**DAT340 - Applied Machine Learning**
**Writing Assignment**

# A graph placement methodology for fast chip design

**Stefano Ribes**
Chalmers University of Technology
ribes@chalmers.se

## 1 Introduction

This report describes the work of Mirhoseini *et al.* (Mirhoseini et al., 2021), "A graph placement methodology for fast chip design". In it, the authors propose an efficient and fast Reinforcement Learning (RL) method to perform chip floorplanning. Chip floorplanning is the task of placing the individual build blocks of a computer chip onto a two-dimensional grid. The individual blocks typically represent memories, control logic systems, and compute logic and can be described by a *netlist*, *i.e.* a hypergraph of basic circuit components. Such basic circuits are mainly divided into three categories: macros, *i.e.* memory components, standard cells, *i.e.* logic gates, *e.g.* XOR, NAND, NOR, and finally wires, connecting all of the previous. Expert engineers can spend months of work on the floorplanning design phase, trying to optimize the performance of the resulting chip, *i.e.* in terms of power consumption, wirelength and timing, while at the same time meeting the constraints of density and routing congestion.

In order to address the task, the authors exploit a set of neural architectures aiming to *learning transferable representations* of human-made chips. In particular, for training the system, they first designed an encoder network, trained via a Supervised Learning (SL) based algorithm, that feeds an encoded representations of the chips to a value and a policy networks, together referred as an *agent*, trained via a RL-based algorithm.

## 2 Design

### 2.1 Network Architecture

The idea of selecting an RL model is derived from the fact that the number of possible placing of 1000 clusters of nodes onto a 1000 cells-sized grid is greater than $10^{2,500}$, meaning that an exhaustive search would be prohibitive. Even so, heuristic approaches would still have to deal with an incredibly large amount of possibilities. Because of that, the authors decided to instead design an RL-based system able to learn from past chip design and generalize to new placements.

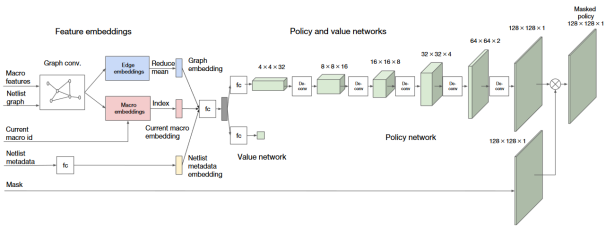


Figure 1: The overall neural architecture. On the left side, the Feature embeddings part features the Edge-GNN, while the policy and value networks are on the right. For the sake of brevity, I'm not discussing all the details of the image.

In order to tackle this challenging task, they modeled the problem as a Markov Decision Process (MDP), in which the agent would be exploited to generate a *probability distribution of placements*. A placement, *i.e.* an action, is then sampled from such distribution and finally applied to generate a new *state*, *i.e.* a grid with the new component macro added.

As mentioned above, the agent exploits an Edge-Based Graph Neural Network (Edge-GNN) to encode the state information into a low-dimensional vector representation to be fed into a value and a policy network. The value network is a feed-forward network, whereas the policy network consists of a series of deconvolution, batch-norm and ReLU layers. The policy network is responsible for generating the aforementioned probability distribution. Figure 1 gives an overview of the three main networks and of how they are interconnected with each other.

### 2.2 Training Data Acquisition

For training the SL part of the system, *i.e.* the Edge-GNN network, the authors assembled a dataset of 10,000 chip placement, with the reward of each placement as a label. In particular, each input formed by a netlist hypergraph represented as an adjecency matrix and a list of node features, the netlist metadata, the identity of the node to be placed and the utilized process technology, *e.g.* 7 nm. The adjecency matrix was generated via the following transformation: for each pair of nodes, *i.e.* components like macros or standard cells, if their distance is greater than a certain threshold, then no edge is created, otherwise, a decaying weight is applied on the created edge.

The 10,000 floorplans were generated by a vanilla policy network properly configured. In practice, the authors collected output snapshots of the policy network while training it via their RL training algorithm. As the network improved via training, so did the floorplans quality, allowing the author to gather many diverse floorplans of varying quality.

## 3 Evaluation

In order to evaluate the system, the authors focused on a series of metrics: Worst Negative Slack (WNS), Total Negative Slack (TNS), area, power, total wirelength and routing congestion. The system was compared to two baseline methods: an academic state-of-the-art tool, RePlAce (), and the previous work of a team of physical designers on the produced TPU design.

