High Performance Lithography Hotspot Detection with Successively Refined Pattern Identifications and Machine Learning

Duo Ding, J. Andres Torres, and David Z. Pan, *Senior Member, IEEE*

***Abstract*—Under the real and evolving manufacturing conditions, lithography hotspot detection faces many challenges. First, real hotspots become hard to identify at early design stages and hard to fix at post-layout stages. Second, false alarms must be kept low to avoid excessive and expensive post-processing hotspot removal. Third, full chip physical verification and optimization require very fast turn-around time. Last but not least, rapid technology advancement favors generic hotspot detection methodologies to avoid exhaustive pattern enumeration and excessive development/update as technology evolves. To address the above issues, we propose a high performance hotspot detection methodology consisting of: 1) a fast layout analyzer; 2) powerful hotspot pattern identifiers; and 3) a generic and efficient flow with successive performance refinements. We implement our algorithms with industry-strength engine under real manufacturing conditions and show that it significantly outperforms state-of-the-art algorithms in false alarms (2.4X to 2300X reduction) and runtime (5X to 237X reduction), meanwhile achieving similar or better hotspot accuracies. Compared with pattern matching, our method achieves higher prediction accuracy for hotspots that are not previously characterized, therefore, more detection generality when exhaustive pattern enumeration is too expensive to perform *a priori*. Such high performance hotspot detection is especially suitable for lithography-friendly physical design.**

***Index Terms*—Lithography hotspot detection, machine learning, manufacturability/yield, pattern classification.**

I. Introduction

WITH THE RAPID shrinking of semiconductor process technology nodes, the minimum feature size of modern integrated circuit (IC) becomes much smaller than the lithographic wavelength [1]. In order to bridge the wide gap between design demands and manufacturing limitations of the current mainstream 193 nm lithography, various design for manufacturability (DFM) techniques [2]–[5] have been proposed to improve product yield and avoid potentially problematic patterns (i.e., process hotspots). However, for 45 nm node and below, hotspot patterns still exist even after design rule checking (DRC) and various resolution enhancement techniques (RET) such as optical proximity correction (OPC) [6], double exposure double patterning lithography [7], and self-aligned double patterning lithography [8], [9].

Therefore, fast and high fidelity hotspot detection engines can play an essential role to enhance physical verification/DRC, and to develop process-aware physical design tools. On the one hand, conventional approaches that employ lithographic simulations [10], [11] are accurate but very costly to run; on the other hand, approaches that utilize pattern/graph matching techniques [12]–[14] are fast but reliant on a set of predefined hotspot patterns. However, general hotspot patterns are hard to define/model in a deterministic manner. Too many patterns lead to high overestimate (false alarms) and too few patterns result in low hotspot coverage. Pattern enumeration could become more problematic as process technology advances and RETs improve, as the definition of the real lithography hotspots is highly dependent on the evolving manufacturing conditions.

In recent years, there have been emerging works that start incorporating modern data mining methods for fast and accurate hotspot detection. A neural network judgment-based detection flow was proposed in [15], where 2-D hotspot image patterns were directly used to train an artificial neural network (ANN) kernel. In [16], data mining algorithms are developed for hotspot pattern (2-D images) clustering. While these early attempts have shown promising potential for lithography hotspot detection using data mining methods, there are still limitations to overcome, such as high training noise and low hotspot detection fidelity.

Later in [17], a support vector machine (SVM)-based hotspot detection method is utilized through performing 2-D distance transform and histogram extraction on pixel-based layout images. Also in [18] and [19], SVM is employed for hotspot detection through extraction and classification of certain special layout density-related metrics. References [17]–[19], as improvements over [15] and [16], demonstrate higher detection accuracy and lower classification noise, due to the introduction of high fidelity metrics. However, these approaches have limited efficiency in runtime and/or detection coverage, since 2-D transforms and density extractions can be quite expensive to perform, meanwhile detection windows (or hotspot candidate locations) for the layout images can be

Manuscript received March 2, 2011; revised May 31, 2011; accepted July 12, 2011. Date of current version October 19, 2011. This work was supported in part by NSF under Grant CCF-0644316, SRC under Task 1833.001, IBM Faculty Award, Fujitsu, and Oracle. This paper was recommended by Associate Editor Y.-W. Chang.

D. Ding and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: ding@cerc.utexas.edu; dpan@ece.utexas.edu).

J. A. Torres is with the Design-to-Silicon Division, Mentor Graphics Corporation, Wilsonville, OR 97070 USA (e-mail: andres_torres@mentor.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2164537

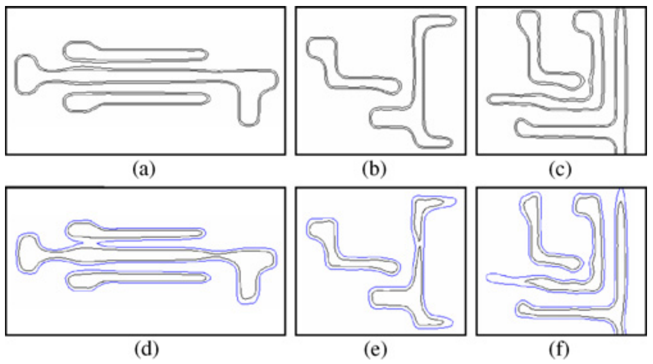


Fig. 1. Illustrative example of process variability bands depicting manufacturing variations under different manufacturing conditions [23]. (a)–(f) Design pattern under process variations.

very hard to anchor for full-chip area detections. In practice, these windows are slid, scanned, or sampled across the entire layout with a certain amount of overlap (or blank interval) between each other. As an inevitable result, detection performance becomes a tradeoff between runtime and detection coverage. In [20], *critical hotspot signature* is proposed and extracted through certain special edge-based metrics. Although such edge-based extractions operate much faster compared with [17], [18], their chip level applications still face similar problems such as scanning window coverage, and others.

Moreover, very few existing studies deal with the detection challenges under the real manufacturing conditions in which hotspots become increasingly harder to detect. In order to be practically employed in modern IC physical design, a successful hotspot detection engine must demonstrate superior speed compared to full lithography simulation (>100 CPUs running in the order of days) and DRC (tens of CPUs running for a few hours), as well as comparable performance to meet the real design and manufacturing requirements. Unfortunately, under such situations, the hotspot evaluation models in [15]–[18], [20] suffer from severe performance degradation. This is because detecting real hotspots under industry-strength PDK/manufacturing conditions require more than just one straightforward model, but multiple levels of identification models for performance refinement.

To better address the issues and challenges above, we extend our work in [21] and propose a generic hotpot detection methodology that is capable of fast online data learning and high performance hotspot pattern identifications. This methodology provides a full layout, feature-centric analysis without being penalized in runtime or coverage by conventional sliding window [18] or raster scanning [17] related techniques. Under such a methodology framework, we define novel layout analyzing algorithms that process the layouts in a fragment-based manner. We implement the framework in a leading industrial geometry processing engine via a shared object library [22]. The proposed framework is tested with enhanced ANN and SVM kernels on large industry layouts under real manufacturing conditions, demonstrating very promising performance in detection accuracy enhancement, false-alarm suppression, and CPU time reduction.

The remainder of this paper is organized as follows. In Section II, we further motivate a few key challenges in lithography hotspot identification and summarize our major contributions. Section III gives an overview of our proposed methodology. In Section IV, we propose a *layout analyzer* for the extraction of hotspot-related features with high detection coverage and speed, followed by Section V, where we describe in detail our *hotspot identifiers* with special machine-learning models for enhanced accuracy. In Section VI, we propose a novel flow to integrate, configure, and validate multiple successive levels of hotspot identifiers for ultralow false alarms under current real manufacturing conditions. Simulation results on various placed and routed industry layouts are assessed and analyzed in Section VII. Section VIII concludes this paper.

II. Motivation and Contributions

To visualize the aforementioned challenges, in Fig. 1 we show the printed images of three layout patterns under two different manufacturing conditions: Figs. (a)–(c) are printed under a real production 45 nm process which provides insights on the frequency of occurrence of hotspots under real manufacturing conditions, while Figs. (d)–(f) are under simplified but widely accessible manufacturing conditions (e.g., free PDK 45 nm), which lack the detailed information in optical, resist, and actual OPC and RET recipes used during production tending to exacerbate process variations. In this case, we observe significantly less poor-printability areas in Figs. (a)–(c) than in Figs. (d)–(f). The significance of this motivational example is manifold: first, a real hotspot pattern by definition is strongly dependent on manufacturing conditions, and second, the number of real hotspots under continuously improving manufacturing conditions becomes less (assuming improving RETs) but hard to fix at post-layout stages. Moreover, non-hotspot detection inaccuracy (false alarms) increases the burden of hotspot correction processes, and should therefore be minimized. Last but not least, the huge data volume of large area design layouts requires ultrafast detection speed and good runtime scalability.

To sum up, the key challenges for lithographic hotspot detection under real manufacturing conditions include:

1) lithographic hotspot patterns that are highly dependent on manufacturing conditions and very hard to enumerate exhaustively;
2) hotspots that become harder to identify in early design stages and harder to fix at post-layout stages;
3) non-hotspot detection accuracy (indicating false alarms) that becomes vitally important as excessive false alarms are severely penalized in post-layout corrections;
4) ultrafast detection speed that is desired for large layouts and for guiding lithography-friendly physical design;
5) sliding window techniques that can be highly penalized by considerable loss of accuracy and detection coverage.

In face of the aforementioned challenges, this paper proposes a novel methodology and a set of highly effective techniques for fast and accurate lithographic hotspot detection under real manufacturing conditions. Our main contributions are summarized as follows.

