Efficient Layout Hotspot Detection via Neural Architecture Search

YIYANG JIANG and FAN YANG, State Key Lab of AISC & System, School of Microelectronics, Fudan University
BEI YU, Chinese University of Hong Kong
DIAN ZHOU, University of Texas at Dallas
XUAN ZENG, State Key Lab of AISC & System, School of Microelectronics, Fudan University

Layout hotspot detection is of great importance in the physical verification flow. Deep neural network models have been applied to hotspot detection and achieved great success. Despite their success, high-performance neural networks are still quite difficult to design. In this article, we propose a bayesian optimization-based neural architecture search scheme to automatically do this time-consuming and fiddly job. Experimental results on ICCAD 2012 and ICCAD 2019 Contest benchmarks show that the architectures designed by our proposed scheme achieve higher performance on hotspot detection task compared with state-of-the-art manually designed neural networks.

CCS Concepts: • **Hardware → Electronic design automation**

Additional Key Words and Phrases: Hotspot detection, neural architecture search

ACM Reference format:
Yiyang Jiang, Fan Yang, Bei Yu, Dian Zhou, and Xuan Zeng. 2022. Efficient Layout Hotspot Detection via Neural Architecture Search. *ACM Trans. Des. Autom. Electron. Syst.* 27, 6, Article 62 (June 2022), 16 pages.
https://doi.org/10.1145/3517130

1 INTRODUCTION

The lithographic printability is of great significance in integrated circuits as the shrinking transistor feature size significantly challenges the manufacturing yield.

Due to the light diffraction, the **critical dimension (CD)** (the minimal transistor feature size) is defined in Equation (1)

$$CD = k\frac{\lambda}{NA}, \tag{1}$$

This research is supported partly by National Key R&D Program of China 2020YFA0711900, 2020YFA0711903, partly by the National Natural Science Foundation of China (NSFC) Research Projects under Grant 61822402, Grant 62141407, Grant 62090025 and Grant 61929102. partly by the Research Grants Council of Hong Kong SAR (Project No. CUHK14208021).
Authors' addresses: Y. Jiang, F. Yang (corresponding author), and X. Zeng (corresponding author), State Key Lab of AISC & System, School of Microelectronics, Fudan University, China; emails: {yangfan, xzeng}@fudan.edu.cn; B. Yu, Chinese University of Hong Kong, China; D. Zhou, University of Texas at Dallas.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
1084-4309/2022/06-ART62 $15.00
https://doi.org/10.1145/3517130

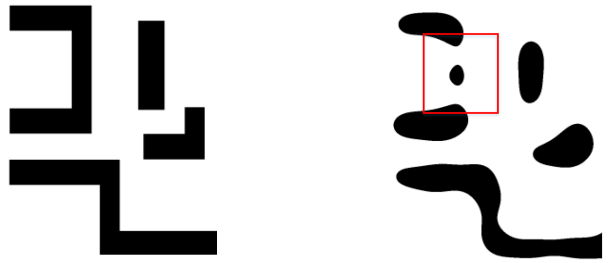

Fig. 1. A hotspot example. The pattern marked by the red box can cause open circuit after the lithography process.

where λ is the lithography system wavelength and NA is the numerical aperture of the lithography system. As the technology node shrinks down, there is a large gap between the layout feature size and the wavelength.

Layout patterns sensitive to process variations are called hotspots. Figure 1 shows an example of the hotspot in the layout. The pattern marked by the red box can cause open circuit after the lithography process. The hotspots should be detected in early stage to improve the lithographic printability.

There are two main classes of hotspot detection approaches: pattern matching-based [8, 11, 13, 29, 30] and machine learning-based [4–7, 9, 10, 12, 18, 19, 32] approaches. In [13], the hotspots are encoded by strings. Reference [11] characterizes the hotspots as modified transitive closure graphs. In [29], hotspots are encoded by the density and the encoding is further improved by **principle components analysis** (**PCA**). Reference [30] proposes a new distance metric based on tangent space to encode the hotspot patterns. The weak point of pattern matching-based approaches is that they cannot detect the hotspots, which are never seen before.

To solve this problem machine learning-based approaches have been developed. The never seen hotspots can be detected via the generalization capability of machine learning approaches. Although they improve the hotspot detection accuracy compared with the pattern matching-based approaches, they may cause false alarms which can decrease the efficiency [9, 10]. Reference [5] proposes to extract critical features and train a multi-kernel SVM on the extracted data for hotspot detection. Reference [7] extracts features of the layouts based on the histogram and adopts an unsupervised SVM to predict hotspots. Reference [4] proposes to detect hotspots with both ANN and SVM models. Reference [18] improves feature extraction with feature space index and detects hotspots with Adaboost algorithm. In [32], hotspot detection performance is further improved with a new online learning scheme.

Deep learning-based approaches have also achieved great success in hotspot detection problem. Reference [31] proposes to extract layout features with discrete cosine transform and utilize a CNN architecture for hotspot detection. The scheme is further adjusted with the proposed bias learning algorithm because of the imbalanced dataset. Reference [1] proposes to detect multiple hotspots at a time inspired by the object detection problem. In [14], binarized neural network is utilized to speed up the hotspot detection flow. New network architecture is designed based on residual networks to achieve higher detection accuracy and performance. Reference [2] proposes a semi-supervised scheme to utilize samples with/without labels for higher hotspot detection performance.

Despite the success, designing and tuning these neural network architectures still require a great deal of time. People continuously try slight variations of the known architectures, which have

Table 1. Confusion Matrix of Hotspot Detection Problems

Prediction	Ground Truth	
	Non-hotspot	Hotspot
Non-Hotspot	*TN*	*FN*
Hotspot	*FP*	*TP*

proven to be practical. Recently, **neural architecture search** (**NAS**) techniques have achieved great success. The automatically searched neural networks outperform the handcrafted architectures in both efficiency and performance.

