Understanding Negotiated-Congestion Routing for FPGAs

Mikhail Asiatici, Samuel Bosch, Zander Harteveld, and Stefan Nikolić

1 INTRODUCTION

Arguably the most important stage for timing closure in the *Field-Programmable Gate Array* (FPGA) CAD flow is placement. Most of the timing-driven placement algorithms strive to minimize some function of total connection criticality and delay [7]. Whereas in ASIC placement the final routed delay of a connection is highly correlated with its Manhattan length, the discrete nature of FPGA routing structure makes the Manhattan model very unrealistic. For that reason, individual connection delays are usually extracted by actually routing single connections with different tail and head locations. These delays are then stored in a look-up table which is indexed by the connection endpoints during the placement process [7]. This approach provides a much more realistic model than any based solely on geometric properties. However, in routing individual connections, the phenomenon of *congestion*, occurring when optimum routes of two or more connections use the same routing tracks, forcing one (or several) of them to be diverted, is entirely ignored. Although *negotiated-congestion* routing algorithms, which have been the state-of-the-art for almost 30 years, do try to implement the critical connections using the shortest possible routes, in practice, the discrepancy between the delays predicted using the place-time model that ignores congestion and the final routed delay can reach several percent of the critical path delay, which is significant by modern standards. It is clear that simultaneously solving the placement and the routing problem is infeasible for today's problem sizes. Nevertheless, having a better understanding of what causes this discrepancy and why it is more pronounced in some circuits and less in others can enable us to create better models that will lead to either better solution quality, or reduced runtime. The goal of this project is to harness the methods of network science and data analysis in an attempt to deepen this understanding.

2 BACKGROUND

Let us first give informal definitions of some terms that will be used throughout the text.

Definition 1. *A net is the set of connections between an output of a particular gate and inputs of all the gates that it drives.*

In the circuit graph, a net is a star graph, while after routing, it could be a more general tree (Figure 1).

Definition 2. *The slack of a connection is the maximum amount of delay that can be added to its current delay, before it becomes part of the slowest path in the entire (circuit) graph, assuming that all other connections keep their current delay.*

Definition 3. *The criticality of a connection is* $1 - Slack/D_{max}$*, where* D_{max} *is the weighted length of the slowest path in the circuit.*

Definition 4. *The criticality of a net is the maximum criticality of its connections.*

We say that a net or a connection is *critical* if its criticality is 1.

3 REPRESENTATION

Now that we have clarified the basic terms, we can start building the representation of the problem at hand. Congestion will be

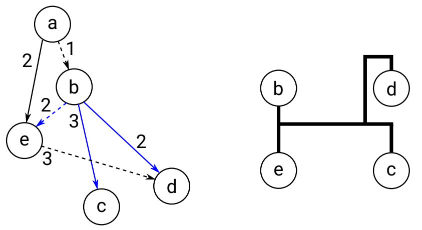

Figure 1: Examples of the defined terms. A dummy circuit graph is shown on the left. The delay of each connection is represented by the number shown next to it. The critical (slowest/longest) path is composed of dashed connections. Its delay is 6. Since the slowest path on which connection (b, d) lies is (a, b, d), with delay 3, we can delay it by 3 before it becomes critical. Hence the slack of connection (b, d) is 3, while its criticality is $1 - 3/6 = 0.5$. The criticality of net b is 1, because slack of connection (b, e) is 0, as it lies on the critical path. In general, the critical path does not need to be unique. On the right, a possible placement of nodes participating in net b is shown along with a possible routing. We can see that the net is no longer a star, but a more general tree.

problematic if, for the given placement of a given circuit, two nets that are (close to) critical must be routed using the same track in order to achieve the shortest route. Hence, we should describe the "preference" of each net to use a particular track.

We can construct a bipartite graph where each net of the circuit to be placed on the FPGA represents a node in one partition, while each routing track of the FPGA is represented by a node in the other (Figure 2). The weight of an edge between the net u and the routing track v should describe how likely it is that the final solution will be of good quality if the net u is routed through the track v. We can calculate the distance $d(u, v)$ as the sum of the delays of the shortest paths between the net's source and each of its targets, traversing the track v. Each of the addends in the sum is weighted by the criticality of the corresponding connection, normalized by the highest criticality among all the connections of the net. This is illustrated in Figure 3. The inverse of the distance is being used to weight the corresponding edge.[1] We also weight the net-representing nodes, in proportion to their criticality.

