Toward Reinforcement Learning–based Rectilinear Macro Placement Under Human Constraints

Tuyen P. Le
tuyenple@agilesoda.ai
AgileSoDA Company
Seoul, South Korea

Hieu T. Nguyen
trong@agilesoda.ai
AgileSoDA Company
Seoul, South Korea

Seungyeol Baek
sybaek@agilesoda.ai
AgileSoDA Company
Seoul, South Korea

Taeyoun Kim
klumblr@agilesoda.ai
AgileSoDA Company
Seoul, South Korea

Jungwoo Lee
justinlee@agilesoda.ai
AgileSoDA Company
Seoul, South Korea

Seongjung Kim
seongjung@asicland.com
Asicland Company
Suwon, South Korea

Hyunjin Kim
tkv93@asicland.com
Asicland Company
Suwon, South Korea

Misu Jung
msjung92@asicland.com
Asicland Company
Suwon, South Korea

Daehoon Kim
dhkim@asicland.com
Asicland Company
Suwon, South Korea

Seokyong Lee
sean.lee@asicland.com
Asicland Company
Suwon, South Korea

Daewoo Choi
daewoo.choi@hufs.ac.kr
Hankuk University of Foreign Studies
Yongin, South Korea

ABSTRACT

Macro placement is a critical phase in chip design, which becomes more intricate when involving general rectilinear macros and layout areas. Furthermore, macro placement that incorporates human-like constraints, such as design hierarchy and peripheral bias, has the potential to significantly reduce the amount of additional manual labor required from designers. This study proposes a methodology that leverages an approach suggested by Google's Circuit Training (G-CT) to provide a learning-based macro placer that not only supports placing rectilinear cases, but also adheres to crucial human-like design principles. Our experimental results demonstrate the effectiveness of our framework in achieving power-performance-area (PPA) metrics and in obtaining placements of high quality, comparable to those produced with human intervention. Additionally, our methodology shows potential as a generalized model to address diverse macro shapes and layout areas.

CCS CONCEPTS

• **Hardware → Placement**; **Physical design (EDA)**.

KEYWORDS

Reinforcement Learning, Macro Placement, Rectilinear Macros and Layouts, Design Hierarchy

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICCAD '23, October 29–November 2, 2023, CA, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Tuyen P. Le, Hieu T. Nguyen, Seungyeol Baek, Taeyoun Kim, Jungwoo Lee, Seongjung Kim, Hyunjin Kim, Misu Jung, Daehoon Kim, Seokyong Lee, and Daewoo Choi. 2023. Toward Reinforcement Learning–based Rectilinear Macro Placement Under Human Constraints. In *ICCAD '23: Fast Machine Learning for Science Workshop, October 29–November 2, 2023, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Placement of macros is a vital and time-consuming process in chip design and substantially affects the power, performance, and area (PPA) metrics. The increasing demand for customized ASIC chip designs has complicated this task, especially for rectilinear, i.e., non-rectangular, chip layouts and macro shapes. Moreover, development of macro placement algorithms that allow various design constraints, including human-like rules, is challenging and time-consuming.

Classical methods, such as partitioning-based methods and heuristic methods, were widely adopted in macro placers due to their straightforwardness and stability. However, they become time-consuming when dealing with advanced technology nodes and large-scale netlists. Modern macro placers using analytical methods have shown efficiency in representing highly complex objective functions and handling large-scale netlists. However, they lack insights for dealing with the ever-growing diversity of chip designs, continually restarting the process rather than building on prior lessons. Learning-based methods can advance this process because the model learned from past designs so it can predict better placements for unseen designs. Specially, a recent reinforcement learning (RL)-based macro placer proposed by Google [21] has shown potential as a generalized placer to rapidly predict high-quality macro placements for new, unseen designs.

Our methodology, leveraged circuit training (A-CT)-based macro placement, represents an advance to the learning-based approach, but still enjoys advantages from both conventional approaches. Particularly, our placer uses a clustered netlist generated by a grouping engine that uses partitioning-based methods to reduce a large netlist and to maintain some of design hierarchy inherent in the netlist. The placer uses RL to train a deep neural network (the agent) to predict placements for unseen designs and to provide flexibility in dealing with rectilinear layout areas and macros. The placer output is strengthened by a simulated annealing (SA)-based placement engine, which aims for refinements to achieve human-like placement quality by satisfying additional constraints. Finally, our RL-based placement engine trains the agent to maximize the total discounted reward function, which serves as a multiple objective function that integrates various analytical metrics in chip design. Our key contributions are as follows.

- We propose enhancements to CT-based macro placement including a core RL-based engine that is supported by conventional methods for grouping the netlist, and fine-tuning placement to account for human-like constraints (such as placing macros based on design hierarchy guided from the netlist, placing macros at the periphery, and pin accessibility constraints).
- We present methods to unify macro placement using macros and layout areas for general rectilinear shapes. To the best of our knowledge, this is the first work dealing with rectilinear layout areas and macro shapes using RL.
- We propose an enhanced RL model and demonstrate that our RL-based placer can use fewer resources than previously reported and still achieve competitive PPA metrics both from proxies and standard commercial tools.

