Explainable DRC Hotspot Prediction with Random Forest and SHAP Tree Explainer

Wei Zeng, Azadeh Davoodi
Department of Electrical and Computer Engineering
University of Wisconsin–Madison, Madison, WI, USA
{wei.zeng, adavoodi}@wisc.edu

Rasit Onur Topaloglu
IBM
Hopewell Junction, NY, USA
rasit@us.ibm.com

***Abstract*—With advanced technology nodes, resolving design rule check (DRC) violations has become a cumbersome task, which makes it desirable to make predictions at earlier stages of the design flow. In this paper, we show that the Random Forest (RF) model is quite effective for the DRC hotspot prediction at the global routing stage, and in fact significantly outperforms recent prior works, with only a fraction of the runtime to develop the model. We also propose, for the first time, to adopt a recent explanatory metric—the SHAP value—to make accurate and consistent explanations for individual DRC hotspot predictions from RF. Experiments show that RF is 21%–60% better in predictive performance on average, compared with promising machine learning models used in similar works (e.g. SVM and neural networks) while exhibiting good explainability, which makes it ideal for DRC hotspot prediction.**

***Index Terms*—design rule check, machine learning, random forest, explainability, global routing**

I. Introduction

Today's VLSI fabrication technologies require satisfying many complex design rules to ensure manufacturability. Creating a layout that is clean of design rule violations is now a cumbersome task, which may require many iterations in the design flow. Within the design flow, Design Rule Check (DRC) is typically applied after detailed routing. However, the process of detailed routing can be rather tedious and expensive, which typically takes several hours, if not days, to finish. Hence it is highly desirable that an inexpensive DRC predictor is developed so that DRC hotspots on the layout may be predicted accurately at the earlier stages in the design flow. In addition, it is beneficial if predictions for individual DRC hotspots can be properly *explained* in order to point to the root cause behind each individual violation. In this way, designers may leverage this early feedback without going through detailed routing and DRC phases each time.

Recent researches have focused on predicting routability and DRC hotspots [1]–[7] with machine learning. They have identified various features at the placement and/or global routing stages which can contribute to DRC violations. Several researchers [2], [3], [5] adopted support vector machines (SVMs) with radial basis function (RBF) kernels, Tabrizi *et al.* used a boosting ensemble model in [4] and a feedforward neural network (NN) in [6]. Xie *et al.* proposed RouteNet [7], a convolutional neural network (CNN) with transfer learning.

This research is supported by National Science Foundation grant #1608040.

It is worth noting that, these works mainly focused on the predictive performance, without much consideration on other important aspects in practice, such as the model development cost and model explainability as well as data availability. Specifically, with a large number of features and samples, the SVM model with RBF kernel is expensive to train and the predictions are difficult to explain. As for data availability, some researches [4], [6] assumed that the accurate DRC results are partially available for random regions of the layouts; they split samples in the *same* design into training and testing sets based on this optimistic assumption. Moreover, unlike most other works, where each data sample contains layout information in a small region, RouteNet [7] used features in the *entire* layout as a single input to the model, and thus it requires hundreds of different placements from *each* design and the corresponding DRC results (after detailed routing each placement) for model training, assuming all macros are movable in placement. For this reason, data acquisition in [7] may be computationally expensive. Finally, no prior work provided model explainability to analyze *individual* violations.

To address these issues, **in this paper**, we propose to use the random forest (RF) classifier [8] as an ideal candidate to predict DRC hotspots, considering predictive performance and computational cost for model development. For the first time, we also provide consistent explanations for *individual* DRC hotspot predictions, with a recent explanatory model named SHAP [9] that is compatible with RF. Each explanation identifies top-ranked features and their amounts of contribution to a predicted DRC hotspot, which suggest the root causes behind each violation. Our contributions are summarized below.

- We propose model evaluation metrics that are tailored for DRC hotspot prediction where the number of DRC violations may be relatively very small, including area under the precision-recall curve for predictive performance, and number of predictive operations for model complexity.
- With these metrics, we carry out a comparative study on RF and machine learning models from recent works on DRC hotspot prediction with similar settings [2]–[6], with standard procedures of cross validation and hyperparameter tuning, where data availability is carefully considered.
- By exploiting recent advances in explanatory analysis, we provide reasonable explanations for individual DRC hotspots predicted by RF, validated with real examples.

