Effective Macro Placement for Very Large Scale Designs Using MCTS Guided by Pre-trained RL

Jai-Ming Lin
National Cheng Kung University
Tainan, Taiwan ROC
jmlin@ee.ncku.edu.tw

Zong-Ze Lee
National Cheng Kung University
Tainan, Taiwan ROC
n26124963@gs.ncku.edu.tw

Nan-Chu Lin
National Cheng Kung University
Tainan, Taiwan ROC
n26101622@gs.ncku.edu.tw

***Abstract*—Macro placement plays a critical role in modern designs. Some researchers have applied reinforcement learning (RL) techniques to handle this problem. This paper proposes an effective placer based on the Monte Carlo Tree Search (MCTS) algorithm, guided by a pre-trained RL agent. To reduce the complexities of RL and MCTS, we transform the macro placement problem into a macro group allocation problem. Additionally, we propose a new reward function to facilitate training convergence in RL. Moreover, to reduce runtime without affecting placement quality, we use the pre-training result to directly evaluate the placement quality in MCTS. Experiments show that our MCTS-based placer can achieve high-quality results even in the early stages of RL training. Moreover, our method outperforms state-of-the-art placers.**

I. Introduction

Macros occupy a substantial portion of chip area in the VLSI designs, which directly impacts chip performance. Thus, macro placement has become a significant challenge in VLSI design.

A. Previous Works

Various macro placement algorithms have been developed over recent decades. These can be broadly categorized based on their methodologies. The earliest research in macro placement relies on non-deterministic algorithms, primarily using simulated annealing (SA) with different representations to tackle the problem, as seen in works such as [6], [7], [8], [9], [20], and [36]. However, non-deterministic approaches often require significant runtime, making them impractical for large-scale designs.

The second category begins with a global distribution of mixed-size cells through analytical placement formulations, followed by legalizing macros using various techniques. Notable methods in this category include AutoDMP [4], RePlAce [10], [24], [26], and IncreMacro [31]. RePlAce [10] employs the SA algorithm to refine macro positions, while [24] and [26] integrate a simulated evolution algorithm (SE) with corner stitching representation. After achieving a global placement result with [25], IncreMacro [31] applies a gradient-based macro-shifting technique to move certain macros toward the chip boundaries if they are placed near the center. Additionally, AutoDMP [4] utilizes automated parameter tuning to enhance placement outcomes.

Recently, RL-based placement has emerged as a prominent research direction. Mirhoseini et al. [27] combine graph neural networks (GNN) with reinforcement learning to train an agent for macro placement while minimizing wirelength, termed CT. Gu et al. [13] further develop this concept by incorporating convolutional neural networks (CNN) alongside GNN for agent training, resulting in LAMPlace. Lastly, Lai et al. [19] introduce the "wiremask" technique during training to improve wirelength estimation accuracy after each macro is placed, resulting in MaskPlace.

B. Motivation

Silver et al. [33] introduced an innovative approach to playing GO by integrating RL and MCTS. This hybrid approach leverages RL to enhance decision-making efficiency during gameplay without requiring exhaustive explorations. Their method performs several MCTS simulations simultaneously to generate samples for training RL. Subsequently, they use the RL-trained agent to guide MCTS. This iterative process is continuous over time, enabling the RL agent to progressively improve its strategy based on insights gained from samples generated by MCTS.

Considering each macro as a stone and a placement region as a goban, it is straightforward to apply the approach of Silver et al. [33] to handle the macro placement problem. However, as the number of macros increases, the node count in the search tree of MCTS grows exponentially, significantly prolonging the search for optimal solutions. Additionally, cell placement must be executed multiple times within each MCTS sampling. The total runtime will increase significantly as more MCTS processes are executed simultaneously.

C. Our Contribution

Triggered by the approach [33], this paper proposes an innovative macro placement approach by integrating RL and MCTS techniques. To avoid the problems mentioned in Sec I-B while achieving better placement quality, we adopt the following strategies:

- Performing MCTS once after the RL agent is trained.
- Proposing a new reward function to facilitate faster training convergence in RL.

