PROS: A Plug-in for Routability Optimization applied in the State-of-the-art commercial EDA tool using deep learning

Jingsong Chen
The Chinese University of Hong Kong
jschen@cse.cuhk.edu.hk

Jian Kuang
Cadence Design Systems
jkuang@cadence.com

Guowei Zhao
Cadence Design Systems
guowei@cadence.com

Dennis J.-H. Huang
Cadence Design Systems
dhuang@cadence.com

Evangeline F. Y. Young
The Chinese University of Hong Kong
fyyoung@cse.cuhk.edu.hk

ABSTRACT

Recently the topic of routability optimization with prior knowledge obtained by machine learning techniques has been widely studied. However, limited by the prediction accuracy, the predictors of the existing related works can hardly be applied in a real-world EDA tool without extra runtime overhead for feature preparation. In this paper, we revisit this topic and propose a practical plug-in for routability optimization named PROS which can be applied in the state-of-the-art commercial EDA tool with negligible runtime overhead. PROS consists of an effective fully convolutional network (FCN) based predictor that only utilizes the data from placement result to forecast global routing (GR) congestion and a parameter optimizer that can reasonably adjust GR cost parameters based on prediction result to generate a better GR solution for detailed routing. Experiments on 19 industrial designs in advanced technology node show that PROS can achieve high accuracy of GR congestion prediction and significantly reduce design rule checking (DRC) violations by 11.65% on average.

1 INTRODUCTION

In physical design of integrated circuits (ICs), routing is an essential step where metal wires are layout to connect the pins of all the nets. The objective of routing is to optimize circuit performance while avoiding DRC violations that will prevent a design from being taped out successfully. To solve this optimization problem, EDA tools utilize a two-step method: global routing (GR) and detailed routing (DR). By allocating routing resources to each net properly, the GR tool generates routing plan to guide the DR tool to complete the final routing solution. GR solution is a coarse-grained solution on a data structure named grid graph that divides a design into grid cells (G-cells), consisting of cell-to-cell paths for all the nets. One of the main objectives of GR is to minimize the total wire length of all the nets. However, the amount of routing resources on a design is limited. The competition of routing resources between different nets will result in GR congestion in some parts of the design.

Table 1: Comparison between previous works and PROS: whether the predictor is applied in an EDA tool or not, and if yes whether it will introduce extra runtime overhead for feature preparation.

Works	Applied in an EDA tool?	Introduce extra runtime overhead?
[6]	No	N/A
[7]	No	N/A
[8]	No	N/A
[9]	Yes	Yes
[10]	Yes	Yes
PROS	Yes	No

The quality of a GR solution has a great impact on that of the resulted DR routing solution. Congestion in a GR solution is one of the major causes of DRC violations in the DR solution since most of DRC violations are due to overcrowded wires and vias [1, 2]. As advanced process nodes come, new design rules become even more complex, so it becomes much harder for the DR tool to fix all the DRC violations. Thus, a better GR solution with less congestion is needed to lower the probability of getting DRC violations in advance. To this end, placement engines [3–5] which take routing congestion into consideration are applied, and most GR tools will invoke several rip-up and reroute iterations after the initial routing to further reduce congestion. However, if the initial GR solution is not good and has a lot of congestion, the GR tool can hardly tackle the problem by rip-up and reroute. As the complexity of routing increases, nowadays, EDA tools urgently need to obtain useful prior knowledge before routing to generate better solutions.

Recently, researchers explore machine learning techniques to acquire prior knowledge. Most related works [6–10] aim at predicting the distribution of DRC violations. They utilize various machine learning (ML) techniques: multivariate adaptive regression [6, 10], fully convolutional network [7], learning-based framework [8], and support vector machine [9]. Among them, only the work [8] does not require GR solution as input feature. For others, they needs to perform GR first, which hinders their applications in placement and routing. For the work [8], the prediction precision is low (33%), i.e., there will be a lot of violation false alarms, and their predictor was not integrated into EDA tools. As shown in Table 1, two pieces of

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 14202218).

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415662

work [9, 10] applied their predictors in a placer and a global router respectively. But applying their predictors will incur long extra runtime since GR needs to be invoked first for feature preparation. Therefore, it is impossible to use any of the above predictors in commercial EDA tools as a zero overhead plug-in to reduce DRC violations.

To resolve the above dilemma, we propose a deep learning-based plug-in named PROS for routability optimization which can be embedded into the state-of-the-art commercial EDA tool (Cadence Innovus v20.1) with negligible runtime overhead. In the EDA flow, PROS is placed between placement and GR, which predicts congestion before GR and then uses the prediction result to facilitate a better GR solution with less congestion. By eliminating congestion in the GR solution, PROS can ultimately lead to a reduction of DRC violations in the DR solution, supported by our experimental results in Section 4. The main advantage of PROS is that it can avoid extra runtime overhead of feature preparation since it uses only the data from the placement result but the prediction accuracy is high enough to have impacts on the DR solution. In fact, the topic of congestion estimation in GR has been around for decades and probabilistic approaches [11–14] are commonly used. However, probabilistic estimations are not accurate enough especially for the newly advanced technology node. In contrast, the fully convolutional network (FCN) based predictor in PROS can learn the behavior of a router, giving very accurate predictions. After being well trained, PROS can maintain good prediction performance for other unseen designs in the same technology node.