978-3-9819263-4-7/DATE20/©2020 EDAA

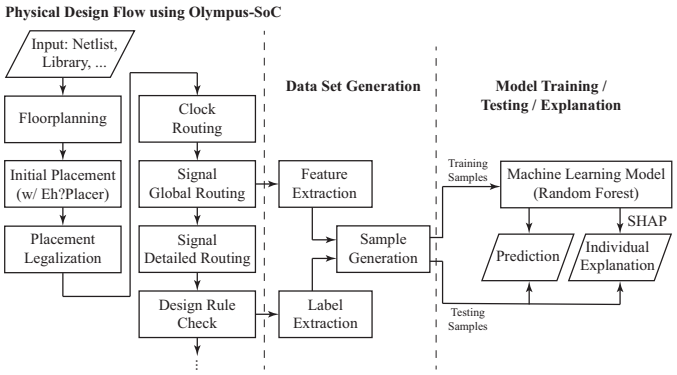

Fig. 1. Workflow of explainable DRC hotspot prediction.

II. DRC Hotspot Prediction Workflow

In the DRC hotspot prediction problem, given a global routing (GR) outcome, we predict whether a global routing cell (g-cell) [10], after detailed routing, will contain at least one DRC violation. The prediction is made based on other routed designs with the same technology and same design flow.

We show the overall workflow in Fig. 1. Our approach is to formulate this problem as a supervised classification problem. We extract features from placement and GR (shown in the left panel of Fig. 1) to form a feature vector (a.k.a. data sample) for each g-cell, and then feed it to a machine learning model, which accepts this feature vector as input and produces an output indicating how likely the g-cell is a DRC hotspot.

The data acquisition process is shown in the middle panel in Fig. 1. We use 14 designs in 65 nm technology with five routing layers from the ISPD 2015 contest benchmark suite [11][1]. Each design is first fed into `Eh?Placer` [12], which produces a placed `.def` file. Then with Olympus-SoC, we follow a standard SoC flow with the steps shown in the figure. After GR of signal nets, the intermediate results are used to generate the feature vectors, and the DRC errors reported in the last step are used to determine whether they are actual DRC hotspots. Refer to Section II-A for details.

Model training and testing work as follows. The 14 designs are randomly divided into five groups with roughly equal number of samples, as shown in Table I. For predicting the DRC hotspots in a specific design, we exclude the group that contains the design (i.e. testing group) and use all designs in the other four groups (i.e. training groups) for model training and tuning. The aforementioned procedure respects data availability by guaranteeing the *entire design* under test is never foreseen in the training stage. It avoids potential optimism in evaluation as in [4], [6], and better matches the practice in physical design, where the actual DRC errors become available after a design is detail-routed in its entirety.

In the training stage of our workflow, we use grid search with 4-fold cross validation to find the best hyperparameters. This process consists of four passes. For each pass, designs in one of the four training groups are held out for validation and the designs in the remaining three groups are used for training. The performances from the four passes of cross validation are averaged. Then we compare and select the hyperparameter set with the best performance, and retrain the final model with the whole training set (i.e. all 4 training groups) before testing. As such, no *designs* (rather than samples) for validation is foreseen in the training process, which resembles the training-testing split and thus avoids the optimism bias in validation.

With predictions from the trained model, we explain them by inspecting the feature contributions to individual DRC hotspots with SHAP [9], as will be described in Section III-C.

[1]Design `edit_dist_a` is excluded from our experiments since it took more than 10 days to detail route and is therefore considered unroutable. `superblue` designs are excluded because the technology is different. We include two hidden designs `mult_2` and `mult_c` available at the contest website, which were released after the contest.

TABLE I
The Profile and Grouping of Designs

Design	# G-cells	# DRC hotspots	# Macros	# Cells (k)	Layout size (μm)
Group 1	**29994**	**364**	—	—	—
des_perf_b	10000	0	0	112.6	600×600
fft_2	3249	17	0	32.3	265×265
mult_1	8281	154	0	155.3	550×550
mult_2	8464	193	0	155.3	555×555
Group 2	**28263**	**547**	—	—	—
fft_b	6506	534	6	30.6	800×800
mult_a	21757	13	5	149.7	1500×1500
Group 3	**27826**	**669**	—	—	—
mult_b	24257	613	7	146.4	1500×1500
bridge32_a	3569	56	4	29.5	400×400
Group 4	**29689**	**738**	—	—	—
des_perf_1	5476	676	0	112.6	445×445
mult_c	24213	62	7	146.4	1500×1500
Group 5	**30318**	**298**	—	—	—
des_perf_a	11498	246	4	108.3	900×900
fft_1	1936	50	0	32.3	265×265
fft_a	6491	2	6	30.6	800×800
bridge32_b	10393	0	6	28.9	800×800

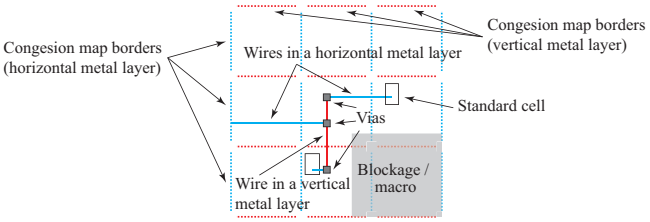

Fig. 2. A 3 × 3 g-cell window with standard cells, wires (different colors indicate different metal layers), vias, congestion map borders, blockage/macro.

