# RLPlace: Deep RL Guided Heuristics for Detailed Placement Optimization

Uday Mallappa[1], Sreedhar Pratty[2], David Brown[2]
[1]University of California, San Diego [2]Nvidia Corporation
umallapp@ucsd.edu, {spratty, dbrown}@nvidia.com

***Abstract*— The solution space of detailed placement becomes intractable with increase in thenumber of placeable cells and their possible locations. So, the existing works either focus on the sliding window-based optimization or row-based optimization. Though these region-based methods enable us to use linear-programming, pseudo-greedy or dynamic-programming algorithms, locally optimal solutions from these methods are globally sub-optimal with inherent heuristics. The heuristics such as the order in which we choose these local problems or size of each sliding window (runtime vs. optimality tradeoff) account for the degradation of solution quality. Our hypothesis is that learning-based techniques (with their richer representation ability) have shown a great success in problems with huge solution spaces, and can offer an alternative to the existing rudimentary heuristics. We propose a two-stage detailed-placement algorithm *RLPlace* that uses reinforcement learning (RL) for coarse re-arrangement and Satisfiability Modulo Theories (SMT) for fine-grain refinement. With global placement output of two critical IPs as the start point, *RLPlace* achieves upto 1.35% HPWL improvement as compared to the commercial tool's detailed-placement result. In addition, *RLPlace* shows at least 1.2% HPWL improvement over highly optimized detailed-placement variants of the two IPs.**

## I. Introduction

In the physical design flow, the detailed placement step follows the global placement step. The job of the detailed placer is to legalize the cells in valid placement sites. While doing this, several objective functions such as wirelength, power, timing, area, etc. are used to improve the solution quality. Since the design is already optimized for one of the above cost metrics before the detailed placement step, the cost improvement during the detailed placement is incremental and low in magnitude as compared to the optimization of global placement step; thus making it a difficult problem. As the number of placeable cells and their possible placement locations increase, heuristic-based solvers serve as a good alternative for the detailed placement problem. Region-wise optimization involves dividing the layout into multiple smaller and tractable regions and work on each of these small local regions in some sequence. The inherent heuristics such as order of selecting the regions plays an important role in the final solution quality. Our hypothesis is that Reinforcement Learning (RL) techniques can help us in determining the optimal sequence of choosing the regions for such region-based optimizations. The most important feature of a such a learning-based placer is its ability to learn from past experience and improve its performance over training. Besides the learning ability, there are also two other promising properties of learning-based placers. Firstly, the trial-and-error mechanism of RL makes it easier to deal with lack of data training data. Secondly, the machine learning models in general, tend to have a better state representation ability with flexible representation forms such as image (matrix), vector and graph, which is usually not available for heuristics-based methods. Recent works in RL for solving Combinatorial Optimization (CO) problems [8] combine RL with existing combinatorial search algorithms and demonstrate promising results. These techniques leverage deep RL's ability to recover reasonable rough solution fast, and then exploit the ability of exact solvers to incrementally improve via local refinement.

**Problem Statement:** Given a global placement solution, we divide the layout into multiple grids (a set of grids form a window or a region). The goal of sliding-window based *RLPlace* algorithm is to determine optimal arrangement of grids in each window and distribute the cells within each grid such that the resulting overlap-free solution is optimal in terms of total wirelength.

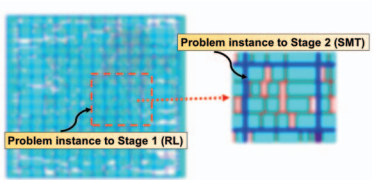


Fig. 1: The problem instances of *RLPlace*, shown on a blurred layout of our design.

## II. Related Work

The proliferation of machine learning algorithms coalesced with GPU acceleration offer an alternative to solve existing EDA problems [1] with huge solution space, along with generalization to unseen problems. Many previous works use supervised learning techniques for various VLSI optimization problems; [2], [3] for prediction of congestion and DRC violations, [4], [5] for predictions associated with static timing analysis, [6] for power-delivery network synthesis and [7] for leakage power optimization. In active learning category, deep RL [8], [9] has proven its success in problems formulated as MDP. Previous works [10], [11], [12], [13], [14] demonstrate the efficacy of deep learning-driven prediction problems and reward-driven optimization tasks such as macro-placement, transistor sizing, tool parameter optimization. Recently, Lin et al. [16] propose batch-based concurrent algorithms ABCDPlace that exploits GPU acceleration.

