EPIC: Efficient Prediction of IC Manufacturing Hotspots With a Unified Meta-Classification Formulation

Duo Ding, Bei Yu, Joydeep Ghosh and David Z. Pan
ECE Dept. Univ. of Texas at Austin, Austin, TX 78712
{ding, bei}@cerc.utexas.edu, {ghosh, dpan}@ece.utexas.edu

ABSTRACT

In this paper we present *EPIC*, an efficient and effective predictor for IC manufacturing hotspots in deep sub-wavelength lithography. EPIC proposes a unified framework to combine different hotspot detection methods together, such as machine learning and pattern matching, using mathematical programming/optimization. EPIC algorithm has been tested on a number of industry benchmarks under advanced manufacturing conditions. It demonstrates so far the best capability in selectively combining the desirable features of various hotspot detection methods (3.5-8.2% accuracy improvement) as well as significant suppression of the detection noise (e.g., 80% false-alarm reduction). These characteristics make EPIC very suitable for conducting high performance physical verification and guiding efficient manufacturability friendly physical design.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuit**]: Design Aids

Keywords

Design for Manufacturability, Lithography Hotspots, Meta Classification, Machine Learning, Pattern Matching

1. INTRODUCTION

Due to the widening gap between the continuous scaling of feature-size and the limited lithography capability [1], the semiconductor industry is critically challenged in both IC design and manufacturing. To address these challenges, design-aware manufacturing and manufacturing-friendly design techniques have been developed to avoid high variability design patterns (process hotspots) and to ensure high product yield at post Silicon stage. During such processes, printing masks are usually re-targeted and optimized through powerful resolution enhancement techniques (RETs) such as Sub-Resolution Assist Features, Optical Proximity Correction, etc. At the same time, various merging lithography technologies are under active research and development, including Double (Multiple) Patterning lithography, E-beam lithography and EUV lithography. However, these technologies still suffer from different degrees and types of printing variabilities, therefore layout dependent lithography hotspots remain a challenging issue.

To optimize the masks of a design for better printability, one approach is to first locate the lithography hotspots in a layout, then fix them in a construct-by-correction manner. Meanwhile in recent years, CAD methodologies have evolved to incorporate RET models into early design stages (e.g., detailed routing) to avoid lithography-unfriendly patterns in a correct-by-construction manner [2–6]. Consequently, fast and accurate lithography hotspot detection becomes a common and critical issue for a wide range of applications in both design and manufacturing.

However, the quests for such detection methods have been critically challenged in many aspects: (1) designs are getting more complex; (2) under the evolving manufacturing conditions, the number of real hotspots is only a very small fraction of the entire design, making it very difficult to achieve high detection accuracies and low false-alarms simultaneously; (3) detections are seriously run-time constrained due to short turn-around-time, etc.

Current state-of-the-art hotspot detection methods mainly fall into 3 categories. (1) lithography simulations are very accurate but CPU intensive. (2) Machine learning techniques [7–13] with good noise suppression capability are still in need of further accuracy improvement. (3) Pattern matching techniques [14–17] that are very good at detecting pre-characterized hotspot patterns lack the capability to predict never-before-seen hotspots. This is especially problematic when new types of designs are involved after the original pattern library is built.

Recently in [18], a hotspot detection flow was proposed to hybridize the strengths of machine learning models and pattern matching models. Such a flow feeds data samples to a pattern matcher first, then employs machine learning classifiers to further examine the non-hotspot data set produced by the pattern matcher. It demonstrates good performance trade-off between detection accuracies and false-alarms suppression compared to the previous works. However, its ad-hoc nature can make the performance fine-tuning and optimization processes very costly.

In order to better address the problem, we propose *EPIC*: an efficient meta-classification formulation (Fig. 1) to combine various hotspot detection techniques into a unified and automated framework that selectively adopts their strengths and suppresses their drawbacks. Based on the theoretical framework in Order Statistics [19], we propose a new CAD flow with different types of *base classifiers* and optimize the flow via constrained quadratic programming.

The rest of the paper is organized as follows. In Section 2 we further motivate the *meta-classification* methodology and summarize our main contributions. Section 3 details the overall CAD flow together with an overview of the

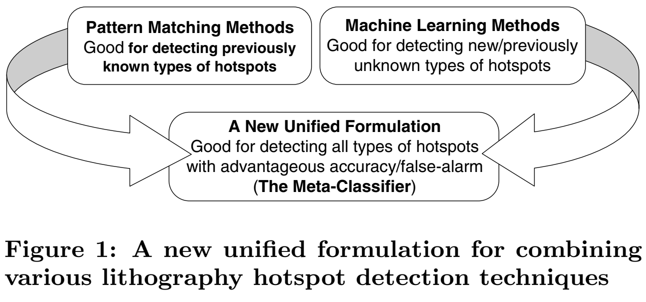

Pattern Matching Methods
Good for detecting previously known types of hotspots
Machine Learning Methods
Good for detecting new/previously unknown types of hotspots
A New Unified Formulation
Good for detecting all types of hotspots with advantageous accuracy/false-alarm
(The Meta-Classifier)

Figure 1: A new unified formulation for combining various lithography hotspot detection techniques

This work is supported in part by NSF, SRC, Oracle and NSFC.
Asia and South Pacific Design Automation Conference (ASPDAC), Jan. 30 – Feb. 2 2012, Sydney, Australia.

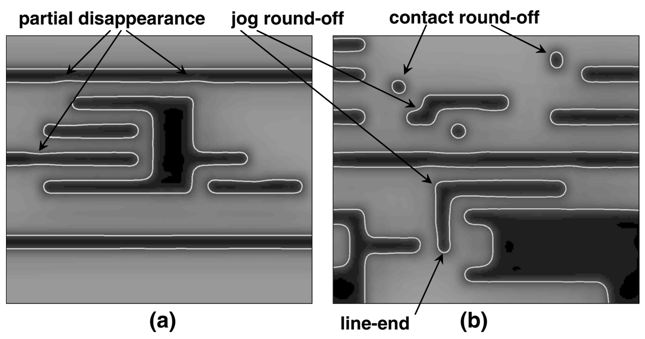


Figure 2: Examples of lithography hotspot patterns

meta-classifier construction. Section 4 gives a brief description of several classes of building-block detection techniques, followed by Section 5, where these techniques are combined under the *meta-classifier* with mathematical programming and optimization techniques. Section 6 presents the results and analysis. Section 7 concludes the paper.

2. MOTIVATION AND CONTRIBUTIONS

In the previous section we discussed the need for a systematic and unified meta-classification methodology to selectively combine certain features of multiple hotspot detection engines. In this section, we use an example to further motivate such a meta-classifier.