2 PRELIMINARY

2.1 Review of Previous Work

Non-rectangular macro and areas placement. Macros can have arbitrary shapes, and many studies [6, 16, 20, 22] have proposed algorithms to place rectilinear macros, a special kind of non-rectangular macros with only 90 degrees angles. Given the increasing interest and investment in quantum computing, non-rectangular macro placement is likely to continue to be an important area of research in chip layout[4].

Methods for macro placement. Macro placement methods are classified into three categories: partitioning-based methods, heuristic methods, and analytical methods. Partitioning-based methods [5, 23] use a divide-and-conquer strategy to recursively divide the chip areas and netlist into smaller sub-regions and sub-netlists and then assign each sub-netlist to a sub-region via min-cut objective functions. Partitioning-based methods are scalable, but the min-cut objective function has the drawback that it does not take explicitly into account common performance metrics such as wirelength, density, and congestion. Heuristic methods [8, 28] can consider performance metrics in their optimization functions and potentially reach a good placement. However, they are time-consuming and struggle to deal with very large circuit netlists. Analytical methods [7, 9, 19] model the placement problem using mathematical techniques and use optimization methods to improve an objective function. Modern analytical methods can be efficient and scalable via parallelization on multi-threaded CPUs [9], and by utilizing multiple GPUs [18]. In addition to these three classical categories, learning-based methods [11, 17, 21, 24, 26] have been an active topic in academic research in recent years because of their potential to create a generalized placement model "averaged" from many designs. The Google method[21] trains a graph neural network (GNN) agent using an RL algorithm to place macros, and the trained agent has been shown to adapt well to unseen designs.

Human-like constraints. Physical designers consider various design features to produce high-quality placements. Some placers [12, 13, 27] locate macros following the design hierarchy derived from the RTL model or directly from the cell-level netlist. Other placers locate macros near the periphery of the area to minimize the effects of wire resistance on performance [12, 13]. Pin constraint awareness [13] is also critical in macro placement to ensure that macros are placed in locations that meet their pin connectivity requirements.

2.2 Problem Formulation

We formulate the macro placement problem as a sequential Markov decision process (MDP) in which an RL agent sequentially places T macros ($\mathcal{M}_0$ to $\mathcal{M}_{T-1}$) onto a layout area or chip canvas (we use both terms). The problem has the following components.

- **States:** s_t encodes the observed information collected from the RL environment at the current placement step t.
- **Position action:** a_t^P is drawn from an action space $\mathcal{A}^P$, represented by $N_{max} \times N_{max}$ discrete actions (N_{max} is set to 128) that correspond to all possible locations on the chip canvas where the current macro could be placed without overlapping with already placed elements. However, to reduce the search space, the canvas is generally divided into smaller areas of $N_r \times N_c$ grid cells, where N_r are the grid rows and N_c are the grid columns. N_r and N_c should not be greater than N_{max}.
- **Position mask:** m_t^P is an $N_{max} \times N_{max}$ matrix with 0 or 1 entries that represent which positions are free for placement (value 1) or occupied (value 0). Positions outside the chip canvas are marked (masked) as unplaceable areas.
- **Reward:** the $\mathcal{R}$ is calculated at the end of a placement, meaning once all macros and standard cell clusters have been placed on the chip canvas. It is a negative weighted sum of several proxy costs: wirelength (C_W), congestion (C_C), and density (C_D)

The RL agent modeled by the deep neural network is trained to maximize an expected cumulative reward as follows:

$$\mathcal{J}(\theta) = \mathbb{E}_{p\sim\pi(\theta)}\Big[\mathcal{R}_p\Big], \tag{1}$$

where θ represents the parameters of the RL model, and p represents episodes drawn from the policy distribution $\pi(\theta)$. The placement is constrained by the following four requirements: (1) macros and layout areas can be rectilinear polygons [6, 16, 20, 22]; (2) macros are placed in groups based on the design hierarchy (**design hierarchy bias**); (3) macros are restricted to be placed near the periphery (**peripheral bias**); (4) macros are placed so that pins are not blocked

for their connections with other standard cells and macros (**pin accessibility**).

3 METHODOLOGY

As shown in Figure 1, our framework consists of three distinct engines designed to optimize the processes of standard cell and macro grouping, macro placement, and post-processing placement. First, the grouping engine groups millions of standard cells into several clusters and classifies all the macros into groups based on the design hierarchy. It can be guided by human or automatically inferred from the netlist (as we did not have access to the original RTL). Second, the RL-based placement engine receives input from the grouping engine and produces near-final placements. This engine uses methods to handle rectilinear macros and layout areas, and to satisfy constraints about the design hierarchy, and peripheral bias. Third, the SA-based post-placement engine fine tunes the results generated by the RL placement engine for better pin accessibility, and dead-space minimization.