A. Feature and Label Extraction

Fig. 2 illustrates the types of layout information available after the placement and GR stages. Using these, prior works have defined different types of features related to routability and thus the DRC hotspot prediction, including

- Location of g-cells in the layout [4], [6],
- Density-related information (e.g., cell density [3]–[6], [13], pin density [1]–[5], [13], [14], pin spacing/distribution [2], [3], [6]),
- Special pins and cells, which may have constraints in routing (e.g., clock pins [6], pins in nets with non-default rules (NDRs) [6], [13], and multi-height cells [3]),
- Connectivity, where complex connections around the g-cells may complicate routing (e.g., local nets [1]–[3], [5], [13], and cross-border nets [2], [3], [5], [6]),
- Congestion map [1], [3], [5], [6], [13], which indicates supply and demand of routing resources,

- Blockages for placement and/or routing [1], [4], [6], which further limit the routing resources.

Recent works [3]–[6] also extract features in a window including neighboring g-cells to consider their contributions to DRC error due to potential routing detours.

Inspired by these prior works, **in this paper**, we extract the following features in the designs, from the placed cells and the congestion map after signal GR, which are explained in Fig. 2. Each data sample corresponds to a g-cell in the layout, which is expanded to a 3×3 *window* consisting of this g-cell (referred to as "central g-cell") and its 8 neighbors[2].

- For each of the 9 g-cells in the window, we extract
  - The center x- and y-coordinates, normalized to $[0, 1]$.
  - The numbers of standard cells, pins, and clock pins that are fully inside the g-cell.
  - The number of local nets, defined as nets whose all pins are inside the same g-cell.
  - The number of pins that belong to any local net.
  - The number of pins that have NDRs, as defined in the ISPD 2015 contest benchmarks in our experiments.
  - The pin spacing, defined as the arithmetic mean of pair-wise Manhattan distances of pins inside g-cell.
  - The percentage of area occupied by blockages.
  - The percentage of area occupied by standard cells.
- For each of 12 congestion border edges (i.e., segments with blue/red dots in Fig. 2) on *each metal layer*, and for each of 9 g-cells inside the window on *each via layer*,
  - The capacity C, defined as the maximum allowed number of wires/vias across the edge.
  - The load L, defined as the number of wires that are already across the edge (for metal layers) / the number of vias inside the g-cell (for via layers).
  - The resource margin, i.e., the difference of C and L.

We include almost all applicable features from prior works. This results in 387 features in total. To determine the labels, we examine the bounding boxes of DRC errors as reported by Olympus-SoC. A g-cell is a DRC hotspot if and only if the g-cell overlaps with any DRC error bounding box. A sample is positive if and only if the central g-cell is a DRC hotspot.

III. Proposed Approaches

In this section, we elaborate our approaches to modeling predictions with Random Forest, defining metrics for evaluation, and providing explainability for individual predictions.

A. Random Forest and Its Benefits for DRC Hotspot Prediction

Random Forest (RF) classifier [8] is a well-known ensemble learning model based on decision trees, which typically trains hundreds of randomized decision trees and aggregates their outputs to generate a final prediction.

Compared to other classifiers with similar problem formulation for DRC hotspot prediction [2]–[6], RF has good and robust performance in prediction, owing to the ensemble mechanism. Moreover, because of the randomization in choosing the features to split, RF is robust in the presence of uninformative and redundant features. This property is especially good for our problem with a large number of features whose relative importance is not known beforehand.

It has relatively low computational cost because the training and testing of underlying decision trees are much more straightforward than optimization-based models like neural network, support vector machine, etc., as used in [2], [3], [5], [6]. Furthermore, the aggregation process is naturally good for parallelism, which further reduces the computational time where multiple computing cores are available. The low computational cost also makes it feasible to perform extensive searching for the optimal hyperparameters. Last but not least, the tree-based structure of RF makes it transparent in making decisions, providing good explainability, which we utilize to explain *individual* predictions, as elaborated in Section III-C.