Fig. 2 presents the printed images of 2 local regions from certain design at $32nm$ technology node after applying RETs. We can make several observations from Fig. 2(a) and (b). First, there are various types of process hotspots, featuring complex patterns related to line-ends, jogs, corners or contacts, etc. Second, hotspot patterns suffer from different amount of manufacturing variation, which is usually measured by the Edge Placement Error (EPE). Therefore by setting different EPE thresholds, we can classify lithography hotspots into multiple categories. This allows us more efficiency to study the manufacturability and yield effects of each category. We first define lithography hotspots:

DEFINITION 1. ***Hotspot***: A pattern (or part of a pattern) in an IC design layout that suffers excessive EPE/variation under lithography printing variation at fabrication stage.

The definition of lithography hotspots is dependent on the EPE error tolerance of a design. Excessive EPE can lead to electrical errors (parasitics variation, timing issues, etc.) or even logic errors (shorts, opens, etc.). To avoid these issues and assist design sign-off and manufacturing closure, lithography hotspots should be properly predicted and avoided during early design stages with short turn-around-time. In this paper, we use the following two targets to quantitatively calibrate the prediction performance:

DEFINITION 2. ***Hotspot Accuracy***: The rate of correctly predicted hotspots among the set of actual hotspots.

This rate characterizes the success rate of hotspot prediction within the set of actual hotspots. We also use *Hit* to represent the actual count of correctly predicted hotspots, or equivalently, the rate of *Hotspot Accuracy* percentage wise.

DEFINITION 3. ***Hotspot False-Alarm***: The rate of incorrectly predicted non-hotspots over the set of actual hotspots.

This rate represents the over-shoots of the prediction, i.e., the set of non-hotspots predicted incorrectly as hotspots. We also use *Extra* to denote the actual count of such a set, or equivalently, the rate of *Hotspot False-Alarm* in percentage.

Next we motivate a meta-classification flow to concurrently optimize *Hit* and *Extra* on top of powerful hotspot prediction methods. During our prediction process, each fragment geometry in the layout will be processed and analyzed by multiple hotspot detection engines. Suppose we input pattern i to a machine learning classifier ML and a pattern matcher PM at the same time and the prediction results are x_i^{ML} and x_i^{PM}, respectively. x_i^{ML} takes certain value between -1 (non-hotspot) and +1 (hotspot), while x_i^{PM} usually is either -1 (non-hotspot) or +1 (hotspot). Thus the simplified meta-classification problem becomes the following motivational problem:

***Given* decisions x_i^{ML} and x_i^{PM} over the same design pattern i, *decide* the final hotspot target label T_i^{meta} to simultaneously maximize *Hotspot Accuracy* and minimize *Hotspot False-Alarms*.**

First, it is easy to see that T_i^{meta} is +1 if both x_i^{ML} and x_i^{PM} are (very close to) +1; T_i^{meta} is -1 if both x_i^{PM} and x_i^{ML} are (very close to) -1. Second, in the cases when x_i^{ML} and x_i^{PM} disagree with each other, we introduce the *weighting functions* $f^{ML}(x)$ and $f^{PM}(x)$ to adjust the weights and improve detection performance.

$$T_i^{meta} = \digamma^{meta}\{x_i^{ML} \cdot f^{ML}(x_i^{ML}) + x_i^{PM} \cdot f^{PM}(x_i^{PM})\} \quad (1)$$

If we define T_i^{meta} as in Eqn.(1) above, we can pre-calibrate the *weighting functions* with accurate lithography simulations as golden targets. Then we can use the calibrated functions onto new layout fragments by applying Eqn.(1), where $\digamma^{meta}$ is a threshold cut-off function defined as follows,

$$\digamma^{meta}(x) = \begin{cases} +1 \ (hotspot), & \text{if } x \geq \theta \\ -1 \ (nonhotspot), & \text{if } x < \theta \end{cases} \quad (2)$$

Such a formulation combines machine learning and pattern matching techniques meanwhile preserves generality to cover both cases, as if $f^{ML}(x)$=1 and $f^{PM}(x)$=0, then Eqn.(1) degenerates into a machine learning classifier ML; similarly for a pattern matcher PM. Therefore the solution to the above motivational problem lies in the configuration and optimization of the *weighting functions* such that the overall hotspot prediction performance exceeds each individual predictor. Built upon such a motivation, this paper proposes a systematic CAD flow to construct and optimize a *meta-classifier* integrating multiple types of powerful hotspot prediction techniques (known as disparate *base classifiers*). Our key contributions are as follows,

- We propose for the first time a unified *meta-classifier* to seamlessly combine the advantages of various hotspot detection techniques for enhanced accuracy and reduced false-alarms.
- We develop high performance hotspot detection engines as *base classifiers* to leverage state-of-the-art machine learning and pattern matching techniques.
- We employ Quadratic Programming techniques to achieve efficient configuration and performance optimization of the meta-classifier.
- We perform exhaustive assessment on the proposed method using various industry-strength benchmarks under advanced RET and manufacturing conditions.

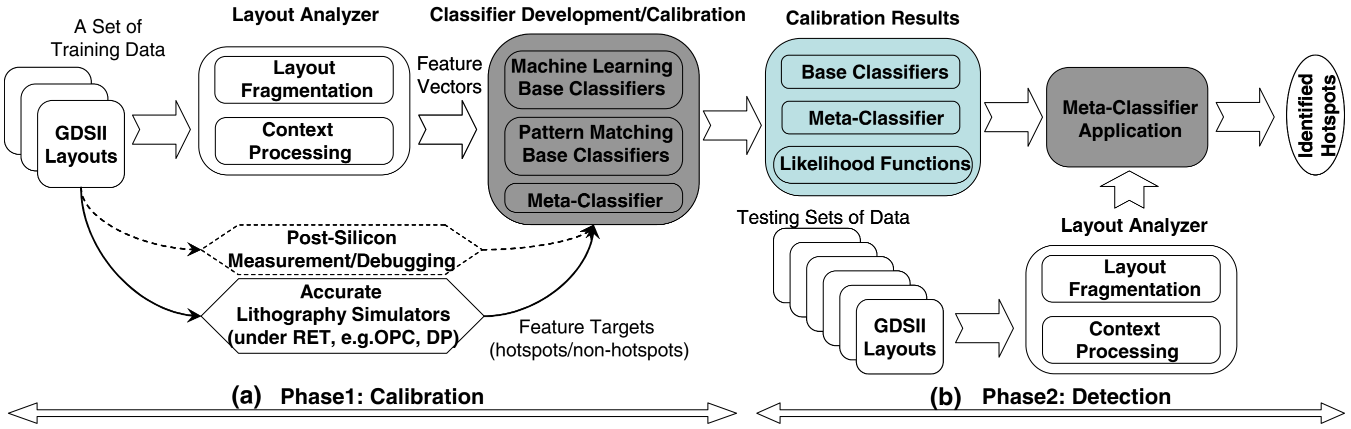


Figure 3: The overall CAD flow proposed for hotspot detection based on meta-classification formulation

3. META-CLASSIFICATION OVERVIEW

3.1 Overall Flow

Fig. 3 shows the overall flow for calibrating and applying the *meta-classifier*. It consists of 2 steps, the calibration and the detection phases. Before going into details, we introduce the following key components of the proposed flow.