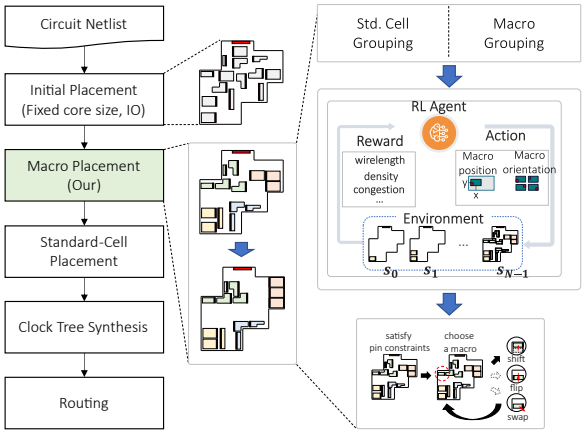

Figure 1: Our macro placer in physical design flow.

3.1 Grouping Engine

The grouping of standard cells is done in a similar way to Google method [21], which utilizes hMETIS [15] as its underlying partitioning algorithm. Our main focus is on grouping macros, where we allow for human guidance. When human guidance is not possible, we propose an alternative method which analyzes the names of all macros in the netlist and identifies the group information by constructing a tree data structure composed of common sub-strings. This is based on the expectation that names will reflect the original hierarchy to some degree. Figure 2 illustrates the resulting tree data structure, which consists of nodes that represent sub-strings found in the macro names. To identify proper groups, a recursive search procedure is implemented at each depth level of the tree, such that all nodes in the same depth level are traversed. If a node at a given depth level has more than one child, it is considered a group. Otherwise, the search continues to deeper depth levels. In this way, a list of unique groups are generated and used in the RL placement engine.

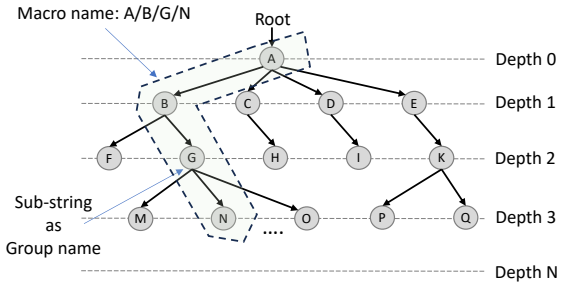

Figure 2: An example of tree data structure.

3.2 Deep RL-based Placement Engine

This engine takes the outputs of the grouping engine to initialize the placement simulator (e.g., environment) in which it performs algorithms to handle rectilinear macros in rectilinear layout areas.

3.2.1 Rectilinear Macros and Area Handling. We propose two algorithms for handling the placement of rectilinear macros. The first algorithm identifies non-placeable areas, and the second algorithm decomposes each rectilinear shape (non-placeable areas and rectilinear macros) into multiple rectangles as illustrated in Figure 3. This representation allows the use of a grid-based masking algorithm (Section 3.2.2) to work with "primitive", i.e. rectangular, blocks and maximize the use of the layout area.

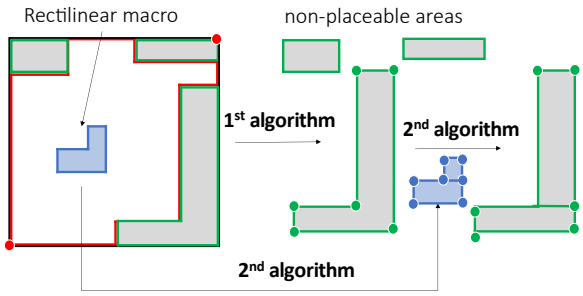

Figure 3: Rectilinear macros and layout handling.

3.2.2 Masking Control Algorithm. Our next step is to control the position mask to ensure that the currently placed macro adheres to the design hierarchy and periphery bias. It is performed repeatedly at beginning of each placement step, and it returns the position mask (m_j) of the current macro to be placed ($\mathcal{M}_j^i$). As illustrated in Figure 4, if the macro is the first from its group, the position mask is the boundary mask ($M_{boundary}$), which allows the macro to be placed only by the closest peripheral grid cells. Beginning with the second macro of a group, to increase pin accessibility and follow the design hierarchy, the algorithm restricts the placeable grid cells to be in close proximity to macros from the same group that have already been placed. To do that, the algorithm loops through all placed macros from that group and selects grid cells with intersecting rows and columns according to the following criteria.