4 PRUNING

When routing a particular net, the router will not exit its bounding box by more than a fixed amount on each side (the default value for VPR is 3 rows/columns). Hence, for a large enough circuit, we can expect that the resulting graph will be very sparse. However, there are some high-fanout nets (usually inputs, or simple functions of inputs) that have very large bounding boxes. Moreover, given the number of tracks present even in a small FPGA, the resulting graphs are very large. The pruning method we employ is to observe the routed wire length of each net (in the number of segments), L, and keep only $L \times CHAN\_W$ tracks connected to the net representing node by the edges of the highest weight, where $CHAN\_W$ represents channel width.

[1] In principle, we should normalize the weights of each individual net, as otherwise the smaller, more compactly placed nets will have overall higher incident edge weights. Doing so, however, gave some counter-intuitive results, so in the final experiments, we do not do such normalization.

5 CONCRETE GOALS

Now that we have developed a basic model to capture the problem at hand, we can formulate concrete questions that can guide our exploration towards truly gaining a deeper understanding of it.

5.1 The Main Question

Let us start from a loose conjecture.

Conjecture 1. *The routed delay will be noticeably larger than the post-placement prediction iff there are heavy clusters in the projection of the net-representing partition.*

Careful choice of weights of the edges in the projection is crucial: if two nets share a potentially used track, it does not mean that an optimum routing solution cannot be achieved without both of them using that track. Let us designate the weights in the original bipartite graph as $w(u, v)$ and those in the projection as $w_p(u, v)$. Let us also choose two nets u_1 and u_2 and designate their neighborhoods in the bipartite graph as N_1 and N_2, respectively. Then we can define $w_p(u_1, u_2)$ as follows:

$$\langle w_c(u_1)\rangle = \frac{\sum_{v\in\{N_1\cap N_2\}} w(u_1, v)}{|N_1 \cap N_2|} \tag{1}$$

$$\langle w_{uc}(u_1)\rangle = \frac{\sum_{v\in\{N_1\setminus N_2\}} w(u_1, v)}{|N_1 \setminus N_2|} \tag{2}$$

$$w_p(u_1, u_2) = min(\{\langle w_c(u_1)\rangle - \langle w_{uc}(u_1)\rangle, \langle w_c(u_2)\rangle - \langle w_{uc}(u_2)\rangle\}) \tag{3}$$

Finally, we can drop the negative edges. Defined in this way, the weights tell us how much a net would lose by giving up its preferred resources to its opponent. To increase the realism of the model, when computing w_c, we consider only those tracks that have degree larger than $THR \times CHAN\_W$, in the bipartite graph, where THR is a constant somewhat arbitrarily set to 0.75. The rationale is that if the channel is largely unused, a different track from it can easily be chosen instead of the considered one.

Testing the conjecture now reduces to clustering the projection, weighing the clusters and comparing the weights to an appropriate threshold. We define the cluster weight as follows:

$$W = \frac{\sum_{(u,v,w_p)\in E} uvw_p}{|E|} \tag{4}$$

u and v in the expression designate the weights of the corresponding nodes and E is the set of edges contained in the cluster.

5.2 Are the heavy clusters inherent or created by the placer?

If the heavy clusters are really the cause of the critical path delay increase in presence of congestion, it would be useful to know how to eliminate them, or whether they can be eliminated at all. We can embed the circuit graph in a plane, using standard techniques such as t-SNE and observe the proximity of the nodes forming the heavy cluster. If the embedding obtained in this way is similar to that produced by the placer, we could have some confidence that the clusters are inherent to the structure of the circuit itself, and not created artificially through a suboptimal placement process.