The main contributions of this paper include:

- We propose a practical plug-in named PROS for routability optimization that can be applied in the state-of-the-art commercial EDA tool with negligible runtime overhead. PROS consists of an FCN-based predictor that requires only data from the placement result to predict GR congestion and a parameter optimizer to optimize GR cost parameters based on the prediction result. Meanwhile, besides the commercial tool used in this paper, PROS can be easily embedded into any other routers as a plug-in.
- To the best of our knowledge, this is the first work that can demonstrate its effectiveness on the state-of-the-art commercial EDA tool and on industrial designs in advanced technology node. Experimental results show that PROS can achieve a high accuracy of GR congestion prediction and lead to a significant reduction (11.65%) of DRC violations.

The remainder of this paper is organized as follows. Section 2 briefly introduces the definition of DRC violation and provides some background knowledge of FCN. Section 3 presents the overall flow of PROS in Section 3.1 and details two main components of PROS: an FCN-based predictor for GR congestion in Section 3.2 and an optimizer for GR cost parameters in Section 3.3. Section 4 presents experimental results, followed by a conclusion in Section 5.

2 PRELIMINARIES

2.1 DRC Violations

Each technology node of IC design defines a set of design rules which are geometric constraints imposed on circuit layouts. After DR, the routing solution will go through a process named design rule checking (DRC). In general, the solution that violates any design rules cannot be taped out successfully. The instances of violated design rules in the solution are called DRC violations. For example, there are two common design rules from [1, 2] as shown in Figure 1, which are simplified for better understanding. The parallel spacing rule indicates that two wires running in parallel on the same metal layer cannot be too close to each other. The end-of-line spacing rule indicates that there is a reserved region at the end of each wire segment which cannot be occupied by any other wires. We can see that most DRC violations are due to overcrowded wires and vias. A GR tool will make great efforts to eliminate GR congestion to make routing easier for the DR tool to avoid DRC violations.

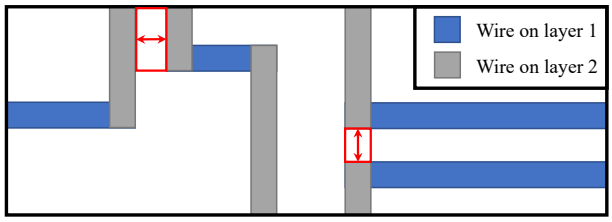

Figure 1: The DRC violations are marked as red boxes. The left one violates the parallel spacing rule. The right one violates the end-of-line spacing rule.

2.2 FCN Background

Nowadays CNN-based models can achieve best performance in several tasks of computer vision. Among these tasks, semantic segmentation is considered a challenge as a per-pixel classification problem in an image. Most of the leading semantic segmentation systems [15–19] utilize fully convolutional networks (FCNs) which is first introduced in [15]. FCNs take an arbitrary input size and produce an output with exactly the same size. GR congestion prediction can also be treated as a pixel-wise binary classification problem (congested or not) on a chip design of arbitrary size. Thus, a FCN-based predictor can be naturally applied in PROS.

Pixel-wise classification needs to handle two issues simultaneously: (1) classification, marking each object in the input with an associated class; (2) localization, detecting the boundary of each object. To this end, many FCNs [15–19] adopt an encoder-decoder framework. In the encoder, by downsampling operators, the depth and spatial dimensions (width and height) of the feature map gradually get deeper and smaller respectively so that the network can capture a long range of information and learn complex representations. In the decoder, the width and height of the feature map are gradually recovered to those of the input by upsampling operators. The predictor of PROS is based on such a encoder-decoder FCN framework as well and we will detail its architecture in Section 3.2.3.

3 THE ARCHITECTURE OF PROS

3.1 Overall Flow

To embed PROS into an EDA tool, first of all, the predictor of PROS needs to be trained on designs of the same technology node. Input

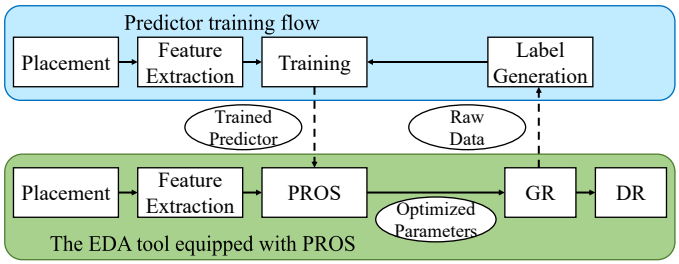

Figure 2: The overall flow

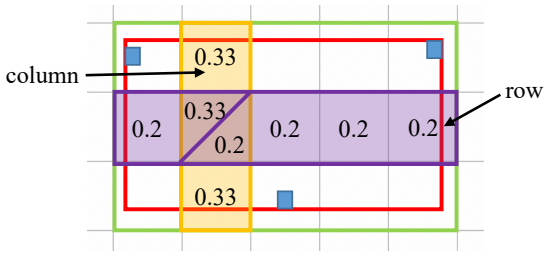

Figure 3: The horizontal/vertical net density of a 3-pin net

features are extracted from placement results, and congestion labels are generated from the raw data of GR solutions. All the features and labels are two-dimensional[1] ($W \times H$) maps with the same width and height as those of the GR grid graph. Congestion labels are binary maps in which one or zero indicates whether a G-cell has congestion issue or not respectively.