Additionally, we generate a coarsened netlist by addressing the macro group allocation problem rather than placing macros individually. This approach not only reduces the search complexity in MCTS but also minimizes the training steps in each RL episode. Moreover, we apply the Actor-Critic method [29] to pre-train RL agent in order to steer MCTS away from suboptimal paths. Furthermore, by leveraging agent's predictive capabilities for placement quality at non-terminal nodes in MCTS, the number of cell placement runs is significantly reduced, effectively mitigating the runtime challenges.

Experimental results show that our approach can obtain good solutions even in the early training stages of RL. Furthermore, RL training can converge more quickly through our new reward function than setting it as minus wirelegnth. Most importantly, our approach achieves better placement quality than previous works.

978-3-9826741-0-0/DATE25/© 2025 EDAA

II. Our Macro Placement Methodology

Our methodology is composed of three stages as follows:

- Preprocessing Stage:
  - Partitioning into Grids: This step divides a placement region into a grid-based structure.
  - Coarsened Netlist Generation: This step simplifies a netlist into macro groups and cell groups so that the complexity of a design can be reduced while retaining essential connectivity information.
- Pre-training Stage by RL: This stage employs RL to develop an optimal strategy for placing macro groups within the predefined grids.
- Placement Optimization Stage by MCTS: This final stage utilizes MCTS to explore better solutions for the allocation of macro groups based on the strategies learned in the previous stage.

A. Preprocessing

To reduce the complexity of machine learning training and decision-making in macro allocation, we first partition the placement area into $\zeta \times \zeta$ grids (with ζ set to 16 in our experiments). Next, macros and cells are grouped together using a clustering algorithm [24], which considers both the design hierarchy and the spatial distances during placement. The placement is generated using the analytical global placement method [23] prior to the clustering step.

Initially, each macro is treated as an individual group, denoted by g_i^M. At each iteration, the two macro groups, g_i^M and g_j^M, that result in the highest cost based on the score function $\Gamma(g_i^M, g_j^M)$ in (1) are merged. This process continues until either the size of each group exceeds the size of a grid or the value of Γ drops below a user-specified threshold ν (with ν set to 0.001 in our experiment).

$$\Gamma(g_i^M, g_j^M) = \frac{1}{\Delta D(g_i^M, g_j^M)} + \delta H(g_i^M, g_j^M) + \epsilon w(g_i^M, g_j^M) + \kappa \frac{1}{\Delta A(g_i^M, g_j^M) + 1}, \quad (1)$$

where $\Delta D(g_i^M, g_j^M)$ denotes the distance between g_i^M and g_j^M in an initial placement. $H(g_i^M, g_j^M)$ denotes the common parts of the hierarchy names of g_i^M and g_j^M, $w(g_i^M, g_j^M)$ denotes the connectivity between g_i^M and g_j^M, and $\Delta A(g_i^M, g_j^M)$ denotes the area difference between g_i^M and g_j^M. δ, ϵ and κ are user-specified parameters (they are set to 0.001, 0.0003, and 1 in our experiment, respectively).

Similarly, we start by treating each cell as an individual group, denoted by g_i^C. At each iteration, the two cell groups, g_i^C and g_j^C, that result in the highest cost according to the score function $\varphi(g_i^C, g_j^C)$ in (2) are merged. The termination condition for cell grouping is identical to that for macros.

$$\varphi(g_i^C, g_j^C) = \frac{1}{\Delta D(g_i^C, g_j^C)} + \varrho \frac{w(g_i^C, g_j^C)}{A(g_i^C) + A(g_j^C)}, \quad (2)$$

where $\Delta D(g_i^C, g_j^C)$ denotes the distance between g_i^C and g_j^C in an initial placement, $w(g_i^C, g_j^C)$ denotes the connectivity between g_i^C and g_j^C, and $A(g_i^C)$ denotes the area of the cell group. ϱ is a user-specified parameter (it is set to 1 in our experiment).

B. Macro Legalization

This section introduces a method to determine exact locations of macros after macro groups are allocated to grids by RL or MCTS.