B. Metrics for Model Evaluation in DRC Hotspot Prediction

We propose several metrics for evaluating the predictive performance, complexity and computational cost of a model.

As can been seen in Table I, DRC-violated g-cells are much fewer than DRC-free g-cells. Therefore, accuracy (i.e. the percentage of correctly predicted samples) is not a good indicator of model performance [15] for DRC hotspot prediction. To address this, the following metrics are used in prior works.

- True positive rate $TPR = TP/(TP+FN)$, a.k.a. recall,
- False positive rate $FPR = FP/(TN + FP)$,
- Precision $Prec = TP/(TP + FP)$,

where TP and FP are the numbers of samples that are correctly and incorrectly predicted as positive, respectively; TN and FN are the numbers of samples that are correctly and incorrectly predicted as negative, respectively.

Although these metrics are better alternatives to accuracy, all of them can change when different thresholds of classification are applied [15]. Most works in DRC hotspot prediction [2], [3], [5], [6] used TPR and FPR only at a single threshold. In practice, however, the designer is free to adjust the threshold to get different prediction results with the same model. With this consideration, threshold-independent metrics, such as the areas under the receiver operating characteristic (ROC) curve and the precision-vs-recall (P-R) curve [15], are better indicators of overall model quality.

For the problem of DRC hotspot prediction in particular, one usually cares more about the performance when FPR is low (as the number of negative samples is typically large, a moderate FPR can imply a large number of undesired false alarms). For this reason, the area under ROC curve A_{roc}, which weights TPR over all FPRs equally, may not be effective enough. Therefore, **in this paper** we use the area under P-R curve (AUPRC) as the main performance metric, since it focuses more on the positive samples (both actual and predicted ones). For the same reason, AUPRC is used when we tune hyperparameters with cross validation. To understand how different models perform at a low FPR, we *also* report TPR and $Prec$ values at the classification threshold when $FPR = 0.5\%$. A similar FPR is seen in prior works [3], [7]. In summary, we use the following metrics for evaluation.

[2]If the central g-cell is on the boundary of the layout, neighbors outside the layout are padded with blank g-cells.

- TPR^*: true positive rate (a.k.a. recall) at the classification threshold such that $FPR = 0.5\%$.
- $Prec^*$: the precision at the same threshold as above.
- A_{prc}: the area under the precision-vs-recall curve.

C. Individual Explanations for Predicted DRC Hotspots

We propose to use a recent metric from machine learning community named **SH**apley **A**dditive ex**P**lanation (SHAP) values [16] and an efficient way to compute it for tree-based models [9], to analyze individual hotspot samples predicted by the RF model. With SHAP values, we estimate how much each of the 387 features contributes to the DRC errors found at *individual* predicted DRC hotspots. *Therefore, the designer is empowered to both predict and root cause individual DRC hotspots at an early design stage.*

More specifically, the RF model maps each sample with feature vector $\mathbf{x} \in \mathbb{R}^M$, M being the number of features, to a probability $f(\mathbf{x}) \in [0, 1]$ indicating how likely the sample is a DRC hotspot. Let $f(\mathbf{x}_i)$ denote the prediction of i-th sample with feature vector $\mathbf{x}_i$. To add explainability, we write it as

$$f(\mathbf{x}_i) = \mathbb{E}[f(\mathbf{x})] + \sum_{j=1}^{M} c_{i,j}, \tag{1}$$

where $\mathbb{E}[f(\mathbf{x})]$ is the expected prediction based on all training samples, and $c_{i,j}$ is the contribution of the j-th feature of the i-th sample, which can be positive, negative, or zero. In other words, each $c_{i,j}$ indicates to what extent the j-th feature deviates the i-th sample's prediction from the average. Such an additive decomposition is not as trivial as it might seem, since there are usually complex feature interactions in the prediction, which must be captured to ensure its accuracy.

The SHAP value [16] is proposed as an excellent candidate to compute the $c_{i,j}$s. Specifically, the SHAP value of the j-th feature for the i-th sample equals the expected difference of predictions with and without knowing the feature value $x_{i,j}$, averaged over all orders in which each feature is available to account for feature interactions, formally

$$c_{i,j} = \sum_{S \subseteq F \setminus \{j\}} \left\{ \frac{|S|!(M - |S| - 1)!}{M!} \cdot \{\mathbb{E}[f(\mathbf{x}_i) \mid \mathbf{x}_{i,S\cup\{j\}}] - \mathbb{E}[f(\mathbf{x}_i) \mid \mathbf{x}_{i,S}]\} \right\}, \tag{2}$$

where F is the set of all feature indices, and $\mathbf{x}_{i,S}$ denotes the subvector of $\mathbf{x}_i$ with feature indices in set S.