1) We propose a generic methodology for lithography hotspot detection whose flow is compatible with evolv-

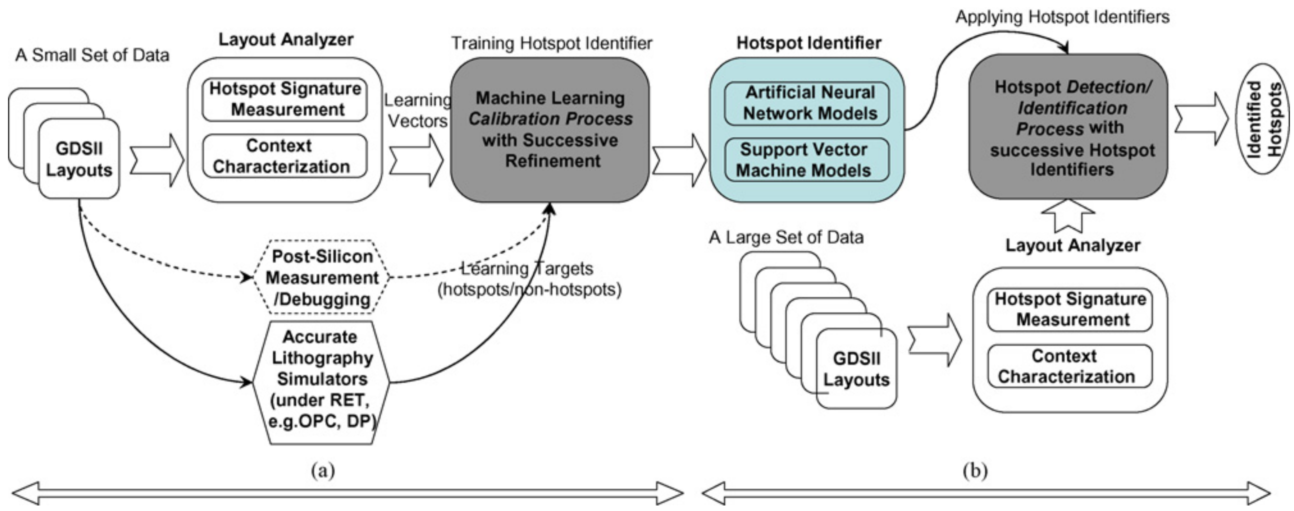

A Small Set of Data
GDSII Layouts
Layout Analyzer
Hotspot Signature Measurement
Context Characterization
Learning Vectors
Training Hotspot Identifier
Machine Learning Calibration Process with Successive Refinement
Post-Silicon Measurement /Debugging
Accurate Lithography Simulators (under RET, e.g.OPC, DP)
Learning Targets (hotspots/non-hotspots)
Hotspot Identifier
Artificial Neural Network Models
Support Vector Machine Models
Applying Hotspot Identifiers
Hotspot Detection/ Identification Process with successive Hotspot Identifiers
Identified Hotspots
A Large Set of Data
GDSII Layouts
Layout Analyzer
Hotspot Signature Measurement
Context Characterization
(a)
(b)

Fig. 2. Overview of our proposed hotspot detection methodology. (a) Calibration stage. (b) Detection stage.

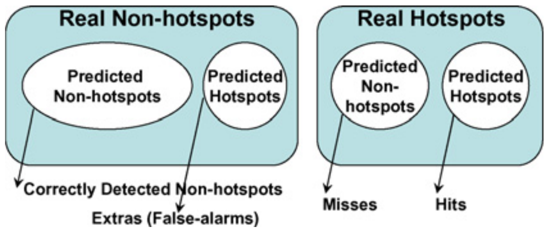


Fig. 3. Illustration of the hotspot detection hits, misses, and extras.

ing RETs and manufacturing conditions.

2) We define special hotspot signature measurements for ultrafast, full layout detection without sliding window or raster scanning techniques.
3) We introduce fast layout analyzers and generic hotspot identifiers with powerful machine-learning models especially for capturing new/unknown types of hotspots.
4) We develop a generic and efficient flow with multiple levels of successive pattern identifiers for ultralow identification false alarms.
5) We perform thorough qualification using real industry examples for a 45 nm METAL1 process under real manufacturing conditions.

III. Methodology Overview

The ultimate objective of our methodology is to correctly classify hotspots and non-hotspots with low false-alarm rates at a fast speed.

Fig. 3 depicts the relations between the actual hotspots/non-hotspots and the predicted hotspots/non-hotspots, where hit number is the number of correctly predicted hotspots, miss number is the complement of hit in the actual hotspot set, and extras is the number of non-hotspots predicted as hotspots. Based on Fig. 3, we define several important terms used throughout this paper, as in (1)–(4).

Definition 1: Hhit: the hotspot detection accuracy rate

$$Hhit = \frac{\text{correctedly\_detected\_hotspots}}{\text{real\_hotspots}}. \quad (1)$$

Definition 2: Hmiss: the hotspot detection inaccuracy rate (1s complement of *Hhit*)

$$Hmiss = \frac{\text{undetected\_hotspots}}{\text{real\_hotspots}}. \quad (2)$$

Definition 3: Hextra: the hotspot detection overshoot rate (false-alarm rate)

$$Hextra = \frac{\text{falsely\_detected\_hotspots}}{\text{real\_hotspots}}. \quad (3)$$

Definition 4: Nhit: non-hotspot detection accuracy rate

$$Nhit = \frac{\text{correctedly\_detected\_nonhotspots}}{\text{real\_nonhotspots}}. \quad (4)$$

Since the number of falsely predicted hotspots is $(1 - Nhit) \cdot \text{real\_nonhotspots}$, we have

$$\frac{Hextra}{\text{real\_nonhotspots}} = \frac{(1 - Nhit)}{\text{real\_hotspots}}. \quad (5)$$

From (5), we can see the relation between *Hextra* and *Nhit*, both of which are measurements of the hotspot detection false alarms. With *Hextra* we can clearly see the total false-alarm counts, while using *Nhit*, we can better appreciate the occurring frequency of the false alarms among all non-hotspots. For this reason, we will use both of them in the analysis and discussions of this paper.

Before going further to details, we first illustrate an overview of our proposed methodology in Fig. 2, which is divided into the *calibration stage* and the *detection stage*.

The calibration stage involves: 1) a relatively small set of layouts for configuring the hotspot identifiers via supervised learning techniques; 2) a layout analyzer for characterizing layout geometries; 3) a lithography simulator (or post-silicon measurement) to provide accurate information of the real hotspots as learning targets; and 4) a novel calibration process for the training and validation of hotspot identifier models via successive refinements. The result of the calibration stage is a set of hotspot identifiers established at a one time computation cost. In the detection stage, we apply the hotspot identifiers to search for hotspot patterns over very large volume of design layouts with high efficiency given that the identifiers have

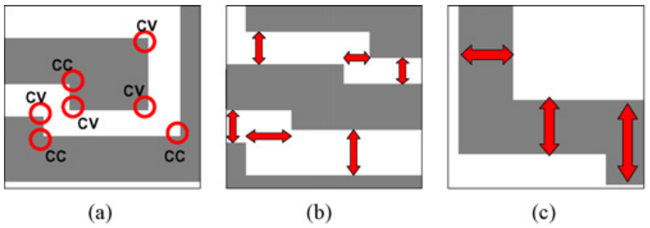


Fig. 4. Three major types of hotspot feature measurements. (a) Corner information. (b) External length. (c) Internal length.

been setup *a priori*. This stage mainly consists of: 1) a layout analyzer, and 2) a hotspot detection process that utilizes the multiple hotspot identifiers via levels of refinements.

Note that the function of layout analyzer is to characterize the layout and convert patterns to compact 1-D vectors. With these data vectors, special hotspot identifiers can be: 1) trained and validated under the supervision of our employed production simulator [22], and 2) applied with very high speed and fidelity over new layouts. Note that the hotspot identifiers must be updated by re-running the calibration stage if the design rules or manufacturing conditions are changed for the new layouts. In the following sections, we will further explain the details of each block from Fig. 2.

IV. Feature-Centric Layout Analyzer

Layout analyzer performs the function to characterize the layout context and extract hotspot-related features in the format of data vectors. Here, we define hotspot feature metrics (or hotspot signature measurements) as a set of special measurements which contribute strongly to the decision-making process of hotspot detection. Unlike special restricted design rules, a layout analyzer does not decide whether a certain pattern is hotspot or not, but it leaves the decision-making process to the recursively refined supervised training of hotspot identifiers using machine-learning techniques. In face of the aforementioned challenges under real manufacturing conditions, a successful layout analyzer must first define a proper set of hotspot signature measurements. Unlike previous studies utilizing 2-D transforms or density calculations or sliding window techniques, we propose novel metrics and special data structures for significant runtime reduction and satisfactory accuracy.

A. Hotspot Signature Measurements

The hotspot signature measurement step aims at scanning the layout and extracting useful information/data from all geometries in the layout. To achieve this goal, we first define several classes of measurements as shown in Fig. 4, including: 1) corner information (convex or concave); 2) distance to an externally facing polygon edge; and 3) distance to an internally facing polygon edge. In a design layout where the fragments (polygon edges) are indexed numerically, these basic measurements are programmed and optimized to reach high speed and memory efficiency using the function library provided via advanced programming interface of [22].

To perform the above signature measurements, we propose four different types of feature-centric operator functions, summarized in Table I. Given a fragment *Frag* in certain

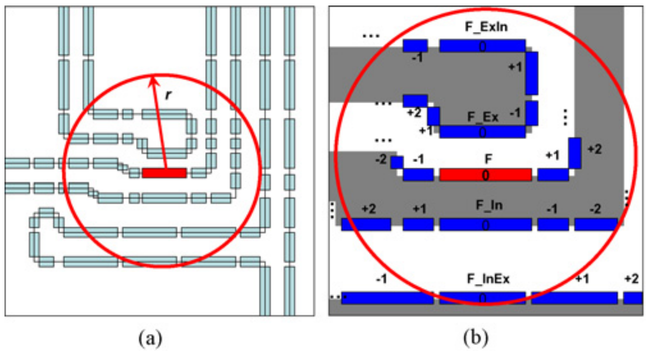


Fig. 5. Fragmentation-based hotspot signature extraction. (a) Effective radius centered at each fragment. (b) Fragmentation-based context characterization.

TABLE I
Hotspot Signature Measurement Operators

Operators	Operation Description (Features to Measure)
$f_{corn}(\cdot)$	Corner information: CV (convex)/CC (concave)
$f_{ext}(\cdot)$	External inter fragment distances
$f_{int}(\cdot)$	Internal inter fragment distances
$f_{misc}(\cdot)$	Miscellaneous information

layout, *operator* $f_{corn}(\cdot)$ extracts information of convex and concave corners touching *Frag*, and *operator* $f_{ext}(\cdot)$ returns the distance(s) between *Frag* and the fragments facing *Frag* on the external side. Similarly, *operator* $f_{int}(\cdot)$ returns the distance(s) between *Frag* and the fragment facing *Frag* on the internal side. The *Operator* $f_{misc}(\cdot)$ requests extra information regarding *Frag*, such as fragment orientation (x or y-axis) and the length of *Frag*.

With a proper combination of these measurement operators, we can accurately characterize the entire layout at a one-time cost. In the real practice, the hotspot signature measurement of the layout geometries is performed through establishing a table-structure database that can be indexed in constant time, so that the next step—the *context characterization* process—can be carried out with good runtime and memory efficiency. More details will be discussed at the end of this section.