DEFINITION 4. ***Base Classifier***: An individual hotspot classifier that is optimized under certain performance metric, such as detection accuracy, or false-alarms, or adaptivity to new unknown designs, etc.

DEFINITION 5. ***Weighting Function***: A function that properly weights and compensates the prediction result of a *base classifier* such that the overall combinations of individual *base classifiers* can be configured for better accuracy and smaller noise.

DEFINITION 6. ***Meta-Classifier***: A classifier that is formulated and optimized via proper combinations of multiple *base classifiers* under a set of *weighting functions* to further enhance hotspot prediction performance.

According to Fig. 3, Phase1 is the calibration stage where the *base classifiers* and the *weighting functions* are configured and optimized using training data sets. This stage requires the supervision of accurate lithography simulators or real silicon debugging data. Phase2 is the stage when the established *meta-classifier* is applied onto new testing data sets. This stage operates at very high speed without accurate lithography simulations.

3.2 Constructing the Meta-Classifier

The construction and optimization of the *meta-classifier* are the two key contributions of this paper. In this section we give an illustrative overview of the proposed methodology, leaving detailed analysis to Section 5.

The development of a *meta-classifier* is illustrated in Fig. 4, which is mainly divided into 3 levels. For every layout pattern geometry i, certain key hotspot features are extracted then fed into each *base classifier*. *Base classifiers* generate the prediction decisions (x_i's) of pattern i, then the weight of each classifier's decision is generated by the *weighting functions*. The final meta-decision is the weighed sum of *base classifiers*. Generalizing from the motivational example, we define the following:

$$T_i^{meta} = \digamma^{meta}\{\sum_{k=1}^{N} x_i^{(k)} \cdot f^{(k)}(x_i^{(k)})\} \qquad (3)$$

where T_i^{meta} is the final decision value of pattern i, N is the total number of *base classifiers*, $f^{(k)}(\cdot)$ is the *weighting function* of the kth *base classifier*, $x_i^{(k)}$ is the output from the kth *base classifier* when pattern i is the input. $\digamma^{meta}$ is the same as in Eqn.(2).

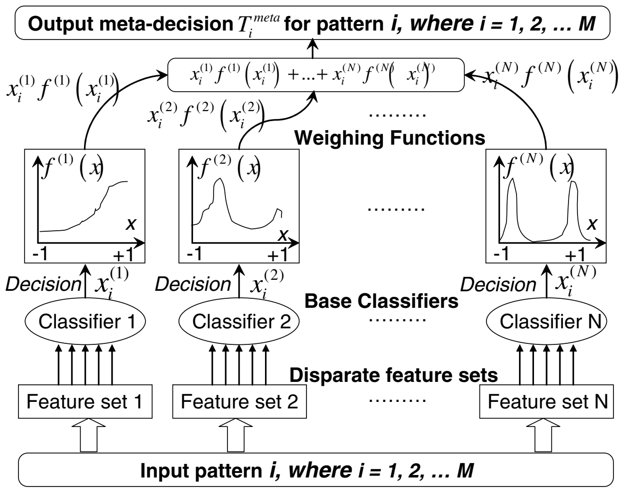

Output meta-decision T_i^{meta} for pattern i, where i = 1, 2, ... M
$x_i^{(1)} f^{(1)}(x_i^{(1)}) + ... + x_i^{(N)} f^{(N)}(x_i^{(N)})$
$x_i^{(1)} f^{(1)}(x_i^{(1)})$
$x_i^{(2)} f^{(2)}(x_i^{(2)})$
$x_i^{(N)} f^{(N)}(x_i^{(N)})$
Weighing Functions
$f^{(1)}(x)$
$f^{(2)}(x)$
$f^{(N)}(x)$
x
-1
+1
Decision $x_i^{(1)}$
Decision $x_i^{(2)}$
Decision $x_i^{(N)}$
Base Classifiers
Classifier 1
Classifier 2
Classifier N
Disparate feature sets
Feature set 1
Feature set 2
Feature set N
Input pattern i, where i = 1, 2, ... M

Figure 4: *Meta-classifier* construction via a combination of disparate *base classifiers*

By examining the corner cases, we notice that if the *weighting function* of SVM *base classifier* $f^{SVM}(x) \equiv 1$, and all other *base classifiers* have 0 value *weighting functions*, then the *meta-classifier* degenerates into a SVM *base classifier*. Therefore by adjusting the *weighting functions* we can achieve a performance trade-off between different types of hotspot detection techniques, such as machine learning and pattern matching. In the following sections, we will discuss the development and optimization of each classifier involved in the proposed flow.

4. CONSTRUCTING BASE CLASSIFIERS

In this section, we elaborate the *base classifiers* using machine learning techniques (Artificial Neural Network and Support Vector Machine) and pattern matching techniques.

4.1 Artificial Neural Network Classifiers

The ANN *base classifier* is built via fine tuning [10, 13]. We briefly describe the formulation and our specific feature contents as follows.

$$objective : minimize\{\sum_{p=1}^{N} E^p\} \quad w.r.t \quad \omega_{ij}, \omega_{jk} \tag{4}$$

$$E^p = \frac{1}{2}[out_p - y_p]^2 \tag{5}$$

$$out_p = f_{out}\{\sum_j \omega_{jk} \cdot f_{hid}(\sum_i V_p^i \cdot \omega_{ij})\} \tag{6}$$

$$\frac{\partial E^p}{\partial \omega_{jk}} = (out_p - y_p) \cdot f_{hid}\{\sum_i V_p^i \cdot \omega_{ij}\} \tag{7}$$

$$\frac{\partial E^p}{\partial \omega_{ij}} = (out_p - y_p) \cdot \omega_{jk} \cdot V_p^i \cdot (1 + out_{hid}^j)(1 - out_{hid}^j) \tag{8}$$

$$f_{hid} = \frac{2}{(1 + e^{-2x})} - 1, \qquad f_{in} = f_{out} = x \tag{9}$$

$$sign\_func(x) = \begin{cases} -1 & \text{x} < 0 \\ 0 & \text{x} = 0 \\ +1 & \text{x} > 0 \end{cases} \tag{10}$$

$$Est_{\tilde{p}} = F^{ann}\{f_{out}[\sum_j \omega_{jk} \cdot f_{hid}(\sum_i V_{\tilde{p}}^i \cdot \omega_{ij})]\} \tag{11}$$

An ANN classifier (predictor) works by calculating an outcome out_p for a data sample vector V_p based on established weights (ω_{jk}) and biases assigned to a neural network structure, such that the summed square error is minimized according to Eqn.(4) and that out_p approximates certain target y_p. The models shown here 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 nonhotspot 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, $i = 1$ 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. F^{ann} is the threshold adjustment for performance fine-tuning. Once the ANN *base classifier* is fully calibrated, we can apply it to estimate $Est_{\tilde{p}}$ according to Eqn.(11) without using costly lithography simulations.