The exact evaluation of (2) is expensive as the number of terms increases exponentially with the number of features M. Hence, practical implementations in [16] are proposed based on assumptions like feature independence and approximations by sampling, which compromise the accuracy. Even with these strategies, the computation still takes a long time with large numbers of features and training samples as in our problem, making it impractical to be called repeatedly in the physical design flow and during the iterative optimization of routability.

Fortunately, a recent extension [9], referred to as *SHAP tree explainer*, suggests that the *exact* evaluation of SHAP values can be done in polynomial time exclusively for tree-based models (including RF, but not models like SVM and NNs), by exploiting the information stored in the tree structure. Note the SHAP tree explainer does *not* assume feature independence, since feature interactions are already captured in the underlying trees.

In this paper, we adopt the SHAP tree explainer [9] to explain individual predictions made from RF, by looking into the most contributing features in specific DRC hotspots. We validate these explanations by comparing with actual DRC errors and detailed-routed layouts.

IV. Experimental Results

We run experiments in a Linux desktop with an Intel 6-core 2.93 GHz CPU, an Nvidia 1080Ti GPU, and 24 GB memory.

A. Performance of DRC Hotspot Prediction

We compare the predictive performance of RF with all machine learning models applied in previous works that have similar problem formulations to ours [2]–[6], including SVM with RBF kernel (SVM-RBF), random undersampling boosting (RUSBoost) with decision trees as base learners, and feedforward NNs. The inputs to all machine learning models are the 387 normalized features described in Section II-A. Since we have no access to the exact post-route designs used in prior works, it is not possible to take their results directly. Instead, we implement each machine learning model with available information in these papers, train and tune the models with our best effort with cross validation, and evaluate with our dataset. All models are implemented using Python with scikit-learn (except for NNs: with Keras, backed by TensorFlow and GPU acceleration). We do not compare our work with [7] because the performances cannot be compared fairly due to different assumptions on macro movability in placement, as well as significant differences in the amount, scope, and acquisition cost of training data, as mentioned in Section I.

The performance is reported in Table II for 12 designs[3] in terms of TPR^*, $Prec^*$ and A_{prc}, as defined in Section III-B. For example, the RF model shows an average TPR^* of 50.6%, meaning that RF predicts 50.6% of positive samples correctly on average while guaranteeing that $1 - 0.5\% = 99.5\%$ of negative samples are also predicted correctly. Comparing the average performance in Table II, we can find that RF is the best among all models in terms of all three performance metrics. Specifically, it is 21% better in A_{prc} than the popular SVM-RBF model on average and up to 60% better than other models. We highlight in bold the best performances among compared models for each design in Table II. Then for each model and performance metric, we count the number of "winning designs" where the model performs the best among all models. Results show that RF wins the most designs in all three metrics, especially in the main performance metric A_{prc}.

Although the SVM-RBF model performs well in some designs, this model has a fairly large number of parameters, the longest training time (7× more than RF), and the largest number of operations for a single prediction (110× more than RF). This is due to its high model complexity from the RBF kernel, hence the need to store many high dimensional support vectors as parameters and the complex calculations in both

[3]Two other designs (`des_perf_b` and `pci_bridge_b`) are not included because there is no DRC errors and thus these metrics are undefined.

TABLE II
COMPARISON OF PERFORMANCES OF RF AND DIFFERENT MACHINE LEARNING MODELS USED IN PRIOR WORKS [2]–[6]