B. Fragmentation-Based Context Characterization

The ultimate goal of layout analyzer is to provide high resolution layout characterization with full scanning coverage. To achieve this goal, we first perform the hotspot signature measurement on a per fragment basis, followed by the fragmentation-based context characterization step. The goal of this step is to generate a 1-D vector that serves as the input data of our machine-learning engines. Hotspot detection decision for a certain fragment will later be made based on the context information provided by such a data vector. In the real practice, the context characterization is performed as a table-lookup procedure.

Given a properly fragmented layout and any fragment of interest F, we illustrate the concept of an effective radius r in our proposed context characterization procedure. As shown in Figs. 5(a) and (b), r centers at (each) fragment F. By definition, effective radius r covers the neighboring fragments which need to be considered in the context characterization of F. In its empirical nature, r depends on the lithography processes, and it is generally easy to pick in the training

TABLE II
SHORTHAND FRAGMENT NOTATIONS

Notation	Descriptions of the Shorthand Notation
F	Current fragment of interest (detection anchor point)
F_In	Fragment(s) facing F internally
F_Ex	Fragment(s) facing F externally
F_In_Ex	Fragment(s) facing F_In externally
F_{+i}	ith neighbor traced from F clockwise
F_{-i}	ith neighbor traced from F counter-clockwise

and validation procedure given the manufacturing conditions. According to Fig. 5(b), we illustrate our proposed context characterization process as follows.

First, suppose the current fragment of interest is F (colored red in the center of the effective circle), we define several shorthand notations whose indices are used for indexing throughout the fragments lying within the effective region of F. We elaborate their details in Table II. Note that F can also be denoted as F_0. We use F_In to represent Fs internally facing fragment, F_Ex meaning the externally facing fragment. Similarly, F_ExIn is the internally facing fragment of F_Ex and F_InEx is the externally facing fragment of F_In. Also for each fragment, we mark its adjacent neighbors with ascending indices in clockwise order. For example, the first adjoining neighbor of F in clockwise order is denoted as F_{+1}, and F can also be denoted as F_0, as shown in Fig. 5(b). These indices are used for indexing the fragments lying within the effective region of certain polygon of interest in the context characterization process.

Next, we present the characterized context of fragment F in the format of a 1-D data vector defined as

$$V_F = \coprod_{i}^{\widetilde{F_i}\in\delta_r^F} \{f_{ext}(\widetilde{F_i}) \oplus f_{int}(\widetilde{F_i}) \oplus f_{corn}(\widetilde{F_i}) \oplus f_{misc}(\widetilde{F_i})\} \quad (6)$$

$$\widetilde{F} = [F, F\_Ex, F\_In, F\_ExIn, F\_InEx...] \quad (7)$$

where F is an integer identification (ID) number representing a certain fragment in the layout, and δ_r^F is the effective region of F. To ensure an efficient learning process meanwhile maintaining the consistency of the extracted hotspot signatures, we generate the final learning vector V_F via properly combining the individual measurements of each fragment using procedures $\bigoplus$ and $\coprod$. Together they determine the order of the learning features in V_F so that the same patterns (with the same surrounding geometries) maintain the same V_F after rotation and mirroring, should the surroundings rotate or mirror at the same time. The length of V_F is the number of features M.

We further illustrate the above equations with a simple example shown in Fig. 6. Assume fragment A to be the fragment of interest and r to be two-fragment deep, meaning to look for a maximal depth of two neighboring fragments whenever a search algorithm is applied. In this case, fragment A is F, B is F_Ex, and C is F_In. This is because B is facing A on the external side, and C is facing A on the internal side (on the same metal polygon). D is on the internal side of B, which is equivalently the internal of the external side of A: F_ExIn. Similarly, E can be written as F_InEx, and so on. Since r is set to 2, we only look at two neighbors on each side of A, or else we need to look further than D, B and C, E. In other words, r controls the range of our context characterization process.

To generate the vector V_F, we follow two major steps: first, in the order of A, B, C, D, and E, select two neighbors counter-clockwise then two neighbors clockwise for each fragment and store the ID number of each resulting fragment in a queue. For simplicity, Fig. 6 marks the order in which the queue is filled using numbers from 1 to 25. Second, apply the signature measurement operators in order to each of the fragment in the queue and put the results in a parameter vector V_F.

There are several things worth noting for this procedure.

1) The same fragment in a layout may be queried multiple times thus showing at different indices/locations in the queue, such as (10, 16), (9, 17), (6, 18), (1, 13), (4, 12), and (5, 11) in Fig. 6. This is desirable information since they usually indicate line-end shapes.
2) A large value L_{MAX} will be used in case when F_Ex or F_In does not exist. For example, in Fig. 6, index 9 (or 17) has no facing fragments on either internal or external side.

After all these considerations, we have the V_F as a compact parameter vector to represent the context information around fragment A. Such a generation procedure repeats until every fragment of interest in the layout is processed.

In this paper, the vector V_F formed by the context characterization is defined as the hotspot signature, and the context characterization for each F is also referred as feature extraction process. Note the set of all V_Fs over the entire layout forms the final output of the layout analyzer. It filters out noise and provides a compact data set for hotspot identifiers (with machine-learning models) to be properly established.

Apparently, the total number of fragments could be huge in a chip-level layout, especially for high density METAL1 layers. With Fig. 6, we can figure out that the V_Fs of close-by fragments share quite some common context information thus need not be processed all over again. Based on such an observation, we can achieve significant speedups by avoiding processing the layout in a fragment-by-fragment manner. In practice, we build a lookup table database of the entire design inside [22]. Using the advanced programming interface of [22], the memory cost of the lookup table is about 28 MB/mm^2 design, which is well acceptable for modern workstations. This way, (6) can be realized via table lookup with constant time complexity for the entire layout. This allows us to achieve up to hundreds of times of runtime reduction compared with some previous studies. Simulation results and further discussions will be presented in Section VII.

V. HOTSPOT IDENTIFIERS AND ROBUST LEARNING MODELS

Our hotspot identifiers employ machine-learning models that play essential roles unlike previous works. Using powerful machine-learning techniques, we build efficient models specially suitable for classifying lithography hotspot data. In particular, we modify and enhance two types of machine-learning techniques, ANN and SVM, in mainly two aspects: first, robustness and accuracy in the weight update process,

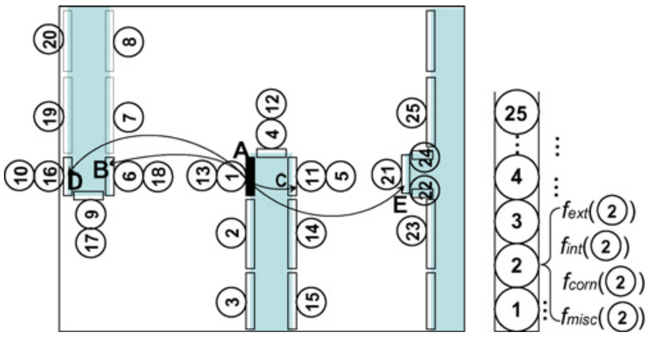


Fig. 6. Illustrative example of the vector generation process for fragment A (black color), where r is set to cover a depth of two neighboring fragments.

TABLE III

ANN/SVM KERNEL-RELATED VARIABLES

Variables	Descriptions
N	Total number of input sample vectors
M	Feature number per sample vector
V_p	Input sample vectors, p=1 to N
V_p^i	ith element(feature) of V_p, i=1 to M
y_p	Hotspot label for V_p in *calibration*, p=1 to N
f_{in}	Input transfer function for ANN kernel
f_{hid}	Hidden layer transfer functions for ANN kernel
f_{out}	Output layer transfer function for ANN kernel
out_p	ANN output prediction value from V_p input
out_{hid}^j	ANN hidden layer jth neuron prediction output
$\overrightarrow{\varpi}$	ANN kernel matrix of neuron connection weight
$K(V_i, V_j)$	SVM kernel function between V_i and V_j
α	SVM weight vector for input V_ps
$\Delta(\cdot)$	Threshold function for hotspot decision making
$Est_{\tilde{p}}$	Machine-learning estimation for a new input $V_{\tilde{p}}$

and second, detection threshold $\Delta(\cdot)$ optimizations for simultaneous *Hhit* improvement and *Hextra* suppression.

Generally speaking, ANN and SVM have similar performance for most of binary classifications. SVM guarantees the global optimum in its formulation if the kernel function satisfies *the Mercer's condition* defined as follows.

Definition 5: Mercer's Condition: A kernel function $K(x, y)$ satisfies the *Mercer's condition* if and only if (8) holds for any square integrable function $f(x)$ as follows:

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} K(x, y) f(x) f(y) dx dy \geq 0 \tag{8}$$

where $f(x)$ satisfies

$$\int_{-\infty}^{+\infty} |f(x)|^2 dx < \infty. \tag{9}$$

On the one hand, SVM is usually prone to data noise and may also result in longer runtime for high-dimensional data sets when the number of support vectors becomes large. ANN, on the other hand, provides more noise robustness, compact kernel models (neuron weight), and flexible network structures. In theory, the algorithms that ANN uses to update the neuron weights usually do not guarantee global optimum. In practice, however, a very close-to-optimum solution can be found and validated through careful calibration and convergence control.

With these considerations, we incorporate both classes of models into our hotspot identifiers with special modifications.

Algorithm 1 Pseudocodes for training and configuring hotspot identifier-ANN( )

```
Require: Training vectors $V_p$s and training targets $y_p$s
  Scale each feature column of the training row vectors to $[-1, +1]$
  Divide training set into learning, cross-valid, cross-test sets
  Set converging speed parameters: $\eta_+ = 1.5$, $\eta_- = 0.5$, $\delta_{\max} = 50$
  Initialize conditions for all variables to be updated
  while error target not met do
    for each $V_p$ in the learning set do
      calculate gradients $\partial E^p/\partial\omega_{ij}$ and $\partial E^p/\partial\omega_{jk}$ for $V_p$
    end for
    calculate the average gradient value for each link as $\partial E^p/\partial\omega_{ij}$
    for all weights and biases do
      if $\partial E^p/\partial\omega_{ij}(t-1) * \partial E^p/\partial\omega_{ij}(t) > 0$ then
        $sign = \eta_+$
      else if $\partial E^p/\partial\omega_{ij}(t-1) * \partial E^p/\partial\omega_{ij}(t) < 0$ then
        $sign = \eta_-$
      else
        $sign = 1.0$
      end if
      $\delta(t) = \min(sign*\delta(t-1), \delta_{max})$
      $\omega_{ij}(t) = -\delta(t)*sign\_func(\partial E^p/\partial\omega_{ij}(t))+\omega_{ij}(t-1)$
    end for
    update error for current epoch $t$
    break if(early stopping criteria met in valid and test sets)
  end while
  return A hotspot identifier model with $\omega_{ij}$, $\omega_{jk}$.
```

For ANN kernels, we modify the resilient backpropagation update method [24] with enhanced robustness and better parameter tradeoffs between convergence speed and detection accuracy. We also propose strategies for optimizing the detection threshold of each ANN model. For SVM kernels, we combine a C-type SVM formulation with higher accuracy working set selection based on [25], together with the detection threshold optimizations. We describe our special hotspot identifier model formulations and implementations briefly as follows, with related symbols and variables summarized in Table III.