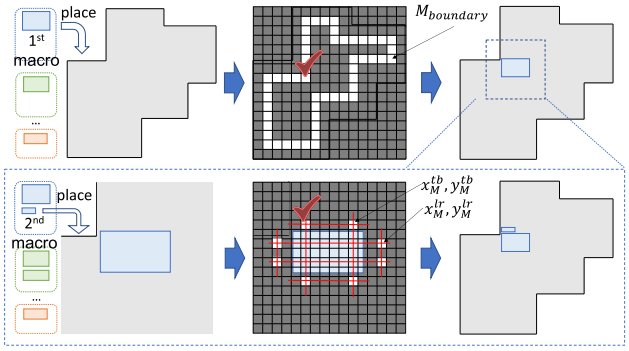

Figure 4: An illustration of the masking control algorithm.

$$
\begin{aligned}
x_{\mathcal{M}}^{lr} &= x_{\mathcal{M}_P} \pm (\lfloor \frac{w_{\mathcal{M}_P}}{2} \rfloor + \lfloor \frac{w_{\mathcal{M}_C}}{2} \rfloor) \\
y_{\mathcal{M}}^{lr} &= y_{\mathcal{M}_P} \pm \lfloor \frac{h_{\mathcal{M}_P} - h_{\mathcal{M}_C}}{2} \rfloor \\
x_{\mathcal{M}}^{tb} &= x_{\mathcal{M}_P} \pm \lfloor \frac{w_{\mathcal{M}_P} - w_{\mathcal{M}_C}}{2} \rfloor \\
y_{\mathcal{M}}^{tb} &= y_{\mathcal{M}_P} \pm (\lfloor \frac{h_{\mathcal{M}_P}}{2} \rfloor + \lfloor \frac{h_{\mathcal{M}_C}}{2} \rfloor),
\end{aligned} \tag{2}
$$

where x, y are the center coordinates and w, h the size of the macros, and $\mathcal{M}_P$ and $\mathcal{M}_C$ are the placed macros and the current macro, respectively. $\{x_{\mathcal{M}}^{lr}, y_{\mathcal{M}}^{lr}\}$ are left and right cells of the placed macros, while $\{x_{\mathcal{M}}^{tb}, y_{\mathcal{M}}^{tb}\}$ are top and bottom cells. Intuitively, the selected cells enable the edge of the current macro to align with the edges of already placed macros.

3.2.3 Neural network model. Our neural network model (Figure 5) uses an Edge-GNN to encode the observed information sent from the environment into a low-dimensional vector representation, enabling it to adapt to unseen data. Our proposal makes a key contribution by incorporating additional information that significantly enriches the macro and design features. Specifically, we introduce three crucial elements: the **group index**, which identifies the group to which a macro belongs; the **pin side**, which specifies the side of the pin on placed macros; and the **corner list**, which represents all points in clockwise order from the area of the macro and layout area so that the rectangular and non-rectangular shapes of macros and area can be unified.

Furthermore, as an additional advancement, we upgrade our model to a **two-head policy** that allows the macro position and orientation to be predicted simultaneously. Actions drawn from the two-head policy create a joint distribution of two categorical distributions that is updated via gradient descent. The PPO[25] algorithm was chosen to train the RL agent because it is robust to hyperparameters and works well with distributed training systems, allowing to speed up the collection of simulation data.

3.2.4 Reward function. Our reward function $\mathcal{R}$ is defined as a negative weighted sum of four proxy costs as follows.

$$
\mathcal{R} = -(\alpha C_W + \beta C_C + \gamma C_D + \omega C_H) \tag{3}
$$

where C_W, C_C, and C_D are the three common proxy costs of wirelength, congestion, and density, respectively. The calculation of wirelength cost, density cost, and congestion cost follows known methods (see [21], [10] for example)in which the wirelength cost is approximated as the normalized half-perimeter wirelength (HPWL); the density cost is approximated as the average density of the densest 10% of grid cells, and the congestion cost is approximated as the average of the top 5% most congested grid cells. In addition to these conventional proxy costs, we propose a novel proxy cost C_H, which is an extra proxy cost added to the reward function to encourage closeness between macros in the same design hierarchy. It is formulated as follows.

$$
C_H = \frac{1}{G} \sum_{g=0}^{G} \frac{\sum_{i,j \in N^g, i \neq j} dist_{ij}}{\sum_{i,j \in N^g, i \neq j} \min(w_{ij}, h_{ij})}, \tag{4}
$$

where G, and N^g are the number of groups and number of macros in group g, respectively. $dist_{ij}$ is the Euclidean distance between two macros in a group and w_{ij}, and h_{ij} are the sum of the width and sum of the height of two macros, respectively. Intuitively, the cost will be high if macros of the same group are far apart on the canvas, and the cost will be lower as the macros move toward their expected positions, with each macro close to the other macros from its group.

3.3 SA-based Post Placement Engine

To achieve human-quality placement in terms of pin accessibility and dead-space minimization, we propose an SA-based post-placement engine, which incorporates the following key ideas: (1) The engine operates on a dense grid of cells that is linearly scaled with the chip canvas, typically around 2000×2000 cells, providing sufficient precision for macro placement in a reasonable runtime; (2) For each iteration, each macro is applied one of three actions with equal probability: *shift* (moving a macro to the closest boundary), *swap* (swapping two macros of the same size, shape, and hierarchy group), or *flip* (flipping a macro along the x or y axis); (3) If there are any pin accessibility violations present after applying an action, we will revert back to the previous situation.

4 EXPERIMENTS

Implementation: We built our framework on top of Google Circuit Training[2]. We use `DreamPLACE`[18] to read designs from `LEF/DEF` files and modify it to get the die-area from the macros and layout areas.

