Predicting Routing Congestion using Machine Learning Techniques

Rim El Jammal

Abstract

Designing and optimizing complex applications is very challenging due to the many challenges that are faced along the way. One major challenge is routing congestion. Estimating routing congestion on an early stage is very crucial in order to avoid major complications while manufacturing. In this work, we propose estimating routing congestion not only on FPGAs, but on VLSI systems as well. We use machine learning techniques to predict the congestion level. The techniques were improved through hyperparameter tuning.

1 Introduction and Problem Statement

In routing channels, there exists tracks which are used for routing wires on a chip. If the number of tracks available for routing in one particular area is less than the number of routing tracks required, then the area is said to be congested.
This will be a limit for the number of nets that can be routed through the particular area or channel.
Routing congestion contributes to a lot of delay and resource utilization. In a congested design, wires have to be detoured for connections, generating longer delays and occupying more routing resources. Thus, congestion will degrade the design performance and might lead to implementation failures.
Routing constraints became a significant factor in the VLSI design. It significantly impacts quality metrics, such as area and timing performance. But the problem lies in not being able to accurately pinpoint or detect the congestion until a late stage in the design cycle.
This is why, early feedback is required in order to avoid any sudden problems while manufacturing.

2 Literature Review

Given that routing congestion is an important aspect of VLSI design, and its early detection is crucial, many studies tackled congestion estimation. Most of these studies work towards

prediction on FPGA, we will mention some of them.
First, and in most recent studies, Zhao et. al (2019) proposed a novel method to predict routing congestion in the high level synthesis (HLS) step using machine learning techniques. Their aim is to locate the highly congested regions in the source code. The early and accurate congestion estimation assists in guiding the optimization in HLS and futther improve the efficiency of implementation. In their paper, they describe how they generated their own dataset using C/C++ source code. Another study by Kirby et. al (2019) proposed a novel graph neural network based approach to congestion estimation. They also performed their technique on FPGAs.
Another way of tackling this problem is by using imaging. In Yeager et. al (2007) they used multiple image blending methods and multiple imaging techniques in order to detect the most congested region before routing.
We believe that this can also be applied to VLSI and ASIC. However, the correct benchmarks are needed in order to do that with their actual congestion estimate. Unfortunately, no benchmark with these specifications were found so we need to process and create our own dataset with the available benchmarks.

3 Methodology

In our study, we want to improve the results of previous studies by trying other machine learning techniques, performing hyper-parameter tuning, and by extracting other meaningful features from the datasets. We also would like to test it on VLSIs and ASICs, not just FPGAs.

3.1 Supervised learning

Assuming that we have a benchmark containing some information about pins and cut sizes in specific regions in a system, we would like to do the following:

1- Read and process the benchmark by extracting features such as, wire length area, pin count, number of net cuts, net count, ... As well as calculating the estimate congestion.

2- Splitting the data into training and testing. Since machine learning techniques can be prone to overfitting, we will be performing 10-fold cross validation on the training set, then we'll validate on the unseen testing set. We will also perform 30 runs because some of the techniques might introduce some randomness into the prediction. So, in order to eliminate that randomness, we run the experiments 30 times and get the average of the runs to have a good and fair evaluation.

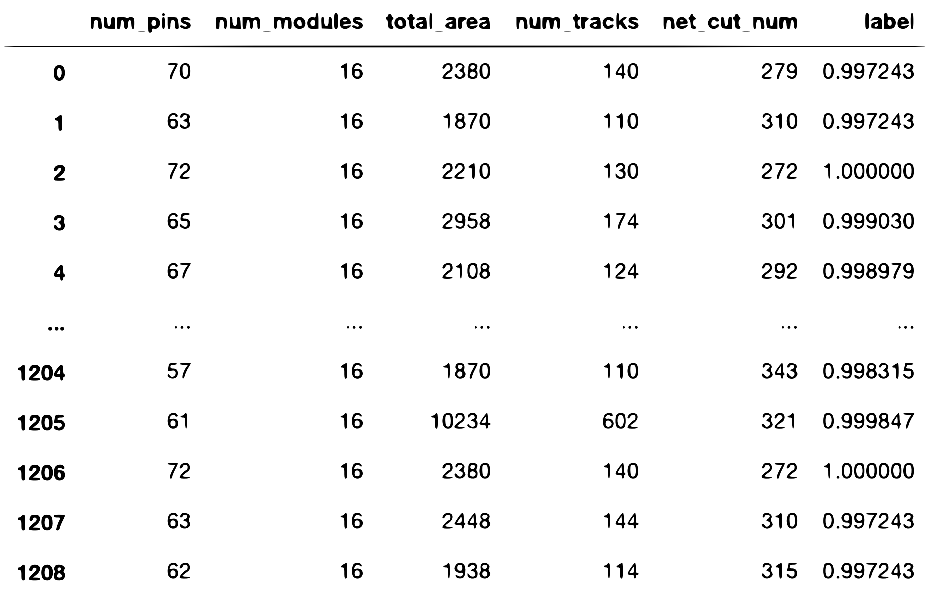
3- Step 3 is connected to step 2, because we want to perform hyper-parameter tuning, and this is done while training the models. We will be using randomized search for the tuning, because based on previous studies, randomized search can reach better hyper-parameters in much less time than other tuning techniques such as grid search for example.