A. ANN: Artificial Neural Network Models

In this section, we present hotspot identifiers with ANN models, together with the techniques used to configure (Algorithm 1) and apply (Algorithm 2) these novel identifiers.

A typical ANN classifies data by predicting a value for each V_p based on an established set of weights and biases assigned to certain neural network structure. Our ANN kernels are customized with single hidden layer of neurons, with transfer functions denoted as f_{hid}. Inputs V_p to the ANN kernels are the extracted feature vector samples labeled with values (y_p) indicating hotspot or non-hotspot patterns (these values can be continuous for variability prediction). We use p to represent feature vector index with $p = 1$ to N, V_p^i denotes the ith element of vector V_p, and $i = 1$ to M, where M is the total number of features for each sample vector. We use f_{in} and f_{out} to represent input and output layer transfer functions, and index i, j, k to indicate neuron indices in the input, hidden, and output layer, respectively. In particular, we first choose certain *sigmoid* functions for the hidden layer and a linear function for the output layer. Then we formulate the ANN calibration

Algorithm 2 Pseudocodes for applying hotspot identifier-ANN( )

```
Require: $V_p$s from the layout analyzer
  Scale the features of $V_p$s correspondingly by the (min, max) values from Algorithm 1
  Load Hotspot identifier-ANN model
  Calculate (16)
  return A hotspot estimate $Est_{\tilde{p}}$.
```

process in (10)–(16) as follows:

$$\text{objective :minimize}\{\sum_{p=1}^{N} E^p\} \quad \text{w.r.t.} \quad \omega_{ij}, \omega_{jk} \tag{10}$$

$$E^p = \frac{1}{2}[\text{out}_p - y_p]^2 \tag{11}$$

$$\text{out}_p = f_{\text{out}}\{\sum_j \omega_{jk} \cdot f_{hid}(\sum_i V_p^i \cdot \omega_{ij})\} \tag{12}$$

$$\frac{\partial E^p}{\partial \omega_{jk}} = (\text{out}_p - y_p) \cdot f_{hid}\{\sum_i V_p^i \cdot \omega_{ij}\} \tag{13}$$

$$\frac{\partial E^p}{\partial \omega_{ij}} = (\text{out}_p - y_p) \cdot \omega_{jk} \cdot V_p^i \cdot (1 + \text{out}_{hid}^j)(1 - \text{out}_{hid}^j) \tag{14}$$

$$f_{hid} = \frac{2}{(1 + e^{-2x})} - 1, \qquad f_{\text{in}} = f_{\text{out}} = x \tag{15}$$

$$Est_{\tilde{p}} = \Delta\{f_{\text{out}}[\sum_j \omega_{jk} \cdot f_{hid}(\sum_i V_{\tilde{p}}^i \cdot \omega_{ij})]\}. \tag{16}$$

As shown in (10), the objective function is set to the summed square error (SSE) among all N input sample vectors. Such a minimization is achieved through iterative update of weight matrix $\overrightarrow{\varpi}$ using modified resilient backprop method. We call every iteration as one epoch, defined as one complete representation of V_1 through V_N to the ANN kernel. out$_p$ is the ANN prediction result for each V_p vector, while out_{hid}^j is the predicted output from the jth node in the hidden neuron layer, both of out$_p$ and out_{hid}^j are iteratively updated across epoches.

The training and configuration of each hotspot identifier-ANN model is achieved by executing Algorithm 1, which takes in data set V_ps after the signature extraction process (the layout analyzer) and returns final neuron weight coefficient matrix $\overrightarrow{\varpi}$. In particular, there are three steps involved for hotspot detection accuracy enhancement. First, the input data set is normalized for each feature across the whole sample space. Second, the data set is divided into learning (80%), cross-validation (10%), and cross-testing (10%) subsets for the considerations of kernel establishment robustness, data over-fitting prevention, and proper early stopping criteria. In the third step, network output is adjusted incrementally with a stepwise update of network weight matrix toward minimal SSE value with parameters and gradient update procedures modified for hotspot detection under real manufacturing conditions. With the calculations of the gradient values using (12)–(15) and the arithmetic smoothing steps, the hotspot identifier-ANN is trained iteratively until a certain error target is met. Note in Algorithm 1, the training targets are derived by running accurate lithography simulations at a one-time runtime cost. Once the ANN model is fully trained and configured, we can

Algorithm 3 Pseudocodes for training and configuring hotspot identifier-SVM( )

```
Require: Training vectors $V_p$s and training targets $y_p$s
  Scale each feature column of the training row vectors to $[-1, +1]$
  Set control parameters: $\gamma = -1/M$, $C = 1.5$, stopping tolerance $\epsilon$ = 1e-3, min floating number $\tau$ = 1e-12
  Initialize weight vector $\alpha$ and gradient vector $G$
  while 1 do
    Working set ($i$, $j$ pair) selection based on [25]
    if $j == -1$ then
      break
    end if
    Calculate $\eta_1 = \max(\tau, Q_{i,i} + Q_{j,j} - 2\, y_i\, y_j\, Q_{i,j})$
    Calculate $\eta_2 = y_j \cdot G_j - y_i \cdot G_i$
    Update weight: $\varpi_i \mathrel{+}= y_i \cdot \eta_1/\eta_2$, $\varpi_j \mathrel{-}= y_j \cdot \eta_1 / \eta_2$
    $\varpi_i = slop\_func(\varpi_i)$,$\varpi_j = slop\_func(\varpi_j)$
    Update gradients for all $k = 1$ to $M$: $G_k \mathrel{+}= Q_{k,i}\, (\varpi_i - \varpi_i^{prev}) + Q_{k,j}\, (\varpi_j - \varpi_j^{prev})$
  end while
  Calculate $\rho$ and $bias$ values for prediction processes
  return A hotspot identifier model of nonzero elements of $\alpha$ and corresponding $V_p$s.
```

apply it to identify hotspots according to Algorithm 2 without using costly lithography simulations.

B. SVM: Support Vector Machine Models

In this section, we present hotspot identifiers with SVM models, together with the techniques used to configure (Algorithm 3) and apply (Algorithm 4) these novel identifiers.

SVM classifies sample vectors by calculating a (hyperplane) boundary with maximum separation margin in-between of different classes. With such an optimized margin, only the sample vectors forming the boundaries are considered as contributing factors for new sample classifications. These vectors are called support vectors; they are assigned different weights and they perform classification tasks through certain kernel function $K(V_i, V_j)$. For this paper's high fidelity detection flow, we combine a typical 2-class soft error-tolerant SVM kernel, a special working set selection technique using second-order information [25] and a detection threshold Δ optimization procedure toward simultaneous accuracy enhancement and false-alarm suppression.

The dual problem of our quadratic formulation of C-type SVM is given as follows:

$$\text{objective : minimize}\{f(\alpha) = \frac{1}{2}\alpha^T Q \alpha - e^T \alpha\} \quad \text{w.r.t.} \quad \alpha \tag{17}$$

$$\text{subject to : } 0 \leq \alpha_i \leq C, i = 1, \ldots, N \tag{18}$$

$$y^T \cdot \alpha = 0 \tag{19}$$

$$K(V_i, V_j) = exp\{\gamma \cdot \|V_i - V_j\|^2\} \tag{20}$$

$$Est_{\tilde{p}} = \Delta\{\sum_i \alpha_i y_i K(V_{\tilde{p}}, V_i) + bias\}. \tag{21}$$

Given V_i, $i = 1$ to N sample vectors, with label y_i (either $+1$ or -1 for 2-class SVM). e is a vector of all 1s. C is a preset upper bound to constrain feasible regions for hotspot

Algorithm 4 Pseudocodes for applying hotspot identifier-SVM( )

```
Require: $V_p$s from the layout analyzer
  Scale the features of $V_p$s correspondingly with the (min, max) values from Algorithm 3
  Load hotspot identifier-SVM model
  Calculate (21)
  return  A hotspot estimate $Est_{\tilde{p}}$.
```

detection under real manufacturing conditions. Q is N by N positive semidefinite matrix defined as $Q_{ij} = y_i y_j K(V_i, V_j)$, where $K(V_i, V_j)$ is defined in (20) as the kernel function to meet the Mercer's condition. α is the N element weight vector for V_ps. Note α is generally sparse and the nonzero weights correspond to the final support vectors. Due to the fact that Q is usually dense and large, decomposition methods are usually used to solve the formulation iteratively rather than directly dealing with the quadratic (17).

The training and configuration of hotspot identifier-SVM models are achieved through performing Algorithm 3, which intakes data set V_ps and returns the supporting vectors and corresponding weight coefficients. There are three major steps involved: first, data set normalization for detection robustness; second, high order working set selection for enhanced detection accuracy particularly for our special hotspot detection requirements; and third, update weight and gradient vectors. The last two steps are carried out in an iterative manner until certain error target is met. For implementation details regarding the higher order working set selection, please refer to [25]. When the SVM model is fully trained and configured, we can apply it to evaluate a new design pattern using Algorithm 4, without using accurate lithography simulations.

To sum up, our hotspot identifiers with modified ANN and SVM kernels hold their respective advantages as two of the most important machine-learning classifiers, which are fine-tuned for our hotspot detection requirements under real manufacturing conditions. From a data mining point of view, evaluating both types of classifiers can help us quantitatively interpret the nature of lithography hotspot features/signatures. Such tools can also assist the design rule development process.

VI. Integrative Flow for Successive Identification Refinements

A. Overview

Due to the relatively small number of real hotspots and the highly noisy detection environment under real manufacturing conditions, we propose a novel hierarchical flow with successive levels of refinements to integrate our proposed layout analyzer and hotspot identifiers. In its nature, such an approach hybrids the strength of hotspot identifier models and the successive levels of pattern classifications, contributing to significant detection performance boost in runtime, detection accuracy, and false alarms, when compared with previous approaches such as using straightforward machine-learning techniques.

