k*. In our implementation, *k* is empirically set to 10,000, which results in RMS error and training time that are comparable to those of proposed method as shown in Table III. Sampling itself however takes more than 5 hours; if 20,000 are sampled out (instead of 10,000), sampling time increases to about 30 hours, which raises a scalability issue.

OPTICS sequentially clusters data points based on absolute distance from chosen points. At first, a point of interest is randomly chosen, and its closest neighbor is clustered together if the closest one is within a certain distance threshold. The closest one then becomes the point of interest, and the same process is repeated until all points are contained in clusters. The first interesting point of each cluster is designated as a representative. To find an optimum distance threshold, we sweep the threshold from 0.01 to 0.2; the optimum threshold turns out to be 0.1, which gives the shortest training time while the smallest RMS error can be kept. OPTICS sometimes yields a big cluster chain (see Fig. 14(b)), whose representative may hardly cover all the points in the chain. This may reduce the number of sampled data and training time (see third and fourth rows in Table III). RMS error however increases (see the last row), because a representative of a big cluster chain may hardly cover all the points in the chain. In ML-OPC, disadvantage of the accuracy loss is more significant rather than the benefit of short training time, because poor accuracy negatively affects manufacturing yield due to lithography defect, while training is usually performed only once. Note that in the same context, both WPCA and sampling method in Section V-B also aim to reduce training time while keeping (or even improving) accuracy.

D. Assessment of Synthetic Data Generation

We examine the effect of the synthetic data generation by training another NNC with new data set including sampled data (in Section V-B) and synthetic data. Out of 31 classes, 22 small classes are augmented in which the maximum proportion of synthetic data in each class is set to 15x; 540 synthetic data are in turn added in total. We average out the RMS errors in the predictions for each class across the 10 test layouts. We then take the standard deviation of the resulting 31 average class errors (SDoE) as a measure of the equality of results across classes.

The first two pairs of bars in Fig. 15 are reference SDoE and RMS error, which is obtained from NNC-OPC in Section V-A without any techniques to handle imbalanced data. SDoE is reduced by 3.5nm due to the synthetic data generation (see first two gray bars). This however comes at the cost of 0.9nm prediction accuracy loss (see first two black bars) because adding synthetic data reduces RMS errors of small classes but increases those of large classes, which dominate overall prediction accuracy.

We empirically determined maximum proportion of synthetic data by training and testing NNCs while varying the proportion. The results obtained by adding different amounts of synthetic data, up to maxima of between 5x and 20x the original amount of data in each class, are shown in Fig. 16. The benefit of adding synthetic data in terms of SDoE is clear, but it tails off. RMS error increases as the proportion becomes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

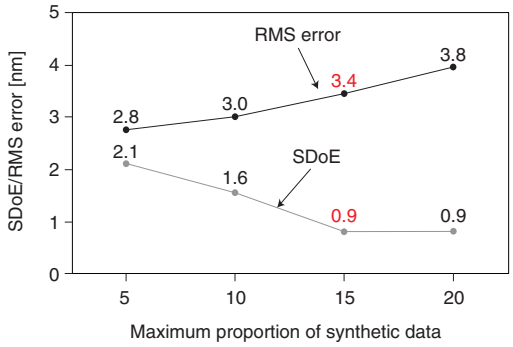

Fig. 16. SDoE and RMS error for NNC-OPCs with various maximum proportion of synthetic data.

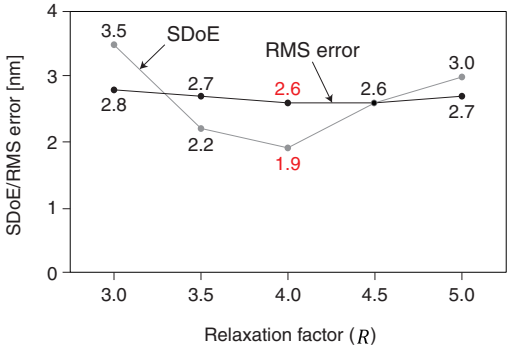

Fig. 17. SDoE and RMS error along relaxation factor (R).