The flow of steps would look as following:

Algorithm 1: Machine learning approach

```
data processing
split training and testing data
define sets of hyper-parameters values to evaluate
for n_runs 1 -> 30 do
    for each iteration in 10-fold cross validation do
        hold-out specific data sample
        fit the model on the remaining 9 samples with different set of
          hyper-parameters
        predict on the hold-out sample
    end
    compute average performance among the 10 folds
end
compute average performance among the 10 folds determine the optimal
  hyper-parameter set
fit the final model on initial training data using the optimal hyper-parameter set
```

Given that we didn't have any existing data, we tried it on some dummy data. The dummy data was extracted from The ISPD98 Circuit Benchmark Suite. We used the imb02, we assumed that for example, there will be a grid of 4x4 so we took the total number of modules and divided it by 16. We got the number of pins of each module, and by doing so, we were able to get the number of pins in each region.
As for the labels, we got it by doing some calculation over the existing features on each row. Then we normalized using a min max scaler to normalize between the values -1 and 1.
In the table below [1], we can see a small sample of the dataset.
We extracted the following as features: number of pins, number of modules, total area of the region, number of tracks, and number of net cuts. Each is per region, this way we can detect the most congested part of our system.



	num_pins	num_modules	total_area	num_tracks	net_cut_num	label
0	70	16	2380	140	279	0.997243
1	63	16	1870	110	310	0.997243
2	72	16	2210	130	272	1.000000
3	65	16	2958	174	301	0.999030
4	67	16	2108	124	292	0.998979
...	...	...	...	...	...	...
1204	57	16	1870	110	343	0.998315
1205	61	16	10234	602	321	0.999847
1206	72	16	2380	140	272	1.000000
1207	63	16	2448	144	310	0.997243
1208	62	16	1938	114	315	0.997243

FIGURE 1: Sample of data

3.2 Unsupervised learning

However, if the data instances, after being processed, didn't have a label, then we will go with unsupervised learning. In that case we can use clustering to group the data points based on their features. We can then assume from the different clusters which are congested and which are not through assumptions based on their number of pins or cut nets.
Logically speaking, a region with a high cut nets would most probably be congested due to the multiple wires passing through it, causing the high number of cut nets. So, this would be a significant indicator of detecting a congested region.

4 Results

We performed 10-fold cross validation 30 times, and we got the average for each.
For the evaluation metrics, we using mean squared error, mean absolute error, and the root mean squared error. These are good to assess regression tasks, moreover, the root mean squared error usually accounts for big errors in the prediction, so in case of outliers for example.

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(\overline{y}_i - y_i)^2} \tag{1}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|\overline{y}_i - y_i|) \tag{2}$$

$$MRE(1) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|y_i - \overline{y}_i|}{y_i + 1}\right) \tag{3}$$

	MRE	MAE	RMSE
Random Forest	0.0002	0.0004	0.0006
Support Vector Machine	0.0005	0.001	0.001

TABLE 1: Results on RF and SVM

In the table above [1] we can see the results. The errors compared to other studies is relatively lower. These are promising results and could mean that if trained on benchmarks related to congestion, they could give a good estimation for the VLSI design.

5 Conclusion

Our work aims to estimate the congestion level in each region of the system before going into more advanced stages in the design process. We suggest doing it for VLSIs, as not many studies tackled this problem.
Given the lack of benchmarks, to the best of my knowledge, we used some dummy data extracted from the ISPD Benchmark Suite.
Results showed to be promising. We suggest looking more into this problem by generating more meaningful datasets by exploring more the nature of the VLSI design.

6 References

Yeager, D., Chiu, D., Lemieux, G. (2007, March). Congestion estimation and localization in FPGAs: A visual tool for interconnect prediction. In Proceedings of the 2007 international workshop on System level interconnect prediction (pp. 33-40).

Soni, A., Soni, B., Mehta, R. (2020, May). Congestion Estimation Using Various Floorplan Techniques in 28nm Soc Design. In 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 199-204). IEEE.

Zhao, J., Liang, T., Sinha, S., Zhang, W. (2019, March). Machine learning based routing congestion prediction in fpga high-level synthesis. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)* (pp. 1130-1135). IEEE.

Al-Hyari, A., Abuowaimer, Z., Martin, T., Gréwal, G., Areibi, S., Vannelli, A. (2019). Novel Congestion-estimation and Routability-prediction Methods based on Machine Learning for Modern FPGAs. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 12(3), 1-25.