Design	SVM-RBF [2], [3], [5]			RUSBoost [4]			NN-1 [6]			NN-2			RF (this work)		
	TPR*	Prec*	A_{prc}	TPR*	Prec*	A_{prc}	TPR*	Prec*	A_{prc}	TPR*	Prec*	A_{prc}	TPR*	Prec*	A_{prc}
fft_2	**0.0588**	**0.0588**	**0.1358**	0.0000	0.0000	0.0183	0.0000	0.0000	0.0058	0.0000	0.0000	0.0062	0.0000	0.0000	0.0177
mult_1	**0.3961**	**0.5980**	**0.5248**	0.2143	0.4459	0.2944	0.2143	0.4459	0.3551	0.2857	0.5176	0.4132	0.3442	0.5638	0.4570
mult_2	**0.4767**	**0.6917**	0.5828	0.4041	0.6555	0.4798	0.1865	0.4675	0.3319	0.1658	0.4384	0.2930	0.4352	0.6720	**0.5845**
fft_b	0.1199	0.6809	0.4095	0.0206	0.2683	0.2816	0.0430	0.4340	0.1781	0.0375	0.4000	0.1752	**0.2416**	**0.8113**	**0.4404**
mult_a	0.6923	0.0763	0.2439	0.7692	0.0840	0.4435	0.1538	0.0180	0.0302	0.2308	0.0268	0.0259	**0.8462**	**0.0917**	**0.7445**
mult_b	**0.5498**	**0.7407**	0.6861	0.4633	0.7065	0.6324	0.4274	0.6895	0.5922	0.4356	0.6935	0.6134	0.5269	0.7324	**0.7025**
bridge32_a	**0.8393**	**0.7231**	**0.8703**	**0.8393**	**0.7231**	0.8418	0.7321	0.6949	0.7420	0.6786	0.6786	0.7246	0.7321	0.6949	0.8537
des_perf_1	0.5207	0.9362	0.8947	0.4127	0.9208	0.8129	0.3802	0.9146	0.8427	0.3846	0.9155	0.8488	**0.5459**	**0.9389**	**0.8954**
mult_c	0.5484	0.2194	0.1797	0.7258	0.2711	0.2929	0.6613	0.2531	0.3581	0.7903	0.2882	0.2792	**0.9032**	**0.3164**	**0.7180**
des_perf_a	0.5407	0.7037	0.7313	0.5366	0.7021	0.6740	0.5122	0.6923	0.6957	0.4878	0.6818	0.6309	**0.6748**	**0.7477**	**0.7797**
fft_1	0.1600	0.4706	0.3601	0.0600	0.2500	0.1261	0.0200	0.1000	0.1345	0.0800	0.3077	0.2016	**0.3200**	**0.6400**	**0.5348**
fft_a	**0.5000**	**0.0303**	0.0201	0.0000	0.0000	0.0050	0.0000	0.0000	0.0044	0.0000	0.0000	0.0098	**0.5000**	**0.0303**	**0.1009**
Average	0.4502	0.4941	0.4699	0.3705	0.4189	0.4086	0.2776	0.3925	0.3559	0.2981	0.4123	0.3519	**0.5058**	**0.5200**	**0.5691**
# Win. designs	6	6	3	1	1	0	0	0	0	0	0	0	**7**	**7**	**9**
# Model param.	1252.2k / model			318.5k / model			15.6k / model			15.9k / model			4269.7k / model		
# Prediction op.	3759.7k / sample			24.9k / sample			31.1k / sample			31.8k / sample			34.3k / sample		
Train. CPU time	65.7 min / model			6.9 min / model			24.4 min / model			13.1 min / model			8.9 min / model		
Pred. CPU time	0.24 min / design			0.03 min / design			0.01 min / design			0.01 min / design			0.04 min / design		

training and predicting. These facts also make it difficult to provide explainability with the SVM-RBF model.

The RUSBoost model has 100 iterations of boosting. It has the shortest training time and the fewest operations at prediction time owing to the simplicity of underlying decision trees. However, it is not very powerful in prediction, nor easy to parallelize due to sequential updates of model parameters.

We report two feedforward NN models with different number of hidden layers. NN-1 has a single hidden layer, which is the same architecture as in [6], except that 40 hidden neurons are used in our work as per cross validation. NN-2 has two hidden layers with 40 and 10 neurons, respectively. We use ReLU and sigmoid activations for hidden layers and the output, respectively. They are the simplest but also the least performing models among the five. In fact, we tried feedforwards NNs with more hidden layers with varying number of neurons and found that the predictive performance of feedforward NNs can hardly be improved further by adding more hidden layers beyond two. More complex and deeper NNs like convolutional NNs are not applicable to our dataset. Their complexity could also compromise explainability.

The reported RF model consists of 500 unpruned decision trees. That is why it has the most parameters among compared models. However, owing to the simplicity of decision trees, it does not require many operations in prediction with a negligible predicting runtime. The training time is also fairly short. Furthermore, unlike boosting methods, RF is easy to parallelize for training with more trees, which would not hurt the predicting performance as our cross validation suggests.

B. Explaining Individual Predictions

With the SHAP tree explainer in Python package `shap`, we measure feature contributions to individual RF predictions for hotspots and explain accordingly. Three typical DRC-violated g-cells in two designs are taken as examples in this experiment.