circuit	W	td_inc [%]	uc_td_inc [%]	cls_w
alu2	30	16.66	5.551	0.3586
alu2	34	5.551	4.848	0.1367
alu2	36	3.957	1.659	0.1468
alu2	40	1.403	1.403	0.1467
alu2	42	4.721	1.403	0.1559
alu2	46	8.942	6.061	0.1499
apex6	34	16.02	6.111	0.1029
apex6	38	8.069	2.536	0.0690
apex6	42	10.49	8.532	0.1403
apex6	46	10.72	10.72	0.0808
apex6	48	6.339	3.344	0.1525
apex6	52	2.536	3.344	0.0348
x1	34	8.326	3.524	0.1438
x1	38	20.17	20.17	0.1300
x1	42	4.643	5.764	0.0597
x1	46	20.33	20.33	0.0842
x1	48	8.167	2.243	0.0660
x1	52	2.243	2.243	0.0989

Table 1: Critical path delays and heaviest cluster weights. Numerical values of critical path delays of three different circuits, routed at various channel widths (*W*) are shown. Column *td_inc* shows the percentage increase of the critical path delay estimate after placement, once routing is completed (i.e., all congestion is eliminated). Column *uc_td_inc* is the corresponding delay increase when congestion is entirely ignored (i.e., each connection takes the fastest possible route, ignoring others). It is interesting that this delay sometimes even surpasses the final routed delay. This is because the congestion-oblivious routing does not provide a true lower bound on routed delay, in general [3]. Finally, *cls_w* shows the weights of the heaviest cluster after the respective projection graph is divided into 16 clusters.

6 EXPERIMENTAL SETUP

We use a subset of small, purely combinational circuits from the MCNC [8] benchmark set, to avoid complications with the larger and more complex modern circuits, as well as the associated runtime issues that could be prohibitive in this exploratory phase. A simple FPGA architecture with all tracks spanning four logic blocks and a cluster fairly representative of modern FPGAs is used as the target device. VTR [5] provides the packing, placement, and routing algorithms and all the data is extracted from its output.

7 RESULTS

In this section, we present the results of applying the model described previously. We start with the results of the experiments that directly test the main hypothesis: the postrouting critical path is substantially slower than the posplacement one, if and only if the model graph contains heavy clusters. Then we proceed with exploring a few more phenomena related to the process of placement and routing using the same model and other tools of data analytics.

7.1 An Answer to the Main Question

Table 1 shows the critical path delays of three different circuits, routed with the routing channel width ranging from the minimum necessary to eliminate all congestion to 1.5× that minimum. Sweeping the channel width for the same circuit is useful as it directly controls the amount of routing congestion present in the placed design. To further increase control over congestion, the same placement of the circuit is kept for all channel widths.

One important observation is that delays are not monotonically nonincreasing with respect to channel widening. Most likely, this is due to the differences in the detailed routing architecture [2]. To provide some justification for this conjecture, in the second column of the table, we show the critical path delays found by the router

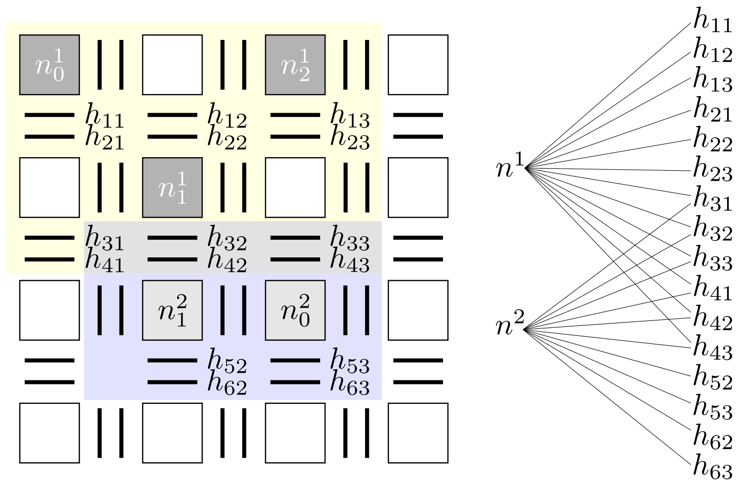
n_0^1
n_2^1
n_1^1
n_1^2
n_0^2
h_{11} h_{21} h_{12} h_{22} h_{13} h_{23}
h_{31} h_{41} h_{32} h_{42} h_{33} h_{43}
h_{52} h_{62} h_{53} h_{63}
n^1
n^2

Figure 2: An example of the construction of the bipartite graph, describing the relations between the nets of a placed circuit and the FPGA's routing tracks. For clarity, only two nets are shown, and only horizontal segments are labeled and represented in the graph.