After the predictor is well trained, it will be deployed in PROS. PROS is placed between placement and GR in an EDA tool. It will input the features extracted from the placement result of a given design that was unseen by its predictor before and it will predict the congestion map before GR. PROS will then optimize the cost parameters of GR based on the prediction result. With the optimized cost parameters, the GR tool can generate a better GR solution with less congestion and facilitate a reduction of DRC violations in the DR solution.

Regarding the runtime overhead of PROS, since the trained predictor can be reused for any unseen designs in the same technology node, and the steps of feature extraction, prediction, and optimization of GR cost parameters can be performed very quickly like in tens of seconds, the amortized runtime overhead of PROS is negligible. Comparing with the previous works [9, 10] that require GR solution as input feature and thus will introduce extra runtime overhead for feature preparation, PROS is more efficient and practical. Moreover, as PROS does not change the original EDA tool a lot, it can be easily embedded into other routers.

Next, we will detail the predictor of PROS in three aspects: feature extraction in Section 3.2.1, label generation in Section 3.2.2, and the network structure in Section 3.2.3. We will then introduce the parameter optimizer of PROS in Section 3.3.

3.2 FCN-based Predictor for GR Congestion

3.2.1 Feature Extraction. Designing effective features has a great impact on prediction accuracy. The key idea is to extract valuable information from a placement result as much as possible. In general, the distributions of routing resources, cells, and nets have strong correlations with GR congestion. Thus, all the features[2] are designed based on these three distributions. They are described in detail as follows.

- **Horizontal/Vertical track capacity map**: Wires are routed on the routing tracks. Before the routing process, the GR tool will estimate the track capacity in each G-cell. The routing tracks on the same metal layer run in the same direction: horizontal or vertical. We thus have two features for the two directions.
- **Cell density map**: After placement, the cell locations are fixed. This feature is built by calculating the total area covered by the cell(s) in each G-cell.
- **Flip-flop cell density map**: This is similar to the cell density map, but we only count flip-flop cells. Compared to other cells, flip-flop cells usually have higher connectivity. Thus, GR congestion can appear around them with higher probability.
- **Fixed cell density map**: Similarly, we pay special attention to fixed cells which cannot be moved during placement. In many cases, the cell densities around them are high, causing congestion with high chance.
- **Cell pin density map**: This feature is built by counting the number of cell pins in each G-cell.

Algorithm 1 Compute pin accessibility map

Input: #pins and #pin accessing plans of each cell, GR grid graph size $W \times H$.
Output: pin accessibility map P.

1: Create an empty $W \times H$ map P
2: **for** each cell c **do**
3: $npin$ = #pins of c
4: $npat$ = #pin accessing plans of c
5: **for** each pin p of c **do**
6: The location of p : x_p, y_p
7: $P_{x_p,y_p} = P_{x_p,y_p} + \frac{1.5^{npin}}{(npat+1)\times npin}$
8: **return** P

- **Pin accessibility map**: Pin accessibility refers to the difficulty of accessing a pin without DRC violation. To connect a pin that is hard to access, the GR tool needs to detour, possibly causing congestion. In line 4 of Algorithm 1, a pin accessing plan of a cell is a combination of the accessing plans of the nets connecting the pins of this cell. For a cell, the more the pins and the fewer the pin accessing plans it has, the higher its pin accessibility value is, and the more difficult it is to access its pins. The pin accessibility value of a cell will be uniformly distributed to each pin location of the cell.

[1] Originally, some input features and congestion labels are three-dimensional $W \times H \times L$ where L is the number of metal layers of the design. They will be converted to two-dimensional maps by accumulating the values of all the layers.
[2] Among all these features, to our best knowledge, flip-flop cell density map, pin accessibility map, and net density map are first proposed by this paper.

Figure 4: Smoothening process from raw data of GR congestion to congestion label

- **Horizontal/Vertical net density map**: These two features estimate how many nets are expected to go through each G-cell horizontally and vertically. For the 3-pin net in Figure 3 as an example, assume that it will not be routed outside its bounding box (the red box), it will go through some G-cells within the green box. To simplify the calculation, the probability of being routed through by a net will be evenly distributed to each G-cell in a column or a row. Thus, in the horizontal (vertical) direction, each G-cell will get the same net density, one divided by the G-cell number in a column (row). For the net in Figure 3, the horizontal (vertical) net density is 0.33 (0.2). By traversing all the nets in this way, horizontal and vertical net density maps can be generated.

Algorithm 2 Compute small/large-net RUDY map

Input: #pins and the BBox of each small/large net, a mapping table Ra from the #pins to the ratio (total edge length of RSMT / half perimeter of BBox), GR grid graph size $W \times H$.
Output: RUDY map R.

```
Create an empty W × H map R
for each small/large net n do
    npin = #pins of n
    The bounding box of n : x_min, x_max, y_min, y_max
    Density_WL = Ra[npin]×(x_max−x_min+y_max−y_min) / ((x_max−x_min)×(y_max−y_min))
    for x ∈ [x_min, x_max], y ∈ [y_min, y_max] do
        R_{x,y} = R_{x,y} + Density_WL
return R
```