In this article, we propose to automatically design a deep learning-based hotspot detector with a bayesian optimization-based neural architecture search scheme. A novel mapping method is proposed to efficiently map the discrete architecture search space to the continuous feature space in order to characterize the distance between different sampled neural architectures, which is important for the Gaussian process model in the Bayesian optimization process. Due to the tremendous time consumption during training and evaluating of the sampled architectures, an accurate prediction model is adopted to help accelerate the searching process as well. Experimental results show that our automatically designed architecture outperforms the state-of-the-art handcrafted architecture in both detecting accuracy and speed.

Lithographic simulation lacks reusability and needs a complete run every time it meets a new sample. This is not efficient. Recent deep learning model-based methods are more efficient because the trained model is reused when dealing with new samples and the inference time of the machine learning model is much less than the lithographic simulation. However, these methods often need much time to design and tune the neural architectures manually. Our algorithm replaces this time-consuming process with a fast auto-searching process, which improves the efficiency. Because architecture searching is a one-off job, once a well-performing architecture is searched it can be used to process numerous similar layout patterns efficiently instead of slow lithographic simulation. Last but not least our searched neural architecture is 42% faster than the previous manually designed architectures. As a result our proposed neural architecture search scheme further improves the efficiency of the layout hotspot detection flow.

The rest of this article is organized as follows. In Section 2, the background of hotspot detection and neural architecture search is presented. In Section 3, we propose the neural architecture search-based hotspot detection approach. In Section 4, we give the experimental results of our proposed approach. In Section 5, we conclude the article.

2 BACKGROUND

In this section, we will define the layout hotspot detection problem and then review the background of neural architecture search.

2.1 Problem Formulation

Process variations in the lithographic process can cause yield reduction. Layout patterns that are sensitive to process variations are defined as *hotspots*. Referring to the practice in previous works, two metrics are adopted to evaluate the performance of our proposed hotspot detection algorithm.

Hotspot detection problem is a binary classification problem whose confusion matrix is presented in Table 1. *TN*, *FN*, *FP*, and *TP* denote True Negative, False Negative, False Positive, and True Positive, respectively. We then define the following metrics with the confusion matrix.

Definition 1 (Accuracy). The ratio of correctly predicted hotspots among the set of actual hotspots [28].

$$\text{Accuracy} = \frac{TP}{TP + FN}. \tag{2}$$

Definition 2 (False Alarm). The ratio of incorrectly predicted non-hotspots among the set of actual non-hotspots [28].

$$\text{False Alarm} = \frac{FP}{FP + TN}. \tag{3}$$

Finally, we present the formulation of the hotspot detection problem.

Problem 1 (Hotspot Detection). *Given a dataset that contains hotspot and non-hotspot instances, train a classifier that can maximize the accuracy and minimize the false alarm.*

2.2 Related Hotspot Detection Approach

In this section, we review the previous hotspot detection approach [31] where two new techniques including biased learning approach and feature extraction algorithm based on **discrete cosine transform** (**DCT**) are proposed.

The biased learning approach is proposed to deal with the imbalanced dataset. The neural network is trained with ground truth $y_{non-hotspot} = [1, 0]$ and $y_{hotspot} = [0, 1]$. After training, the neural network is fine-tuned with $y'_{non-hotspot} = [1 - \epsilon, \epsilon], \epsilon \in [0, 0.5)$, which can help the neural network achieve a better balance between detecting accuracy and false alarm.

The DCT-based feature extraction algorithm transforms the layout to a combination of different frequency components. Firstly, the layout is divided into several sub-regions. Secondly, each sub-region is transformed into frequency domain with DCT. Finally, each frequency vector is flattened, and the first k elements are picked as the extracted feature.

2.3 Bayesian Optimization

In this section, we review the Bayesian Optimization. **Bayesian Optimization** [25] (**BO**) is proposed to optimize black-box functions with high evaluation cost. It consists of probabilistic surrogate models and the acquisition function. The probabilistic surrogate models predict the functions with uncertainty measures and are updated with the sequentially added sampled points. To determine the next point to sample, we use the acquisition function to measure the utility by evaluating f based on the surrogate model predictions.

The most frequently used surrogate model in BO is **Gaussian process** (**GP**) regression model [22]. The GP model can predict both expectation and uncertainty of the objective function. The GP model consists of two functions: mean function $m(x)$ and covariance function $k(x, y)$. Constant function defined in Equation (4) is adopted as the mean function.

$$m(x) = \mu_0. \tag{4}$$

Squared exponential covariance function defined in Equation (5) is adopted as the covariance function in this article.

$$k(x, y) = \sigma_f^2 \exp\left(-\frac{1}{2}(x - y)^T \Lambda^{-1}(x - y)\right), \tag{5}$$

where σ_f and Λ are hyperparameters, $\Lambda = diag(l_1, \ldots, l_n)$ is a diagonal matrix.

The GP model can model the objective function with noise term $y = f(x) + \epsilon_t$. The noise term obeys normal distribution

$$\epsilon_t \sim N(0, \sigma_n^2), \tag{6}$$

where σ_n denotes the variance.

Given the training set $\{X, y\}$ where $X = \{x_1, \ldots, x_N\}$ and $y = \{y_1, \ldots, y_N\}$, the GP model predicts the objective function $f(x)$ as a normal distribution

$$f(x) \sim N(\mu(x), \sigma^2(x)), \tag{7}$$

where $\mu(x)$ is the mean function and $\sigma^2(x)$ is the covariance function. They can be computed via

$$\begin{aligned} \mu(x) &= \mu_0 + k(x, X)[K + \sigma_n^2 I]^{-1}(y - \mu_0), \\ \sigma^2(x) &= k(x, x) - k(x, X)[K + \sigma_n^2 I]^{-1} k(X, x), \end{aligned} \tag{8}$$

where $k(x, X) = [k(x, x_1), \ldots, k(x, x_N)]^T$ and $k(X, x) = k(x, X)^T$. $\mu(x)$ is the predicted value by the GP model while the $\sigma^2(x)$ denotes the uncertainty.