Our approach is composed of three steps. First, the locations of cell groups are determined by the quadratic programming (QP) after the locations of macro groups fixed at center of the corresponding grids. Next, macro groups are decomposed and the relative locations of macros are found by QP after cell groups are fixed. To avoid moving macros far away from their original locations, the boundaries of macros are limited inside their own grids. Finally, movable macros inside each grid are legalized separately in the last step. The horizontal (vertical) geometric relations between macros are identified and recorded by the sequence pair representation [28], which is denoted by (S^+, S^-). Then, the overlaps between macros are removed by the linear programming (LP) while minimizing wirelength as follows [34]:

$$\begin{aligned} &\min \sum_{n \in N} \lambda_n \times hW(n) (\text{or } vW(n)) \\ &\text{s.t. } (S^+, S^-) \text{ is satisfied,} \end{aligned} \quad (3)$$

where $hW(n)$ ($vW(n)$) denotes the wirelength of a two-terminal net n in the horizontal (vertical) direction. Note that the x and y coordinates of macros are estimated respectively since they are independent. λ_n denotes the weight of a net n.

C. Cell Placement

Finally, this section describes the cell placing method. After all the macros have been placed, we leverage mixed-size placer DREAMPlace [25] to generate full placement result, which also returns a measured wirelength value.

III. Pre-training Stage by RL

This section illustrates our pre-training stage by RL.

A. RL and Macro Placement

RL is a machine learning paradigm which can be formed as a Markov Decision problem (MDP), where an **Agent** is trained to make optimal decisions through interacting with a dynamic **Environment** [17] [30]. An agent receives a state s_t at time t from an environment. s_t represents current placement after $t-1$ macro groups have been placed. The agent selects an action a_t according to its own policy π to place a new macro group M_t on a grid. Subsequently, the **Environment** returns a new placement s_{t+1} and provides a reward r_t that reflects the quality of a_t.

B. State representation

A state s_t is composed of three components as follows:

- **Current placement condition** s_p^t: This reflects the placement utilization of each grid. The utilization of a grid ranges from 0 to 1, indicating its occupied area relative to the total area of the grid.
- **Available placement area** s_a^t: This shows the availability of each grid for placing the next macro group M_t. The value ranges from 0 to 1, lower value indicates less availability for placing M_t. The result of s_a^t depends on the shape of M_t.
- **Sequence number** t: This denotes the identification number of the macro group to be placed. This number is served as a position embedding.

To generate s_p^t, we first align all allocated macro groups to the left-bottom corners of the corresponding grids. Then, the utilization of each grid in s_p^t is estimated by dividing the

occupied area of macro groups by the grid area. The value is capped at 1 when the occupied area exceeds the area of grid.

Then, s_a^t is generated based on the next macro group M_t and s_p^t. First, a matrix s_m^t is constructed according to the shape of M_t. The dimension of s_m^t corresponds to the number of grids occupied by M_t, and the utilization of each grid in s_m^t is estimated in a manner similar to that of s_p^t. Finally, the value of each grid g in s_a^t, denoted by $\mathcal{V}(g)$, is estimated using Eq. (4) after aligning s_m^t with the grid g in s_p^t.

$$\mathcal{V}(g) = \sqrt[n]{\prod_{i=1}^{n}(1 - s_m^t(g_i)) \times (1 - s_p^t(g_i))}, \tag{4}$$

where n is the number of grids occupied by M_t. If the value of $\mathcal{V}(g)$ is lower than 0, it is replaced with 0.

Fig. 1 provides an example to illustrate the elements of a state $s_t =< s_p^t, s_a^t, t >$. s_p^t and s_m^t are constructed based on the current placement and the next macro group M_t, respectively. Note that the dimension of s_m^t is 2×1 since M_t occupies two grids (i.e. $n = 2$).

To generate s_a^t, we slide s_m^t over each grid g in s_p^t to estimate its $\mathcal{V}(g)$. For example, the value $\mathcal{V}(g)$ of the grid g in the right-bottom corner is estimated by $\sqrt[2]{\big((1-0.6)\times(1-0.5)\big)\times\big((1-0.3)\times(1-0.25)\big)} = 0.32$ after aligning s_m^t with g.