- **Small/Large-net RUDY map**: Besides net density, we use Rectangular Uniform wire DensitY (RUDY) [14] to estimate wire length density. To estimate wire length more accurately, in Algorithm 2, the ratio of the total edge length of an RSMT to the half perimeter of its BBox, which is a pre-computed factor depending on the pin number of a net [20], is used. The RUDY map is constructed by assigning the wire length density ($Density_{WL}$) of every net to all the G-cells in the net's BBox. The $Density_{WL}$ value of a net is equal to the estimated wire length divided by the G-cell number in its BBox. Since nets with different BBox sizes have different impacts on GR congestion, the nets are divide into two groups (small/large) to construct two RUDY maps. A net is classified as small (large) if the half perimeter of its BBox is shorter than (larger than or equal to) a threshold (15 times G-cell size in our setting).
- **Pin RUDY map**: This feature [7] is used to highlight the role of large nets on congestion prediction. It can be regarded as a combination of cell pin density map and large-net RUDY map. It is built by allocating the $Density_{WL}$ of every large net to the G-cells containing any pins of this net.

3.2.2 Label Generation. The raw data of GR congestion collected after global routing for training as labels can hardly be directly used. As shown in Figure 4, the raw data contains several large congestion *regions* with blurred boundaries and undefinable shapes. In the surrounding of the congestion *regions*, there are many randomly scattered and small congestion *spots*. In fact, PROS does not need very detailed congestion map like those provided by raw data, but some clearly defined congestion *regions* are useful instead. To this end, we propose a two-step smoothening process to convert raw data to desirable congestion labels, which can help to make the prediction task easier.

The first step named *insert* will first traverse all the G-cells and if there are at least six congested G-cells out of the eight in the surrounding of a center G-cell g, g will be labeled as congested. This is reasonable since such G-cells have a high chance to be congested. The second step named *clean* will traverse all the G-cells again and if there are no more than three congested G-cells in the surrounding of a center G-cell g, g will be labeled as non-congested. This step can remove scattered and small congestion *spots* and smoothen the boundaries of congestion *regions*. As Figure 4 shows, after performing one *insert* operation and ten *clean* operations, the raw GR congestion data will be converted to a clear congestion labels.

3.2.3 Network Structure. We adopt an encoder-decoder FCN framework as shown in Figure 5 (a). Compared with the publicly available image datasets in the field of computer vision, the dataset available to us is relatively small. Therefore, several techniques, including transfer learning, dilated convolution, and sub-pixel upsampling, were applied to improve the prediction accuracy with limited training data.

The encoder takes an input feature map of dimension $W \times H \times 12$ where W and H are the width and height of the GR grid graph, and 12 is the feature number. We employ a network (ResNet [21]) pretrained on ImageNet [22] as the fundamental building blocks (Res-1~4 in Figure 5 (a)) in the encoder. This technique, called transfer

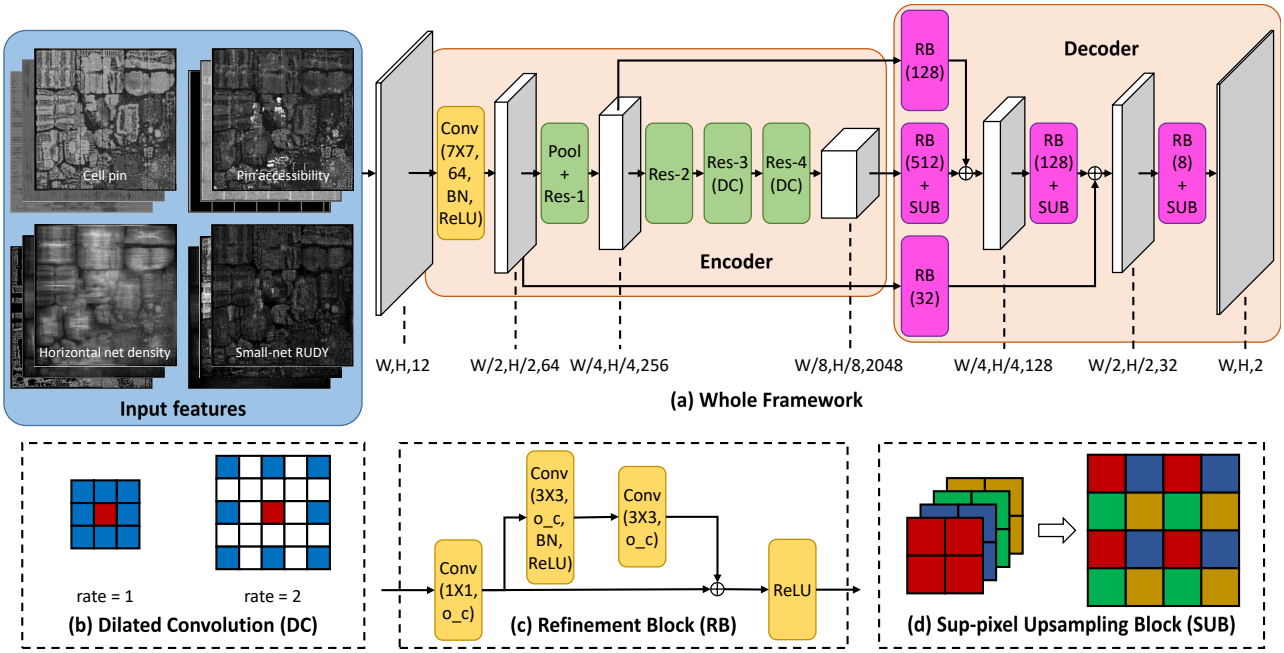

Figure 5: An overview of our prediction model. In (a), "W, H, 12" refers to the width, height, and channel number of the feature map. "Conv (7×7, 64, BN, ReLU)" refers to a convolutional layer with 7×7 kernel size and 64 output channels, followed by batch normalization and an ReLU activation function. "RB (128)" refers to a refinement block with 128 output channels. In (c), "o_c" is the output channel number.