Fig. 3(a)–(c) show the layouts of these example g-cells along with their neighboring cells, with red marks indicating the actual DRC errors (which are not available at prediction and explanation time). The cells, wires and vias are not shown. The colored edges show the GR edge congestion in layers M5 and M4 (a color closer to red indicates higher congestion). The via congestion is not visible. The gray box in (c) indicates a macro, where all wires and vias are blocked. The naming convention, which specifies the type, layer and location of GR congestion, is shown in Fig. 3(d) to relate the feature names and the explanations below. As an example, feature `edM4_4V` is from the **e**dge congestion map. It is the **d**ifference of capacity and load in layer **M4**, on the edge labeled **4V**. Fig. 3(d) shows more examples with corresponding colors.

Based on the RF model, the feature contributions to each hotspot prediction are evaluated and visualized with SHAP in Fig. 4. Recall that the prediction output from the RF model is the probability that the sample is a DRC hotspot, and the threshold of classification is adjustable. Therefore a prediction output is not meaningful unless compared with that of other samples or the base value (i.e. the average). The pink/blue bars represent the positive/negative SHAP values of features, sorted by the absolute value. In other words, the pink/blue bars show how much each feature "pushes" the prediction output higher/lower (i.e. more/less likely to be DRC-violated) from the base value. Since these examples are actual DRC hotspots, the pink bars dominate the prediction and the blue bars can hardly be seen. Referencing the naming convention in Fig. 3(d), we can translate Fig. 4 to the following explanations.

Fig. 4(a) (corresponding to hotspot (a)) shows a lot of features pushing the prediction output from the base value 0.016 to 0.56, making hotspot (a) 35× more likely to be a DRC hotspot than average. The most influential features are the GR edge overflows in layer M5. For example, the fact that `edM5_7H=-4` (i.e. the capacity of edge 7H in layer M5 is 4 tracks less than the load) pushes the output to the positive side by around 0.05, as indicated by the length of the rightmost pink bar. Similar overflows can be found at six other edges, which are all shown in red in Fig. 3(a). Feature value `vlV2_E=35` is also shown in Fig. 4(a), meaning the large load in layer V2

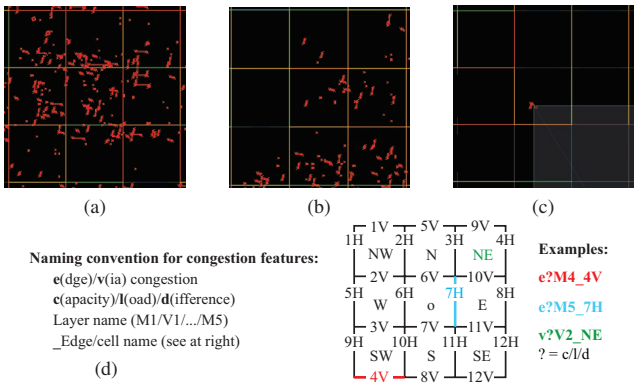

Fig. 3. Example DRC hotspots to be explained. (a) A hotspot in highly congested area from `des_perf_1`. (b) A hotspot with moderate edge congestion from `des_perf_1`. (c) A hotspot near a macro from `matrix_mult_a`. (d) The naming convention for GR congestion features.

in the east neighboring cell also contributes to the DRC errors.

Hotspot (b) is mainly affected by the high via congestion in the north and the central cell in layers V2, as indicated by `vlV2_N=37` and `vlV2_o=29` in Fig. 4(b), and in the northeast cell in layer V3 (`vlV3_NE=24`). The full loads in M4 on edges 6V and 11V (indicated by `edM4_6V=0` and `edM4_11V=0`, corresponding to the two horizontal orange edges in Fig. 3(b)) have secondary contributions.

Hotspot (c) is primarily due to the three edge overflows in M4 (at edges 2V and 3V, the two horizontal red edges in Fig. 3(c)) and M3 (at edge 2H). Other visible elements in the congestion map in Fig. 3(c), such as the blockage of the macro and the overflow in M5 (indicated by the vertical red edge), are not the main reasons for this DRC hotspot.

These explanations are available individually on demand with runtime overhead of 1.4 sec/sample, without requiring actual detailed routing. To verify their validity, we compare them with the actual DRC errors of each example hotspot in Fig. 3(a)–(c) after detailed routing, which are listed below.

(a) 60 errors of different types across metal layers M2 through M5 and via layers V2 through V4.
(b) Two shorts in M2, five end-of-line space errors (EOLs) in M3, two EOLs and a different-net space error in M4.
(c) One short in M3 and one short in M4.

