A Machine Learning Framework to Identify Detailed Routing Short Violations from a Placed Netlist

Aysa Fakheri Tabrizi
University of Calgary
Calgary, Canada
afakheri@ucalgary.ca

Nima Karimpour Darav
Microsemi Corp.
Kitchener, Canada
nkarimpo@microsemi.com

Shuchang Xu
University of Calgary
Calgary, Canada
xsc14thu@foxmail.com

Logan Rakai
University of Calgary
Calgary, Canada
lmrakai@ucalgary.ca

Ismail Bustany
Xilinx Inc.
San jose, CA
ismailb@xilinx.com

Andrew Kennings
University of Waterloo
Waterloo, Canada
akennings@uwaterloo.ca

Laleh Behjat
University of Calgary
Calgary, Canada
laleh@ucalgary.ca

ABSTRACT

Detecting and preventing routing violations has become a critical issue in physical design, especially in the early stages. Lack of correlation between global and detailed routing congestion estimations and the long runtime required to frequently consult a global router adds to the problem. In this paper, we propose a machine learning framework to predict detailed routing short violations from a placed netlist. Factors contributing to routing violations are determined and a supervised neural network model is implemented to detect these violations. Experimental results show that the proposed method is able to predict on average 90% of the shorts with only 7% false alarms and considerably reduced computational time.

CCS CONCEPTS

• **Hardware** → **Placement**; **Wire routing**; • **Computing methodologies** → *Supervised learning by classification*;

KEYWORDS

Design automation, physical design, routing, placement, data mining, machine learning, imbalanced data

1 INTRODUCTION

Routability of a design is one of the most challenging issues in Very Large Scale Integrated Circuits (VLSI) physical design. Detailed routing is very complex and time consuming. Advances in technology node and increasing design rules add to this complexity. Occurrences of violations such as shorts, pin access problems, and other detailed routing violations at the routing stage require remedial procedures such as rip up and reroute process and in some cases the placement and routing stages have to be repeated. This process can take up to several days. Therefore, detecting and preventing routing violations in the early part of the physical design flow, preferably before the routing stage, has become critical in reducing the design time and the possibility of failure.

Conventionally, global routers are used to estimate and evaluate the routability of a placement solution during the placement process. Global routers ignore the effects of local nets, but local nets contribute a high percentage of the total nets and can affect the quality of the routing [12, 14, 18]. There are few methods that take into account local nets during global routing [24]. However, invoking a global router several times is time consuming. In addition, there are no models that directly predict the violations and all the models work based on estimation of congestion. Hence, there is a growing gap between global routers' estimated congestion and the actual routing violations.

In this paper, we propose a machine learning (ML) framework to detect the short violations from a placed netlist without using a global router. The proposed model learns from the actual detailed routing shorts of the routed designs and predicts their occurrence in new designs. We have determined the factors that can contribute to the routing violations and extracted the relevant features at the placement stage. Experimental results show that the proposed detection method is able to predict on average 90% of the shorts on previously unseen data with only 7% false alarms and considerably reduce the computational time. Our main contributions include:

- Proposing a machine learning model to predict the shorts.
- Defining and extracting effective features after placement.
- Predicting the short violations using the defined features with an ML approach compatible with imbalanced data.
- Eliminating the need of a global router for predicting shorts.

We have validated our proposed framework on advanced designs by empirical experiments, and have shown improvement on predicting shorts violations. The rest of this paper is organized

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3195975

as follows. In Section 2, background related to routing analysis in placement, and machine learning are reviewed. The proposed framework is presented in Section 3. The results are discussed in Section 4. Finally, Section 5 concludes the paper.