learning, allows the predictor to acquire a strong classification ability from ResNet. The original ResNet will reduce the width and height of the feature map by 32 times through one max pooling and four convolutions with stride. Such framework is powerful for image classification since it can effectively enlarge the *receptive field* which indicates the field visible to the network. However, a reduction of 32 times will cause a massive loss of local information. To handle this issue, we apply dilated convolution [16] in Res-3 and Res-4. As Figure 5 (b) shows, compared to normal convolution (rate = 1), dilated convolution with rate r is constructed by inserting $r-1$ zeros between each value in the kernel. Hence, by setting the strides to 1 and the rates to 2 and 4 respectively in Res-3 and Res-4, the width and height of the feature map will not be reduced while the receptive field will be the same as that of the original ResNet. Dilated convolution will consume more GPU memory, but since the width and height of the input features in our dataset are not large, the GPU memory usage is still acceptable.

In the decoder, we employ three sub-pixel upsampling [23] blocks (SUBs) in cascade to enlarge the feature map. As shown in Figure 5 (d), given an input feature map of dimension $w \times h \times 4c$, a SUB can reshape the input to give an output feature map of dimension $2w \times 2w \times c$. Besides sub-pixel upsampling, there are two other widely-used upsampling methods: bilinear upsampling [15–17] and deconvolution [19]. Compared with bilinear upsampling which is not trainable, sub-pixel upsampling can learn to recover the local information. Compared with deconvolution, sub-pixel upsampling is parameter-free, so it will not significantly increase the training difficulty. After each of the first two SUBs, as shown in Figure 5 (a), a pair of skip connection and addition operator is added, which can retrieve the feature map from the lower layer to further recover the lost local information. In addition, inspired by the residual blocks used in [17, 18], refinement blocks (RBs) are utilized (as shown in Figure 5 (c)) to refine the feature maps and to control the channel numbers. In our model, the number of trainable parameters can be decided by the channel numbers. In general, a larger number of trainable parameters can lead to a stronger prediction ability but a higher training difficulty. Thus, by controlling the channel numbers, RBs can strive a balance between the prediction ability and the training difficulty of our model.

The output feature map of the decoder, of dimension $W \times H \times 2$, will then go through a softmax layer. At last, an element-wise *argmax* operator is applied to generate the final prediction result.

3.3 Optimizer for GR Cost Parameters

Most GR tools are driven by the cost parameters stored in each G-cell. For a typical path searching-based GR tool, paths formed by a sequence of G-cells are constructed step by step. When arriving at a G-cell g, the tool will compute the cost, called *moving cost*, to move to each of its neighboring G-cells and push these costs into a heap. With optimized cost parameters in G-cells, the GR tool can find better paths and allocate the routing resources to each net more smartly.

For example, the method of rip-up and reroute is based on the optimization of cost parameters. To eliminate congestion, rip-up and reroute will increase the cost parameters of the G-cells that are congested in the last few iterations to prevent these G-cells from being used again in the coming routing iterations. Rip-up and

reroute has been well integrated into various routers [1, 2, 24, 25] due to its effectiveness. However, it cannot effectively eliminate congestion when the initial GR solution has too many congestion regions. The advantage of PROS is that it can optimize the cost parameters before the first routing iteration of GR and thus can give a better GR solution with less congestion.

PROS optimizes two types of cost parameters based on the prediction result, including *overflow cost* and *wire/via cost*. They are widely used by path searching-based GR tools. Recall that the predicted GR congestion is a two-dimensional map while the GR grid graph is three-dimensional. PROS will adjust the cost parameters in the projected congestion regions on all layers.

3.3.1 Overflow Cost. As mentioned in Section 3.2.1, the GR tool will estimate the routing track capacity in each G-cell. Usually the track capacity value is less than the track number in the G-cell since the GR tool will reserve a portion of tracks not to be used due to design rule consideration. When the track usage gets close to or even exceeds the track capacity, the GR tool will include an overflow cost when computing the *moving cost*. For the G-cells that are predicted as congested, PROS will increase their number of reserved tracks so that the overflow threshold will be reached earlier in the potential congestion regions. Note that the distribution of routing tracks are not uniform among different layers and the G-cells on lower layers usually have larger track capacities. Thus, PROS will adopt larger increases of reserved tracks on lower metal layers. Actually the degrees of adding reserved tracks should vary for different technology nodes. For the technology node we used, PROS increases the number of reserved tracks by two in each G-cell on the metal layers in the lower half and by one on the metal layers in the upper half.

3.3.2 Wire/Via Cost. The GR tool uses wire/via cost to estimate the cost of wire length. Increasing the wire/via cost is the most direct and effective way to discourage nets going through some congested G-cells. But it can be harmful to increase the wire/via cost in the potentially congested G-cells for all the nets. In Section 3.2.1, the nets are divided into two groups (small/large) according to their bounding box (BBox) sizes. For a small net that has its BBox's half perimeter less than a threshold, its BBox might be entirely included inside a potential congestion region. In this case, the net must use the routing resources in the potential congestion region. Increasing the wire/via cost for small nets may be useless for congestion reduction and it may even increase the wire length or create new congestion due to detour out of the potential congestion region. In contrast, increasing the wire/via cost for large nets can be helpful since they can select another route within its BBox to completely avoid the potential congestion region. In our setting, PROS increases the wire/via cost of the G-cells that are predicted to have congestion by one when routing large nets.