All three explanations are consistent with the actual outcomes. For hotspot (a), a large number of overflows in the neighborhood pose extreme difficulty in routing. For hotspot (c), the explanation perfectly matches the layers where the errors are located. A closer look at the detailed wire and vias of hotspot (b) reveals that the main errors—EOLs in M3—arise due to the dense presence of vias in V2 and V3, which suggests the explanation to hotspot (b) also makes sense.

Notice that hotspots (a) and (b) have totally different explanations despite being from the same design and predicted by the same RF model. For hotspot (c), there is a GR edge overflow in M5 (vertical red edge in Fig. 3(c)) that looks equally important as other overflows, but it is (*correctly*) not reported as a primary contributing feature. These facts suggest that the RF-based explainer does give reasonable, case-by-case explanations which can be beneficial at an early design stage.

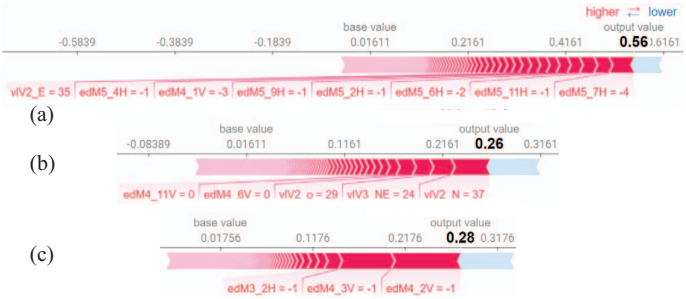

Fig. 4. (a)–(c) Most contributing features for predicting DRC hotspots in Fig. 3(a)–(c), evaluated by the SHAP tree explainer. (The blue regions on the right contain many features that are not visible as the pink bars dominate.)

V. Conclusions

We used RF to predict DRC hotspots at the global routing stage. In terms of AUPRC, a metric tailored for this purpose, RF showed up to 60% better predictive performance on average than machine learning models applied in similar works, with low computational cost. Owing to the transparency of RF, we can further make reasonable and consistent explanations to root cause individual DRC hotspots in an efficient manner. These facts make RF ideal for DRC hotspot prediction.

References

[1] Q. Zhou, X. Wang, Z. Qi, Z. Chen, Q. Zhou, and Y. Cai, "An accurate detailed routing routability prediction model in placement," in *ASQED*, 2015, pp. 119–122.
[2] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, and K. Samadi, "BEOL stack-aware routability prediction from placement using data mining techniques," in *ICCD*, 2016, pp. 41–48.
[3] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *ISPD*, 2017, pp. 15–21.
[4] A. F. Tabrizi, N. K. Darav, L. Rakai, A. Kennings, and L. Behjat, "Detailed routing violation prediction during placement using machine learning," in *VLSI-DAT*, 2017.
[5] L.-C. Chen, C.-C. Huang, Y.-L. Chang, and H.-M. Chen, "A learning-based methodology for routability prediction in placement," in *VLSI-DAT*, 2018.
[6] A. F. Tabrizi, N. K. Darav, S. Xu, L. Rakai, I. Bustany, A. Kennings *et al.*, "A machine learning framework to identify detailed routing short violations from a placed netlist," in *DAC*, 2018, pp. 48:1–48:6.
[7] Z. Xie, Y. Huang, G. Fang, H. Ren, S. Fang, Y. Chen *et al.*, "RouteNet: routability prediction for mixed-size designs using convolutional neural network," in *ICCAD*, 2018.
[8] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, 2001.
[9] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv*, no. 1802.03888, 2018.
[10] J. Westra, P. Groeneveld, T. Yan, and P. H. Madden, "Global routing: metrics, benchmarks, and tools," in *IEEE/DATC Electronic Design Processes Workshop*, 2005.
[11] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *ISPD*, 2015, pp. 157–164.
[12] N. K. Darav, A. Kennings, A. F. Tabrizi, D. Westwick, and L. Behjat, "Eh?Placer: a high-performance modern technology-driven placer," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 3, pp. 37:1–37:27, 2016.
[13] C.-K. Wang, C.-C. Huang, S. S.-Y. Liu, C.-Y. Chin, S.-T. Hu, W.-C. Wu *et al.*, "Closing the gap between global and detailed placement: Techniques for improving routability," in *ISPD*, 2015, pp. 149–156.
[14] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy *et al.*, "GLARE: global and local wiring aware routability evaluation," in *DAC*, 2012, pp. 768–773.
[15] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *ICML*, 2006, pp. 233–240.
[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *NIPS*, 2017, pp. 4765–4774.