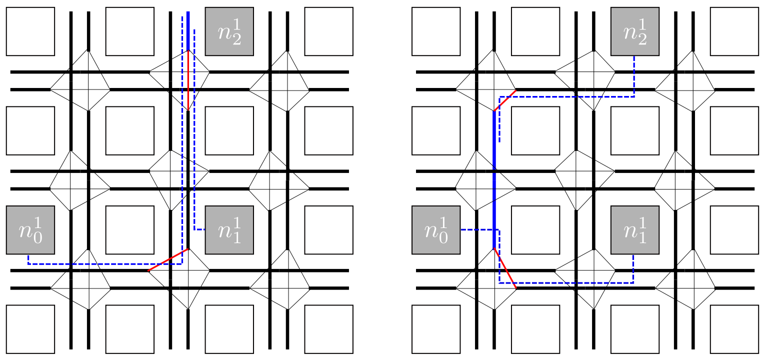

Figure 3: An example of edge-weight calculation. Thin black lines represent the connectivity achievable by the switch boxes. Here we assume that the delay is proportional to the number of switch boxes crossed by a connection. The track v for which we are computing the weight is highlighted in blue. In the left figure, two switch boxes (red) are traversed to reach v; then, to reach the target n_1^1, we need to cross one more switch box, while n_2^1 is immediately accessible. Hence the distances to n_1^1 and n_2^1 through v are 2 and 3, respectively. Assuming that the criticalities of (n_0^1, n_1^1) and (n_0^1, n_2^1) are 0.7 and 0.9, respectively, we obtain the weight of $\frac{1}{3.9}$. In the figure on the right, both targets are one switch box away from v, and v is reachable from the source without traversing any switch boxes, so the weight is $\frac{1}{1.6}$.

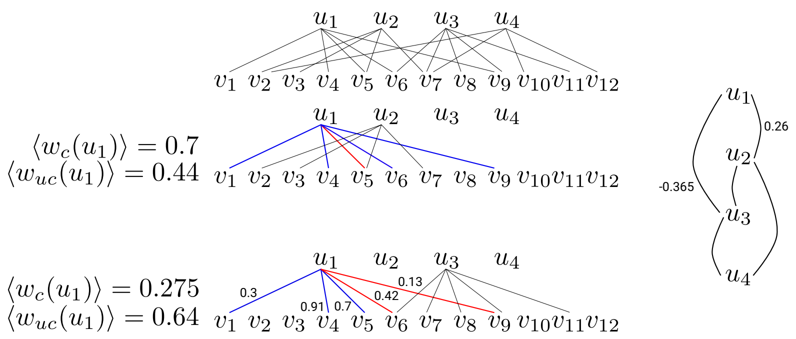
u_1 u_2 u_3 u_4
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12}
$\langle w_c(u_1)\rangle = 0.7$
$\langle w_{uc}(u_1)\rangle = 0.44$
$\langle w_c(u_1)\rangle = 0.275$
$\langle w_{uc}(u_1)\rangle = 0.64$
0.3
0.91
0.7
0.42
0.13
0.26
-0.365

Figure 4: An example of determining the edge weights in the net-representing partition projection. Averages are only computed for the net u_1. Assuming that u_1 has the dominant difference between the average congested and uncongested weights, the projection weights will be as annotated in the graph on the right. Due to its negative weight, the edge (u_1, u_3) does not appear in the final projection.

when congestion is altogether ignored (i.e., when each connection can take the shortest possible path, regardless of whether it intersects with the routing of another connection or not). As we can see, in many cases, there is a substantial difference from the delay predicted by the placer. This is likely because the placer uses a simplified routing architecture model, when populating the delay lookup table. We thus immediately see that the *only if* direction of our hypothesis does not hold in its present formulation.