To minimize the objective function as shown in Equation (9), we describe the BO flow as follows.

$$x_* = \arg\min_x f(x). \tag{9}$$

We use the acquisition function to determine the next point to sample. The acquisition function combines the predicted expectation and the uncertainty to balance the exploration and the exploitation. Some commonly used acquisition functions including the **lower confidence bound** (**LCB**) function, the **probability of improvement** (**PI**) function and the **expected improvement** (**EI**) function are defined in Equation (10). To choose the appropriate acquisition function, a preliminary experiment is performed on a smaller dataset sampled from the original dataset. The experimental results show that the three acquisition functions can get structures with comparable accuracy and FA ratio, but the LCB function needs 13% fewer iterations than the other two acquisition functions to achieve the optimal structure. As a result, we adopt the LCB function as the acquisition function in this article.

$$\begin{aligned} LCB(x) &= \mu(x) - \kappa\sigma(x), \\ PI(x) &= \Phi(\lambda), \\ EI(x) &= \sigma(x)(\lambda\Phi(\lambda) + \phi(\lambda)), \\ \lambda &= \frac{\tau - \xi - \mu(x)}{\sigma(x)}, \end{aligned} \tag{10}$$

where $\mu(x)$ is the predicted expectation and $\sigma(x)$ is the the uncertainty of prediction by the GP model.

The BO procedure is shown in Algorithm 1. We first randomly sample N_{init} points in the search space and use these point to train the initial GP model. Then, in each iteration we find the optimal x that optimizes the acquisition function as the next sample point. Then, we update the GP model with the new sample point. Finally, the best $f(x)$ recorded until the iterations stop is regarded as the optimum of $f(x)$.

2.4 Neural Architecture Search

The choice of the network architecture has proven to be critical for the tasks in computer vision. Recently, deep neural networks have been proposed for hotspot detection [14, 31] as well. Reference [31] applies a plain CNN architecture, which is shown in Figure 2(a) for hotspot detection. Reference [14] designs a ResNet-like BNN model whose structure is shown in Figure 2(b) for hotspot detection.

Nowadays people design neural networks by trying slight variations of the known architectures. Although it has become much easier, designing neural network architectures still requires much time and profound experience, which leads to the development of a wide variety of automated methods for building deep neural networks such as neural architecture search.

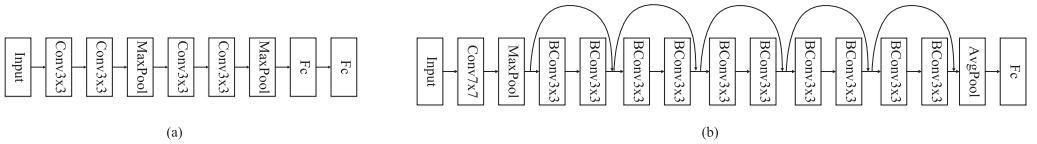

Fig. 2. Examples of neural architectures. Both (a) and (b) are manually designed neural networks for hotspot detection.

ALGORITHM 1: Bayesian Optimization

Input:
1: N_{iter}: number of iterations;
2: N_{init}: number of initial sampling points.
Output: Optimal $f(x)$.
3: Randomly sample N_{init} points in the search space;
4: Construct the initial GP model;
5: **for** $i = 1$ to N_{iter} **do**
6: Find x_i that optimizes the acquisition function;
7: Compute $f(x_i)$ as the ground truth of x_i;
8: Update the GP model;
9: **end for**
10: **return** best $f(x)$ recorded during iterations.

Neural architecture search algorithms mainly contains evolutionary algorithms [16, 23, 26], Bayesian optimization [15] and reinforcement learning [21, 33]. Evolutionary algorithm-based NAS algorithms are more flexible for composing novel neural architectures but hard to perform well when adopted in a large scale and often require many heuristics to work well. Reinforcement learning-based NAS algorithms are widely applied recently whose basic process is presented in Figure 3. The structure of a neural network architecture can be encoded with a variable-length string. The controller (e.g., an RNN) can generate such strings that can be transformed to "child networks". Child networks are then trained on the training set and validated on the validation set. The policy gradient is computed to update the controller with the validation accuracy of the child networks as the reward signal. Thus, the neural architectures with higher accuracy will get higher probabilities in the next iterations. The auto-regressive structure enables the controller to improve the network.

3 PROPOSED NEURAL ARCHITECTURE SEARCH-BASED HOTSPOT DETECTOR

In this section, we will present the proposed neural architecture search method for layout hotspot detection.

3.1 Search Space Formulation

Following the practice of [17], we search for the cells instead of searching for the whole neural network architecture. The searched cell then can be stacked to build the final neural network. The search space is represented by a **directed acyclic graph** (**DAG**). Figure 4 gives an example of a network architecture represented by DAG. Each cell consists of N ordered nodes each of which is a tensor x^i. Each edge (i, j) is an operation $o_{i,j}$, which transforms node x^i to node x^j.

Nodes categories are shown as follows: (1) input nodes, which are the output tensors of the previous cells such as nodes I_0 and I_1 in Figure 4; (2) intermediate nodes such as nodes 2, 3, 4, and 5 in Figure 4; and (3) output nodes such as node *output* in Figure 4. All intermediate nodes

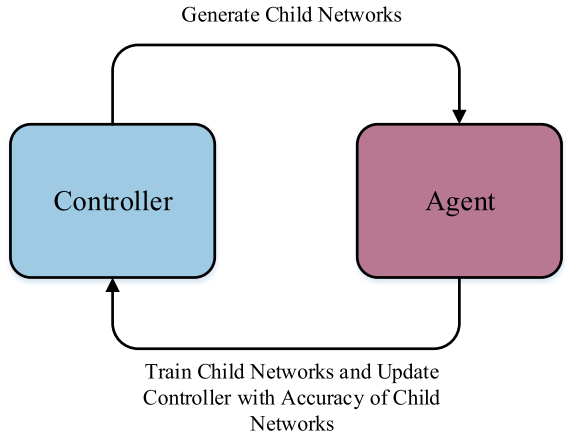

Fig. 3. Overview of neural architecture search.