4 EXPERIMENTAL RESULTS

4.1 Setup

As shown in Table 2, our dataset contains 19 different industrial designs in advanced process node. For each design, besides its original placement, we performed placement (using the same commercial EDA tool in all our experiments) on it repeatedly with different parameters to generate around 10 new placements. Thus including the original placement, each design has around 11 different placements. We then utilized rotating (90°, 180°, and 270°) and flipping to increase the dataset size by 8 times. At last, our dataset contains 1664 design cases in total. Routing was then performed on each placement to obtain the raw data of GR congestion and the DR solution for reference used in Section 4.3. The raw data was smoothened by the method described in Section 3.2.2 to generate the congestion labels. The overall average ratio of the congested G-cell number to the total G-cell number is 28.68%.

Table 2: Benchmark information.

Design	#Cells	#Nets	#G-cells	Design	#Cells	#Nets	#G-cells
D1	167176	181585	54522	D11	335284	338225	120409
D2	361361	412425	103362	D12	303640	307033	92416
D3	546702	620054	161202	D13	329811	332729	100489
D4	433485	513620	137641	D14	180507	188887	45796
D5	321245	321561	92416	D15	182673	191049	47306
D6	335765	335537	100489	D16	185026	193383	48620
D7	173769	200318	35721	D17	189352	197674	50625
D8	184129	211627	39601	D18	172699	181045	35721
D9	182924	191282	68121	D19	190198	198448	42230
D10	306045	309462	110889				

For fairness, the design cases generated from the same original design should not be used for training and testing simultaneously since they can be very similar. To this end, 19 original designs are divided into five groups with sizes (4, 4, 4, 4, 3). When the design cases of one group are used for testing, those of the remaining four groups will be used for training. Thus, in our experiments, the design cases for testing are totally unseen for the trained predictor. In order to test all the design cases in each group, such training and testing process will be repeated five times, and each time, the predictor will be re-initialized and re-trained.

The predictor of PROS is implemented in Python with Tensorflow APIs. The remaining parts of PROS like feature extraction and parameter optimizer are implemented in C++. The Adam Optimizer [26] with 0.001 learning rate (0.0001 for the pre-trained ResNet blocks) is utilized to minimize the loss function when training the predictor. Each training process is consisted of 25 training iterations. In each iteration, all the design cases for training are traversed once. We conduct the experiments with eight Intel Xeon CPUs at 2.3 GHz, 61 GB memory, and one NVIDIA Tesla V100 GPU. The time of training a predictor well is around 4~5 hours, and the runtime overhead of PROS (including feature extraction, prediction, and parameter optimization) per design case is less than one minute. Note that the 4~5 hours training time will only be invoked once and the predictor can then be repeatedly used for all other designs.

4.2 GR Congestion Prediction

4.2.1 Evaluation Metrics. We report three metrics: F1 score (F1), false positive rate (FPR), and accuracy (ACC). To calculate them, first of all, we define the following terms for one congestion label map:

- **Positive** (P): The number of congested (positive) G-cells.

Table 3: Comparison on GR congestion prediction.

Model	F1 (%)	FPR (%)	ACC (%)
LR1×1	63.86	25.39	75.25
LR9×9	66.38	16.70	79.67
OneSUB	70.09	10.45	84.23
ThreeSUB-NoSkipAdd	70.87	8.74	85.32
PROS	73.34	8.92	86.15

- **Negative** (*N*): The number of non-congested (negative) G-cells.
- **True Positive/Negative** (*TP/TN*): The number of G-cells that are correctly classified as positive/negative.
- **False Positive** (*FP*): The number of G-cells that are wrongly classified as positive.

With the above terms, the metrics can be computed by:

- **True Positive Rate** (*TPR*) = *TP* / *P*.
- **Precision** (*PRE*) = *TP* / (*TP* + *FP*).
- **F1-score** (*F1*) = 2 × *TPR* × *PRE* / (*TPR* + *PRE*).
- **False Positive Rate** (*FPR*) = *FP* / *N*.
- **Accuracy** (*ACC*) = (*TP* + *TN*) / (*P* + *N*).

As we can see, F1 combines TPR and PRE so that it can quantify the ability of detecting congested G-cells with low error rate. FPR, also known as false alarm, reflects the error rate from the perspective of being over-reacted. Since our dataset is not extremely imbalanced (28.68% positive), we will also use ACC to evaluate the overall classification accuracy.

4.2.2 Baselines for Comparison. In this section, the comparison is only on the prediction accuracy of GR congestion. Note that the ML-based works [6–10] predict DRC distribution, instead of GR congestion, so our predictor results cannot be compared with theirs directly. For the probabilistic methods [11–14], it is also unfair to compare with them since some of them are input features to our predictor.

Similar to the baselines used in [7, 9], for comparison, we implemented a conventional machine learning model — Logistic Regression (LR) in Python. Based on the classic LR model (**LR1×1**), we enhance it by using a window size of 9 × 9 G-cells to capture neighboring G-cell features (**LR9×9**).