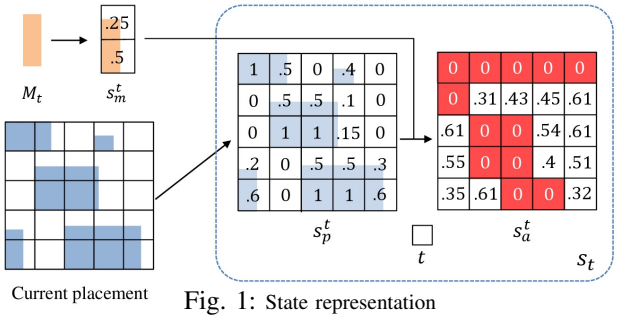


Fig. 1: State representation

C. Agent network

Our RL adopts the Actor-Critic training scheme [29]. Because the reward of a state can be estimated without requiring to place all macro groups, this scheme is more suitable for our MCTS-based placement optimization (see Sec. IV).

The Actor-Critic training scheme consists of policy networks (denoted as π_θ) and value networks (denoted as V_θ), where θ represents the parameters of the agent. Fig. 2 shows the architectures of these two networks. The interaction between the two networks is described as follows:

- **Policy Network** π_θ: Placement characteristics of the current placement condition s_p^t are extracted through a convolution block, a residual tower, a second convolution block, and a fully connected layer. These are then multiplied by available placing area s_a^t. Finally, a Softmax layer is applied to obtain probability distribution $p_{\theta,t}$.
- **Value Network** V_θ: Sequence t is first used for position embedding. Current placement condition s_p^t and the output of the residual tower are then combined to extract state features. These features are processed through a convolution layer and a multilayer perceptron (MLP) to output the predicted value $v_{\theta,t}$.

The information of the blocks in the networks are shown in Table I.

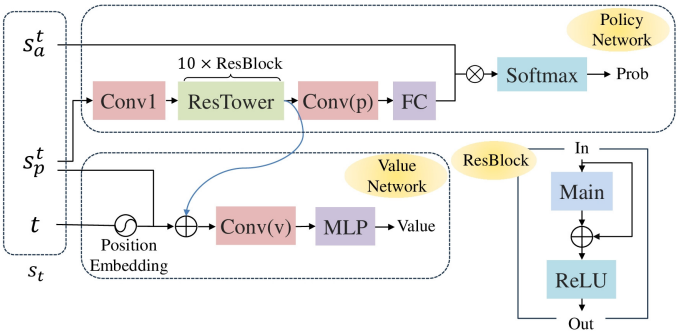


Fig. 2: Architectures of policy and value networks, where the architecture of a ResBlock is shown in the right-bottom side of the figure.

Block	Type	Kernel	Output shape
ResBlock			
Main	Conv2D + BN	3 × 3	(16,16,128)
	ReLU	-	-
	Conv2D + BN	3 × 3	(16,16,128)
Policy Network			
Conv1	Conv2D + BN	3 × 3	(16,16,128)
ResTower	10 × ResBlock	-	(16,16,128)
Conv(p)	Conv2D + BN	1 × 1	(16,16,2)
FC	Linear	-	16 × 16
Value Network			
Conv(v)	Conv2D + BN	1 × 1	(16,16,1)
MLP	Linear + ReLU	-	16
	Linear + ReLU	-	16 × 16
	Linear	-	1

TABLE I: Information of all blocks shown in Fig 2.

D. Training scheme

During the training process, after a partial placement result s_t is received, π_θ and V_θ each produce a probability distribution $p_{\theta,t}$ and an estimated value $v_{\theta,t}$, respectively. The agent then selects an action a_t according to $p_{\theta,t}$, which allocates the next macro group to a grid at step t.