2 PRELIMINARY

2.1 Placement and Routing

Congestion analysis is performed to identify violation prone regions in a design before detailed routing. Global routers such as [4, 28] are mainly used for congestion estimation. The effect of local nets are not considered in global routing based estimators. Although some modified global routers consider the local nets by modeling them, the runtime increases significantly [24]. Works presented in [16, 17, 24] estimate local net utilization by methods such as incorporating Steiner tree wire length estimations and pin density measurements in a global routing based congestion analysis. These estimations are used along with a global router to estimate the overall congestion that has a long runtime. Others use probabilistic congestion estimation techniques [10, 25], which can be highly inaccurate [26]. There are works that aim to detect detailed routing violations by supervised learning [3, 13, 30]. These techniques use the global routing results which can be time consuming and unreliable. The work in [19] also uses supervised learning with limited features which can be inaccurate. In [4, 28], to assess the routability of a design, its overflow is calculated and a congestion plot is generated for each global cell (gcell). A gcell with overflow might be routable because routing resources in neighboring gcells can be used [24]. In addition to the overflow, a routing congestion plot is generated to assess the congestion in the neighbouring gcells.

2.2 Supervised Machine Learning Techniques and Their Metrics

Supervised machine learning refers to the techniques used for developing functions from labeled data that can be used to label new data [4]. In a supervised machine learning technique, first a training set is developed and its labels are determined. The next step is to determine the features that would represent the learned functions. These features are essentially the most important part of a machine learning process. Once the features are determined, the type of the learned function is determined. Some of the most well-known examples of the models are decision trees, support vector machines, and neural networks (NN). A learning algorithm is then performed on the training set and the learned functions are obtained. Finally, the performance of the learned functions are evaluated using a separate test set. Data mining and machine learning techniques have been applied to electronic design automation in recent years [22]. In [21, 23, 29] machine learning is used for lithography, hot-spot detection and timing estimation.

Imbalanced data classification problem is a supervised learning problem where the proportion of the number of data in classes is skewed. In a binary imbalanced data classification problem, the majority of data belongs to one of the classes [5, 6]. The nature of our data is imbalanced with less than 1% of the data belonging to the positive class. Hence, evaluating our model using the traditional metrics for classification such as accuracy is insufficient and can be misleading. In order to evaluate our model's performance, the metrics of imbalance cases are used. In the following a brief summary of these metrics are given.

Confusion Matrix: A confusion matrix is a table that presents the performance of a classifier and is widely used to assess imbalanced data. The rows of a confusion matrix present actual classes while columns show the predictions. The schematic of a confusion matrix is shown in Table 1. The confusion matrix presents the following four cases:

- **True Positive (*TP*/*TN*)**: The number of instances that are correctly classified as positive/negative.
- **False Positive (*FP*/*FN*)**: The number of instances that are negative/positive and incorrectly classified as positive/negative.

Table 1: Confusion Matrix

		Prediction		Total
		Negative	Positive	
Actual	Negative	TN	FP	N
	Positive	FN	TP	P
				Instances

Sensitivity or True Positive Rate (*TPR*): Shows the ability of the model to classify the positive class. In our problem, *TPR* presents the percentage of the tiles, a rectangular area of a placed design, with short violations that are correctly identified as having violation. Sensitivity is defined by: $TPR = \frac{TP}{P} = \frac{TP}{TP+FN} \times 100$.

Specificity (*SPC*): Measures the ability of the model to classify the negative class. In our problem, *SPC* presents the percentage of normal tiles that are correctly identified as not having violation. Specificity is defined by: $SPC = \frac{TN}{TN+FP} \times 100$.

False Alarm (*FP*): Describes cases where the developed function mistakenly identifies an event. In our problem, *FP* describes the tiles with no shorts that are incorrectly identified as having shorts. It is defined as: $FP = \frac{FP}{TN+FP}$.

Accuracy (*ACC*): Shows the overall performance of the classifier and is defined by: $ACC = \frac{TP+TN}{All}$. *All* is the number of instances.

Matthews Correlation Coefficient (*MCC*): Is a metric that is used for binary classification of imbalanced data. *MCC* returns a value between -1 and +1. An *MCC* of +1 represents perfect prediction. *MCC* is calculated as: $MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$.