Evaluation designs: We evaluate the framework using three netlists of Ariane CPU[1] provided by [21], [10], and a version we generated using NanGate45 standard-cell library (NG45). Furthermore, we validated the framework on three industrial designs that contain rectilinear macros and layout areas, implemented originally in an advanced technology node from a commercial foundry. Unfortunately, we are not allowed to disclose exact numbers. Finally, in order to assess the generality of the trained model, we performed training and testing using one Ariane with random rectilinear macros and layouts at NG45 and ASAP7 standard-cell library (ASAP7). Program and evaluation designs are released at https://anonymous.4open.science/r/rl4cad-AE0F.

Infrastructure: Building on top of Google Circuit Training allows our framework to run in distributed fashion across multiple servers and GPUs. However, we constrain our resource utilization to typical

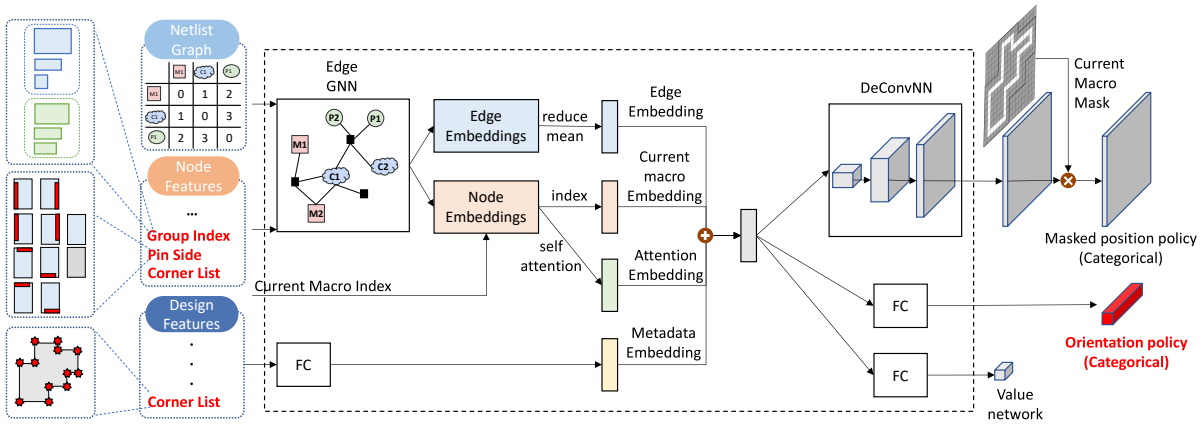

Figure 5: Neural network model of RL agent with extra features and policy head for orientation prediction.

configurations. Specifically, our experiments were conducted on a server with a 64-thread CPU, 512 GB of RAM, and an A5000 GPU with 24 GB of memory. Each run uses 25 collectors to gather simulation data.

Settings: For comparison purposes, we keep almost all training settings the same as the settings from G-CT[21] and TILOS[10]. Additionally, some settings were tuned to work best with our framework: (1) The cost weights α, β, γ, and ω in the reward function[1] were set to 5.0, 1.0, 0.5, and 0.1, respectively. Wirelength weight is set to 5.0 to increase the proportion of wirelength proxy which is hard to optimize under the periphery bias constraint; (2) We select the grid size (N_r and N_c) relative to the chip canvas so that the smallest macro can fit inside a grid cell; (3) The number of nodes and edges in the RL model is chosen relative to the netlist size (e.g., macros, clusters), and does not cause issues during model updates using GPUs.

4.1 Evaluations Using the Netlists of Ariane

Our framework can be applied to designs with rectangular macros and areas without losing generality. Table 1 describes three different netlists for the Ariane CPU (A-GCT, A-TILOS, and A-OURS) and it shows our results for CT Metrics (block placement with proxy costs averaged over nine runs) and the metrics after we complete the layouts (post block placement) with a commercial P&R tool (P&R Metrics, post-route). For the block placement metrics, we compare our results with CT and SA results as published in [2, 10] as well as our own configuration adding the hierarchy cost. Our method can produce placements that show better proxy cost than those published in [2] and [10] for both the original configuration (with 8% and 16.7% improvement on A-GCT and A-TILOS, respectively) and our configuration (with 8.6% and 12% improvement on A-GCT and A-TILOS, respectively).

For the P&R Metrics, we report results for our netlist[2] A-OURS, along with the results using CT, SA, and the block placement stage for three academic placers integrated in OpenROAD[3]: RePLAce, TritonMP and Hier-RTLMP. As indicated, we complete and measure post-route with the same commercial tool for consistency. In three out of four metrics, our framework has the best or second-best results compared to other placers. Figure 6 shows physical placements from the systems we studied. Our placer shows similarities to Hier-RTLMP in term of placing macros based on the design hierarchy, as well as similarities to both Hier-RTLMP and TritonMP in placing macros on the periphery. We do not force macros away from the IO ports and leave that human-like rule for further improvements.