4.2 Support Vector Machine Classifiers

Inside the meta-machine block, we employ a C-class Support Vector Machine (SVM) classifier fine-tuned based on [10, 13]. We brief the problem formulation as follows.

$$objective : minimize\{f(\alpha) = \frac{1}{2}\alpha^T Z\alpha - e^T\alpha\} \quad w.r.t \quad \alpha \tag{12}$$

$$subject \quad to : 0 \le \alpha_i \le C, \quad i = 1, ..., n \tag{13}$$

$$y^T \cdot \alpha = 0 \tag{14}$$

$$K(V_i, V_j) = exp\{\gamma \cdot \|V_i - V_j\|^2\} \tag{15}$$

$$slope\_func(x) = \begin{cases} 0 & x \le 0 \\ x & 0 < x < \text{C} \\ C & x \ge \text{C} \end{cases} \tag{16}$$

$$Est_{\tilde{p}} = F^{svm}\{\sum_i \alpha_i y_i K(V_{\tilde{p}}, V_i) + bias\} \tag{17}$$

Given V_i, i=1 to M sample vectors with n number of features, with label y_i (either *hotspot* or *non-hotspot* for 2-class SVM). e is a vector of all 1's. C is a pre-set upper bound to constrain feasible regions for hotspot detection under real manufacturing conditions. Z is n by n positive semi-definite matrix defined as $Z_{ij} = y_i y_j K(V_i, V_j)$, where $K(V_i, V_j)$ is defined in Eqn.(15) as the kernel function. α is the N element weight vector for V_p's. F^{svm} is a threshold function to adjust and fine-tune the estimation performance of $Est_{\tilde{p}}$.

The configuration of SVM *base classifiers* is achieved through performing a set of algorithms over the calibration data V_p's to identify the support vectors and weight coefficients that construct a classification hyper-plane with maximized separation margin. Once configured, we apply the SVM model to evaluate new data samples according to Eqn.(17) without costly lithography simulations.

4.3 Pattern Matching Classifiers

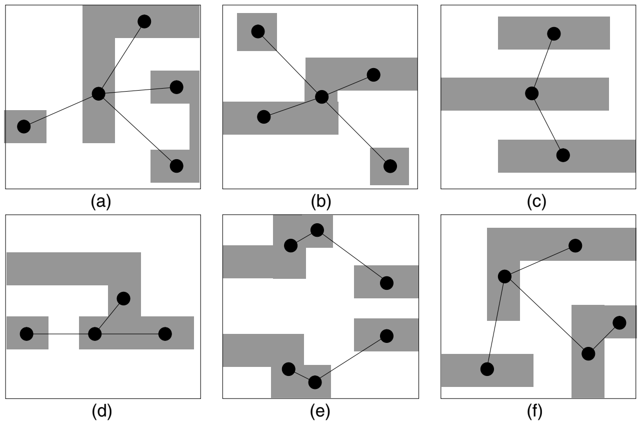


Figure 5: Example patterns in PM *base classifiers*

We explored the current state-of-the-art methods [14–16] and came up with several major classes (each with hundreds of specific hotspot structures) of pattern matching *base classifiers* to cover various types of lithography hotspots, relating to special line-ends, corners, jogs, contact patterns, etc.

Some example hotspot patterns are illustrated in Fig. 5. In particular, we have fine-tuned the pattern matchers to have broader pattern coverage rather than performing exact matching. As a result, the established pattern matchers demonstrate very good hotspot accuracies onto new data sets. Obviously, the penalty of such fine-tuning is the consequent high false-alarms. However, as we will see later in Section 6, the *meta-classifier* performs well in suppressing the false-alarms of such a PM *base classifier*.

5. OPTIMIZING META-CLASSIFICATION

Given the proposed *meta-classifier* in Fig. 4, in this section we first analyze the Mean-Square-Error of the *meta-classifier* introduced by the errors/noises of the *weighting functions*. Then we propose mathematical programming techniques to optimize the *weighting functions* to minimize the detection error.

5.1 Meta-Classification Error Analysis

Depicted in Fig. 6 are 2 sets of curves. Assume the black curves are the optimal *weighting functions* and the intersected point *threshold** is the optimal cutoff value for $F^{meta}(\cdot)$

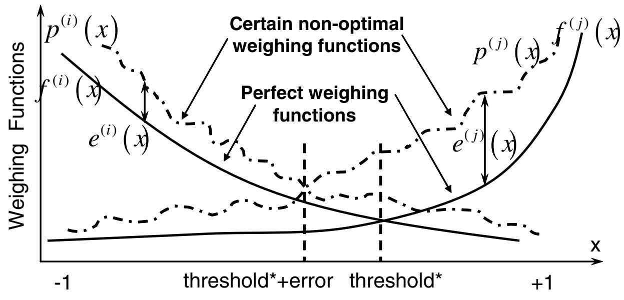


Figure 6: An illustration of *weighting function* error analysis, assuming two *base classifiers* (N=2)

(this happens under 1:1 importance ratio between *hotspot accuracy* improvement and *hotspot false-alarm* reduction). Suppose the dotted curves are the sub-optimal *weighting functions* for the ith and jth *base classifiers*. In this case, the derived cut-off threshold becomes *threshold*+error*, thus the meta-classification flow has an error:

$$MSE^{noise} = \int_x \{\sum_{k=1}^{N} f^{(k)}(x) \cdot x - \sum_{k=1}^{N} p^{(k)}(x) \cdot x\}^2 dx \quad (18)$$

$$= \sum_{k=1}^{N} \int_x \{[f^{(k)}(x) - p^{(k)}(x)] \cdot x\}^2 dx \quad (19)$$

$$= \sum_{k=1}^{N} \int_x \{[e^{(k)}(x)] \cdot x\}^2 dx \quad (20)$$

$$e^{(k)}(x) = f^{(k)}(x) - p^{(k)}(x) \quad (21)$$

From the analysis above, we observe that the classification error accumulates among all *base classifiers* with a quadratic index on each term, should noise/error occur in the *weighting functions*. Therefore, it is critical to find the optimal *weighting functions* to ensure the *meta-classifier*'s noise robustness. In the following section, we will explore the mathematical formulation that optimizes the *weighting functions* given certain calibration data.