To demonstrate the effectiveness of our network structure shown in Figure 5 (a), we created two variants for comparison: (1) replacing three cascaded SUBs by one RB with 128 output channels and one SUB that directly enlarges the feature map 8 times, named **OneSUB**; (2) removing all the skip connections and addition operators, named **ThreeSUB-NoSkipAdd**.

4.2.3 Comparison Results. The prediction results in Table 3 are the overall average of all the design cases when they are used for testing.

Compared to **LR1×1**, **LR9×9** improves F1, FPR, and ACC by 2.52%, 8.69%, and 4.42% respectively. It is because a larger window size can capture more information from the neighboring G-cells so as to improve the prediction performance. But due to the limited ability of learning complex representations of LR models, too large window size will even worsen the prediction result. In our experiments, **LR9×9** reports the best result.

Comparing with **LR9×9**, **OneSUB** has a deeper network structure and stronger classification ability acquired from transfer learning, so it can improve **LR9×9** on F1, FPR, and ACC by 3.71%, 6.25%, and 4.56% respectively. Based on **OneSUB**, **ThreeSUB-NoSkipAdd** is added with three cascaded SUBs and more RBs for a smoother upsampling process that can bring further improvement on F1, FPR, and ACC by 0.78%, 1.71%, and 1.09% respectively. Finally, by retrieving feature maps from the lower network layers to recover lost local information, the ultimate network structure in PROS makes improvement over **ThreeSUB-NoSkipAdd** on F1 and ACC by 2.47% and 0.83% respectively and with a slight degradation of 0.18% on FPR.

4.3 Routing Enhancement

As shown in Figure 2, PROS is integrated into the state-of-the-art commercial EDA tool as a plug-in. The predictor of PROS is well trained as can be seen from the result in Section 4.2. We ensure that the design cases for testing are totally unseen to simulate the scenario when the EDA tool equipped with PROS is used in practice. In Table 4, we compare the routing solution of the original tool (*Orig*) with that of the tool equipped with PROS (*PROS*) in four aspects: *Congested G-cell Ratio*, *#DRC Violations*, *Wire length*, and *Via Count*, which were all reported by the EDA tool itself after detailed routing. The metric *Diff* (%) is equal to $\frac{PROS-Orig}{Orig} \times 100\%$. For *#DRC Violations*, we also show *Diff* (#) that is equal to $PROS - Orig$.

Before a detailed comparison, one more thing to note is that in the experiments of using the state-of-the-art commercial tool (Innovus v20.1), we did not turn off any optimization options like reducing the number of rip-up and reroute steps in GR. Therefore, it is not easy to further improve the routing solutions that were already fully optimized by the tool.

From the leftmost column in Table 4, *Congested G-cell Ratio* is the proportion between the congested G-cell number and the total G-cell number in the GR solution. Note that since *Congested G-cell Ratio* counts on multiple layers, its value is much smaller than the average ratio (28.68%) mentioned in Section 4.1 which stacks the congestion maps of all the layers to one map. Comparing with the original tool, PROS can make an improvement of 3.79% on average.

For the next column – *#DRC Violations*, it is the most important indicator for the performance of PROS. As we can see, with the help of PROS, *#DRC Violations* are reduced on most of the designs. Even for the designs that were worsened, the degree of degradation is relatively minor. On average, the tool equipped with PROS can achieve a significant improvement on *#DRC Violations* by 11.65%. For the last two columns on *Wire length* and *Via Count*, they show a reasonable increase (0.10% and 0.02% respectively) due to some detours for avoiding DRC violations.

To sum up, the experimental results show that PROS can significantly reduce DRC violations by eliminating GR congestion without much increase in wire length or via count. In addition, as discussed in Section 2, the properties of PROS make its running time overhead negligible. PROS is demonstrated as an effective and practical plug-in for routability optimization.

Table 4: Comparison between the original tool (Orig) and the tool equipped with PROS (PROS).