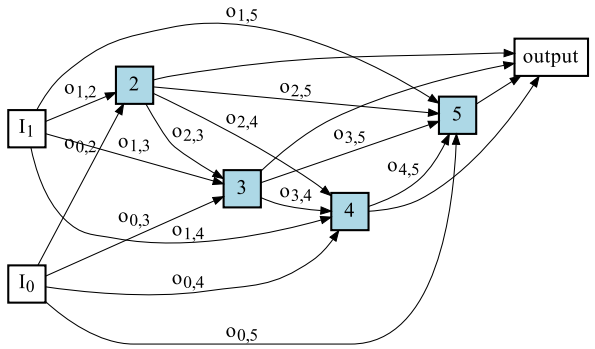

Fig. 4. Example of neural architecture represented by DAG. Nodes I_0 and I_1 are input nodes; nodes 2, 3, 4, and 5 are intermediate nodes; node *output* is the output node.

are concatenated as the output node. There are eight types of operations for each edge to choose: (1) separable convolution layer with 3×3 kernel; (2) separable convolution layer with 5×5 kernel; (3) dilated separable convolution with 3×3 kernel; (4) dilated separable convolution with 5×5 kernel; (5) max pooling with 3×3 kernel; (6) average pooling with 3×3 kernel; (7) identity; and (8) *zero*. The nodes with multiple input edges are computed as the element-wise sum of the input edges.

Each cell can be seen as a convolutional neural network that maps a tensor with shape $c\times h\times w$ to the tensor with shape $c'\times h'\times w'$. In the reduction cells stride of the convolution operations is 2, which leads to the fact that $h=2h'$ and $w=2w'$. In the norm cells, the stride of the convolution operations is 1 so that $h=h'$ and $w=w'$. Following the common heuristic in designing neural architectures that $h\times c=h'\times c'$ (usually $h=w$), in the reduction cells $c'=2c$ and in the norm cells $c'=c$. As mentioned before the final neural architecure is composed of several stacked cells including norm cells and reduction cells.

3.2 Bayesian Optimization-based Neural Architecture Searching Scheme

The architecture search space introduced in Section 3.1 is discrete; thus, it is important to map it to a continuous space. The distance feature needs to be defined to show the difference between different architectures. Since that the search space can be represented by a DAG $G=(V,E)$, inspired by [27] we define three feature matrices to construct the distance feature.

ALGORITHM 2: Feature Matrix

Input:

$G = (V, E)$: graph that represents the architecture;
D: $N_{node} \times N_{node}$ matrix;
w: edge weights.

Output: Feature matrix of the architecture.

Initialize D as all zero;
for each edge (i, j) in E **do**
    $D_{i,i} = D_{i,i} + w(i, j)$;
    $D_{j,j} = D_{j,j} + w(i, j)$;
    $D_{i,j} = D_{i,j} - w(i, j)$;
    $D_{j,i} = D_{j,i} - w(i, j)$;
end for
return D.

ALGORITHM 3: Distance Feature

Input: L: feature matrix of the architecture.

Output: Distance feature of the architecture.

Compute the eigenvectors and the eigenvalues of the feature matrix L;
Sort the eigenvalues and pick the second smallest and the third smallest eigenvalues as v_2 and v_3;
Pick the corresponding eigenvector q_2, q_3 and calculate their absolute values;
return the merged vector $(|q_2|, |q_3|)$.

The first feature matrix D is the Laplacian matrix of the DAG, which is defined in Algorithm 2. For each edge in the DAG, the number at corresponding index is adjusted with the predefined edge weight.

The second feature matrix A defined in Equation (11), which is a diagonal matrix is to scale the matrix D,

$$\begin{cases} A_{i,j} = 0, & i \neq j; \\ A_{i,j} = \frac{1}{\sqrt{\sum_{(k,i)\in E} w(k,i)}}, & i = j. \end{cases} \tag{11}$$

Where each diagonal element is the weighted in-degree of the corresponding node and $w(k, i)$ is the corresponding predefined weight. The scaled feature matrix L is defined in Equation (12),

$$L = A \cdot D \cdot A, \tag{12}$$

where $(.) \cdot (.)$ is inner product of matrices.

The distance feature is defined in Algorithm 3. The eigenvectors that correspond to the second smallest and third smallest eigenvalues of the feature matrix L are picked out. Their absolute values are merged as the final distance feature.

To show the practicality of the distance feature definition, we first randomly sample an architecture $g1$ and generate its distance feature denoted by d_1. Then, we make a few small changes and get a similar architecture $g2$ whose distance feature is denoted by d_2. We then randomly sample another totally different architecture $g3$ and get its distance feature d_3. Finally, $\|d_1 - d_2\|$ and $\|d_1 - d_3\|$ are calculated where $\|.\|$ is the l_2 norm of the vector. An example is shown in Figure 5. For the example in Figure 5, $\|d_1 - d_2\| = 0.498$ and $\|d_1 - d_3\| = 1.241$, which shows that the distance feature can discriminate the similar architectures and different architectures.