5.2 Weighting Function Optimization

We first define the *meta-classification* Mean-Square-Error over the entire calibration data set under the supervision of accurate lithography simulations:

$$MSE^{meta} = \frac{1}{M} \cdot \sum_{i=1}^{M} \| \sum_{k=1}^{N} x_i^{(k)} \cdot p^{(k)}(x_i^{(k)}) - T_i^{litho} \|^2 \quad (22)$$

where M is the total number of calibration samples and T_i^{litho} is the baseline hotspot characterization result given by accurate lithography simulator. Table 1 details the shorthand terms used in our mathematical formulation.

To minimize the Mean-Square-Error among the sample space meanwhile avoid over-fitting of the training data set, we define the performance optimization formulation:

$$To\ minimize:\ MSE^{meta} + PCost \quad w.r.t\ p^{(k)}(x_i^{(k)}) \quad (23)$$

$$PCost = \lambda_0 \sum_i \sum_k (p^{(k)}(x_i^{(k)}) - const)^2 \quad (24)$$

where λ_0 is a non-negative penalty applied to constrain the calibration process such that the *weighting functions* are bounded within certain proximity of a constant parameter. This will prevent numerical instability and preserve detection generality of the *weighting functions* when applied to new testing data. Such proximity is adjustable by varying λ_0.

Table 1: Variables and terms in the QP formulation

Terms	Descriptions
N	Number of the *base classifiers*
M	Number of *meta-machine* calibration sample data
i	Index of each input sample pattern
k	Index of each *base classifier*
$x_i^{(k)}$	Prediction result from the *base classifier* k given input data sample i
$f^{(k)}(x_i^{(k)})$	Value of perfect *weighting func.* $f^{(k)}(\cdot)$ at $x_i^{(k)}$
$p^{(k)}(x_i^{(k)})$	Value of non-perfect *weighting func.* $p^{(k)}(\cdot)$ at $x_i^{(k)}$
$L^{(k)}$	Total quantization levels of *base classifier* k
l	Index of each quantization level
$p_k^{(l)}$	Quantized weight value from $f^{(k)}(\cdot)$ at level l of the *base classifier* k, $p_k^l \in [1,\ L^{(k)}]$
$\Theta(\cdot)$	The quantization mapping function: $x_i^{(k)} \rightarrow$ quantization level index l
$\alpha_k^{(l)}(i)$	Prediction result given by *base classifier* k at level l with input sample i (set to 0 if NULL) i.e., the value to which $p_k^{(l)}$ is to be applied
L^{total}	Total number of independent $p_k^{(l)}$
Q	A definite positive matrix $\in \Re^{L^{total} \times L^{total}}$
c	A vector $\in \Re^{L^{total} \times 1}$
X	Variable vector for the quadratic programming formulation, where $X = [p_1^{(1)} \ ...p_k^{(l)} \ ...p_N^{(L^{(N)})}]^T \in \Re^{L^{total} \times 1}$
T_i^{meta}	*Meta-machine* prediction result for input sample i
T_i^{litho}	Prediction baseline for input sample i by accurate lithography simulator
λ_0	Parameter to avoid over-fitting/instability

To assist numerical optimization, we quantized the original continuous *weighting functions* $p^{(1)}(x) \sim p^{(N)}(x)$, each into $L(k)$ levels, with each level being a single weight value denoted as $p_k^{(l)}$, where $l \in [1,L(k)]$.

After the *weighting function* quantization process, we have the following modified formulation:

$$To\ minimize:\ \overline{MSE} + \overline{PCost} \quad w.r.t\ p_k^{(l)} \quad (25)$$

$$\overline{MSE} = \frac{1}{M} \sum_{i=1}^{M} \| \sum_{k=1}^{N} p_k^{(\Theta(x_i^{(k)}))} \cdot x_i^{(k)} - T_i^{litho} \|^2 \quad (26)$$

$$\overline{PCost} = \lambda_0 \sum_{k=1}^{N} \sum_{l=1}^{L^{(k)}} (p_k^{(l)} - 1)^2 \quad (27)$$

where $p_k^{(l)}$'s are the optimization variables (quantized weight values) for fine-tuning the overall classification quality. In $\overline{PCost}$, we set the constant parameter to 1.0 since it is the boundary factor of numerical up-scaling and down-scaling. Due to the $\overline{PCost}$ term, each weight variable will be scattered not far away from 1.0 meanwhile be optimized under predication error minimization objective. This benefits us in two ways: first, avoiding close to zero weights for the calibration data yields better classification generality over testing data; second, avoiding large weights can maintain good balance among hotspot features meanwhile prevent numeric instability over testing data. For further details of the notations please refer to Table 1.

Finally we can write the following quadratic programming problem formulation:

$$f(x) = \frac{1}{2} X^T Q X + c^T X \quad (28)$$

$$X \geq lb \quad (29)$$

$$lb = [0\ 0\ 0\ 0\ 0\ \ldots\ 0]^T \in \Re^{L^{total}\times 1} \tag{30}$$

where X is the optimization variable vector defined as follows,

$$X = [p_1^{(1)}\ \ldots\ p_1^{(L^{(1)})}\ \ldots\ p_k^{(l)}\ \ldots\ p_N^{(L^{(N)})}]^T \in \Re^{L^{total}\times 1} \tag{31}$$

where L^{total} is the total number of $p_k^{(l)}$'s:

$$L^{total} = \sum_{k=1}^{N} L^{(k)} \tag{32}$$

Matrix Q is defined as follows,

$$Q \;= \begin{pmatrix} \beta_1^{(1)}(i) & \gamma_{1,1}^{(1,2)}(i) & . & \gamma_{1,k}^{(1,l)}(i) & \gamma_{1,N}^{(1,L^{(N)})}(i) \\ \gamma_{1,1}^{(2,1)}(i) & \ddots & . & . & . \\ . & . & \beta_1^{(L^{(1)})}(i) & . & . \\ \gamma_{k,1}^{(l,1)}(i) & . & . & \beta_k^{(l)}(i) & . \\ \gamma_{N,1}^{(L^{(N)},1)}(i) & . & . & . & \beta_N^{(L^{(N)})}(i) \end{pmatrix} \tag{33}$$

Vector c is defined as the linear term coefficients vector from the quadratic formulation objective:

$$c = [\omega_1^1(i)\cdots\omega_1^{L^{(1)}}(i)\cdots\omega_k^{(l)}(i)\cdots\omega_N^{L^{(N)}}(i)\ ]^T \tag{34}$$

where the related terms are defined as follows, and $\alpha_k^{(l)}(i)$ is an intermediate term to link the *base classifier*'s prediction values with $p_k^{(l)}$, i.e., $\alpha_k^{(l)}(i)$ is the $x_i^{(k)}$ value that falls into level l of the quantized weighting function relating to the *base classifier k*. Given certain i and k, if there is no output values corresponding to level l, then $\alpha_k^{(l)}(i)$ is set to 0.