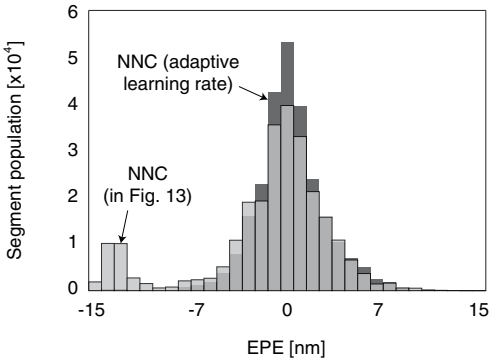

Fig. 18. EPE distributions of NNC-OPCs with and without adaptive learning rate (for layout 1).

high, which can be attributed to overfitting due to excessive number of similar data. If SDoE no longer decreases with increasing the proportion, we pick the numbers (here it is 15) as our best choice.

E. Assessment of Class Reorganization

We will now look at the impact of class reorganization by applying it to reference NNC. Starting with 31 classes, 8 large classes are divided into 21 classes, and 13 small classes are merged into a single class; producing a new set of 32 classes, all in the central band (in Fig 10). We then construct a new main NNC with 32 output nodes, and train it with all the sampled training data which are recategorized into the new 32 classes. In order to find the mask bias for the segment belonging to the merged output node, we construct a sub-NNC with 13 output nodes, and train it with the data from the merged class. This main NNC, which has only one more output node than the reference NNC, takes less than 1% longer to train. Training the sub-NNC is quick because the 13 small classes which are merged only have 96 training segments between them.

Once the two NNCs are trained, testing segments from 10 test layouts are first submitted to the main NNC. If a test segment is classified to the merged class, it is subsequently inputted to the sub-NNC, which in turn determines the predicted mask bias of the segment. As shown in Fig. 15, class reorganization method reduces the SDoE by 2.2nm, but the RMS error increases by 0.4nm, compared to the reference.

F. Assessment of Adaptive Learning Rate

We construct another NNC, like the reference NNC, but train it using an adaptive learning rate with a relaxation factor $R = 4$, so learning rates increase up to 462%. This increases the average RMS error by 0.1nm but significantly reduces the SDoE, by 2.5nm as shown in Fig. 15; what is not shown in the figure is that the maximum of class RMS error falls by 50%. Its impact is also shown in EPE distribution; an hump near EPE of -14nm is shifted to the center as shown in Fig. 18. The training time is unchanged because the complexity of the NNC and the amount of training data remain the same.

While the techniques described in Section V-D and V-E involve the additional endeavor of generating synthetic data or constructing sub-NNCs, this method only requires a small change to the training procedure.

We attempt this method with several relaxation factors (R) to figure out the best value. As shown in Fig. 17, RMS error plot is almost flat across various R, but SDoE forms convex trend. The fact is observed that RMS error of large classes are almost intact but those of small classes change. With too small R, divergence happened for the small classes because adaptive learning rates of the small classes are large, so SDoE is high. In the opposite situation, all adaptive learning rates becomes close to 1, so accuracies of small classes are not effectively improved. Thus, we pick R at the minimum point of convex SDoE trend.

To show the advantage of this method, we also compare it to simply replicating members of small class. In our experiment, members of the smallest class should be replicated by 3.62 times, which is corresponding to 462% learning rate from Eq. (13). All members of the smallest class are first replicated by 3 times, and then 62% of them are randomly picked and replicated. We perform this for all small classes, and construct NNC again. It also reduces SDoE to 1.9nm, which is same SDoE in adaptive learning rate (see right most gray bar in Fig. 15), but RMS error turns out to be 3.2nm, which is 23% larger than that of adaptive learning rate.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824255, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