Following our previous discussions, we present the pseudocodes in Algorithm 5 and Algorithm 6 for the calibration stage and the detection stage overviewed in Section II. For both

Algorithm 5 Pseudocodes for the calibration stage

```
Require: A small set of input design layouts
  Setup optical models, fragmentation specifications
  Generate training targets by accurate lithographic simulation
  Invoke the layout analyzer
  for each fragment in the design layout do
    Perform hotspot signature measurements
    Update lookup tables
  end for
  Invoke hotspot identifier-ANNs (or -SVMs) (signatures, targets) for supervised learning processes with successive refinements (Fig. 7)
  return  Compact hotspot identifiers models
```

Algorithm 6 Pseudocodes for the detection stage

```
Require: A large set of new input layouts
  Setup (the same) optical models, fragmentation, and others
  Invoke the layout analyzer
  Load the hotspot identifiers
  for each fragment in the design layout do
    Apply the hotspot identifiers inside a novel successive refinement hierarchy (Fig. 8)
  end for
  return  Identified hotspot patterns.
```

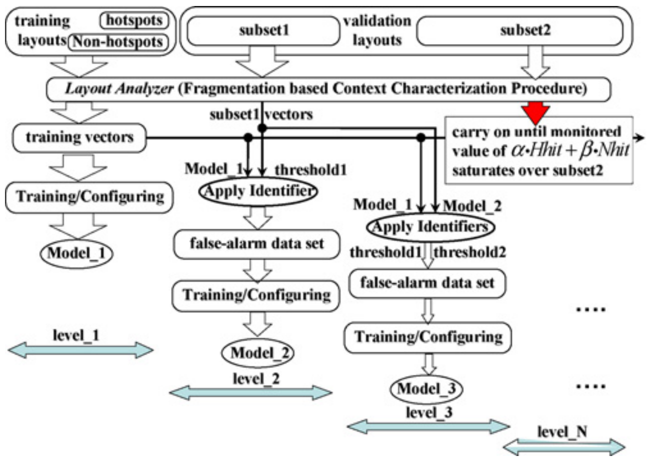


Fig. 7. Training and configuring successive hotspot identifiers and thresholds with cascaded refinements.

algorithms, the proposed hierarchy of successive identification refinements plays a critical role in both detection accuracy and runtime. Such a hierarchy takes slightly different forms in the calibration than in the detection stage. As shown in Fig. 7, in the calibration stage, the hierarchy takes the form of a multilevel cascade with each level contributing an unique hotspot identifier model. As illustrated in Fig. 8, various successive levels of hotspot identifiers derived from the calibration stage are applied in the detection stage in a similar hierarchically refined manner to help reduce the false alarm rate *Hextra* without penalizing the hotspot detection rate *Hhit*. In the following sections, we explain such a hierarchy in detail by dividing it into two terms: a *global* term and a *local* term.

B. Global Calibration and Detection

Here, we refer to the first-level training in Fig. 7 as the global training, since hotspot identifier *Model_1* (abbreviated as *Model_1*) is trained with the whole training data set (on the global scale); similarly, in Fig. 8, detection with only *Model_1*

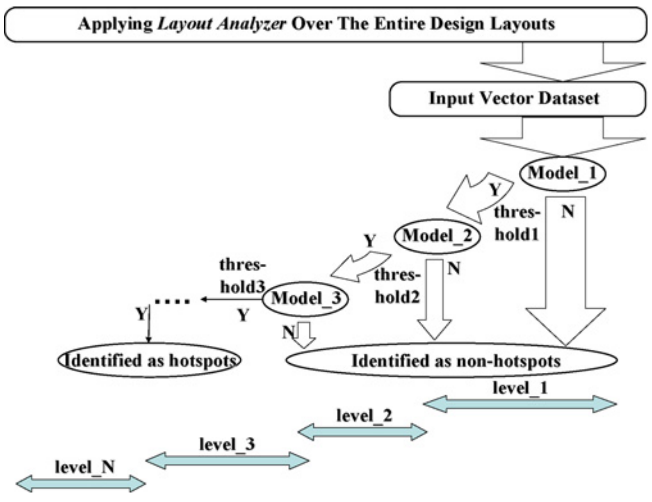


Fig. 8. Applying successive hotspot identifiers and thresholds for hotspot detection.

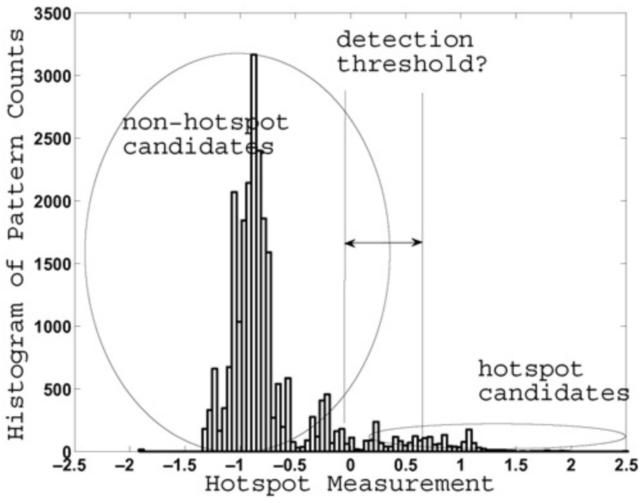


Fig. 9. Motivation of the threshold function in hotspot identification.

is defined as global detection, since the whole testing data go through this model. As we will show later in Section VII, using the layout analyzer and hotspot identifiers, the global term alone achieves very satisfactory hotspot detection accuracy *Hhit* but low non-hotspot detection accuracy *Nhit*. Under the evolving manufacturing conditions, hotspot and non-hotspot patterns are highly unbalanced in quantity, which results in huge number of non-hotspot patterns. Under such a scenario, even a small fraction of identification error can lead to highly excessive false alarms therefore heavy workload of post-design hotspot removal. Consequently, global term alone is not enough to ensure overall satisfactory detection performance.

C. Successive Local Refinements

To further suppress false alarms meanwhile maintaining satisfactory hotspot detection rate, we extend the global term with sublevels of identification hierarchies, which we refer to as the successive local refinements. As illustrated in Figs. 7 and 8, we apply the *level_2* to *level_N* hotspot identifiers that serve as successive stages of refinements in both calibration and prediction stages. In the calibration stage, the refinement flow consists of several key steps.

1) Training, configuring, and validating multiple hotspot identifiers using the entire training data set plus the false-alarm data sets accumulated with each additional level.
2) Stopping criteria to decide when to stop adding more hotspot identifiers.
3) Optimizations of the thresholds associated with the hotspot identifers. In the detection stage, all the hotspot identifier models and thresholds are applied successively, and hotspots are detected as those patterns that are eventually identified as "hotspots" after all levels of refinements. We describe related key steps in more detail in the following sections.

1) *Configuring Successive Hotspot Identifiers:* The configuration of each hotspot identifier involves the training and validation of both the learning model and a detection threshold above which a pattern is identified as a hotspot. Fig. 9 shows some motivations on the importance of threshold selection. By now we have a hotspot identifier *Model*_1 derived from global stage using Algorithms 1 and 3, and the training layouts. Then we apply *Model*_1 (using Algorithms 2 and 4) over some validation data *subset1* to derive a *threshold*1. With this threshold, we collect all the identification false alarms (the configuration mistakes) and use them to configure a *Model*_2. Subsequently, an extra hotspot identifier *Model*_3 can be derived over another false-alarm set of data generated by applying *Model*_1 and *Model*_2 successively (as in Fig. 8). Consequently, *threshold*2 can be derived. Such a refinement goes on until our predefined performance metric saturates over the validation data set *subset*2.

2) *Stopping Criteria:* To quantify the stopping criteria for the local successive refinements, we introduce a user-defined performance metric Ψ_{perf} as follows:

$$\Psi_{perf} = \alpha \cdot Hhit + \beta \cdot Nhit \qquad (22)$$

where α and β are user-defined weights, *Hhit* is the hotspot detection accuracy, and *Nhit* is the non-hotspot detection accuracy. Therefore, Ψ_{perf} represents the weighed summation of hotspot and non-hotspot detection accuracies. With each additional level, we re-evaluate Ψ_{perf} over *subset*2 using all the hotspot identifiers derived so far, according to Algorithms 2 and 4, and Fig. 8. We stop configuring additional hotspot identifiers when Ψ_{perf} saturates or starts to degrade.

3) *Threshold Optimizations:* The important role of threshold optimization has been illustrated in Fig. 9. In this paper, we employ a heuristic approach, that is to exhaust the solution space with grid-based simulations and select the threshold combinations giving the best Ψ_{perf}. The main justification is twofold: a) the calibration stage is performed only once *a priori*, therefore, does not lead to runtime overhead for the detection stage, and b) based on our experiments in Section VII, Ψ_{perf} saturates at level 3, therefore the total solution space is limited and the heuristic approach would suffice. For more details of the simulation results please refer to Section VII.

4) *Detections and Testings:* Testing is carried out over a new set of layouts using the layout analyzer, the hotspot identifiers, and the refinement architecture in Fig. 8. Our proposed

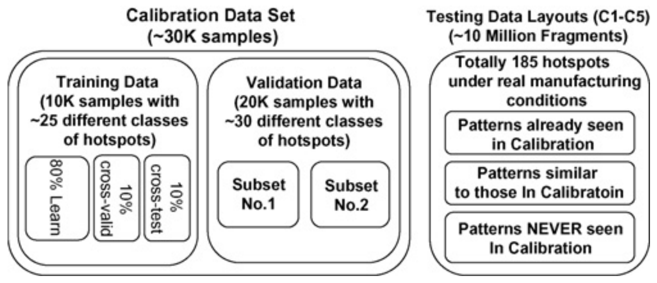

Calibration Data Set
(~30K samples)
Training Data
(10K samples with ~25 different classes of hotspots)
80% Learn
10% cross-valid
10% cross-test
Validation Data
(20K samples with ~30 different classes of hotspots)
Subset No.1
Subset No.2
Testing Data Layouts (C1-C5)
(~10 Million Fragments)
Totally 185 hotspots under real manufacturing conditions
Patterns already seen in Calibration
Patterns similar to those In Calibratoin
Patterns NEVER seen In Calibration

Fig. 10. Illustration of the calibration and testing data sets.

flow differentiates the data streams and guides them through successive levels of identifications, resulting in satisfactory overall performance.

VII. Simulation and Testing

The simulation process involves the calibration stage and the detection stage. The data used in these stages to calibrate (train/validate) and test the proposed models are referred to as the calibration data and the testing data, which are illustrated in Fig. 10. Note that in the real practice, the testing layouts generally have more hotspot patterns than those the calibration data set can represent, since it is usually too costly to enumerate all the possible hotspot patterns. In this section, we will test and analyze our proposed methodology in terms of both hotspot detection accuracy and prediction generality over new testing layouts.

A. *Layout Calibration Stage*

We break down the calibration stage into two major steps: 1) under real manufacturing conditions, multiple hotspot identifiers are trained and configured over a relatively small set of layouts that are fully placed and routed in 45 nm technology, and 2) under the successive local refinement framework, the threshold optimizations are carried out over the validation data set to reduce detection inaccuracies introduced by the unbalanced hotspot/non-hotspot ratio.