$$\beta_k^{(l)}(i) = \frac{2}{M}\sum_{i=1}^{M}[\alpha_k^{(l)}(i)]^2 + 2\lambda_0 \tag{35}$$

$$\gamma_{m,k}^{(n,l)}(i) = \frac{2}{M}\sum_{i=1}^{M}\alpha_m^{(n)}(i)\cdot\alpha_k^{(l)}(i) \tag{36}$$

$$\omega_k^{(l)}(i) = -\frac{2}{M}\sum_{i=1}^{M}T_i^{litho}\cdot\alpha_k^{(l)}(i) - 2\lambda_0 \tag{37}$$

The values and parameters in the above equations are derived properly so that the original problem in Eqn.(23) becomes the minimization of a quadratic problem in Eqn.(28). Once it is solved, we apply the resulting *weighting functions* over some calibration data to properly select a threshold function $\digamma^{meta}(\cdot)$ to further balance *hotspot accuracy* and *hotspot false-alarm*. After calibration, the *meta-classifier* will be tested over new design layouts based on Eqn.(3).

5.3 Complexity Analysis

Theorem 1: *Matrix Q is positive definite under certain conditions of λ_0 and the formulated quadratic programming problem has the following properties: (1) it can be solved in polynomial time complexity; (2) if it has a local minimum, then this local minimum is also the global minimum.*

Proof For notation simplicity, we assume a vector $\vec{X} \in \Re^{L^{total}\times 1}$ and $X \geq \vec{0}$. Let ρ_i be the coefficient of χ_i, where χ_i is the ith element of $\vec{X}$. Let L^{total} be the total number of quantization levels among all *base classifiers*. Therefore we have the following:

$$\vec{X}^T Q \vec{X} = \frac{1}{M}\sum^{M}\sum_{j=i+1}^{L^{total}}\sum_{i=1}^{L^{total}}(\rho_i\chi_i + \rho_j\chi_j)^2 + \Delta \tag{38}$$

Algorithm 1 ***Meta-Classifier*-Calibration**

```
Require: data sample vectors and over-fit penalty $\lambda_0$
  Initialize $Q$, $c$, $\beta_k^{(l)}(i)$, $\gamma_k^{(l)}(i)$, $\omega_k^{(l)}(i)$
  Build Hierarchical MLK-ANN
  Build Hierarchical MLK-SVM
  Build Pattern Matchers
  for All input data samples do
    Generate the base classifiers
    Update $Q$, $c$, $\beta_k^{(l)}(i)$, $\gamma_k^{(l)}(i)$, $\omega_k^{(l)}(i)$
  end for
  Formulate Quadratic Programming Problem
  if $Q$ not positive definite then
    Increase calibration data volume
    Improve hotspot feature quality
    Adjust parameter $\lambda_0$
    Consider matrix pre-conditioning
  end if
  Solve the Quadratic Programming Problem
  Optimize the detection threshold function $\digamma^{meta}(\cdot)$
  return  weighting functions $p_k^{(l)}$ and $\digamma^{meta}(\cdot)$
```

Algorithm 2 ***Meta-Classier*-Prediction**

```
Require: data sample vectors
  Load weighting functions $p_k^{(l)}$ and $\digamma^{meta}(\cdot)$
  Load all base classifiers
  Generate vector $\vec{x_i}$ from base classifiers outputs
  for Each data vector $\vec{x_i}$ do
    Calculate $T_i^{meta}$=$\digamma^{meta}(\sum_{k=1}^{N} p_k^{(\Theta(x_i^{(k)}))}\cdot x_i^{(k)})$
  end for
  return  Meta-decision $\{T_i^{meta}\}$
```

where

$$\Delta = -\frac{L^{total}-2}{M}\sum^{M}\sum_{i}^{L^{total}}\rho_i^2\chi_i^2 + \lambda_0\sum_{i}^{L^{total}}\chi_i^2 \tag{39}$$

We can adjust λ_0 to achieve positive Δ (in practice usually a λ_0 slightly larger than $\frac{L^{total}-2}{M}\sum^{M}$). Therefore $\vec{X}^T Q\vec{X}$ is always positive given non-zero $\vec{X}$. Thus Q is a positive definite matrix and has no negative eigenvalues under the specified condition. Numerical simulations further validate this proof by showing all positive eigenvalues for matrix Q. Consequently, the Quadratic Programming problem can be solved by the *ellipsoid method* [20] in polynomial time.

Since Q is a symmetric positive-definite matrix, $f(\cdot)$ is now a convex function. Thus the quadratic program has a global minimizer if there exists some feasible vector X^n satisfying the constraints and if $f(\cdot)$ is bounded below on the feasible region ($X^n \in \Re_+^n$). Therefore in search of a local minimum, if found, will guarantee the optimal global minimum.

We solve the quadratic programming problem using a proper λ_0 to optimize the *weighting functions* during the calibration phase. Then we use a heuristic approach to search for the optimal $\digamma^{meta}(\cdot)$ function. In Algorithm 1 and Algorithm 2, we show the details for the calibration and application of the *meta-classifier*.

6. SIMULATION AND TESTING

6.1 Benchmarks and Simulation Setups

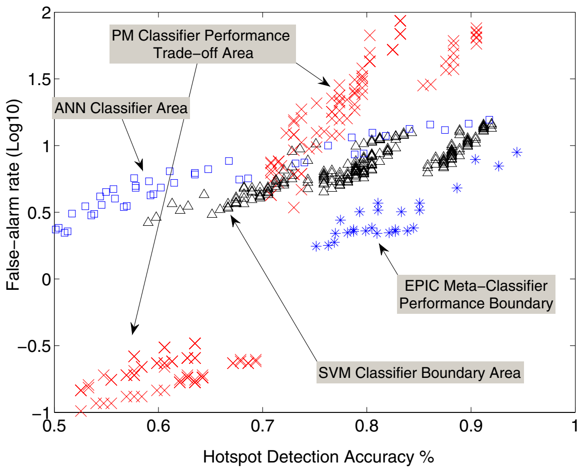


Figure 7: Trade-off capabilities between hotspot accuracy and false-alarms using various methods on C0 hotspots

To fully evaluate *EPIC*, we employed a number of training data sets and 3 new testing circuit benchmarks in 32nm. These testing circuits include new hotspot patterns that were not present in the training data. We labeled 2 classes of 'real' lithographic hotspots based on 2 EPE thresholds. In Table 2, C0 is the class0 hotspot patterns whose printed images suffer from above 6nm of EPE; C1 refers to the patterns whose printed images have EPE from 4.5nm to 6nm. Further details of the 3 testing benchmarks are in Table 2. To properly evaluate the proposed methods, we perform accurate lithographic simulations as baseline to identify the actual hotspots under industry-strength RETs.