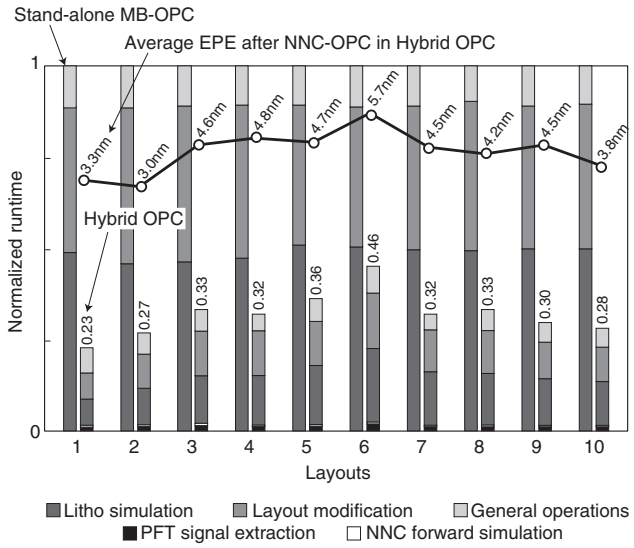

Fig. 19. Runtime comparison between stand-alone MB-OPC and hybrid NNC- and MB-OPC. Both methods are applied until the final maximum EPE becomes less than 1nm.

G. Hybrid NNC- and MB-OPC

Proposed NNC-OPC is much faster than conventional MB-OPC, but RMS error may not be small enough. We propose hybrid NNC- and MB-OPC, in which NNC-OPC is applied first to reduce EPE substantially and MB-OPC is applied next with only a few iterations to further reduce EPE. We apply stand-alone MB-OPC and the hybrid OPC to 10 test layouts in metal 1 layer, and runtime to meet the EPE criteria (maximum EPE < 1nm) is compared as shown in Fig. 19. While the MB-OPC takes 40.2 hours (with 16 iterations) on average, the hybrid OPC takes 12.9 hours, which is about 68% runtime reduction (see short bars). In the hybrid OPC, less than 2% of runtime is spent for NNC-OPC including PFT signal extractions and NNC forward simulations, and most is spent for subsequent MB-OPC (with only 4 iterations) including lithography simulation, mask layout modification, and general operations, e.g. arithmetic operations and data input and output. Layout 1, 2, and 10 benefit most because of the small remaining EPEs after NNC-OPC (see a line with white circles), which can be eliminated with less iterations in subsequent MB-OPC. Note that NNC-OPC result is from the rightmost two bars in Fig. 15.

VI. Conclusion

ML-OPC has been considered to be promising to replace conventional MB-OPC. A standard ML-OPC based on regression methods however has shown limited prediction accuracy. We have proposed NNC-OPC, in which NNC serves as a mask bias model. A number of techniques have been proposed to enhance NNC-OPC. These include parameterization of layout segments using PFT signals, dimensionality reduction through weighted PCA, and sampling of training layout segments. Training segments are typically not well distributed over the range of mask biases, which causes large prediction error for segments that appear less frequently. This has been resolved by synthetic data generation, class reorganization, and adaptive learning rate. Proposed NNC-OPC has been demonstrated using layouts in metal 1 layer of 20nm node. Experiments have indicated that NNC-OPC allows us to reduce RMS error of predicted mask bias and training time by 29% and 80%, respectively, compared to state-of-the-art ML-OPC.

The NNC-OPC has a potential to be applied to pixel-based OPC, which is left as a future work.

References

[1] S. Choi, S. Shim, and Y. Shin, "Machine learning (ML)-guided OPC using basis functions of polar Fourier transform," in *Proc. SPIE Advanced Lithography*, Mar. 2016, pp. 1–8 (97800H).

[2] Samsung Electronics Corp. OPC principal engineer *personal communication*, May 2016.

[3] M. C. Simmons, "Pattern matching optical proximity correction," Mar. 2014, US Patent 8,683,394.

[4] P. Verma, F. Batarseh, S. Somani, J. Wang, S. McGowan, and S. Madhavan, "Pattern-based pre-opc operation to improve model-based OPC runtime," in *Proc. SPIE Photomask Technology*, Oct. 2014, pp. 1–11 (923506).

[5] P. Verma, S. Somani, Y. Y. Ping, P. Pathak, R. S. Ghaida, C. P. Babcock, F. Batarseh, J. Wang, S. Madhavan, and S. McGowan, "Hybrid OPC flow with pattern search and replacement," in *Proc. SPIE Advanced Lithography*, Mar. 2015, pp. 1–18 (942611).