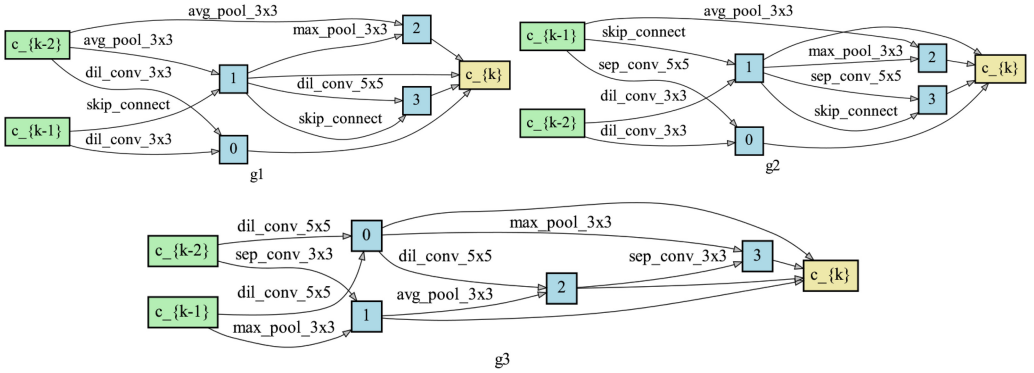

Fig. 5. The distance features of similar and different architectures. $g1$ is a randomly sampled architecture in the search space. $g2$ is similar to $g1$ with some small modifications. $g3$ is another randomly sampled architecture, which is totally different from $g1$.

ALGORITHM 4: Bayesian Optimization based Searching Scheme

Input:

N_{iter}: number of iterations;

N_{init}: number of initial sampling points.

Output: Optimal architecture.

Randomly sample N_{init} architectures in the search space;

Map these architectures to the distance vectors with Algorithm 3;

Train and evaluate these architectures;

Construct the initial GP model with the generated data;

for $i = 1$ to N_{iter} **do**

    Construct the acquisition function with existing GP model;

    Find optimal architecture that optimizes the acquisition function using evolutionary algorithm (EA);

    Map this architecture to the distance vector with Algorithm 3;

    Train and evaluate the new architecture;

    Update the GP model;

end for

return best architecture during iterations.

After the definition of the distance feature, we illustrate the whole process of our searching algorithm in Algorithm 4. Firstly, N_{init} architectures are randomly sampled from the search space as the initial points. Then train and evaluate these architectures and build the initial GP model with the accuracies and distance features mapped from the architectures. The LCB acquisition function is then constructed. In each iteration of the BO procedure, we find the architecture that optimizes the acquisition function with **evolutionary algorithm** (**EA**). The optimal architecture is then trained and evaluated. The GP model is updated using the distance vector mapped from the new architecture and its accuracy. Finally, we select the architecture with best performance during the iterations as the final architecture. It is then trained from scratch and tested.

Table 2. Performance Degradation Caused by Accuracy Prediction Model

	1 epoch	Model (5 epochs)	100 epochs	200 epochs
Mean Square Error	–	0.054	–	–
Accuracy	98.0	98.4	98.6	98.3
FA	15.7	14.1	14.5	13.8

Since training a CNN architecture from scratch is quite time-consuming, a linear regression model is applied to help accelerate the training procedure. The model predicts the performance of the architecture based on the training results of first k epochs as shown in Equation (13),

$$\begin{aligned} h_\theta(x) &= \sum_{i=0}^{n} \theta_i x_i, \\ J(\theta) &= \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2, \\ \theta_* &= \arg\min_\theta J(\theta), \end{aligned} \tag{13}$$

where $x^{(i)} = x_1 \dots x_n$ is the training and validating accuracies of the ith architecture in the first k epochs and $y^{(i)}$ is the the groundtruth accuracy of the ith architecture. The model parameter θ is optimized to minimize the loss denoted by $J(\theta)$.

To study the performance degradation caused by the accuracy prediction model, a preliminary experiment is performed on a smaller dataset randomly sampled from the ICCAD 2012 dataset. Table 2 shows the performance of neural architectures searched under different settings. Column "1 epoch" shows the result of training for 1 epoch during searching. Column "5 epochs" shows the result of training for five epochs with accuracy prediction model during searching. Columns "100 epochs" and "200 epochs" show the results of training for 100 and 200 epochs during searching. The experimental results show that the accuracy prediction model can achieve comparable accuracy and false alarm although it might lead to some precision loss (the mean square error is 0.054).

Thus, we only need to train the architectures for k epochs ($k = 5$ in this article) during the search procedure, which can save much time. In addition to save the time of pretraining, we can also set the parameters of the linear model as hyperparameters to tune. To further improve the searching efficiency shallower architectures with fewer stacking cells are adopted during the search procedure.

3.3 Customizations for Hotspot Detection

The hotspot detection problem is different from the problems that normal NAS algorithms handle such as language modeling and image recognition. Thus there are some customizations for hotspot detection.

(1) The feature extraction process is significant for hotspot patterns. For the CNN architecture-based hotspot detector we apply the DCT-based feature extraction [31].

(2) Because of the imbalanced dataset, the hotspot samples are up-sampled during the searching and training process. The hotspot samples get more chance to be sampled during training so that each mini-batch contains nearly the same amount of hotspots and non-hotspots. We also adopt the biased learning algorithm in [31] to achieve a better balance between the detecting accuracy and false alarm. The trained model is fine-tuned with biased label.

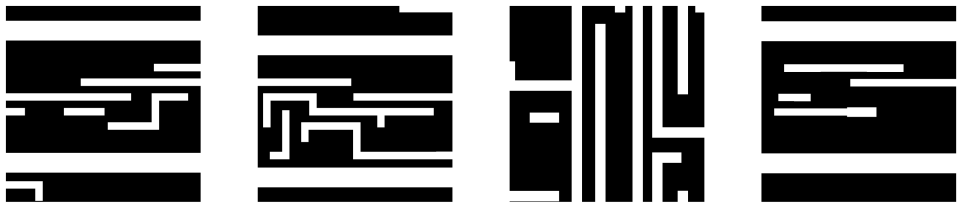

Fig. 6. Clip samples in the dataset. Figure (a) and (b) contains hotspots; figure (c) and (d) does not conatin hotspots.