Table 1: Results on different Ariane netlists

Designs	Netlist information				Model Configuration			
	Core Size	# Macros	# IOs	# Clusters	Ori. Grid	Our Grid	# Nodes	# Edges
Ariane (GCT)	356.592 356.640	133	1231	799	35x33	12x18	1200	10000
Ariane (TILOS)	1347.1 1346.8	133	495	810	23x28	23x10	1200	12000
Ariane (OURS)	1445.9 1444.8	133	495	41	-	25x10	200	1100
CT metrics								
Designs	Placer	WL Cost	Den. Cost	Cont. Cost	Hier. Cost	CT Cost	Our Cost	Inference time(h)
Ariane (GCT)	CT[2]	0.1013	0.5502	0.9174	-	1.1102	-	-
	$CT_{(12\times18)}$	**0.0886**	0.5345	0.8852	2.2115	-	1.6411	0.02
	$SA_{(12\times18)}$	0.0963	**0.5057**	0.8446	1.4281	-	1.5523	14
	Our_{RL}	0.0973	0.5088	0.8507	1.0571	1.0315	1.5264	0.02
	Our_{POST}	0.0933	0.5070	**0.8414**	**1.0565**	**1.0209**	**1.4997**	0.1
Ariane (TILOS)	CT[10]	0.1060	0.5280	1.0470	-	0.8932	-	-
	SA[10]	0.0860	0.4990	0.8350	-	0.7533	-	12.5
	$CT_{(23\times10)}$	**0.0975**	0.5860	0.7881	2.9580	-	1.7635	0.02
	$SA_{(23\times10)}$	0.1061	**0.5038**	0.7761	1.5988	-	1.5820	10
	Our_{RL}	0.1092	0.5121	0.7701	**1.3207**	0.7503	1.5752	0.02
	Our_{POST}	0.1045	0.5156	**0.7643**	1.3211	**0.7444**	**1.5522**	0.1
P&R Metrics (post-route)								
Designs	Placer	Area (mm^2)	WNS (ns)	TNS (ns)	# DRC	Power (mW)	Proxy cost	Inference time (h)
Ariane (OURS)	$CT_{(25\times10)}$	1.2806	-0.91	-4833.9	9	585	1.8570	0.02
	$SA_{(25\times10)}$	1.2850	-0.93	-5320.6	9	586	1.7879	14
	RePLAce	1.2812	-1.04	-5423.7	9	584	1.7244	1
	TritonMP	1.2839	-0.89	-5068.2	9	586	1.9621	1
	Hier-RTLMP	1.2823	-0.84	-4632.2	7	586	1.6482	8
	Our	1.2803	-0.86	-4731.0	6	586	1.5807	0.1

Discussions: Here we propose explanations for the results: (1) Our masking strategy and model enhancement show advantages in reserving as much space as possible for standard cells in the middle area. This brings a reduction in density and congestion

[1]The weights α, β, γ in CT weighted cost are 1.0, 1.0, 0.5, respectively, and 1.0, 0.5 and 0.5 in TILOS weighted cost

[2]All metrics from the P&R Tool are reported after the post-route optimization stage without cleaning DRC errors.

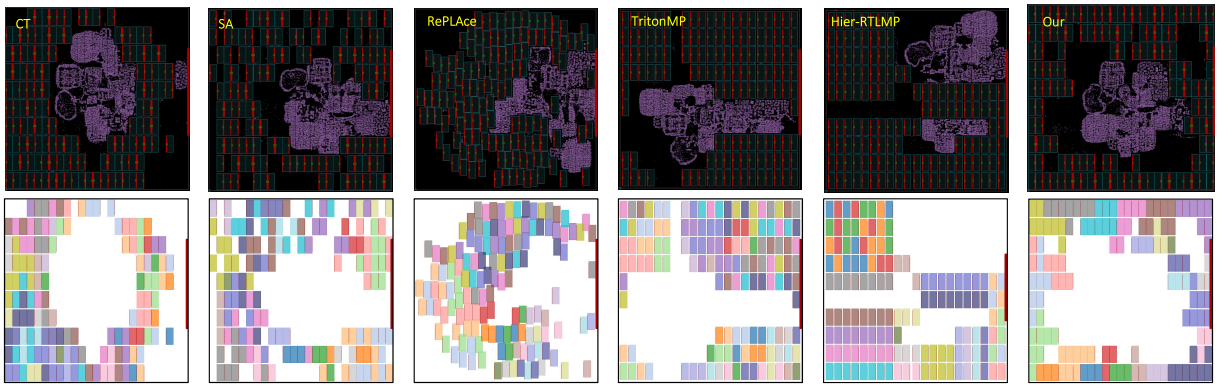

Figure 6: Placements on Ariane-NG45 (freq.=770 MHz, density = 64%) using different academic placers.

costs, and consequently, the total weighted cost. (2) Our placer finds placements that better preserve the design hierarchy, leading to a better hierarchy cost and thus the total weighted cost. (3) Our strategy for choosing grid size results in a significant reduction in density and congestion when compared to previously reported work.