The comparison was made over 5 TPU blocks which weren't used during the network training. The authors' AI system outperforms RePlAce in terms of timing, *i.e.* WNS and TNS, and also outperforms or matches the human design in terms of area, power and wirelength. In terms of execution time instead, RePlAce completed placement in around one hour, but produced low quality results, whereas the authors' system completed the task in under 6 hours. On the other hand, the physical design team concluded their work in a period of several month. The complete list of results is reported in Figure 2.

| Name | Method | Timing | | Total area (μm²) | Total power (W) | Wirelength (m) | Congestion | |
|---|---|---|---|---|---|---|---|---|
| | | WNS (ps) | TNS (ns) | | | | H (%) | V (%) |
| Block 1 | RePlAce | 374 | 233.7 | 1,693,139 | 3.70 | 52.14 | 1.82 | 0.06 |
| | Manual | 136 | 47.6 | 1,680,790 | 3.74 | 51.12 | 0.13 | 0.03 |
| | Our method | 84 | 23.3 | 1,681,767 | 3.59 | 51.29 | 0.34 | 0.03 |
| Block 2 | RePlAce | 97 | 6.6 | 785,655 | 3.52 | 61.07 | 1.58 | 0.06 |
| | Manual | 75 | 98.1 | 830,470 | 3.56 | 62.92 | 0.23 | 0.04 |
| | Our method | 59 | 170 | 694,757 | 3.13 | 59.11 | 0.45 | 0.03 |
| Block 3 | RePlAce | 193 | 3.9 | 867,390 | 1.36 | 18.84 | 0.19 | 0.05 |
| | Manual | 18 | 0.2 | 869,779 | 1.42 | 20.74 | 0.22 | 0.07 |
| | Our method | 11 | 2.2 | 868,101 | 1.38 | 20.80 | 0.04 | 0.04 |
| Block 4 | RePlAce | 58 | 11.2 | 944,211 | 2.21 | 27.37 | 0.03 | 0.03 |
| | Manual | 58 | 17.9 | 947,766 | 2.17 | 29.16 | 0.00 | 0.01 |
| | Our method | 52 | 0.7 | 942,867 | 2.21 | 28.50 | 0.03 | 0.02 |
| Block 5 | RePlAce | 156 | 254.6 | 1,477,283 | 3.24 | 31.83 | 0.04 | 0.03 |
| | Manual | 107 | 97.2 | 1,480,881 | 3.23 | 37.99 | 0.00 | 0.01 |
| | Ours | 68 | 141.0 | 1,472,302 | 3.28 | 36.59 | 0.01 | 0.03 |

Here, we compare our method with the state-of-the-art method (RePlAce[9]) and with manual placements using an industry-standard EDA tool. For all metrics in this table, lower is better. H, horizontal; V, vertical.

Figure 2: Evaluation results.

## 4 Discussion

### 4.1 Authors' Opinion

The authors believe that their method has the potential to improve other hardware optimization problems, such as allowing for a fast architectural exploration. In addition to that, they claim it could be used to speed up the development of new techniques to improve critical metrics such as timing and power. Moreover, their algorithm for optimizing placing can be extended to other fields of study like city planning, compiler optimization and environmental engineering.

### 4.2 Personal Opinion

Having worked as a hardware engineer myself, I'm aware of the high amount of human-time required to iterate over a chip design. Despite the availability of such a system in commercial tools won't come in the near future, I believe this work is quite impressive. Shrinking months of work down to a few hours is a huge productivity improvement that can lead to even better chip designs. Simply put, the time spent on having a production-ready chip could instead be potentially spent on improving even further an automatically generated and highly performing baseline.

However, the authors do not mention a possible drawback in utilizing such AI technology. Despite generating impressive chips, I believe that expert designers would anyway need to double check the automatically generated chip. This is because any Application Specific Integrated Circuit (ASIC) such as the Google TPU is, in the end, a physical object that cannot be changed once fabricated. Because of that, any possible bug found in an already manufactured ASIC translates to an enormous economic loss for the company designing the chip. This means that human designers must need to be extremely confident that the design is free from any errors. Hence, completely

relying on an automated algorithm, might still require significant human effort.

## 5 Conclusions

This document reported the work of Mirhoseini *et al.* (Mirhoseini et al., 2021), "A graph placement methodology for fast chip design". The proposed RL algorithm is able to efficiently perform chip floorplanning in under six hours, matching or overtake expert human designers.

## References

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. 2021. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212.