For the first step, we perform sparse samplings over a number of different size training layouts (summing up to $\sim$3000 μm^2 in area) and generate a data set with 5K non-hotspot samples, equivalent to the total number of fragments/patterns of a 500 μm^2 design. We also sample the hotspot patterns (totally around 25 classes) from all the training layouts and replicate them to around 5K. Then we combine the hotspot and non-hotspot data sets together to generate our final training data set, which contains 10K data samples with hotspot versus non-hotspot ratio 1:1. The validation data set is built similarly with 20K samples, including about 30 different classes of hotspot samples.

The main goal of the sampling is to properly select representative samples from the training and the validation layouts for more effective learning process. In our simulation setups, this sampling is implemented by first marking the layouts into very small grid regions, then selecting several fragments of interest from different locations within each region. Note that the training and validation sets do not include all possible types of hotspot patterns. In other words, we must build a detection

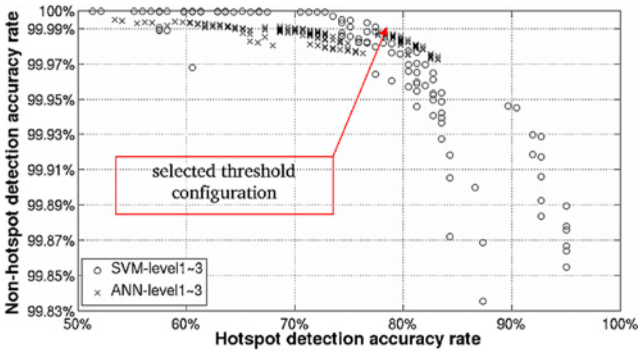


Fig. 11. Spectrum of detection accuracies in the threshold optimization.

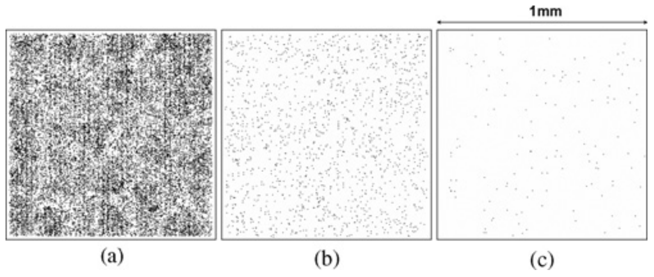


Fig. 12. Visualizations of false-alarm locations when simulating (a) [20] on C5, (b) [26] on C5, and (c) our method on C5 (barely visible: $\sim$100 false-alarm spots on 1 mm^2 layout).

engine to best predict new types of hotspots that have not been calibrated in the training stage. For this purpose in the second step, we will select the Bayesian-decision thresholds of the machine-learning models.

For the second step, Fig. 11 plots a fine-grid simulation result of the threshold optimization process in the calibration stage. The x-axis is the hotspot detection accuracy $Hhit$ over the validation set $subset2$, and y-axis is the non-hotspot detection accuracy $Nhit$. Every point on the plot represents a different combination of the thresholds as in Fig. 7, the data depicted with cross markers is derived through applying hotspot identifier-ANNs and the circles are through SVMs. As a quick observation, ANNs give more robustness against noises, while the SVMs achieve better performance ψ_{perf}. In our case, we pick the upper-right corner threshold combinations on ANN and SVM curves, respectively, in Fig. 11 for the purpose of generating the identifiers: $Model\_1$ to $Model\_3$.

The simulation time cost of the layout calibration stage is around 2.0 CPU h, which is about 35 min on a quad-core Linux workstation. We break down the calibration runtime as follows in Table IV, where the layout analyzer runs at an average speed of around 6.6K samples per second among all calibration layouts on a single core workstation; and the core ANN/SVM model generations take about 5–10 min for 10K–20K samples per hotspot identifier model.

B. *Hotspot Detection Stage*

With the three levels of hotspot identifiers ready from the calibration stage, we perform hotspot identifications over five industry design layouts under the real manufacturing conditions. Details of layouts C1–C5 are in Table V, where the sizes of the test cases range from 900 μm^2 to 1 mm^2. We can tell from the total number of geometry fragmentation in

TABLE IV

RUNTIME BREAKDOWN OF THE CALIBRATION STAGE

Layout Analyzer	Hotspot Identifier Models (Three Levels for ANN/SVM)	Threshold Opt.
<0.08 CPU h	<0.42 CPU h	∼1.5 CPU h

TABLE V

DETAILS OF TESTING LAYOUTS

	Dimension	Hotspot Count	Non-Hotspot Count
C1	$30\times30\,\mu m^2$	4	4.955k
C2	$50\times50\,\mu m^2$	0	17.37k
C3	$200\times200\,\mu m^2$	6	293.5k
C4	$500\times500\,\mu m^2$	38	1779k
C5	$1000\times1000\,\mu m^2$	137	7175k

C1–C5 (hotspot count + non-hotspot count) that they contain very densely placed and routed METAL1 layer. Note that C1–C5 contain various types of lithography hotspots whose geometric patterns are very hard to enumerate *a priori* as a library, i.e., precise geometry matching will result in significant loss of generality to new hotspot patterns (outside the training set), while other graph-based methods (e.g., dual-graph matching [12]) usually lead to large amount of detection false alarms.

In the following sections, we test and evaluate the proposed methodology on the above design layouts in terms of *Hhit*, *Hmis*, *Hextra*, *Nhit*, and runtime under a real set of industry-strength manufacturing conditions at 45 nm process. Note that these designs did not participate in the calibration stage. Also note that our machine-learning flow is designed to be generic to apply to other technology nodes.

Table VI shows the simulation results of hotspot identification accuracies using our proposed methodology, where GD represents the global detection stage alone (using only hotspot identifier *Model*_1) and GD + LR represents the combination of the global detection and the successive local refinements (using *Model*_1-*Model*_3). We observe that although GD leads to satisfactory hotspot identification accuracy, it is LR that plays a vital role in bringing down the false alarms. Also in these results, we better appreciate the detection challenges under real manufacturing conditions due to highly unbalanced quantities of hotspot and non-hotspot patterns. Fig. 12 provides a good visualization of detection false alarms on the 1 mm^2 area design C5 by using two previous methods and our proposed methodology. From the figure it is obvious that our approach achieves the least amount of false alarms (without penalties on the hotspot accuracy *Hhit*), therefore the workload overheads can be kept minimum in the post-layout hotspot removal stages. We can also appreciate that although a 95% of non-hotspot identification accuracy seems good percentage wise, we have to raise it to above, e.g., 99.85% to avoid excessive post-layout corrections under the real manufacturing conditions.

Table VII shows the runtime breakdown of the hotspot detection stage, where the context calibration (a table lookup procedure) runs at a constant speed of about 8.3K samples per second, and the hotspot identifier applies the SVM or ANN model to each sample at a constant speed of 1.2K(SVM) or 1.7K(ANN) samples per second.

Since the speeds of the context calibration and the hotspot identifier dominate the detection stage, the detection runtime scales linearly to the total number of fragments in the design layout. Since we use a small shielding factor in the fragmentation rules, the fragmentation process becomes localized and the number of fragments per area is proportional to the local metal track density. Given design layouts with similar metal densities, the detection runtime therefore scales linearly to the areas of the layouts. So far we have proved the linear runtime scalability of the proposed machine-learning method, which will be further validated with simulation results.

TABLE VI

SIMULATION RESULTS OF OUR PROPOSED METHODOLOGY ON INDUSTRY LAYOUTS UNDER REAL MANUFACTURING CONDITIONS

	Using Hotspot Identifiers-ANN				Using Hotspot Identifiers-SVM			
	GD		GD + LR		GD		GD + LR	
	Hhit[a]	*Nmis*	*Hhit*	*Nmis*	*Hhit*	*Nmis*	*Hhit*	*Nmis*
C1	4	315	3	18	4	251	4	5
C2	–	109	–	11	–	81	–	3
C3	6	493	5	31	6	355	5	7
C4	32	3020	30	195	34	1983	31	38
C5	121	10 960	111	485	122	7535	114	135

[a] *Hhit* is the number of correctly identified hotspots; *Nmis* is the number of incorrectly identified non-hotspots (false alarms).

TABLE VII

RUNTIME BREAKDOWN OF THE HOTSPOT DETECTION STAGE

Machine-Learning Models	Layout Analyzer		Hotspot Identifier
	Signature Measurement	Context Calibration	Levels 1 + 2 + 3
ANN	∼4%	∼16%	∼80%
SVM	∼3%	∼12%	∼85%

Table VIII and Fig. 13 show comparisons of accuracies and runtime between our approach and some existing machine-learning-based studies. The simulation results of [20] and [26] are collected by directly running the original source codes on our testing cases C1–C5. We also implement the original method of [17] inside [22] via an advanced programming interface in C/C++. Due to environment compatibility issues, we modified the original approach in [17] slightly for better memory efficiency, which could possibly end up with more runtime. However, it would suffice as a first-order estimation.

As shown in Table VIII, our proposed methods demonstrate better performance with superior runtime under real manufacturing conditions. With similar or slightly better hotspot detection rate *Hhit* of 82–89%, we achieve hotspot false-alarm reductions ranging from 2.4X (between [26] and hotspot identifiers-ANN-GD + LR) to 2300X (between [17] and hotspot identifiers-SVM-GD + LR). As visualized in Fig. 13, the simulation runtime speedups range from 5X (between [20] and hotspot identifiers-SVM) to 237X (between [17] and hotspot identifiers-ANN), when calibred in CPU h/mm^2 unit. Such speedups mainly owe to our proposed identification methodology that is free of time-consuming data transformations, such as grid density extraction, distance transform, and histogram calculations.

Inside our locally refined detection methodology, ANN models result in faster runtime than SVM models while SVM models outperform ANN models in both hotspot and non-hotspot detection accuracy. We also notice that the runtime overhead introduced by the successive local refinements is negligible, owing to: 1) the ultrafast speed of the layout analyzer and the hotspot identifiers; and 2) the exponential reduction of false alarms achieved by each additional level of

TABLE VIII
PERFORMANCE COMPARISON BETWEEN PREVIOUS HOTSPOT IDENTIFICATION METHODS AND OUR METHOD

	DAC09 [17][a]	SPIE09 [26][b]	ICICDT09 [20][b]	Identifiers-ANN		Identifiers-SVM	
				GD	GD + LR	GD	GD + LR
Average hotspot detection accuracy *Hhit*	88%	80%	87%	88%	82%	89%	83%
Average Nonhotspot detection accuracy	95.818%	99.985%	99.809%	99.847%	99.994%	99.895%	99.998%
Average false-alarm count per mm^2	300K	1.1K	13.5K	10K	0.45K	7.5K	0.13K
Average CPU runtime[c] per mm^2	356	30	10	1.5	1.5	2.0	2.0
Average real-time runtime[d] per mm^2	100	8.50	2.80	0.40	0.40	0.52	0.52

[a]Implemented within the same geometry engine framework [22] with slight modifications for compatibility reasons. Calibrated on a total 100×100 μm^2 region from layout C5 due to runtime constraint.
[b]Results collected by running the original source codes on C1–C5.
[c]Runtime calibrated in the unit of CPU h/mm^2 on Linux station with 2.8 GHz quad-core processors.
[d]Runtime calibrated in the unit of real-time h/mm^2 on Linux station with 2.8 GHz quad-core processors.