In our simulation, *EPIC* incorporates two types of machine learning methods based on [10,13] and several pattern matching techniques based on [14–17]. We implement *EPIC* in C++ on 3.2GHz quad-core Linux workstations.

6.2 Result Analysis and Comparison

After the quadratic programming problem is solved, we properly select the decision threshold function $F^{meta}(\cdot)$ using some calibration data to balance between *hotspot accuracy* and *hotspot false-alarm*. To illustrate such performance trade-off, we test *EPIC*, *ANN* and *SVM* over C0 data with a set of varying thresholds and plot the results in Fig. 7. We also plot the performance region of the employed pattern matchers, which include up to 4 major classes of hotspot patterns. As we enrich the pattern library gradually with up to more than hundreds of specific patterns and structures, the overall performance becomes a trade-off between enhancing detection accuracy and suppressing false-alarms, especially when there are new unseen types of hotspots in the testing data.

From Fig. 7 we observe that in the region of above 70% accuracy, *EPIC* shows higher *hotspot accuracy* than other methods with very similar *hotspot false-alarm*, it also achieves lower *hotspot false-alarm* given similar *hotspot accuracy*. We also see that pattern matching methods are not good at detecting new types of hotspots without obvious penalty in *hotspot false-alarm*. In this sense, machine learning can make pattern matching more robust to predict new/unknown hotspots, especially when pattern enumeration becomes costly.

Based on Fig. 7, we calculate the following for each method:

$$\Psi = \alpha \cdot Accuracy^{hotspot} + \beta \cdot False\_alarm^{hotspot} \qquad (40)$$

where α (positive) and β (negative) are user defined parameters to quantify the importance ratio between *hotspot accuracy* and *hotspot false-alarm*. In Table 3 and Table 4, we report the detection result of each method corresponding to the peak of their respective Ψ function. We observe that *EPIC* reaches the highest performance over both C0 and C1 categories of hotspots in both *hotspot accuracy* and *hotspot false-alarm*. To be specific, it improves ANN and SVM by 3.5-7% in *hotspot accuracy* and reduces up to 50% in *hotspot false-alarm* counts. *EPIC* also outperforms PM by 4.5-8.2% in *hotspot accuracy* and 53-81% in *hotspot false-alarm* reduction. This demonstrates very promising potential of the *meta-classification* flow with respect to *weighting function* optimizations. Moreover, *EPIC* runs at the speed of around 45 min per mm^2 design on a 3.2GHz quad-core workstation, which is typically hundreds of times faster than accurate lithography simulator.

Table 2: Circuit benchmarks for testing *EPIC*

Benchmarks	CK1	CK2	CK3
Layout Size um^2	100×100	150×150	800×800
Fragment number	58K	94.5K	2.5M
Class0[a] Hotspots number	9	21	122
Class1[b] Hotspots number	61	134	2.8K

[a] Class0 hotspots: EPE $\geq 6nm$ for $32nm$ process.
[b] Class1 hotspots: $4.5nm \leq$ EPE $< 6nm$ for $32nm$ process.

Table 3: Performance of hotspot detection methods

Circuits	Class	Perf.	ANN	SVM	PM	*EPIC*
CK1	C0	Hit	6	7	7	9
		Extra	79	41	280	48
	C1	Hit	52	54	53	57
		Extra	0.55K	0.33K	1.5K	0.3K
CK2	C0	Hit	18	17	17	19
		Extra	0.2K	0.11K	0.7K	0.1K
	C1	Hit	119	120	120	125
		Extra	1.2K	0.75K	3.4K	0.65K
CK3	C0	Hit	109	105	104	112
		Extra	1.2K	0.6K	3.9K	0.65K
	C1	Hit	2.45K	2.5K	2.5K	2.63K
		Extra	24K	16K	73K	13.5K

In Fig. 8 we give two summary plots on the performance comparisons of various hotspot prediction methods according to (a): *hotspot accuracy*(*Hit rate*)/run-time, and (b): *hotspot false-alarm*(*Extra*)/run-time, respectively. Here we make some further observations.

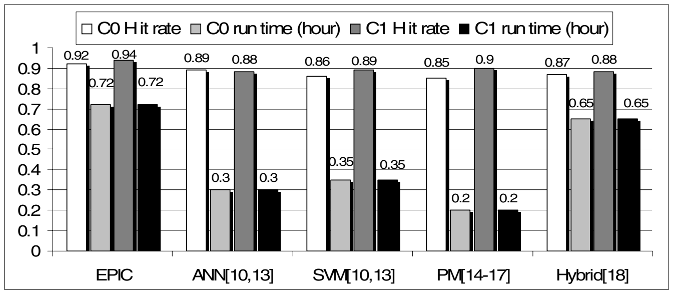
First, *EPIC* achieves much enhanced performance in hotspot prediction accuracy and false-alarms reduction, meanwhile the extra CPU run-time is only 20-30 minutes per mm^2 layout in the worst case (versus pattern matching methods).

Second, in comparing C0 category of hotspots with C1 category, we see *EPIC* achieves higher *Hit rate* and lower *False-Alarm ratio* (in the unit of X times of real hotspots) over C1 than C0. In other words, *EPIC* gives more enhancement in accuracy and false-alarm on C1. This is mostly because C1 class represents the set of lithography hotspot with

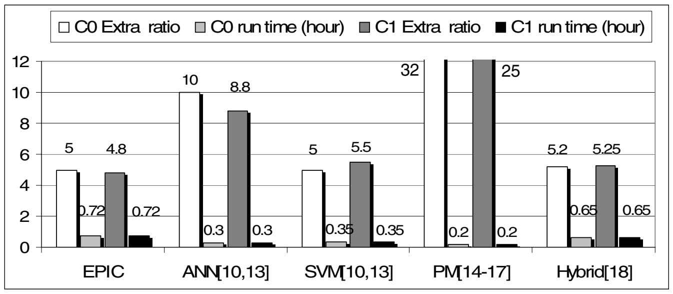
Table 4: Comparison between *EPIC* and previous works

Hotspot	C0			C1		
Avg. Perf.	Hit	Extra	Time	Hit	Extra	Time
EPIC	92%	5X	0.72	94%	4.8X	0.72
ANN [10,13]	89%	10X	0.3	88%	8.8X	0.3
SVM [10,13]	86%	5X	0.35	89%	5.5X	0.35
PM [14–17]	85%	32X	0.2	90%	25X	0.2

Time calibrated in $hour/mm^2$ unit on 3.2GHz quad-core Linux workstation.