4.2 Evaluation of Industrial Designs

For each design, we selected the top three placements from the training phase and evaluated them using a commercial, state-of-the-art P&R tool. The best results are reported in Table 2 together with results from the timing-driven placer from the commercial P&R tool (Comm) and from the same tool but aided by designers (Human). We only applied reasonable efforts (no "benchmarking") , meaning we wanted to see if results were comparable, and not to try to prove if any such approach could "beat" the others. The netlists cannot be disclosed. Our placer achieved PPA results that are better than those obtained by the designers within a few evaluations and are quite comparable to those achieved by the timing-driven placer from the P&R tool. The output from our placement is entered into the commercial tool without any additional modification (See Fig 1) nor prior knowledge about the power grid or later steps. This may create some DRC errors. Improving this aspect is a long-term plan for our future work.

4.3 Generalization between Technology Nodes

The last experiment assessed the possible generalization of our model to designs containing rectilinear macros and areas. In this study, we restricted the macro shapes to L, J and T patterns[14], and avoided modifying macro shapes on their IO sides. The top plot from Figure 7 shows the training and evaluation curves from 80 synthesized designs used for training and 20 synthesized designs used for testing. The model is well trained to handle rectilinear designs from NG45. When we evaluated with 100 synthesized designs at ASAP7, the model-generated placements improved outperforming the random placements after 100K policy updates (middle plot of Figure 7). Finally, we tested the transferability of the best model trained on NG45 to a more challenging synthesized design at ASAP7. The bottom plot of Figure 7 shows that adapting from a pre-trained model enabled the model to converge faster than training the model on that design from scratch. Figure 8 depicts placements on unseen designs of Ariane at NG45 and ASAP7, which are generated by the trained model and are fine-tuned with SA-based post-placement.

Table 2: Results of industrial designs

Designs	# Macro	# Types	# IOs	# Cells	# Nets	Recti. Layout	Recti. Macros
ic1	89	59	1125	1.5M	1.7M	✓	
ic2	169	97	630	3.8M	4.3M	✓	
ic3	94	21	2207	1.8M	1.8M	✓	✓
Layout Metrics							
Designs	Placer	Area (mm^2)	WNS (ns)	TNS (ns)	# DRC	Power (mW)	Run time(h)
ic1	Human	0.4550	-0.6201	-0.6201	2559	44.6	weeks
	Comm	**0.4495**	**-0.6044**	**-0.6044**	**2491**	46.8	0.5
	Our	0.4548	-0.6178	-0.6178	2695	**43.7**	14
ic2	Human	1.0331	-0.0709	-376.68	**6619**	62.6	weeks
	Comm	1.0256	-0.0739	-302.11	23088	**58.5**	12
	Our	**1.0206**	**-0.0698**	**-288.59**	23542	59.8	28
ic3	Human	5.7972	-0.4193	-1.4651	**3924**	284	weeks
	Comm	5.7965	-0.4544	-15.5075	5038	274	1.7
	Our	**5.7961**	**-0.1402**	**-0.5792**	4313	**269**	14

4.4 Runtime analysis

Table 1 and Table 2 reports the inference time and training time of various placers. Our learning-based placer only needs a few minutes to obtain a good placement. However, to generate a well-trained agent, the agent needs a few hours of training. Specifically, 14 hours are needed with the G-CT netlist and 10 hours with the TILOS netlist, which increases the total runtime. It's worth noting that with the same amount of training time, the reported work required a farm of CPUs and GPUs (20×96vCPUs and 8×V100s), while our placer has been optimized to consume minimal computing resources and conform to common EDA server configurations. Using an adaptation technique from a pre-trained agent is a way to reduce the training time to 6 hours[21]. However, keeping the runtime of a RL-based placer to be the same as that of traditional placers is a challenging problem. We will address this in our future work.

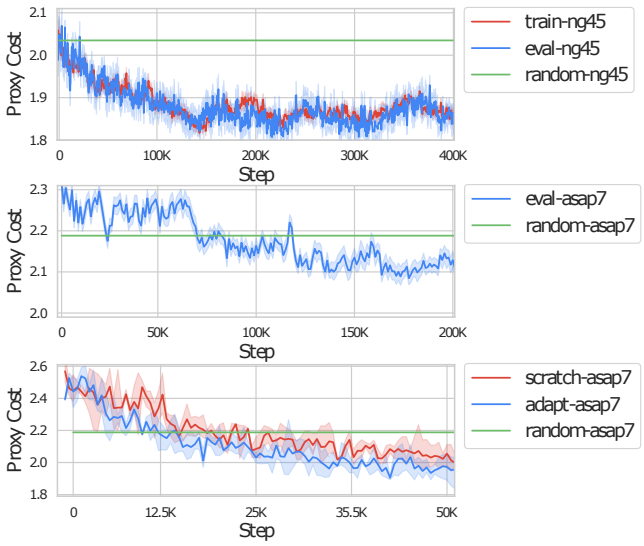

Figure 7: Evaluation curves on Ariane-NG45 (top), Ariane-ASAP7 (middle), and adaptation to a harder Ariane-ASAP7.