**Our Contribution:** Though previous works use RL for macro-placement (piggybacked by forced directed standard-cell placement), we are the first to formulate the standard-cell detailed-placement as MDP and solve it using RL.

## III. Background

### A. Markov Decision Process (MDP)

Optimization problems such as VLSI placement can be viewed as a sequence of decisions; decisions to gain some long-term reward such as wirelength, area, performance or power. Markov Decision Process (MDP) gives us a formal way to solve these sequential decision making processes. Formally, MDP is a discrete-time state-transition system that can be described with four components (i) States ($s \in S$) that are necessary for choosing possible actions (ii) Actions $a \in A$ that are chosen from a policy $\pi : S \rightarrow A$ (iii) Transition Model $P(s'|s,a)$ that describe the dynamics of the world and (iv) Reward that is a real-valued function $R(s)$ on states representing the goodness of a state, from the agent's perspective. The value of a state $V^{\pi}(s)$ keeps track of the expected long-term discounted rewards of the state. The discount factor $\gamma < 1$ guarantees the convergence of expected reward. Practically, this makes sense in the placement problem, because we are not certain of long-term future and is rational to weigh them less as compared to immediate rewards.

$$V^{\pi}(s_0) = \mathbb{E}_{\pi}\Big(\sum_{i=0}^{\infty} \gamma^i R(s)\Big) = \sum_{i=0}^{\infty} \gamma^i * R(s) \leq \frac{R_{max}}{1-\gamma} \qquad (1)$$

### B. Reinforcement Learning

The value function for every state has an interesting local recursive relationship, represented by "Bellman Expectation" Equation 2. The first term $R(s)$ is deterministic, given the state. The second term is a distribution over all possible next-state transition probabilities.

$$V^{\pi}(s) = R(s) + \gamma * \sum_{s'} P\big(s'|s,\pi(s)\big)V^{\pi}(s') \tag{2}$$

For every state, we are interested in the optimal value function $V^*(s)$ and then derive the corresponding policy.

$$V^*(s) = \max_{\pi}\big(V^{\pi}(s)\big) = R(s) + \max_{a\in A(s)}\Big(\sum_{s'} P\big(s'|s,a\big)V^*(s')\Big) \tag{3}$$

In the absence of transition probabilities of the environment, learning from simulated experiences a powerful method to determine the value function. The action-value states denoted by $Q$ states help us to overcome the sampling issue associated with the max operator of Equation 3. Intuitively, imagine taking an action $a$ and then following the default policy thereafter.

$$Q(s,a) = R(s,a) + \gamma\Big(\sum_{s'} P\big(s'|s,a\big)\max_{a'} Q(s',a)\Big) \tag{4}$$

We can use the Temporal Difference (TD) update rule of Equation 5, to incrementally improve the Q value estimates as we get more and more samples. Once the $Q$ values converge, the best value of a state can be obtained by Equation 6.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(R(s) + \gamma\max_{a'} Q(s',a') - Q(s,a)\Big) \tag{5}$$

$$V^*(s) = \max_{a} Q(s,a) \tag{6}$$

### C. Deep Q Learning

To derive the optimal policy, we can either maintain a table of $Q$ values for all state-action $(s,a)$ pairs or use a function approximator (using a neural network). For both the methods, the idea is to iteratively improve the stored current $Q$ values using the TD update. The obvious advantage of having a function approximator is the scalability (without having to deal with large tables), along with generalizability.

### D. Satisfiability Modulo Theories (SMT)

For optimization problems such as placement, in addition to satisfiability, we also need to support for predicate logic along with an ability to systematically search for the optimal solution. To solve such Boolean Optimization problems, Satisfiability Modulo Theories-based framework (SMT) often termed as the generalization of Boolean SAT solvers offer a richer modeling language and framework; AND, OR, at-most-1 (AMO), at-least-1 (ALO), exactly-1 (EO), if-then-else (ITE), BitVector (BV) representations and lexicographic objective functions. In our flow, once we determine optimal sequence of grid-swaps of each window, we use the SMT framework for fine-grain refinement.