[6] Samsung Electronics Corp. OPC senior engineer *personal communication*, Sep. 2017.

[7] D. Ding, J. A. Torres, and D. Z. Pan, "High performance lithography hotspot detection with successively refined pattern identifications and machine learning," *IEEE Trans. on CAD*, vol. 30, no. 11, pp. 1621–1634, Nov. 2011.

[8] J.-R. Gao, B. Yu, and D. Z. Pan, "Accurate lithography hotspot detection based on PCA-SVM classifier with hierarchical data clustering," in *Proc. SPIE Advanced Lithography*, Mar. 2014, pp. 1–10 (90530E).

[9] S. Shim and Y. Shin, "Machine learning-guided etch proximity correction," *IEEE Trans. on Semiconductor Manufacturing*, vol. 30, no. 1, pp. 1–7, Feb. 2017.

[10] S. Shim and Y. Shin, "Fast verification of guide-patterns for directed self-assembly lithography," *IEEE Trans. on CAD*, vol. 36, no. 9, pp. 1522–1531, Sep. 2017.

[11] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical Bayes model," in *Proc. SPIE Advanced Lithography*, Mar. 2015, pp. 1–10 (94260X).

[12] M. Jeong and J. W. Hahn, "Prediction of biases for optical proximity correction through partial coherent identification," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 15, no. 1, pp. 1–12, Mar. 2016.

[13] A. Gu and A. Zakhor, "Optical proximity correction with linear regression," *IEEE Trans. on Semiconductor Manufacturing*, vol. 21, no. 2, pp. 2263–271, May 2008.

[14] T. R. Ender and S. Balestrini-Robinson, "Surrogate modeling," in *Modeling and Simulation in the Systems Engineering Life Cycle.* Springer, 2015, pp. 201–216.

[15] T. Shah and O. Dabeer, "Fast inverse lithography using machine learning," in *Proc. Indian Workshop on Machine Learning*, Jul. 2013, pp. 21–22.

[16] K.-S. Luo, Z. Shi, X.-L. Yan, and Z. Geng, "SVM based layout retargeting for fast and regularized inverse lithography," *Journal of Zhejiang University SCIENCE C*, vol. 15, no. 5, pp. 390–400, Apr. 2014.

[17] R. Luo, "Optical proximity correction using a multilayer perceptron neural network," *Journal of Optics*, vol. 15, no. 7, pp. 708–713, Jun. 2013.

[18] Y. L. Murphey, H. Guo, and L. A. Feldkamp, "Neural learning from unbalanced data," *Applied Intelligence*, vol. 21, no. 2, pp. 117–128, Sep. 2004.

[19] G. H. Nguyen, A. Bouzerdoum, and S. L. Phung, "A supervised learning approach for imbalanced data sets," in *Proc. Int. Conf. on Pattern Recognition*, Dec. 2008, pp. 1–4.

[20] R. Batuwita and V. Palade, "Class imbalance learning methods for support vector machines," in *Imbalanced Learning: Foundations, Algorithms, and Applications.* Wiley & Sons, 2013 pp. 83–99.

[21] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: a deep learning approach," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 16, no. 3, pp. 1–14, Aug. 2017.

[22] S. Shim and Y. Shin, "Topology-oriented pattern extraction and classification for synthesizing lithography test patterns," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 14, no. 1, pp. 1–13, Jan. 2015.

[23] T. Matsunawa, B. Yu, and D. Z. Pan, "Laplacian eigenmaps-and Bayesian clustering-based layout pattern sampling and its applications to hotspot detection and optical proximity correction," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 15, no. 4, pp. 1–11, Oct. 2016.

[24] Q. Wang, O. Ronneberger, and H. Burkhardt, "Fourier analysis in polar and spherical coordinates," Albert-Ludwigs-UniversitÃd't Freiburg, Institut fÃijr Informatik, Tech. Rep., 2008.