Table 3. Statistics of the ICCAD 2012 and 2019 Benchmarks

Benchmark	#Train HS	#Train NHS	#Test HS	#Test NHS
ICCAD 12	1,204	17,096	2,524	13,503
ICCAD 19-1	467	17,758	1,001	14,621
ICCAD 19-2	467	17,758	64,310	65,523

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

All our training and testing code is built with Python based on the PyTorch [20] library. The testing platform is a workstation with Intel Xeon E5 CPU and Nvidia Tesla K80 GPU.

Two benchmarks are employed including ICCAD 2012 benchmark [28] and ICCAD 2019 benchmark [24]. Figure 6 shows some samples in the benchmarks. Figure 6(a) and (b) contains hotspots; Figure 6(c) and (d) does not contain hotspots. The details of ICCAD 2012 dataset are listed in Table 3. The ICCAD 2019 benchmark is divided into two subsets, which share the training set: ICCAD19-1 and ICCAD19-2 benchmarks. The statistics of the ICCAD 2019 dataset are listed in Table 3 as well. In the table, columns "# Training HS" and "# Training NHS" denote the numbers of hotspots and non-hotspots in training set, respectively. Columns "# Testing HS" and "# Testing NHS" denote the numbers of hotspots and non-hotspots in testing set.

4.2 Implementation Details

4.2.1 Data Preprocessing. We follow the practice in the previous article [31] for data preprocessing. The DCT-based feature extraction method is applied before sent to the convolutional neural networks. Random horizontal and vertical flipping and Cutout [3] are performed for data augumentation.

Because of the imbalance between the hotspot class and non-hotspot class, we choose the strategy with minority class (hotspots) over-sampled and majority class (non-hotspots) down-sampled during training.

4.2.2 Training Hyperparameters. We use SGD with momentum to optimize the weights. Table 4 shows the hyperparameter settings for training. The learning rate is annealed down to 0 with the cosine schedule. We set the path dropout probability to 0.2 following the practice in previous work [17]. The cutout length is set to 16 for data augumentation. The search scheme runs for 200 iterations to get the optimal neural architecture. The searched architecture is then trained from scratch.

Table 4. Training Hyperparameter Settings

Hyperparameter	Value
Batch Size	128
Bias Term (ϵ)	0.2
Data Augumentation Method	Random flipping and cutout
Initial Learning Rate (λ)	0.025
Optimizer	SGD with Momentum
Momentum	0.9
Weight Decay	2×10^{-4}
Cutout Length	16

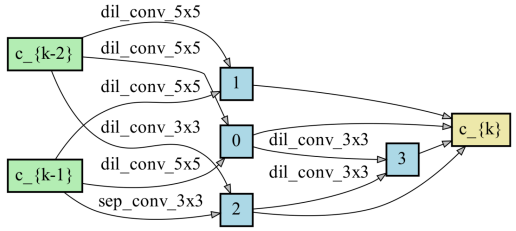

Fig. 7. The searched norm cell for ICCAD12.

4.2.3 Other Details. The number of input nodes in a cell is set as 2, intermediate nodes number is set as 4, and output node number is set as 1. Each node will be connected with two previous nodes following the practice in [17].

All outputs of each channel are globally averaged after the last convolution layer as the input of the softmax layer. This is to reduce the number of the parameters in the dense connection so as to avoid the overfitting problem.

While a layer is connected to several previous layers, the outputs of these previous layers are concatenated in their depth dimension and convolved with kernel size 1×1.

4.3 Performance Comparison with State-of-the-art

Figures 7 and 8 are our generated cell architectures searching on the ICCAD 2012 dataset. Figures 9 and 10 are our searched cells based on the ICCAD 2019 benchmark. The two green nodes are the input nodes and the four blue nodes are the intermediate nodes. The yellow node `c_{k}` is the output node. The operations between the nodes is labeled on the corresponding edges. Each intermediate node gets two input edges from the previous nodes.

Table 5 summarizes the results of our proposed algorithm and three state-of-the-art hotspot detectors. Columns "Accu", "FA", "F1", and "Time" show the hotspot detection accuracy, false alarm ratio, F1 score and runtime of the algorithm. Columns "TCAD'19 [31]", "DAC'19 [14]", "TCAD'19 [2]", and "Ours" show the results of the selected state-of-the-art hotspot algorithms and our proposed hotspot detection scheme. Our proposed algorithm gets higher accuracy than other state-of-the-art algorithms on average (87.6% of TCAD'19 [31], 89.9% of DAC'19 [14], and 76.4% of TCAD'19 [2] v.s. 92.1% of ours). Our proposed hotspot detector achieves comparable average false alarm rate (34.6% of ours compared with 34.3% of TCAD'19 [2]). As a result, the average F1 score of our searched neural network is the highest among the four approaches. Our proposed method is of great efficiency as well. The average runtime of our searched architectures is the lowest among all

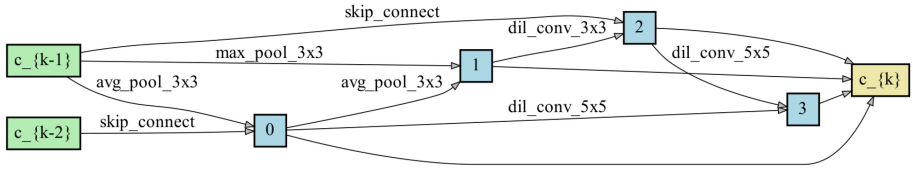

Fig. 8. The searched reduction cell for ICCAD12.

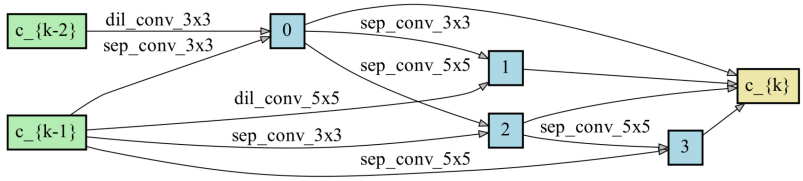

Fig. 9. The searched norm cell for ICCAD19.