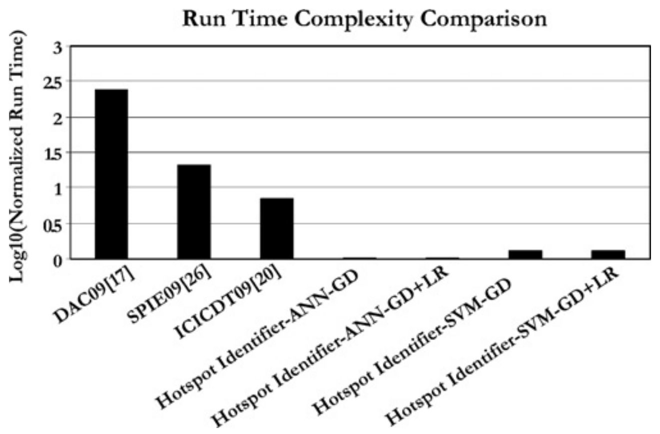


Fig. 13. Detection stage runtime comparison between our approach and previously existing works, in the unit of $Log10$(h/mm^2).

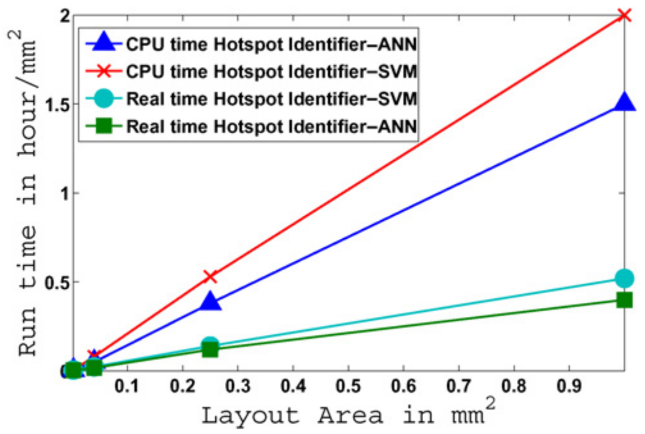


Fig. 14. Runtime scalability of our methodology in the detection state.

refinement. These make our methodology especially suitable for guiding lithography-friendly physical design.

To evaluate the runtime scalability of our methodology on multicore platforms, we implemented the layout analyzer and the hotspot identifiers inside [22] with parallel processing friendly functions and procedures. Assisted by the layout segmentation and refactoring features provided by [22], we plot in Fig. 14 the runtime of our approach when simulated on a Linux workstation with a Intel quad-core processor. In Fig. 14, C1–C5 have increasing areas from 900 μm^2 to 1.0 mm^2. We can see that our methodology shows linear runtime complexity as design layouts scale to full-chip size. In comparing the CPU time and the real time of both hotspot identifiers-ANN and hotspot identifiers-SVM, we observe that our methodology demonstrates very good multicore scalability. From "CPU Time" to "Real Time," ANN models and SVM models achieve 73.3% and 74.0% runtime reduction on quad-core machines, respectively. The main reasons of such complexity and scalability owe to: 1) good memory/workload management and database refactoring mechanisms of [22]; 2) the novel measurement operators we propose in the layout analyzer; and 3) the ultrafast calculations involved with the hotspot identifiers.

Based on the results in Table VIII, we summarize and compare several classes of machine-learning-based hotspot identification methods in Table IX, where we further highlight the key contributions of our proposed methodology. First, we employed powerful hotspot identifiers to achieve high hotspot detection accuracy. Second, we proposed efficient successive refinements to suppress detection false alarms. Third, we employed ultrafast layout analyzer for advantageous runtime efficiency. Without sliding windows or raster scanning, our methodology enables full design layout analysis without leaving any cold spots. Consequently, there is no need to seek tradeoffs between runtime and detection accuracy/coverage. It is obvious that our method is most suitable to guide lithography-aware physical design due to these performance advantages. Although the machine-learning methodology still misses some hotspots, it is acceptable as a fast and high-fidelity prediction at early (physical) design stages.

TABLE IX
COMPARISONS BETWEEN EXISTING METHODS

	[17]	[26]	[20]	Ours
Identifier model	SVM	Regression	ANN	**ANN, SVM**
Accuracy	High	Medium	High	**High**
False alarms	High	Low	High	**Low**
Window-based	Yes	No	Yes	**No**
Scanning coverage	Tradeoff	Full	Tradeoff	**Full**
Runtime	Usually slow[a]	Medium	Medium[a]	**Very fast**

[a]For scanning window-based approaches, runtime can be traded-off at the cost of detection accuracy and coverage.

C. Quantitative Comparison with Pattern Matching

In this section, we further assess our proposed machine-learning methodology in comparison to a typical pattern-matching-based detection flow, shown in Fig. 15. The main objective of such a comparison is to demonstrate the advantageous capabilities of the machine-learning methods in terms of predicting new hotspot patterns that were not previously characterized in the calibration stage. This is very important because in the real practice, exhaustive and complete pattern enumeration is usually too expensive to perform *a priori*.

Typical pattern-matching flows take three steps: 1) analyze the patterns in the calibration data set and build a special

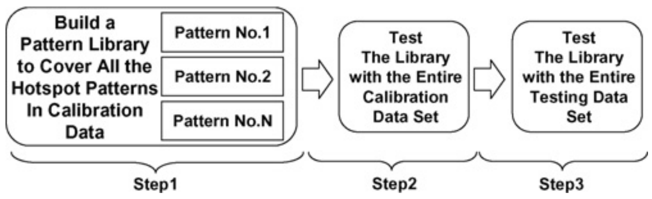

Build a Pattern Library to Cover All the Hotspot Patterns In Calibration Data
Pattern No.1
Pattern No.2
Pattern No.N
Test The Library with the Entire Calibration Data Set
Test The Library with the Entire Testing Data Set
Step1
Step2
Step3

Fig. 15. Typical hotspot detection flow using a pattern-matching library.

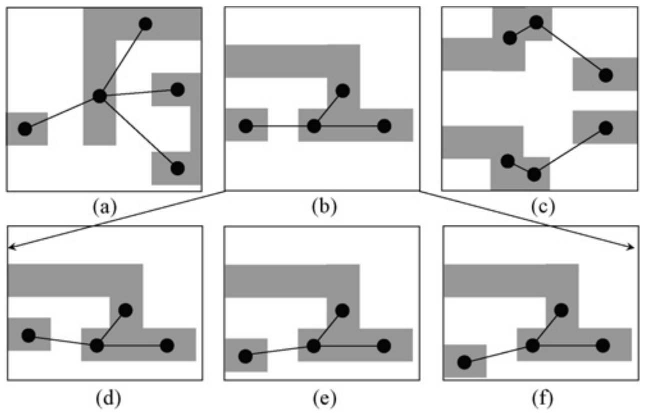


Fig. 16. Some illustrative examples of hotspot patterns. (a)–(c) Example pattern in PM library. (d)–(f) Specific pattern under category (b).

pattern library to cover all the known hotspot patterns; 2) apply the pattern library over the entire calibration data set and adjust the library patterns to reduce the false alarms; and 3) apply the final pattern library to the entire testing layouts C1–C5.

Initially in Step 1, we processed the hotspot samples and developed about ten types of structure elements based on various line-end/corner/jog/contact geometries. Their combinations form a pool of patterns that covers the entire set of hotspots seen in the calibration data. However, the false alarms are unacceptably high, thus further refinement is required.

In Step 2, we develop a number of special complex pattern combinations based on fast dual-graphs [12] and range pattern methods [14], then merge them together with the existing patterns. After some fine-tuning, we finalize the library with about 20 categories of special hotspot patterns. Within each category there are a number of specific patterns. For each specific hotspot pattern, we further enumerate a number of "similar" hotspot patterns that share the same structure skeleton but with slight differences in the layout. This way we can enhance the prediction generality of the library over new testing samples. A few specific example patterns are depicted in Fig. 16. In Table X, we see that this library provides 100% *Hhit* and 96.2% *Nhit* over the calibration data set. Such an *Hhit* outperforms that of the machine-learning methods over the same data set, but the *Nhit* indicates that the detection false alarms from such a pattern matcher is much worse than most of the existing machine-learning models.

In Step 3, we apply the special pattern library to the testing layouts C1–C5 and show the average detection performance in Table X. Note there are new hotspot patterns in the testing layouts that were not present in the calibration data set. From the results we can see that the *Hhit* reduces to about 74% meanwhile the false alarms further worsens. This tells us

TABLE X
PERFORMANCE OF THE EMPLOYED PATTERN-MATCHING METHOD

Calibration Data			Testing Data			
Hhit	*Hextra*	*Nhit*	*Hhit*	*Hextra*	*Nhit*	Runtime
100%	1.9K	96.2%	74%	2.3K	95.6%	0.9 CPU h/mm^2

that the pattern-matching techniques employed have significant disadvantage in predicting new/unseen hotspots. Comparing with such a pattern matcher, machine-learning techniques achieve 11% (82% versus 74%) to 20% (89% versus 74%) better *Hhit* meanwhile much better false-alarm rate. The total runtime for scanning the pattern library in the testing stage takes about 0.9 CPU h/mm^2, which is around 12 min for a mm^2 design on a quad-core workstation. This gives the pattern matcher 40% (1.5 versus 0.9) to 55% (2.0 versus 0.9) of runtime advantage compared with the machine-learning methods. However, we also need to consider the time overhead spent in Step 1 of Fig. 15, which includes the development and optimization of the pattern library.

Overall speaking, our proposed machine-learning methods have great advantage in predicting new/unseen types of hotspots and suppressing detection noise to achieve very low false alarms. The pattern-matching methods on the other hand, are very good at precise detection of already characterized hotspots with very fast speed. As a future work, it will be very interesting to combine machine learning and pattern matching together for the ultimate hotspot detection.