3 PROPOSED METHODOLOGY

3.1 Framework Overview

We have formulated the violation detection task as a binary classification problem with imbalanced data, where the input is a tile t_i and the output is a binary label, $p_i \in \{0, 1\}$, indicating the absence or presence of violation. The flow of our framework for violation prediction and its integration in physical design flow is presented in Figure 1. In this figure, the blocks represent the processes and the clouds represent the inputs and the outputs. The solid arrows represent the common paths for training and prediction. The dashed arrows are exclusive to the training flow.

In training the model, first a design is placed by a placer. Then, using a grid similar to global routing grid, the layout of the placed design is divided into non-overlapping rectangular areas, tiles. The

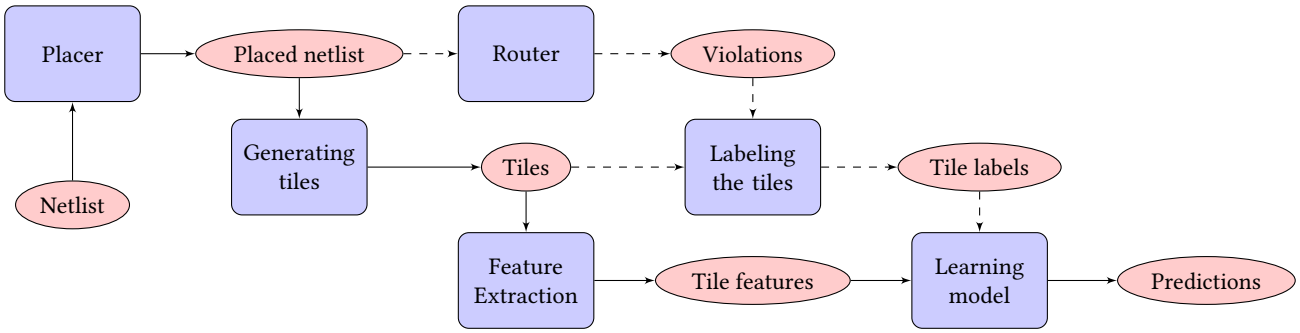


Figure 1: The flow of the proposed violation detection framework and its integration in physical design flow

tiles that completely overlap the macros are excluded from the set. The features of the tiles are extracted in the feature extraction process. The placed netlist is given to a router to be routed and the violations along with their locations are extracted. The tiles are labeled using the location of the violations in the tile labeling step. The tiles that have shorts reported by detailed router are labeled as positive instances. All other tiles are labeled as negative instances.

The tile features and labels are fed to the learning model to train the system. Once the model is trained, similar tiles are generated from different designs and their features are extracted. Then, the trained model decides if there will be a violation in the tile or not.

3.2 Feature Extraction

Features can have a high impact on the performance of the machine learning algorithm. In this research, factors that significantly contribute to violations are determined through extensive experimentation. These features are then formulated and included in the feature set. The list of features that are extracted and used in this framework is presented in the following. All the following features, except for the first two, are calculated for each individual tile and its neighborhood. The neighborhood of a tile is defined as the main tile and its eight surrounding tiles. The area of a neighborhood is nine times larger than the tile.

Relative location of the tile in the design: Tiles located in the center of a design are prone to become congested as more global nets crossing over the area.

Design Cell utilization: The density of a design affects the performance of both the placer and the router. Hence, design cell utilization is included as a global feature in the design.

Routing accessibility: This feature relates to the proportion of the tile area covered by macros, placement blockages, and routing blockages in different metal layers (M2:M9).

Number of local nets: Number of the nets that have at least two pins in the tile.

Number of global nets: Number of the nets that have at least one pin in the tile and at least one pin outside the tile.

Narrow channels effects: This features captures the effects of narrow channels and is an estimate of the number of the nets passing over the tile horizontally/vertically. It is computed as follows: For each column/row of the constructed tile grid, the number of the nets having at least one pin on both sides of the column/row is calculated and divided by the number of the tiles in that column/row.

Pin distribution in the tile: This feature is computed as the standard deviation of the x and y locations of the pins in the tile.