## IV. Problem Formulation

In this work, we follow a two-stage hierarchical approach to solve the problem of detailed placement optimization. We divide the layout into windows or regions (2D space); similar to the sliding window approach. Each window is further partitioned into grids that contain many cells. In Stage 1, instead of working at the cell-level, we work on rearranging these coarse grids that contain one or more cells. When we rearrange the grids inside each window, the cells associated with the grid move along with the grid. It is important to observe that any possible arrangement of the grids in the window can be obtained by a sequence of grid-swaps. Therefore, the eventual goal of Stage 1 is to determine the optimal sequence of these grid swaps and handover to Stage 2 to perform the fine-grain overlap-free redistribution. Figure 1 summarizes our RL model training and the two-stage *RLPlace* flow-deployment.

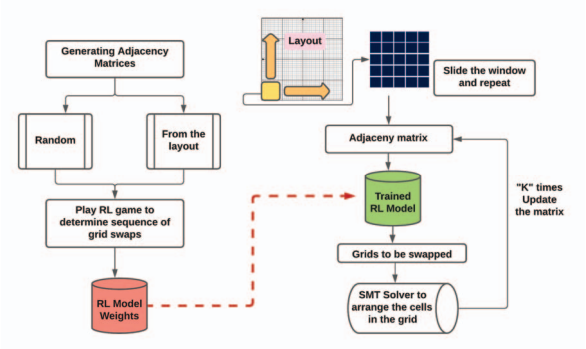


Fig. 2: The flow on the left shows the learning phase of our RL framework and the right is model deployment.

### A. Stage 1 (coarse-grained placement)

We work on a window of the layout at a time, in the pursuit of generating an optimal grid arrangement in the window. Even a $5\times5$ window results in 25! possible grid arrangements, indicating the complexity of the problem. As this window is swept across the layout, the exercise of grid re-arrangement needs to be performed multiple times. We observe that any re-arrangement of grids in a window can be realized using a sequence of grid swaps. This results in a simple and finite action-space. By formulating this re-arrangement task as an MDP (states, actions, rewards and state-transitions), existing RL algorithms can help us in finding optimal sequence of swaps. The advantage of such a learning-based sliding window approach is that the past experiences can be generalized and used to solve new windows. Figure 3 summarizes the state, action and reward representations used in this work.

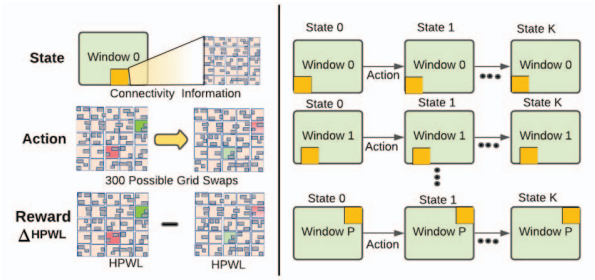


Fig. 3: The components of MDP (left). As the window is slided across the layout, RL agent finds the optimal grid arrangement in each window.

*1) States:* Since Graph Convolution Networks (GCN) adds a layer of approximation and needs to learn the optimal embedding, we use a simple adjacency matrix of the window for the state representation. The elements of the matrix comprehend the connectivity information between the grids of the window; normalized (for generalizability) summation of pin-pin net connections between the grids. For a window with $k\times k$ grids, the size of adjacency matrix $\boldsymbol{C}$ is $k^2\times k^2$, where an element $C_{ij}$ indicates the number of pin-pin connections (normalized in 0-1 range) between grid $i$ and grid $j$.

$$s = \boldsymbol{C}_{k^2\times k^2}, \quad C_{ij} = \frac{\sum Connections(i,j)}{max(\boldsymbol{C})}$$

*2) Actions:* We represent the action using $a_{ij} = Swap(i, j)$ indicating the swap of grid $i$ and grid $j$. In a window with $k \times k$ grids, number of actions $|A|$ can be obtained using the following equation.

$$|A| = \binom{k^2}{2}$$

*3) Rewards:* For minimal wirelength as the objective, one possibility for the reward function is the inverse of wirelength of the associated cells in the window. Since we do not care about absolute goodness while generating the sequence of swaps, we define reward as the HPWL (Half-Perimeter Wire Length) difference of the current state $s$ (in an episode) as compared to the initial state of the episode $s_0$ (from where we initiated the episode). In this milieu, an episode is just a sequence of actions between an initial and terminal state.

$$R(s) = HPWL(s) - HPWL(s_0)$$

### B. Stage 2 (fine-grained placement)

During Stage 1, movement of associated cells in these grids could result in cell overlaps. Figure 4 show the overlap-free placement problem-instance to the SMT Solver [15]. Formally, given placement site coordinates $(s_x, s_y)$ and grids to be swapped $g1, g2$; with grid origins $(X_{g1}, Y_{g1})$ and $(X_{g2}, Y_{g2})$, grid dimensions $(W_{g1} H_{g1})$ and $(W_{g2}, H_{g2})$, the goal is to determine optimal overlap-free locations $(x_i, y_i)$ for the cells of these two grids. The binary auxiliary variables $l_{ij}, r_{ij}, u_{ij}, d_{ij}$ represent the relative placement between cells $i, j$.