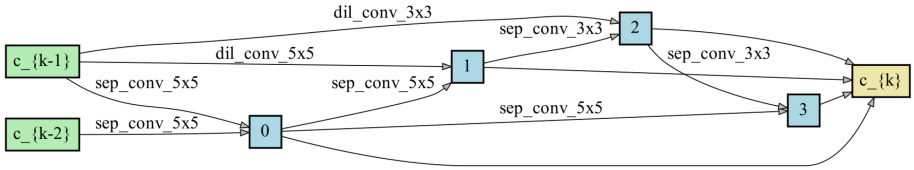

Fig. 10. The searched reduction cell for ICCAD19.

Table 5. Performance Comparison with State-of-the-art Approaches

Dataset	TCAD'19 [31]				DAC'19 [14]				TCAD'19 [2]				Ours			
	Accu (%)	FA (%)	F1	Time (s)	Accu (%)	FA (%)	F1	Time (s)	Accu (%)	FA (%)	F1	Time (s)	Accu (%)	FA (%)	F1	Time (s)
ICCAD12	98.4	26.2	0.581	397	99.2	20.6	0.641	62	97.7	17.9	0.666	401	98.9	9.9	0.785	30
ICCAD19-1	76.0	2.6	0.710	53	80.9	3.5	0.667	63	46.9	1.5	0.556	56	87.2	9.7	0.530	44
ICCAD19-2	88.4	87.8	0.636	429	89.7	84.1	0.651	496	84.5	83.6	0.627	445	90.3	84.1	0.654	363
Average	87.6	38.9	0.642	293	89.9	36.1	0.653	207	76.4	**34.3**	0.616	301	**92.1**	34.6	**0.656**	**146**
Ratio	0.95	1.12	0.98	2.01	0.97	1.04	0.995	1.42	0.83	**0.99**	0.94	2.06	**1**	1	**1**	**1**

The bold values are the best performed ones.

hotspot detectors (293s of TCAD'19 [31], 207s of DAC'19 [14], and 301s of TCAD'19 [2] v.s. 146s of our architecture).

To show the performance improvement brought by our proposed algorithm, we perform an ablation study. We randomly sample 10 architectures in our proposed search space and train them from scratch on both ICCAD 2012 and 2019 benchmarks. We also train the best searched architecture in 50, 100, 200, and 400 iterations from scratch to study the relation between performance and search iterations. Table 6 illustrates the performance of randomly generated architectures and searched architectures at different iterations. Column "random" shows the average performance of 10 randomly sampled architectures. Columns "50 iterations", "100 iterations", "200 iterations", and "400 iterations" refer to the performance of the best architecture searched in 50, 100, 200, and 400 iterations. Columns "Accu", "FA" show the hotspot detection accuracy, false alarm ratio. The final searched architecture achieves 5.3% higher accuracy and 3.1% lower false alarm rate than the randomly sampled architectures on average. The performance significantly imrpoves as the number of iterations grows before 200 iterations. The search scheme converges at 200 iterations and searching for 400 iterations does not make any change.

Table 6. Performance Comparison with Randomly Generated Architectures and Searched Architectures at Different Iterations

Dataset	random		50 iterations		100 iterations		200 iterations		400 iterations	
	Accu (%)	FA (%)	Accu (%)	FA (%)	Accu (%)	FA (%)	Accu (%)	FA (%)	Accu (%)	FA (%)
ICCAD12	95.2	13.1	97.9	10.7	98.3	9.1	98.9	9.9	98.9	9.9
ICCAD19-1	81.2	10.3	84.6	10.5	86.7	10.1	87.2	9.7	87.2	9.7
ICCAD19-1	83.9	89.7	87.1	86.1	90.3	87.4	90.3	84.1	90.3	84.1
Average	86.8	37.7	89.9	35.8	91.8	35.5	**92.1**	**34.6**	**92.1**	**34.6**
Ratio	0.942	1.090	0.975	1.034	0.996	1.027	**1**	**1**	**1**	**1**

The bold values are the best performed ones.

Table 7. Performance Comparison with Baseline NAS Algorithm

Dataset	DARTS [17]				Ours			
	Accu (%)	FA (%)	Runtime (s)	Search Time (h)	Accu (%)	FA (%)	Runtime (s)	Search Time (h)
ICCAD12	98.5	9.9	31	19	98.9	9.9	30	12
ICCAD19-1	85.1	4.8	40	18	87.2	9.7	44	12
ICCAD19-2	92.0	89.5	349	30	90.3	84.1	363	12
Average	91.9	34.7	**140**	22.3	**92.1**	**34.6**	146	**12**
Ratio	0.998	1.004	**0.959**	1.86	**1**	**1**	1	**1**

The bold values are the best performed ones.

The proposed NAS approach shares the same search space with the baseline method DARTS [17] but search for the optimal neural architecture with a different algorithm (Bayesian optimization-base algorithm). To show the advantage of our proposed search algorithm, we compare our result with DARTS as well. Table 7 summarizes the results of the two methods. "Accu", "FA", "Runtime", and "Search Time" show the hotspot detection accuracy, false alarm ratio, runtime of the algorithm, and search time of the algorithm. The experimental results show that our proposed search approach can generate a competitive architecture (0.2% higher accuracy, 0.1% lower false alarm rate, and 6s more runtime on average) with 1.86× speed on average.

5 CONCLUSION

Deep learning-based hotspot detection approaches have shown great potential recently. However, designing and tuning the deep learning models can be quite time consuming. For the first time, we propose a bayesian optimization-based neural architecture search scheme for automated hotspot detection. Time-consuming manual tuning is replaced with efficient searching process. The searched neural architecture can be reused while dealing with more layout patterns instead of slow lithographic simulation. The experimental results on ICCAD 2012 and ICCAD 2019 benchmarks show that the architectures designed by our proposed scheme outperform the manually designed architectures in both accuracy and efficiency as well. With the transistor size shrinking rapidly, we wish to apply our scheme for a more convenient and efficient hotspot detector designing.