Clock Network Effect: This feature represents the effect of the clock network during routing and is calculated as the number of the clock pins in the tile.

Track availability: This feature is an estimation of number of the maximum horizontal/vertical track using line scan algorithm [20].

Non-default Routing nets: This feature represents the effects of Non-Default Routing (NDR) nets, and is equal to the number of the pins with NDR nets.

The mentioned feature set has been selected from a set of candidate features. The selection was based on which feature would have significant impact on detailed routing violations. Some of these features have direct impact on the presence of these violations, where others do not have a direct correlation with violations but the occurrence of a few features at the same instance can affect the routability. It should also be mentioned that before inputting the instances into the network, the features were normalized based on the mean and standard deviation of each feature in the data set.

3.3 Learning Model

As No Free Lunch theorem [27] states, there is no one model that works best for all the problems. Several machine learning models were tried in order to find the best model. The main selection criteria was to use a model that could consider imbalanced data.

Models that can take into account the imbalance data or can be adopted to do that were tested to determine their suitability for the data set. After trying different models with different configurations, NN was found to be the best match as the main learning model. However, neural networks are not tailor-made for imbalance data. Therefor, the neural network model was modified by assigning weights to the instances of minority class in the loss function. This modification enabled the network to take into account the skew in data. Finally, a weighted neural network with one hidden layer consisting of 20 nodes in the hidden layer was selected and tuned.

4 EXPERIMENTAL SETUP AND RESULTS

We have assessed the efficacy of our proposed framework by applying it to the ISPD15 routability driven benchmark set designs [2]. The benchmark circuits are placed and routed by Eh?Placer [8], and Mentor Graphics Olympus router [11], respectively. Eh?Placer is an academic placer that achieved 2nd place in the ISPD 2015 contest

Table 2: Benchmark Circuits and their charachteristics

Design	# Macros	# Cells k	#Nets k	FR	# IO k	AU SC	AU Macro	Density limit (%)
perf_1	0	113	113	0	0.4	91	91	91
perf_a	4	108	110	4	0.4	57	72	57
perf_b	0	113	113	12	0.4	56	50	56
dist_a	6	127	131	1	2.6	54	62	54
fft_1	0	32	33	0	3.0	84	84	84
fft_2	0	32	33	0	3.0	50	50	65
fft_a	6	31	32	0	3.0	29	74	50
fft_b	6	31	32	0	3.0	31	74	60
mm_1	0	155	159	0	4.8	80	80	80
mm_a	5	150	154	0	4.8	45	77	60
mm_b	7	146	152	3	4.8	34	73	60
b32_a	4	30	30	4	0.4	64	51	64
b32_b	6	29	29	3	0.4	27	51	27
sb11_a	1.5k	926	936	4	27.4	35	73	65
sb12	89	1,287	1,293	0	5.9	44	57	65
sb16_a	419	680	697	2	17.5	50	74	55

[7]. In this section, first we review the benchmark set characteristics. Then, we present the details of the implementation of our model. Finally, we present the prediction results and comparisons.

4.1 Implementation Details

The data set is generated from the ISPD 2015 benchmark designs [2]. The characteristics of these benchmark circuits are given in Table 2.

In this section the details of extracting features, generating training and test sets and setting the learning model hyperparameters are given to enable the reader to regenerate the codes.

Determining the grid size: The size of a grid was determined empirically. Since we have a limited number of shorts in our data set, considering a big grid size results in absorption of multiple shorts in one tile and having less positive instances for training. On the other hand, the small grid size results in noise in data and the data extracted from such a small grid are not meaningful. We tried different sizes and selected 3 rows by 3 rows as the most suitable grid size.

Generating training and test sets: In order to train and evaluate our model, the instances are divided into training set and test set. In dividing the tiles to training and test sets, 80% of the instances of each design, except *mgc_fft_2*, were randomly selected for training and the remaining 20% of instances are assigned to the test set. Design *mgc_fft_2* is completely excluded from training as an example to observe the performance of the tool on totally unseen circuits. All the instances of this design are assigned to test set.