[25] N. Cobb and D. Dudau, "Dense OPC and verification for 45nm," in *Proc. SPIE Advanced Lithography*, Mar. 2006, pp. 1–7 (61540I).

[26] Synopsys, "Progen," Dec. 2013.

[27] Synopsys, "Proteus," Dec. 2013.

[28] M. Feldman, *Nanolithography: the art of fabricating nanoelectronic and nanophotonic devices and systems.* Woodhead, 2014.

[29] J. Li, X. Li, R. Lugg, and L. S. Melvin III, "Kernel count reduction in model based optical proximity correction process models," *Japanese Journal of Applied Physics*, vol. 48, no. 6, pp. 1–5, Jun. 2009.

[30] P. Gong, S. Liu, W. Lv, and X. Zhou, "Fast aerial image simulations for partially coherent systems by transmission cross coefficient decomposition with analytical kernels," *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, vol. 30, no. 6, pp. 1–7, Nov. 2012.

[31] H. H. Yue and M. Tomoyasu, "Weighted principal component analysis and its applications to improve FDC performance," in *Proc. Conf. on Decision and Control*, Dec. 2004, pp. 4262–4267.

[32] H. Moon and P. J. Phillips, "Computational and performance aspects of PCA-based face-recognition algorithms," *Perception*, vol. 30, no. 3, pp. 303–321, Mar. 2001.

[33] H.-B. Deng, L.-W. Jin, L.-X. Zhen, and J.-C. Huang, "A new facial expression recognition method based on local Gabor filter bank and PCA plus LDA," *Int. Journal of Information Technology*, vol. 11, no. 11, pp. 86–96, Nov. 2005.

[34] M. Daszykowski, B. Walczak, and D. L. Massart, "Looking for natural patterns in analytical data. 2. Tracing local density with OPTICS," *Journal of chemical information and computer sciences*, vol. 42, no. 3, pp. 500–507, May 2002.

[35] P. Golik, P. Doetsch, and H. Ney, "Cross-entropy vs. squared error training: A theoretical and experimental comparison," in *Proc. Interspeech*, Aug. 2013, pp. 1756–1760.

[36] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[37] P. Baldi, "Gradient descent learning algorithm overview: A general dynamical systems perspective," *IEEE Trans. on Neural Networks*, vol. 6, no. 1, pp. 182–195, Jan. 1995.

[38] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. Int. Conf. on Computational Statistics*, Sep. 2010, pp. 177–186.

[39] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, Jun. 2002.

Suhyeong Choi received the B.S. and M.S. degrees in electrical engineering from KAIST, Korea, in 2016 and 2018, respectively. His research interests include computational lithography, DFM, and neuromorphic hardware as well as machine learning.

Seongbo Shim received B.S. and M.S. degrees in physics from Seoul National University, Seoul, Korea, in 2004 and 2006 respectively, and a Ph.D. from the School of Electrical Engineering, KAIST, Korea, in 2016. Since 2006, he has been with the Semiconductor R&D Center, Samsung Electronics, as a Senior Engineer working on computational lithography, OPC, and DFM for advanced technologies. He has authored more than 40 papers on semiconductor manufacturing (lithography and OPC) and DFM. He is the holder of 12 patents on OPC and lithography. His research interests include mask synthesis (OPC) algorithms, DFM, VLSI CAD for the design-manufacturing interface, and design technology co-optimization (DTCO) for emerging technologies.

Youngsoo Shin (F'17) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Seoul National University, Korea. From 2001 to 2004, he was a Research Staff Member with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He joined the School of Electrical Engineering, KAIST, Korea, in 2004, where he is currently a Professor. His recent research interests include low-power design, computational lithography, design for manufacturability, and neuromorphic circuit design. Dr. Shin has received the Best Paper Award at ISQED in 2005. He was a General Chair of ASP-DAC 2018 and has served as Program Chair of ICCD 2014 and VLSI-SoC 2015. He has served as a member of technical program committees and organizing committees of DAC, ICCAD, ISLPED, ASP-DAC, ICCD, and so on. He is an Associate Editor of the IEEE Transactions on CAD and IEEE Design & Test.