(a)



(b)

Figure 8: Overall performance comparison in *Hit rate*, *Extra ratio* and run-time over C0 and C1 hotspot data

4.5nm to 6.0nm of EPE, while C0 is the set of hotspots with above 6.0nm EPE values. Under our employed RETs, C0 translates to a set of hotspots that have high variability and small quantity (a few hundred out of a layout with totally millions of patterns); whereas C1 is a set of hotspots with less severe variability and much larger quantity. This is why sometimes detecting C0 could be slightly harder than C1. However it also depends on how the trade-off solution is selected using Ψ based on Fig. 7 to balance between *hotspot accuracy* and *hotspot false-alarm*, e.g., ANN shows a slightly higher (by 1%) accuracy rate in C0 than C1 category.

An important advantage of *EPIC* is that it is capable of high performance hotspot prediction under varying EPE thresholds and given large scale design layouts. We see that *EPIC* exhibits very similar CPU run-time when targeting at different categories of hotspot under different EPE thresholds. We have also observed linear run-time complexity when design layout up-scales in area. These properties make our flow efficient for optimizing large industry designs.

Moreover, *EPIC*'s unified formulation covers the static hybrid detection flow proposed in [18] as a special case, i.e., when *weighting function* $f^{MLK1}(+1) = 0.5$ and 0 elsewhere, $f^{MLK2}(+1) = 0.5$ and 0 elsewhere, $f^{PM}(+1) = 1.0$ and 0 elsewhere, $\theta = 1.0$, then *EPIC*'s formulation Eqn.(3) will be equivalent to the hybrid flow in [18]. Here *EPIC*'s advantage lies in the dynamic/automated optimization techniques, thus it can easily reach an optimized solution. Comparing with the static hybrid flow over the employed test cases, *EPIC* observes around 5.7-6.8% of improvement in *hotspot accuracy* and 3.9-8.6% of *false-alarm* reduction at a small cost of 10% extra run-time. Depending on designs, *EPIC*'s advantages could be even higher.

7. CONCLUSION

In this paper we examined several different types of lithography hotspot detection techniques and proposed EPIC, a new formulation to selectively combine their respective advantages for further accuracy improvement. Under a meta-detection flow, we first used mathematical programming techniques for systematic performance optimization over a set of calibration data, then applied the flow onto new testing cases for further evaluation. EPIC's accuracy, flexibility and false-alarm suppression capability show very promising potential for efficient litho-friendly design.

8. REFERENCES

[1] International Technology Roadmap for Semiconductors. 2011.

[2] Joydeep Mitra, Peng Yu, and David Z. Pan. RADAR: RET-Aware Detailed Routing using Fast Lithography Simulation. In *Proc. Design Automation Conf.*, June 2005.

[3] Minsik Cho, Kun Yuan, Yongchan Ban, and David Z. Pan. ELIAD: Efficient Lithography Aware Detailed Router with Compact Printability Prediction. In *Proc. Design Automation Conf.*, June 2008.

[4] Tai-Chen Chen, Guang-Wan Liao, and Yao-Wen Chang. Predictive Formulae for OPC with Applications to Lithography-Friendly Routing. In *Proc. Design Automation Conf.*, June 2008.

[5] David Z. Pan, Minsik Cho, and Kun Yuan. Manufacturability Aware Routing in Nanometer VLSI. In *Foundations and Trends in Electronic Design Automation*, 2010.

[6] Duo Ding, Jhih-Rong Gao, Kun Yuan, and David Z. Pan. AENEID: A Generic Lithography-Friendly Detailed Router Based on Post-RET Data Learning and Hotspot Detection. In *Proc. Design Automation Conf.*, 2011.

[7] Norimasa Nagase, Kouichi Suzuki, Kazuhiko Takahashi, Masahiko Minemura, Satoshi Yamauchi, and Tomoyuki Okada. Study of Hotspot Detection using Neural Network Judgement. In *Proc. of SPIE*, volume 6607, 07.

[8] Duo Ding, Xiang Wu, Joydeep Ghosh, and David Z. Pan. Machine Learning based Lithographic Hotspot Detection with Critical Feature Extraction and Classification. In *IEEE Int. Conf. on IC Design Technology*, Austin, TX, 2009.

[9] Dragoljub Gagi Drmanac, Frank Liu, and Li-C. Wang. Predicting Variability in Nanoscale Lithography Processes. In *Proc. Design Automation Conf.*, San Francisco, CA, 2009.

[10] Duo Ding, J. Andres Torres, Fidor G. Pikus, and David Z. Pan. High Performance Lithographic Hotspot Detection Using Hierarchically Refined Machine Learning. In *Proc. Asia and South Pacific Design Automation Conf.*, 2011.

[11] Jen-Yi Wuu, Fedor G. Pikus, J. Andres Torres, and Malgorzata Marek-Sadowska. Detecting Context Sensitive Hot Spots in Standard Cell Libraries. In *Proc. of SPIE*, 2009.

[12] Jen-Yi Wuu, Fedor G. Pikus, and Malgorzata Marek-Sadowska. Rapid Layout Pattern Classification. In *Proc. Asia and South Pacific Design Automation Conf.*, 2011.

[13] Duo Ding, J. Andres Torres, and David Z. Pan. High Performance Lithography Hotspot Detection with Successively Refined Pattern Identifications and Machine Learning. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2011.

[14] Jingyu Xu, Subarna Sinha, and Charles C. Chiang. Accurate Detection for Process Hotspots with Vias and Incomplete Specification. In *Proc. Int. Conf. on Computer Aided Design*, 2007.

[15] Andrew B. Kahng, Chul-Hong Park, and Xu Xu. Fast Dual Graph based Hotspot Detection. In *Proc. of SPIE*, volume 6349, 2006.

[16] Hailong Yao, S. Sinha, C. Chiang, X. Hong, and Y. Cai. Efficient Process Hotspot Detection using Range Pattern Matching. In *Proc. Int. Conf. on Computer Aided Design*, 2006.

[17] Ning Ma, Justin Ghan, Sandipan Mishra, Costas Spanos, Kameshwar Poolla, Norma Rodriguez, and Luigi Capodieci. Automatic Hotspot Classification using Pattern-based Clustering. In *Proc. of SPIE*, 2007.

[18] Jen-Yi Wuu, Fedor G. Pikus, and Malgorzata Marek-Sadowska. Efficient Approach to Early Detection of Lithographic Hotspots Using Machine Learning Systems and Pattern Matching. In *Proc. of SPIE*, 2011.

[19] Kagan Tumer and Joydeep Ghosh. Robust Combining of Disparate Classifiers through Order Statistics. In *Pattern Analysis & Applications, pp. 189-200*, 2002.

[20] M. K. Kozlov, S. P. Tarasov, and Leonid G. Khachiyan. Polynomial Solvability of Convex Quadratic Programming. In *Soviet Mathematics - Doklady 20, pp. 1108-1111*, 1979.