It should be noted that *mgc_edit_dist_a* and *mgc_superblue16a* designs are not included in the training and test sets. they could not be detailed routed due to high global routing congestion.

Setting learning model parameters: Our NN model is developed in Python using tensorflow [1]. We have modeled the problem as a binary classification with one output node. One hidden layer is considered for this model. The number of nodes in the hidden layer is set to 20. For minimizing the loss function of NN we have used Adam Optimizer with the learning rate 0.25 [9]. It should be noted that gradient descent is too slow and mini-batch methods are not good choice for this problem due to the imbalance in data.

Table 3: Confusion matrix for the training and the test set

Training		Prediction		Total
		N	P	
Actual	N	174787 (% 94)	11979	186766
	P	216	3310 (% 94)	3526
				190292
Test		Prediction		Total
		N	P	
Actual	N	46264 (% 93)	3648	49912
	P	90	829 (% 90)	919
				50831

The number of iterations is set to 3000. In order to set this parameter, the change in loss function is monitored in several experiments and the number of iterations is selected. Since our data is imbalanced, we have assigned weights to positive instances in calculation of loss function. The weight parameter is set to 20. That is to say the model penalizes the miss-classification of positive instances by 20 times more than negative instances.

The training is performed 25 times and the model with lowest loss value in training is selected as our predictor.

4.2 Evaluation Results

Our proposed model is trained using the training data described in Section 4.1. The performance of the trained model is evaluated on both the training and test data sets and the results are summarized in Table 3 as confusion matrices. These matrices show that the model is able to detect 94% of the shorts with 6% false alarms in training data, and 90% of the shorts with only 7% false alarms on the previously unseen instances.

The performance of the proposed model on instances of all designs, as well as individual designs, is presented in Table 4. In columns 1-8, the design name, the total number of instances, the total number of positive instances i.e. tiles with shorts, the total number of negative instances i.e. the number of tiles without any shorts, correctly predicted shorts, correctly predicted as normal instances, false alarm i.e. the number of normal tiles incorrectly predicted as tiles with shorts, and the number of tiles with shorts that are not detected are presented, respectively. The last four columns show the *TPR*, *SPC*, *ACC*, and *MCC* metrics, respectively. The proposed method is able to detect 93% of all the tiles with shorts with only 7% false alarm rate i.e. 93% specificity.

To be able to better describe the results, the circuits in Table 4 are ordered based on their routability, starting from most unroutable design. This is done by ordering the designs in the descending order by the number of their positive instances or the number of tiles with shorts. Then, we divided them into three groups, designs with more than 500 shorts, designs with shorts between 100 to 500, and designs with less than 100 shorts.

The results show that our predictor is highly successful in predicting the shorts of designs with high number of shorts with an average of 96%. Average sensitivity for the designs with shorts

Table 4: Prediction results of proposed framework on ISPD 2015 benchmark designs

Design	Instances	P	N	TP	TN	FP	FN	$TPR(\%)$	$SPC(\%)$	$ACC(\%)$	$MCC([-1:1])$
All data	241123	4445	236678	4139	221051	15627	306	93	93	93	0.42
# Shorts > 500											
mgc_des_perf_a	11452	1394	10058	1348	5770	4288	46	97	57	62	0.35
mgc_fft_b	5771	853	4918	810	2373	2545	43	95	48	55	0.31
mgc_fft_a	6491	696	5795	668	2885	2910	28	96	50	55	0.28
mgc_des_perf_1	5476	617	4859	589	4147	712	28	95	85	86	0.60
Average	7298	890	6408	854	3794	2614	36	96	60	65	0.39
100 < # Shorts < 500											
mgc_matrix_mult_b	21433	429	21004	349	19737	1267	80	81	94	94	0.40
mgc_pci_bridge32_a	3569	163	3406	155	2697	709	8	95	79	80	0.36
mgc_superblue12	66010	113	65897	97	65535	362	16	86	99	99	0.42
Average	30337	235	30102	200	29323	779	35	87	91	91	0.40
# Shorts < 100											
mgc_matrix_mult_1	8281	76	8205	52	7535	670	24	68	92	92	0.20
mgc_fft_1	1936	39	1897	37	1651	246	2	95	87	87	0.33
mgc_matrix_mult_a	16512	33	16479	5	16194	285	28	15	98	98	0.05
mgc_fft_2	3249	16	3233	16	3072	161	0	100	95	95	0.29
mgc_superblue11_a	71152	12	71140	10	71079	61	2	83	100	100	0.34
mgc_pci_bridge32_b	9791	4	9787	3	8379	1408	1	75	86	86	0.03
mgc_des_perf_b	10000	0	10000	0	9997	3	0	NA	100	100	NA
Average	18487	30	18457	21	17985	472	10	73	93	93	0.21