Given the difficulties of separating the influence of congestion on delay from all the other possible factors, it is not easy to bring

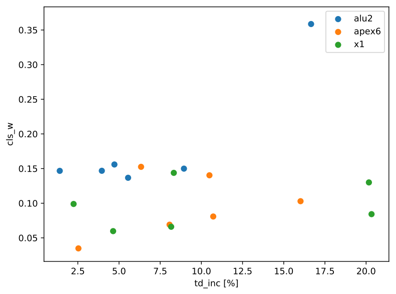

Figure 5: Heaviest cluster weight vs increase in postrouting delay. Weights of the heaviest cluster after clustering the projection graphs into 16 classes are shown against the relative change in the postplacement critical path delay estimate after routing. For routings of the same circuit with different channel width, there seems to be some correlation, with some clear outliers. Bringing conclusions that would apply to all circuits at once is not impossible, however.

any immediate conclusions about the validity of our model that was intended only for congestion analysis. Nevertheless, we show the weights of the heaviest cluster of the corresponding projection graph, for each circuit/channel width combination, after forming 16 clusters using spectral clustering, in the last column of Table 1. This number of clusters was empirically determined by visual observation of the grouping of placed nodes to be reasonably good.

Figure 5 shows a scatter plot of the heaviest cluster weights against the relative change in the critical path delay after routing, in an attempt to better expose any correlation between the two quantities. The plot shows little correlation, far insufficient to make the model useful as a postrouting critical path delay predictor.

Given the quantitative data, we consider the main hypothesis refuted. Let us neglect this for the moment, however, and try to use the network and data analysis techniques to increase our understanding of the structure of circuits, as was our original intention.

7.2 Tracking the Critical Path

A useful assessment of whether the proposed model graph makes sense or not, regardless of its inability to predict the final critical path delay after routing, could be to try to predict the location of the critical path nodes by identifying the heavy clusters of the projection graph. Figure 6 shows the positions of the clusters in a placed circuit, for the number of clusters being gradually increased. The size of the markers is proportional to the weight of the cluster, while the colors are also sorted according to size, the heaviest shown in yellow. Nodes of the critical path, after routing, are represented as fixed-size triangles. As we can see, in most cases, the problematic region is shown as relatively heavy and at least one node of the critical path belongs to the heaviest cluster. It is in fact reasonable that not all of the critical path nodes are part of the heaviest cluster, as it is the conflicts with other nodes that elongate some connection, putting it on the critical path.

Given that we are partially successful in locating the final critical path delay from the posplacement delay predictions, we might be tempted to conclude that the method is indeed useful. Unfortunately, in the particular circuit, there is a single pronounced critical sink[2] that is already detectable after placement, as shown in Figure 7. In more modern circuits with multiple sinks being (close to) critical, we are less likely to succeed. Hence, we can conclude that although the

[2]In general, there could be K^d critical paths, if only one sink LUT is critical, where K is the LUT size and d the (unweighted) length of the critical path. In this case, K is 6 and d is 2, hence up to 36 different intersecting paths could have the maximum delay.