Figure 8: Post-placement on Ariane (NG45 and ASAP7).

5 CONCLUSION

This work proposed an RL-based macro placement framework that exceeds the performance of reported work and achieves comparable results to existing baseline algorithms. It strictly respects crucial human-like constraints, with a specific focus on design hierarchy and peripheral bias. Furthermore, this approach has the potential to generalize a learned model to various designs with rectilinear macros and areas. Lastly, our advances, conducted on standard training machines, can drive the research in RL-based placement towards efficiency and affordability allowing IC design house to adopt it without adding excessive computing resources.

ACKNOWLEDGMENTS

We thank Prof. A. Kahng from UC San Diego for sharing the benchmark data, and Dr. A. Domic from Kepler Computing for his useful feedback. This work was supported by the Korean TIPA R&D Program (S3207298)

REFERENCES

[1] [n.d.]. Ariane. https://github.com/openhwgroup/cva6
[2] [n.d.]. Circuit Training. https://github.com/google_research/circuit_training
[3] [n.d.]. OpenROAD. https://theopenroadproject.org/
[4] Marie C. Baca. 2022. EDA Tools For Quantum Chips. https://semiengineering.com/eda-tools-for-quantum-chips/
[5] Mehmet Can Yildiz and Patrick H. Madden. 2001. Improved Cut Sequences for Partitioning Based Placement. In *Proc. ACM/IEEE DAC*. 776–779.
[6] Ray-I Chang and Pei-Yung Hsiao. 1997. VLSI circuit placement with rectilinear modules using three-layer force-directed self-organizing maps. *IEEE Trans. on Neural Networks* 8 (1997), 1049–1064.
[7] T. Chen et al. 2008. NTUplace3: Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints. *TCAD* (2008).
[8] X. Chen et al. 2016. An Adaptive Hybrid Genetic Algorithm for VLSI Standard Cell Placement Problem. In *Proc. ICISCE*. 163–167.
[9] C. Cheng et al. 2019. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE TCAD* (2019).
[10] C. Cheng, A. Kahng, S. Kundu, et al. 2023. Assessment of Reinforcement Learning for Macro Placement. In *Proc. ISPD*. 158–166.
[11] Ruoyu Cheng and Junchi Yan. 2021. On Joint Learning for Solving Placement and Routing in Chip Design. In *Proc. NeurIPS*.
[12] J. Jung et al. 2022. IEEE CEDA DATC: Expanding Research Foundations for IC Physical Design and ML-Enabled EDA. In *Proc. IEEE/ACM ICCAD*. Article 96.
[13] A. Kahng et al. 2022. RTL-MP: Toward Practical, Human-Quality Chip Planning and Macro Placement. In *Proc. ISPD*. 3–11.
[14] M. Kang et al. 1997. General floorplanning with L-shaped, T-shaped and soft blocks based on bounded slicing grid structure. In *ASP-DAC*.
[15] G. Karypis et al. 1997. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Proc. IEEE/ACM DAC*. 526–529.
[16] Sung-Soo Kim et al. 1992. Circuit placement on arbitrarily shaped regions using the self-organization principle. *IEEE TCAD* 11 (1992), 844–854.
[17] Yao Lai, Yao Mu, and Ping Luo. 2022. MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning. In *Proc. NeurIPS*.
[18] Y. Lin et al. 2019. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *Proc. ACM/IEEE DAC*. Article 117, 6 pages.
[19] J. Lu et al. 2015. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. *IEEE TCAD* (2015).
[20] W. Dai M. Kang. 1998. Arbitrary Rectilinear Block Packing Based on Sequence Pair. In *Proc. IEEE/ACM ICCAD*. 259–266.
[21] A. Mirhoseini, A. Goldie, M. Yazgan, et al. 2021. A graph placement methodology for fast chip design. *Nature* 594 (2021), 207–212.
[22] S. Nakatake et al. 1997. Module Placement on BSG-Structure and IC Layout Applications. In *Proc. IEEE/ACM ICCAD*. 484–491.
[23] J. Roy, D. Papa, S. Adya, et al. 2005. Capo: Robust and Scalable Open-Source Min-Cut Floorplacer. In *Proc. ISPD*. 224–226.
[24] C. Ruoyu et al. 2022. The Policy-gradient Placement and Generative Routing Neural Networks for Chip Design. In *Proc. NeurIPS*.
[25] John. Schulman, Filip. Wolski, Prafulla. Dhariwal, et al. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[26] Yang. Shuwen, Y. Zhihao, L. Dong, et al. 2022. Versatile Multi-stage Graph Neural Network for Circuit Representation. In *Proc. NeurIPS*.
[27] Alex. Vidal, J. Cortadella, J. Petit, et al. 2019. RTL-Aware Dataflow-Driven Macro Placement. In *Proc. IEEE DATE*. 186–191.
[28] M. Wang et al. 2000. Dragon2000: standard-cell placement tool for large industry circuits. In *Proc. IEEE/ACM ICCAD*. 260–263.