VIII. CONCLUSION

Under real and continuously improving manufacturing conditions, lithographic hotspot detection faces many critical challenges. To alleviate the huge runtime cost of current lithographic hotspot simulators, we proposed a fast and high fidelity hotspot detection methodology providing full layout, feature-centric assessment. We incorporated hotspot signature measurements and powerful hotspot identifiers into a successively refined detection flow. Our algorithms were implemented and tested with an industry-strength engine [22] under real PDK/manufacturing conditions, demonstrating significant advantages over previous studies in both detection false alarms and runtime. Our method also showed high prediction generality for hotspot patterns that were not previously characterized, i.e., when exhaustive pattern enumeration becomes too costly.

ACKNOWLEDGMENT

The authors would like to thank Professor J. Ghosh at the University of Texas, Austin, for his insightful comments and feedback.

REFERENCES

[1] *International Technology Roadmap for Semiconductors*. (2010) [Online]. Available: http://www.itrs.net
[2] J. Mitra, P. Yu, and D. Z. Pan, "RADAR: RET-aware detailed routing using fast lithography simulations," in *Proc. Design Automat. Conf.*, Jun. 2005, pp. 369–372.
[3] M. Cho, K. Yuan, Y. Ban, and D. Z. Pan, "ELIAD: Efficient lithography aware detailed router with compact printability prediction," in *Proc. Design Automat. Conf.*, Jun. 2008, pp. 504–509.
[4] T.-C. Chen, G.-W. Liao, and Y.-W. Chang, "Predictive formulae for OPC with applications to lithography-friendly routing," in *Proc. Design Automat. Conf.*, Jun. 2008, pp. 510–515.

[5] D. Z. Pan, M. Cho, and K. Yuan, "Manufacturability aware routing in nanometer VLSI," *Foundations Trends Electron. Design Automat.*, vol. 4, no. 1, pp. 1–97, 2010.

[6] L.-D. Huang and M. D. F. Wong, "Optical proximity correction (OPC) friendly maze routing," in *Proc. Design Automat. Conf.*, Jul. 2004, pp. 186–191.

[7] Y. Borodovsky, "Lithography 2009 overview of opportunities," *Semicon. West*, 2009 [Online]. Available: http://www.semi.org/cms/groups/public/documents/web_content/ctr_030805.pdf

[8] C. Bencher, "SADP: The best option," *Nanochip Tech. J.*, vol. 5, no. 2, pp. 8–13, 2007.

[9] L. W. Liebmann, J. Kye, B. Kim, L. Yuan, and J. Geronimi, "Taming the final frontier of optical lithography: Design for sub-resolution patterning," *Proc. SPIE*, vol. 7641, p. 764105, Apr. 2010.

[10] J. Kim and M. Fan, "Hotspot detection on post-OPC layout using full chip simulation based verification tool: A case study with aerial image simulation," *Proc. SPIE*, vol. 5256, pp. 919–925, Dec. 2003.

[11] E. Roseboom, M. Rossman, F.-C. Chang, and P. Hurat, "Automated full-chip hotspot detection and removal flow for interconnect layers of cell-based designs," *Proc. SPIE*, vol. 6521, p. 65210C, Feb. 2007.

[12] A. B. Kahng, C.-H. Park, and X. Xu, "Fast dual graph based hotspot detection," *Proc. SPIE*, vol. 6349, p. 63490H, Sep. 2006.

[13] J. Xu, S. Sinha, and C. C. Chiang, "Accurate detection for process hotspots with vias and incomplete specification," in *Proc. Int. Conf. Comput. Aided Design*, 2007, pp. 839–846.

[14] H. Yao, S. Sinha, C. C. Chiang, X. Hong, and Y. Cai, "Efficient process hotspot detection using range pattern matching," in *Proc. Int. Conf. Comput. Aided Des.*, Nov. 2006, pp. 625–632.

[15] N. Nagase, K. Suzuki, K. Takahashi, M. Minemura, S. Yamauchi, and T. Okada, "Study of hotspot detection using neural network judgement," *Proc. SPIE*, vol. 6607, p. 66071B, May 2007.

[16] N. Ma, J. Ghan, S. Mishra, C. Spanos, K. Poolla, N. Rodriguez, and L. Capodieci, "Automatic hotspot classification using pattern-based clustering," *Proc. SPIE*, vol. 6925, pp. 692505–692510, Feb. 2008.

[17] D. G. Drmanac, F. Liu, and L.-C. Wang, "Predicting variability in nanoscale lithography processes," in *Proc. Design Automat. Conf.*, Jul. 2009, pp. 545–550.

[18] J.-Y. Wuu, F. G. Pikus, J. A. Torres, and M. Marek-Sadowska, "Detecting context sensitive hot spots in standard cell libraries," *Proc. SPIE*, vol. 7275, p. 727515, Mar. 2009.

[19] J.-Y. Wuu, F. G. Pikus, J. A. Torres, and M. Marek-Sadowska, "Rapid layout pattern classification," in *Proc. Asia South Pacific Design Automat. Conf.*, Jan. 2011, pp. 781–786.

[20] D. Ding, X. Wu, J. Ghosh, and D. Z. Pan, "Machine learning based lithographic hotspot detection with critical feature extraction and classification," in *Proc. IEEE Int. Conf. IC Design Technol.*, May 2009, pp. 219–222.

[21] D. Ding, J. A. Torres, F. G. Pikus, and D. Z. Pan, "High performance lithographic hotspot detection using hierarchically refined machine learning," in *Proc. Asia South Pacific Design Automat. Conf.*, 2011, pp. 775–780.

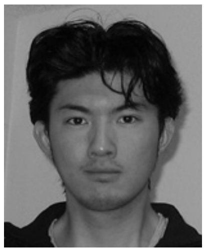
[22] *CALIBRE*, Mentor Graphics Corporation, Wilsonville, OR, 2011.

[23] J. A. Torres and N. C. Berglund, "Integrated circuit DFM framework for deep sub-wavelength processes," *Proc. SPIE*, vol. 5756, pp. 39–50, Mar. 2005.

[24] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Int. Conf. Neural Netw.*, Mar.–Apr. 1993, pp. 586–591.

[25] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working set selection using second order information for training support vector machines," in *J. Mach. Learn. Res.*, vol. 6, pp. 1889–1918, Jan. 2005.

[26] J. A. Torres, M. Hofmann, and O. Otto, "Directional 2D functions as models for fast layout pattern transfer verification," *Proc. SPIE*, vol. 7275, p. 72750N, Feb. 2009.



Duo Ding received the M.S.E. degree in electrical engineering from the University of Texas at Austin, Austin, in 2008.

After his Masters degree, he joined the Computer Engineering Research Center, University of Texas at Austin, working on computer-aided design (CAD) integration of architecture, circuits, and advanced lithography to assist high yield nanometer integrated circuit (IC) design. He has published over 15 refereed papers in international conferences, journals, workshops, forums, and others. His current research interests include CAD physical design and data learning for IC manufacturability/yield/reliability, device modeling, and low-power CAD for on-chip nanophotonic interconnect.

Mr. Ding was the recipient of a Best Student Paper Award at ICICDT'09, and multiple Best Paper Award nominations at ICCAD in November 2011 and at ASPDAC in January 2012.

J. Andres Torres received the B.S. degree from the National Autonomous University of Mexico, Mexico City, Mexico, and the M.S. degree from the University of Wisconsin-Madison, Madison, both in chemical engineering, and the Ph.D. degree in electrical engineering from the Oregon Graduate Institute, Beaverton.

Since 2001, he has been with the Design-to-Silicon Division, Mentor Graphics Corporation, Wilsonville, OR, working in the areas of resolution enhancement technologies, design for manufacturing, and process hardening layout techniques. He is currently the Product Lead Engineer of the Litho Friendly Design Group in the Design-to-Silicon Division. He has published over 50 papers and holds five patents in the area of semiconductor manufacturing. His current research interests include interactions between manufacturing process and electronic design flows to exploit areas of design and process co-optimization that provide more predictable and manufacturable designs.

David Z. Pan (S'97–M'00–SM'06) received the Ph.D. degree (with honors) in computer science from the University of California at Los Angeles (UCLA), Los Angeles, in 2000.

From 2000 to 2003, he was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY. He is currently an Associate Professor (with tenure) with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin. He has published over 140 technical papers in refereed journals and conferences, and holds eight U.S. patents. His current research interests include nanometer physical design, design for manufacturability/reliability, vertical integration of technology/circuits/architecture, and compuetr-aided design (CAD) for emerging technologies.

Dr. Pan has received a number of awards for his research contributions and professional services, including the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the UCLA Engineering Distinguished Young Alumnus Award in 2009, the SRC Inventor Recognition Award (thrice) in 2000 and 2008, the IBM Faculty Award (four times) in 2004–2006, and 2010, the ASPDAC 2010 Best Paper Award, the DATE 2010 Best IP Award, the ISPD 2011 Best Paper Award, the ICICDT 2009 Best Student Paper Award, the SRC Techcon Best Paper in the Session (twice) in 1998 and 2007, and many other Best Paper Award Nominations in top conferences, such as DAC, ICCAD, ASPDAC, and ISPD. He has also received the Dimitris Chorafas Foundation Research Award in 2000, the eASIC Placement Contest Grand Prize in 2009, the ISPD Routing Contest Award in 2007, and the ACM Recognition of Service Award, in 2007 and 2008. He was a Cadence Distinguished Speaker in 2007 and an IEEE CAS Society Distinguished Lecturer from 2008 to 2009. He was an Associate Editor for four premier IEEE journals, including the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS since 2006, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS since 2007, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS from 2008 to 2009, and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS from 2006 to 2007. He is also an Area Editor for the *Journal of Computer Science and Technology* since 2010, an Associate Editor for the IEEE CAS Society Newsletter since 2007, and a Guest Editor of the IEEE COMPUTER-AIDED DESIGN Special Section on "International Symposium on Physical Design" in 2007 and 2008. He is a member of the Design Technology Working Group of the International Technology Roadmap for Semiconductors. He is an Elected Officer of the IEEE CANDE Committee (Workshop Chair in 2007, Secretary in 2008, and Chair in 2009) and has been the Chair of the ACM/SIGDA Technical Committee on Physical Design since 2009. He is the General Chair of ISPD 2008 and the Steering Committee Chair of ISPD 2009. He has served on the technical program committees of major VLSI/CAD conferences, including ASPDAC as the Subcommittee Chair, DAC as the Subcommittee Chair, ICCAD as the Subcommittee Chair, among many others. He is a Technical Advisory Board Member with Pyxis Technology, Inc., Laval, QC, Canada. He is a member of ACM/SIGDA, SPIE, and ASEE.