between 100 and 500 is 87%, and for the design with less than 100 shorts is 73%.

It is very important to obtain a good sensitivity in designs with high number of shorts, as these are the designs that are actually unroutable and detecting the shorts in the placement stage gives the opportunity to plan for avoiding them. There are a few designs with low rate of sensitivity in designs with low number of shorts. The reason is that missing only a few shorts in such a small number of shorts results in big decrease in the percentage. However, these designs are considered routable and the missing shorts can be easily fixed during the routing stage.

According to the table, the specificities of the designs with low number of shorts are high. That means using this predictor as a guide at the placement stage will not result in a large number of changes in the designs that are already routable. There are a few designs with lower rate. These are the design with shorts scattered throughout the circuit and although some of the predicted tiles are not actual shorts, but they are surrounded with actual shorts. This is shown with an example in Figure 2. In this Figure the shorts predicted by the proposed framework for design *mgc_fft_a* is visually compared to an industrial global router congestion estimation map and actual detailed routing shorts.

In order to evaluate the efficiency of proposed framework using the proposed learning model, in Table 5, we compared its performance to the performance of same framework using RUSBoost ensemble ML method [15] and a method that uses different set of features and uses RUSBoost learning model [19].

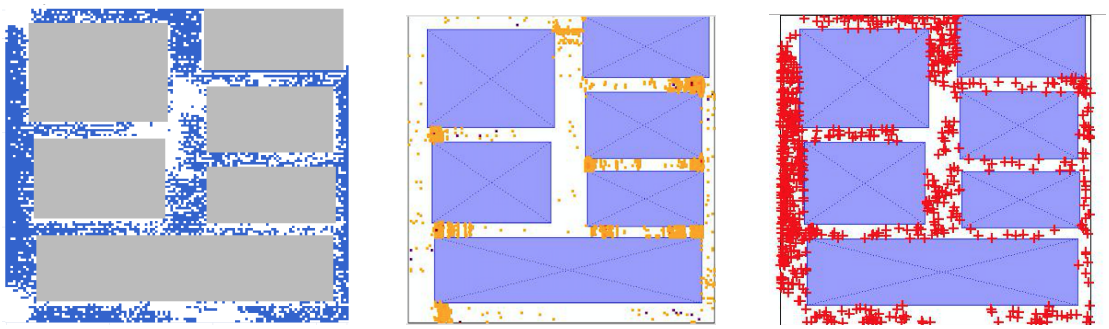
In Table 5, first, we have compared the *TPR* (*T*), *SPC* (*S*), and *MCC* (*M*) of the proposed method to those of the RUSBoost model. It can be seen that the proposed model outperforms RUSBoost for the majority of the designs in terms of *MCC* as a single metric and can maintain a better tradeoff between sensitivity and specificity. We have observed that since neural network is based on combining the features and uses all the features, it can be more accurate in such problems where there are no dominant features and the combination of features are the most important factor. Where as RUSBoost is based on decision trees that do not use the features which it does not find useful.

The next set of columns show the prediction results of the method in [19]. Again, it can be observed that the proposed method outperforms this method by comparing the *MCC* metric. Since the tiling method of the two methods are different, the *SPC* values for designs with macros are not comparable. In calculation of the *SPC* of the proposed model the area used by macros are excluded.