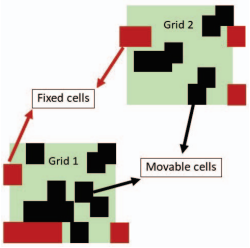


Fig. 4: Stage 2 involves re-arrangement of cells within grids of the window.

**Cell-overlap constraint:**

$$\textbf{ITE}\Big(l_{ij}, (x_i + w_i \le x_j), True\Big), \textbf{ITE}\Big(r_{ij}, (x_i - w_j \le x_j), True\Big)$$

$$\textbf{ITE}\Big(u_{ij}, (y_i - h_j \le y_j), True\Big), \textbf{ITE}\Big(d_{ij}, (y_i + h_i \le y_j), True\Big)$$

$$\textbf{AMO}(l_{ij}, r_{ij}, u_{ij}, d_{ij})$$

**Boundary conditions:**

$$\bigvee \Big\{ \{(x_i \ge X_{g1}) \bigwedge (x_i \le X_{g1} + W_{g1})\}, \{(x_i \ge X_{g2}) \bigwedge (x_i \le X_{g2} + W_{g2})\} \Big\}$$

**Legal placement (snap to grids):**

$$\Big\{ (x_i \; mod \; s_x = 0) \bigwedge (y_i \; mod \; s_y = 0) \Big\}$$

**Net connectivity (bounding box):**
For every net $n \in N$, connecting a set of pins $P(n)$, associated with cells $c \in C$ in grids $g1, g2$; the location of a pin $p$ connected by net $n$ is approximated by $(x_p^n, y_p^n) = (x_c^n + 0.5 * w_c, y_c^n + 0.5 * h_c)$, where $(x_c^n,\ y_c^n)$ are the lower-left coordinates of the associated cell $c$, and $(w_c,\ h_c)$ are the cell dimensions. The upper-right and lower-left xy coordinates of a net $n$'s bounding box $ur_x^n$, $ur_y^n$, $ll_x^n$ and $ll_y^n$ can be expressed as:

$$ur_x^n \ge\ x_p^n\ , ur_y^n \ge\ y_p^n\ \forall p \in P(n), n \in N$$

$$ll_x^n \le\ x_p^n\ , ll_y^n \le\ y_p^n\ \forall p \in P(n), n \in N$$

**Objective:** Half-perimeter wire length (HPWL)

$$min \sum_{n \in N} \Big\{ (ur_x^n - ll_x^n) + (ur_y^n - ll_y^n) \Big\}$$

## V. Experimental Setup

For our policy network, we use Deep Neural Network (DNN + Q Learing) to approximate the Q values. The network takes the adjacency matrix of the window as the input and predicts the Q values for each grid-swap action. To train the DNN, we simulate and generate sets of (State, Action, Reward, Next state, Next Action) with an $\epsilon$-greedy strategy.

**Development Framework:** We use our in-house python-based tool that stores the physical-design database and supports various utilities of a typical RL framework. Since the neural-network libraries and SMT solver is python-based, it offers a single development framework mitigating various tool/data hand-shakes. We perform search over the hyper-parameter space to determine modeling parameters; a fully-connected $300 \times 600 \times 600 \times 300$ neural network, a learning rate of 0.001, discount factor of 0.92, $\epsilon_{initial}$ of 0.5 and $\epsilon_{decay}$ of 0.85, replay buffer size of 1000, training batch size of 128, 1000 training episodes and 200 iterations per each episode.

**Training Setup:** Using the TD update, the policy network is trained to minimize the mean squared loss error (MSE). Each episode of the training process contains a simulated state-transition trajectory, generated with an $\epsilon$-greedy strategy. The MSE decrease during the training process, the cumulative reward at the end of each training episode and the cumulative reward of the final episode is shown in Figure 5. It is important to observe that the model initially explores and eventually learns to exploit good trajectories.