REFERENCES

[1] Ran Chen, Wei Zhong, Haoyu Yang, Hao Geng, Xuan Zeng, and Bei Yu. 2019. Faster region-based hotspot detection. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 146.

[2] Ying Chen, Yibo Lin, Tianyang Gai, Yajuan Su, Yayi Wei, and David Z. Pan. 2019. Semisupervised hotspot detection with self-paced multitask learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 7 (2019), 1511–1523.

[3] Terrance DeVries and Graham W. Taylor. 2017. Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552. Retrieved from https://arxiv.org/abs/1708.04552.

[4] Duo Ding, Andres J. Torres, Fedor G. Pikus, and David Z. Pan. 2011. High performance lithographic hotspot detection using hierarchically refined machine learning. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*.

[5] Duo Ding, Bei Yu, Joydeep Ghosh, and David Z. Pan. 2012. EPIC: Efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation. In *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC), 2012*. IEEE, 263–270.

[6] Duo Ding and David Z. Pan. 2009. Machine learning based lithographic hotspot detection with critical feature extraction and classifications. In *Proceedings of the International Conference on Integrated Circuit Design Technology*.

[7] Dragoljub Gagi Drmanac, Frank Liu, and Li-C Wang. 2009. Predicting variability in nanoscale lithography processes. In *Proceedings of the 46th Annual Design Automation Conference*. ACM, 545–550.

[8] Andrew B. Kahng, Chul-Hong Park, and Xu Xu. 2006. Fast dual graph based hotspot detection. In *Proceedings of the International Society for Optics and Photonics*.

[9] D. Ding B. Yu, J. Ghosh, and D. Z. Pan. 2012. Efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation. In *Proceedings of the 17th Asia and South Pacific Design Automation Conference*. 263–270.

[10] Jen-Yi Wuu, Fedor G. Pikus, and Malgorzata Marek-Sadowska. 2011. Efficient approach to early detection of lithographic hotspots using machine learning systems and pattern matching. In *Design for Manufacturability through Design-Process Integration V*, Vol. 7974. SPIE, 243–250.

[11] Y. T. Yu, Y. C. Chan, S. Sinha, I. H. R. Jiang, and, C. Chiang. 2012. Accurate process-hotspot detection using critical design rule extraction. In *Proceedings of the 49th Annual Design Automation Conference*. 1167–1172.

[12] Yen-Ting Yu, Geng-He Lin, Iris Hui-Ru Jiang, and Charles Chiang. 2015. Machine-learning-based hotspot detection using topological classification and critical feature extraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 3 (2015), 460–470.

[13] Hailong Yao, Subarna Sinha, Jingyu Xu, C. Chiang, Yici Cai, and Xianlong Hong. 2008. Efficient range pattern matching algorithm for process-hotspot detection. *IET Circuits, Devices & Systems* 2, 1 (2008), 2–15.

[14] Yiyang Jiang, Fan Yang, Hengliang Zhu, Bei Yu, Dian Zhou, and Xuan Zeng. 2019. Efficient layout hotspot detection via binarized residual neural network. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 147.

[15] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P. Xing. 2018. Neural architecture search with bayesian optimisation and optimal transport. In *Proceedings of the Advances in Neural Information Processing Systems*. 2016–2025.

[16] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. arXiv:1711.00436. Retrieved from https://arxiv.org/abs/1711.00436.

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. arXiv:1806.09055. Retrieved from https://arxiv.org/abs/1806.09055.

[18] Tetsuaki Matsunawa, Jhih-Rong Gao, Bei Yu, and David Z. Pan. 2015. A new lithography hotspot detection framework based on adaboost classifier and simplified feature extraction. In *Proceedings of the Design-Process-Technology Co-optimization for Manufacturability IX*, Vol. 9427. International Society for Optics and Photonics, 94270S.

[19] Tetsuaki Matsunawa, Bei Yu, and David Z. Pan. 2015. Optical proximity correction with hierarchical bayes model. In *Proceedings of the Optical Microlithography XXVIII*, Vol. 9426. International Society for Optics and Photonics, 94260X.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019) 8024–8035.

[21] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. arXiv:1802.03268. Retrieved from https://arxiv.org/abs/1802.03268.

[22] Carl Edward Rasmussen. 2003. Gaussian processes in machine learning. In *Proceedings of the Summer School on Machine Learning*. Springer, 63–71.

[23] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4780–4789.

[24] Gaurav Rajavendra Reddy, Kareem Madkour, and Yiorgos Makris. 2019. Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 1–8.

[25] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE* 104, 1 (2015), 148–175.

[26] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life* 15, 2 (2009), 185–212.
[27] Yangfeng Su, Fan Yang, and Xuan Zeng. 2012. AMOR: An efficient aggregating based model order reduction method for many-terminal interconnect circuits. In *Proceedings of the 49th Annual Design Automation Conference*. 295–300.
[28] J. Andres Torres. 2012. ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite. In *Proceedings of the 2012 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 349–350.
[29] Wan-Yu Wen, Jin-Cheng Li, Sheng-Yuan Lin, Jing-Yi Chen, and Shih-Chieh Chang. 2014. A fuzzy-matching model with grid reduction for lithography hotspot detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 11 (2014), 1671–1680.
[30] Fan Yang, Subarna Sinha, Charles C. Chiang, Xuan Zeng, and Dian Zhou. 2017. Improved tangent space-based distance metric for lithographic hotspot classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 9 (2017), 1545–1556.
[31] Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline FY Young. 2018. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 6 (2018), 1175–1187.
[32] Hang Zhang, Bei Yu, and Evangeline FY Young. 2016. Enabling online learning in lithography hotspot detection with information-theoretic feature optimization. In *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 47.
[33] Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. arXiv:1611.01578. Retrieved from https://arxiv.org/abs/1611.01578.

Received July 2021; revised November 2021; accepted January 2022