The total runtime spent for training in this experiment using the proposed method is 980 seconds and the prediction time is less than 10 seconds. Training is a one-time task, and once the model is trained the only runtime added to the placer is a few seconds of prediction time. This can save hours of prediction in placement by invoking the global router several times or going back and forth between placement and routing. The model can be updated by adding new training data to the system and retraining the system to achieve even better prediction.

5 CONCLUSION

In this paper, a machine learning framework to predict detailed routing short violations from a placed netlist is presented. The significant contributions of the work are the determination of a rich and diverse feature set, design of a learning model that is capable of handling imbalanced data sets, and development of a weighted neural network model which consists of 20 nodes in the hidden layer. Experimental results show that the proposed method is able to predict on average 90% of the shorts with only 7% false alarms and considerably reduced computational time.



(a) Predicted shorts (b) Global router congestion map (c) Actual detailed routing shorts

Figure 2: Visual comparison of the predicted shorts by the proposed framework with an industrial global routing based congestion estimator and actual detailed routing short violations for design *mgc_fft_a*

Table 5: Comparison of the results of the proposed framework with a different learning method and [19]

Design	Proposed Framework			RUSBoost			RUSBoost less features		
	T	*S*	*M*	*T*	*S*	*M*	*T*	*S*	*M*
	%	%	%	%	%	%	%	%	%
perf_1	95	85	**60**	100	15	14	96	53	21
perf_a	97	57	35	95	58	35	96	86	**37**
perf_b	NA	100	NA	NA	98	NA	NA	71	NA
fft_1	95	87	**33**	90	85	28	78	78	10
fft_2	100	95	**29**	100	74	12	90	47	2
fft_a	96	50	**28**	100	15	14	77	84	11
fft_b	95	48	**31**	100	17	17	82	83	17
multi_1	68	92	**20**	50	95	19	81	50	3
multi_a	15	98	**5**	6	99	2	30	96	1
multi_b	81	94	**40**	79	88	28	78	95	17
B32_a	95	79	**36**	39	91	20	64	73	11
B32_b	75	86	3	25	99	**7**	100	85	2
SB11_a	83	100	**34**	50	100	34	69	81	1
SB12	86	99	**42**	53	100	29	55	65	1

6 ACKNOWLEDGEMENTS

This work was supported by Natural Sciences and Engineering Council of Canada Discovery Grant program and donation by Mentor Graphics. Canadian Microelectronics Corporations and Compute Canada provided computing support.