**Inference:** Once the optimal sequence of grid-swaps is determined, we use SMT solver on-the-fly, to assign each cell within the grid to a legal location, without overlaps. To reduce the problem complexity, SMT solver does not disturb the boundary cells since they will be accounted for, as the sliding windows moves over the layout.

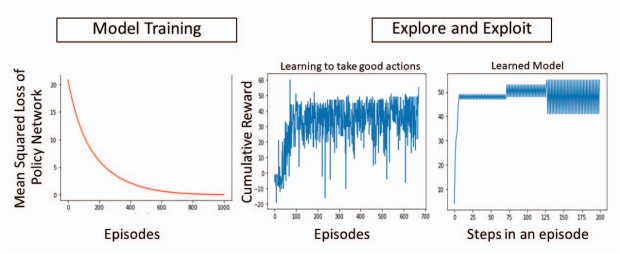


Fig. 5: The MSE of the policy network (left), reward improvement as a function of training episodes (middle) and the final episode's cumulative reward (right).

**Designs:** For our experiments, we use two critical IPs (up to 22K cells) *Multiplier Unit 1* and *Multiplier Unit 2*. In our SOC, we replicate 1000s of these two critical IPs. The improvement in each of these IPs can be scaled by the number of instantiations, to get an estimated improvement on the full-system. We perform two experiments (a) Four optimized global-placement variants of each IP (by varying a combination of physical and logical constraints) serve as the start point. With the same

TABLE I: Wirelength (um) comparison of RLPlace with a commercial tool; optimized output from global placement (Baseline) as the start point for both.

| Design | Baseline | Innovus | RLPlace | Delta (%) |
|---|---|---|---|---|
| Multiplier Unit 1 | | | | |
| *design1* | 11900 | 11520 | 11390 | 1.12% |
| *design2* | 12350 | 11900 | 11780 | 1.01% |
| *design3* | 13460 | 12410 | 12280 | 1.04% |
| *design4* | 14250 | 12520 | 12350 | 1.35% |
| Multiplier Unit 2 | | | | |
| *design1* | 133820 | 126290 | 124810 | 1.19% |
| *design2* | 135790 | 128500 | 126940 | 1.23% |
| *design3* | 139530 | 131540 | 130050 | 1.15% |
| *design4* | 142680 | 134610 | 133210 | 1.05% |

start point, we compare the performance (HPWL) of *RLPlace* with a commercial detailed placer's result. (b) In addition, we also validate if *RLPlace* can improve over two highly-optimized detailed-placement variants resulting from several iterations of tool and manual detailed-placement optimizations.

## VI. Experiments

### A. Improvement over Global-Placement

As shown in Figure 6, we start with a global-placement database from a commercial tool and track the HPWL improvement as the *RLPlace* algorithm is executed. In the plot, X-axis indicates iterations that are "action sequences" (determined by our RL sequence) in each window along with SMT-based refinement. Once we span the entire layout by window sweeping, we repeat the process of sweeping again (within our runtime budget or till the cost metric saturates). The wirelength results of *RLPlace* as compared to the results of a commercial tool's detailed-placement are summarized in Table I. With the same baseline start point, we observe up to 1.35% HPWL improvement as compared to the results of a commercial tool.

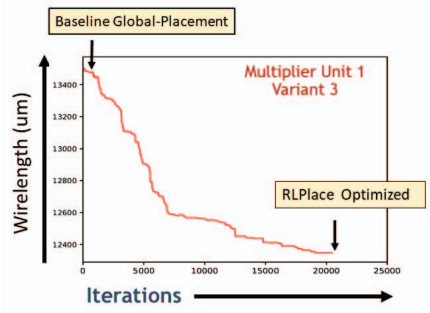


Fig. 6: HPWL improvement of *RLPlace*, with global-placement of *Multiplier Unit 1* as the start point.

.

### B. Improvement over highly optimized detailed-placement implementation

We let the design undergo a commercial tool's detailed-placement optimization, followed by a sequence of manual optimizations. This is used as the start point for our *RLPlace* algorithm. As shown in plots of Figure 7, the resulting solution from *RLPlace* shows at least 1.2% HPWL improvement as compared to the heavily optimized design variants. As shown in Figure 8, the end result of *RLPlace* is overlap-free.