	Congested G-cell Ratio (%)			#DRC Violations				Wire Length			Via Count		
Design	Orig	PROS	Diff (%)	Orig	PROS	Diff (#)	Diff (%)	Orig	PROS	Diff (%)	Orig	PROS	Diff (%)
D1	3.99	3.74	-6.27	40	6	-34	-85.00	1256247	1259854	0.29	1986411	1990470	0.20
D2	3.74	3.78	1.07	62	71	9	14.52	2643585	2644458	0.03	3811903	3808455	-0.09
D3	2.91	2.78	-4.47	266	171	-95	-35.71	4530048	4528118	-0.04	5493976	5491082	-0.05
D4	3.62	3.59	-0.8	59	48	-11	-18.64	3676675	3680954	0.12	4999873	5001267	0.03
D5	7.40	7.25	-2.03	30	39	9	30.00	2767936	2768126	0.01	3883216	3883158	0.00
D6	5.32	5.28	-0.75	31	34	3	9.68	2897940	2896637	-0.04	3958368	3954813	-0.09
D7	11.41	10.99	-3.68	2350	2157	-193	-8.21	891324	893374	0.23	2002840	2003874	0.05
D8	8.83	8.04	-8.95	1251	1306	55	4.40	942272	945013	0.29	2020716	2024463	0.19
D9	3.01	2.91	-3.32	1422	1367	-55	-3.87	956784	957434	0.07	1963423	1961260	-0.11
D10	4.52	4.25	-5.97	453	355	-98	-21.63	2986237	2986843	0.02	4375923	4369190	-0.15
D11	5.48	5.11	-6.75	1105	1022	-83	-7.51	3222854	3223323	0.01	4655382	4651726	-0.08
D12	6.05	6.18	2.15	628	603	-25	-3.98	3081506	3084930	0.11	4379969	4380303	0.01
D13	5.40	5.31	-1.67	65	57	-8	-12.31	3172436	3172003	-0.01	4532385	4534255	0.04
D14	5.35	5.13	-4.11	793	813	20	2.52	944020	943940	-0.01	2240962	2239282	-0.07
D15	4.57	4.44	-2.84	870	722	-148	-17.01	963213	962794	-0.04	2245805	2246101	0.01
D16	4.45	4.19	-5.84	636	502	-134	-21.07	973894	973808	-0.01	2252314	2254554	0.10
D17	4.28	3.94	-7.94	581	556	-25	-4.30	997167	998133	0.10	2277707	2281422	0.16
D18	8.38	7.91	-5.61	267	157	-110	-41.20	898495	904184	0.63	2150738	2153482	0.13
D19	6.09	5.83	-4.27	879	862	-17	-1.93	1001841	1003276	0.14	2255979	2256639	0.03
Avg			-3.79			-49.47	-11.65			0.10			0.02

5 CONCLUSION

In this paper, we propose a practical plug-in named PROS for routability optimization which can be integrated into the state-of-the-art commercial EDA tool with negligible runtime overhead. PROS utilizes the feature information from placement result only to forecast congestion by a FCN-based predictor before GR and then optimizes the cost parameters in the GR tool based on the prediction result. With optimized parameters, the congestion in GR solution gets less, which facilitates the reduction of DRC violations. The significant reduction of DRC violations on industrial designs in advanced process node demonstrates the effectiveness of PROS. In the future, besides routability optimization, we will develop PROS in other directions like timing and power optimization during GR.

REFERENCES

[1] G. Chen, C.-W. Pui, H. Li, J. Chen, B. Jiang, and E. F. Young, "Detailed routing by sparse grid graph and minimum-area-captured path search," in *Proc. ASP-DAC*, 2019.

[2] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Young, "Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *Proc. ICCAD*, 2019.

[3] X. He, T. Huang, W.-K. Chow, J. Kuang, K.-C. Lam, W. Cai, and E. F. Young, "Ripple 2.0: High quality routability-driven placement via global router integration," in *Proc. DAC*, 2013.

[4] C.-W. Pui, G. Chen, W.-K. Chow, K.-C. Lam, J. Kuang, P. Tu, H. Zhang, E. F. Young, and B. Yu, "Ripplefpga: A routability-driven placement for large-scale heterogeneous fpgas," in *Proc. ICCAD*, 2016.

[5] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "Ntuplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE TCAD*, 2014.

[6] Q. Zhou *et al.*, "An accurate detailed routing routability prediction model in placement," in *Proc. ASQED*, 2015.

[7] Z. Xie *et al.*, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. ICCAD*, 2018.

[8] A. F. Tabrizi *et al.*, "Eh? predictor: A deep learning framework to identify detailed routing short violations from a placed netlist," *IEEE TCAD*, 2019.

[9] W.-T. J. Chan *et al.*, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *Proc. ISPD*, 2017.

[10] Z. Qi *et al.*, "Accurate prediction of detailed routing congestion using supervised data learning," in *Proc. ICCD*, 2014.

[11] J. Lou, S. Thakur, S. Krishnamoorthy, and H. S. Sheng, "Estimating routing congestion using probabilistic analysis," *IEEE TCAD*, 2002.

[12] R. T. Hadsell and P. H. Madden, "Improved global routing through congestion estimation," in *Proc. DAC*, 2003.

[13] J. Westra, C. Bartels, and P. Groeneveld, "Probabilistic congestion prediction," in *Proc. ISPD*, 2004.

[14] P. Spindler *et al.*, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proc. DATE*, 2007.

[15] J. Long *et al.*, "Fully convolutional networks for semantic segmentation," in *Proc. CVPR*, 2015.

[16] L.-C. Chen *et al.*, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proc. ECCV*, 2018.

[17] G. Lin *et al.*, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," in *Proc. CVPR*, 2017.

[18] C. Yu *et al.*, "Learning a discriminative feature network for semantic segmentation," in *Proc. CVPR*, 2018.

[19] H. Noh *et al.*, "Learning deconvolution network for semantic segmentation," in *Proc. ICCV*, 2015.

[20] A. E. Caldwell *et al.*, "On wirelength estimations for row-based placement," *IEEE TCAD*, 1999.

[21] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.

[22] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, 2015.

[23] W. Shi *et al.*, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proc. CVPR*, 2016.

[24] C.-H. Hsu, S.-C. Hung, H. Chen, F.-K. Sun, and Y.-W. Chang, "A dag-based algorithm for obstacle-aware topology-matching on-track bus routing," *IEEE TCAD*, 2020.

[25] J. Chen, J. Liu, G. Chen, D. Zheng, and E. F. Young, "March: Maze routing under a concurrent and hierarchical scheme for buses," in *Proc. DAC*, 2019.

[26] D. P. Kingma *et al.*, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015.