REFERENCES

[1] M. Abadi and et all. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.
[2] I. Bustany, D. Chinnery, J. Shinnerl, and V. Yutsi. 2015. ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In *ISPD '15*. 157–164.
[3] W. J. Chan, P. Ho, A. B. Kahng, and P. Saxena. 2017. Routability Optimization for Industrial Designs at Sub-14Nm Process Nodes Using Machine Learning. In *ISPD '17*. 15–21.
[4] Y. Chang, Y. Lee, and T. Wang. 2008. NTHU-Route 2.0: A Fast and Stable Global Router. In *ICCAD '08*. 338–343.
[5] N. Chawla, N. Japkowicz, and A. Kotcz. 2004. Editorial: Special Issue on Learning from Imbalanced Data Sets. *SIGKDD Explor. Newsl.* 6, 1 (June 2004), 1–6.
[6] H. He and E. A. Garcia. 2009. Learning from Imbalanced Data. *IEEE Trans. on Knowl. and Data Eng.* 21, 9 (Sept. 2009), 1263–1284.
[7] ISPD15. 2015. ISPD 2015 Blockage-Aware Detailed Routing-Driven Placement Contest. http://www.ispd.cc/contests/15/ispd2015_contest.html. (2015). Accessed: 2018-04-04.
[8] N. Karimpour Darav, A. Kennings, A. Tabrizi, D. Westwick, and L. Behjat. 2016. Eh?Placer: A High-Performance Modern Technology-Driven Placer. *ACM TODAES* 21, 3 (2016), 37:1–37:27.
[9] D. P. Kingma and J. Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).
[10] J. Lou, S. Krishnamoorthy, and H. S. Sheng. 2001. Estimating Routing Congestion Using Probabilistic Analysis. In *ISPD '01*. 112–117.
[11] Mentor Graphics, Inc. 2015. *Olympus-SoC place and route for advanced node designs.* Technical Report. www.mentor.com/products/ic_nanometer_design/place-route/olympus-soc.
[12] M. Pan and C. Chu. 2007. IPR: An Integrated Placement and Routing Algorithm. In *DAC '07*. 59–62.
[13] Z. Qi, Y. Cai, and Q. Zhou. 2014. Accurate prediction of detailed routing congestion using supervised data learning. In *ICCAD '14*. 97–103.
[14] J. Roy, N. Viswanathan, G. Nam, C. Alpert, and I. Markov. 2009. CRISP: Congestion Reduction by Iterated Spreading During Placement. In *ICCAD '09*. 357–362.
[15] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. 2010. RUSBoost: A Hybrid Approach to Alleviating Class Imbalance. *Trans. Sys. Man Cyber. Part A* 40, 1 (Jan. 2010), 185–197.
[16] D. Shi and A. Davoodi. 2017. TraPL: Track Planning of Local Congestion for Global Routing. In *DAC '17*. 19:1–19:6.
[17] H. Shojaei, A. Davoodi, and J. Linderoth. 2013. Planning for Local Net Congestion in Global Routing. In *ISPD '13*. 85–92.
[18] H. Shojaei, A. Davoodi, and J. T. Linderoth. 2011. Congestion Analysis for Global Routing via Integer Programming. In *ICCAD '11*. 256–262.
[19] A. Tabrizi, N. Darav, L. Rakai, A. Kennings, and L. Behjat. 2017. Detailed routing violation prediction during placement using machine learning. In *VLSI-DAT*. 1–4.
[20] A. Tabrizi, N. Darav, L. Rakai, A. Kennings, W. Swartz, and L. Behjat. 2015. A Detailed Routing-Aware Detailed Placement Technique. In *ISVLSI '15*. 38–43.
[21] J. Andres Torres. 2012. ICCAD-2012 CAD Contest in Fuzzy Pattern Matching for Physical Verification and Benchmark Suite. In *ICCAD '12*. 349–350.
[22] L. Wang. 2017. Experience of Data Analytics in EDA and Test - Principles, Promises, and Challenges. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 36, 6 (June 2017), 885–898.
[23] L. Wang, P. Bastani, and M. Abadir. 2007. Design-silicon Timing Correlation: A Data Mining Perspective. In *DAC '07*. 384–389.
[24] Y. Wei and et. all. 2012. GLARE: Global and Local Wiring Aware Routability Evaluation. In *DAC '12*. 768–773.
[25] Jurjen Westra, Chris Bartels, and Patrick Groeneveld. 2004. Probabilistic Congestion Prediction. In *ISPD*. 204–209.
[26] J. Westra and P. Groeneveld. 2005. Is Probabilistic Congestion Estimation Worthwhile?. In *SLIP '05*. 99–106.
[27] D. Wolpert. 1996. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation* 8, 7 (1996), 1341–1390.
[28] Y. Xu, Y. Zhang, and Ch. Chu. 2009. FastRoute 4.0: Global Router with Efficient via Minimization. In *ASPDAC '09*. 576–581.
[29] Y. Yu, G. Lin, I. Jiang, and C. Chiang. 2013. Machine-learning-based Hotspot Detection Using Topological Classification and Critical Feature Extraction. In *DAC '13*. 67:1–67:6.
[30] Q. Zhou, X. Wang, Z. Qi, Z. Chen, Q. Zhou, and Y. Cai. 2015. An accurate detailed routing routability prediction model in placement. In *ASQED '15*. 119–122.