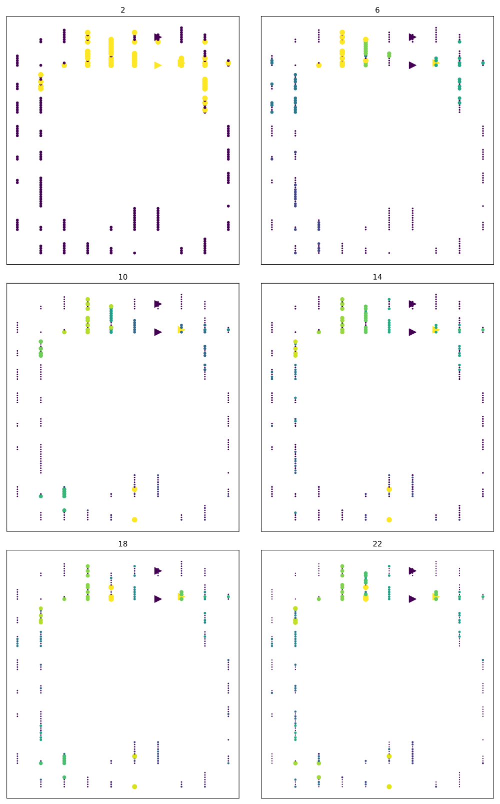

Figure 6: Physical location of the identified clusters on the FPGA grid. The figure shows the locations of the clusters obtained by processing the projection graph of *apex6* circuit with the channel width of 34. Each node represents a net. Nets belonging to the same cluster (or I/O block) of the FPGA are stacked together vertically (appearing as stripes). The size of the markers is proportional to the weight of the respective cluster of the projection. The colors are also sorted according to weight, with yellow being the "heaviest". Triangular nodes of fixed size are members of the final critical path, after routing. The center of the grid is empty because the size of the smallest FPGA into which this circuit can fit is determined by the number of inputs and outputs.

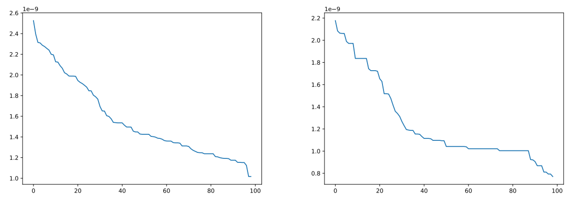

Figure 7: Distribution of sink delays for *apex6* routed with 34 tracks per channel (left) and sink delays for the same circuit predicted after placement. In both plots, there is one clear critical sink.

model appears not to be entirely inappropriate, it is likely neither precise, nor accurate enough to allow us to predict the exact location of the final critical path, much like it failed to predict its delay.

7.3 How Good is the Placer?

Had we managed to find evidence to support the hypothesis of the heavy clusters being the cause of increase of the critical path delay after congestion elimination, it would have been appropriate to investigate where these clusters come from. Is it the structure of the circuit itself that dictates that nodes forming the heavy clusters should be placed close to each other, or is it the placer that fails to

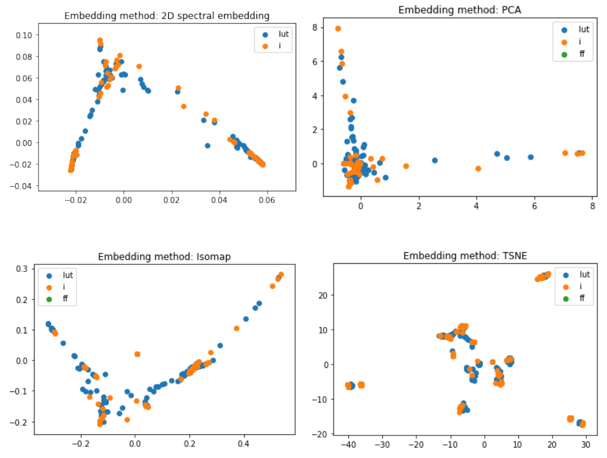

Figure 8: Various embedding methods applied to the *apex6* circuit. The colors represent node types. Since the circuit is combinational, no flip-flops are present. By visual judgment, t-SNE seems to provide the best embedding, as groups of nodes are clearly separable.

converge to an optimal solution? Even if we did manage to find such evidence, answering the above question would be very difficult, as one of the main goals of the placer, and certainly the goal we care the most about here is minimizing delay, which is not necessarily always correlated with the structure of the circuit. If we do find that the structure is preserved throughout the placement process, however, that could give us some confidence that the placer indeed performs well.

To expose the structure of the circuit, we attempt to embed it using one of the standard dimensionality reduction techniques. *Laplacian eigenmaps* is directly applicable because the circuits themselves are graphs, whereas other methods, such as *PCA* and *t-SNE* require points in a space of higher dimension. To make use of them, we first project the circuits using Laplacian eigenmaps to a high-dimensional space, and then use PCA, Isomap, and t-SNE to embed them in the plane.