## VII. Conclusions and Acknowledgments

We use active learning to improve the heuristics of the region-based detailed-placement optimization. With global placement as the start point, we achieve up to 1.35% improvement as compared to commercial tool's detailed-placement. With heavily optimized (commercial tool and manual) detailed-placement as the start point, we achieve ~1.2% HPWL improvement. As part of our future work, we seek to use several interesting state-action representations and simultaneously optimize for multiple rewards (routability, power and performance). We would like to thank Zeki Bozkus, Ghasem Pasandi and James Forsyth of Nvidia Corporation, for their valuable contributions to this work.

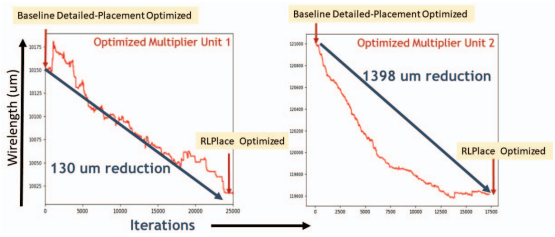


Fig. 7: Plots showing HPWL improvement over highly-optimized detailed-placement design variants of our IPs

.

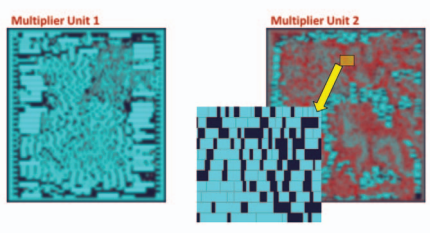


Fig. 8: Overlap-free layout (blurred) of our designs, generated from the RLPlace algorithm.

## References


[1] D. Z. Pan and Y. Lin, "Machine Learning and Its Applications in IC Physical Design", *Proc. EPEPS*, 2016.

[2] R. Kirby, S. Godil, R. Roy and B. Catanzaro, "CongestionNet: Routing Congestion Prediction Using Deep Graph Neural Networks", *Proc. VLSI-SoC*, 2019.

[3] Y. -Y. Huang, C. -T. Lin, W. -L. Liang and H. -M. Chen, "Learning Based Placement Refinement to Reduce DRC Short Violations", *Proc. VLSI-DAT*, 2021.

[4] A. B. Kahng, U. Mallappa, L. Saul and S. Tong, "Unobserved Corner Prediction: Reducing Timing Analysis Effort for Faster Design Convergence in Advanced-Node Design", *Proc. DATE*, 2019.

[5] A. B. Kahng, U. Mallappa and L. Saul,"Using Machine Learning to Predict Path-Based Slack from Graph-Based Timing Analysi", *Proc. ICCD*, 2018

[6] V. A. Chhabria *et al.*, "Template-based PDN Synthesis in Floorplan and Placement Using Classifier and CNN Techniques" *Proc. ASP-DAC*, 2020.

[7] U. Mallappa and C. -K. Cheng, "GRA-LPO: Graph Convolution Based Leakage Power Optimization", *Proc. ASP-DAC*, 2021

[8] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", MIT Press, 1998.

[9] F. Agostinelli, S. McAleer, A. Shmakov and P. Baldi, "Solving the Rubik's cube with deep reinforcement learning and search", *Nature Machine Intelligence* (1), 2019.

[10] A. Agnesina, K. Chang and S. K. Lim, "VLSI Placement Parameter Optimization using Deep Reinforcement Learning", *Proc. ICCAD*, 2020.

[11] Y. Huang, Z. Xie, G. Fang, T. Yu, H. Ren, S. Fang, Y. Chen, and J. Hu, "Routability-driven macro placement with embedded cnn-based prediction model", *Proc. DATE*, 2019.

[12] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H-S. Lee and S. Han. "GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning", *Proc. DAC*, 2020.

[13] A. Goldie, and A. Mirhoseini, "Reinforcement Learning for Placement Optimization". *Proc. ISPD*, 2021.

[14] D. Vashisht, H. Rampal, H. Liao, Y. Lu, D. Shanbhag, E. Fallon, and L. B. Kara, "Placement in integrated circuits using cyclic reinforcement learning and simulated annealing", *Arxiv*, abs/2011.07577.

[15] L. D. Moura and N. Bjorner, "Z3: An efficient SMT solver", *Proc. TACAS*, 2008.

[16] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany and D. Z. Pan, "ABCDPlace: Accelerated Batch-Based Concurrent Detailed Placement on Multithreaded CPUs and GPUs", *IEEE TCAD* (39)12, 2020.