Results of all the methods, applied to *apex6* are shown in Figure 8. Only t-SNE appears to produce any meaningful results. We can see that there are some clear clusters amenable to *K-Means* clustering. After doing so and plotting the results on the FPGA grid, we obtain Figure 9. The clusters obtained directly from the structure of the circuit indeed appear to be placed close together by the placer.

8 CONCLUSION

In this project we tried to apply the network and data analysis techniques to a specific problem in FPGA CAD. Unfortunately, this attempt was not quite successful, but it gave us some valuable insight. The most important conclusion is perhaps that handcrafting the models is no easy task, especially when runtime needed to construct them for particular problem instances has to be taken into account, which is clearly the case in CAD. This may explain why the efforts in the domain of predicting various effects between different stages of CAD flow recently shifted towards supervised learning [6], instead of refining the existing, classical models [4]. One of the use cases is synthesis-level optimization, where one might choose a particular heuristic flow based on classification of the input circuit. Researchers have successfully leveraged the existing image classification techniques by converting truth-tables

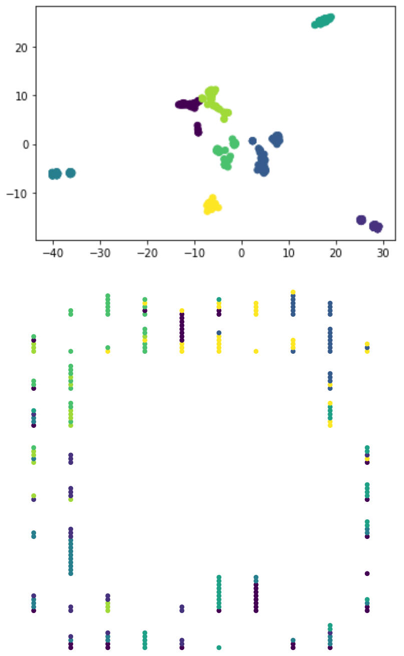

Figure 9: Physical location of the clusters obtained by K-Means clustering of the t-SNE embedding of *apex6*. The top figure shows a K-Means clustering of the t-SNE embedding of the *apex6* circuit, while the bottom one shows the physical locations of the nodes, as determined by the placer, with the colors corresponding to the obtained cluster labels. To a large extent, the circuit structure exposed by the embedding seems to be preserved by the FPGA placer, as indicated by proximity of the nodes of the same color.

of Boolean functions to be optimized to images [1]. While this may be appropriate for synthesis, the downstream phases of the CAD flow are typically driven by the structural representation of the synthesized functions—they care only about graphs. It is quite probable that *learning on graphs* can yield much better results than the current approaches that use more traditional learning methods. The future seems very exciting.

REFERENCES

[1] M. Austin, W. L. Neto, L. Amaru, X. Tang, and P.-E. Gaillardon. Towards a novel logic synthesis framework supervised by convolutional neural network. In *IWLS*, 2019.
[2] V. Betz and J. Rose. Automatic generation of FPGA routing architectures from high-level descriptions. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2000, Monterey, CA, USA, February 10-11, 2000*, pages 175–184, 2000.
[3] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for deep-submicron FPGAs.* Kluwer Academic, 1999.
[4] P. Kannan, S. Balachandran, and D. Bhatia. fgrep - fast generic routing demand estimation for placed FPGA circuits. In *Field-Programmable Logic and Applications, 11th International Conference, FPL 2001, Belfast, Northern Ireland, UK, August 27-29, 2001, Proceedings*, pages 37–47, 2001.
[5] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(2):6:1–6:30, June 2014.
[6] D. Maarouf, A. Alhyari, Z. Abuowaimer, T. Martin, A. Gunter, G. Grewal, S. Areibi, and A. Vannelli. Machine-learning based congestion estimation for modern fpgas. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 427–4277, 2018.
[7] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for fpgas. In *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, FPGA '00, pages 203–213, New York, NY, USA, 2000. ACM.
[8] S. Yang. Logic synthesis and optimization benchmarks user guide, version 3.0. Technical report, Microelectronics Center of North Carolina, Research Triangle Park, N.C., Jan. 1991.