Subsequently, $v_{\theta,t}$ is recorded to influence π_θ. Specifically, the policy loss function is estimated as follow:

$$L_{policy}(\theta) = \sum_t [-log(p_{\theta,t}(a_t)) \times A_t] \tag{5}$$

Here, A_t is the advantage of a_t, which not only considers the actual reward, but also considers the estimated value. This helps π_θ progress more quickly. A_t is estimated using the following equation:

$$A_t = R_t - v_{\theta,t}, \tag{6}$$

where R_t is an actual reward. Moreover, V_θ has its own loss function L_{value}, which is the expected value of A_t. By minimizing L_{value}, we enable V_θ to estimate $v_{\theta,t}$ more accurately.

$$L_{value}(\theta) = E[{A_t}^2] \tag{7}$$

The objective of Actor-Critic algorithm is to minimize the loss function $L(\theta)$ defined as follow:

$$L(\theta) = L_{policy}(\theta) + L_{value}(\theta) \tag{8}$$

This loss function indicates that the policy and the value networks will be trained simultaneously, as they share some common layers, as shown in Fig.2. In our experiment, θ is updated every 30 episodes.

E. Reward function

According to our observation, training in RL will converge more quickly if the average value of rewards is slightly above

zero. To achieve this target, we will play RL randomly for 50 episodes before training and collect the maximum, minimum, and average values, which are denoted by δ, γ, and Δ, respectively. During training, after each episode is complete, a reward value is estimated by the function $\mathfrak{D}(\mathcal{W})$ in the following:

$$\mathfrak{D}(\mathcal{W}) = \frac{-\mathcal{W} + \Delta}{\delta - \gamma} + \alpha \tag{9}$$

where $\mathcal{W}$ denotes the wirelength of a placement which is estimated in the half perimeter wirelegth model. α is a user-specified parameter. To make each reward value above zero, the value α was set in the range [0.5, 1] in our experiment. Our experiment will demonstrate that RL will converge more quickly by a reward estimated by Eq. (9) (see Sec. VI-A).

Finally, the reward value for each non-terminal step in the same episode is set according to the value obtained in the last step. This is because we use the value as the place quality of a partial placement in our post-optimization stage (see Sec. IV).

IV. Placement Optimization Stage by MCTS

This section introduces our placement optimization stage by MCTS algorithm.

A. Concept of MCTS

MCTS allocates macro groups to grids by construction of a searching tree T. Each node in T corresponds to a partial placement result, with the root of T denotes an empty placement. The node in the t-th level represents a partial placement state s_t, indicating that $t-1$ macro groups have been placed. The search process involves consecutively adding new nodes to T, and all macro groups are allocated after the search reaches the final level of the tree. Every edge(s_p,s_q) contains specific information about the transition from the partial state s_p to the next state s_q as follows:

- $N(s_p, s_q)$ represents the number of times that edge (s_p, s_q) has been traversed.
- $P(s_p, s_q)$ represents the probability of going from s_p to s_q. This probability is obtained according to π_θ of agent.
- $W(s_p, s_q)$ records the total value accumulated every time edge(s_p, s_q) is traversed. These values are obtained through **evaluation** (see Sec. IV-B3).
- $Q(s_p, s_q)$ denotes the average value of edge (s_p, s_q), which can be obtained by Eq. (12).

here s_p and s_q denote the parent and child nodes of the edge.

B. Our Algorithm based on MCTS

T is expanded as **exploration** is performed. Each time MCTS needs to call **exploration** for γ times to find the next state s_{t+1} from a target state s_t. Unlike traditional methods, our **exploration** is guided by the pre-trained agent obtained in Sec. III. Each **exploration** is composed of four steps: *selection*, *expansion*, *evaluation* and *backpropagation*, which will be illustrated in the following:

1) Selection: The selection step picks a child state s_q from a target state s_t according to the equation bellow:

$$s_q = \text{argmax}\big(Q(s_t, s_i) + U(s_t, s_i)\big), \forall s_i \in child(s_t) \tag{10}$$

$child(s_t)$ represents the child nodes of s_t. $Q(s_t, s_i)$ is the average value of choosing s_i, with a higher value indicating a higher probability of selecting s_i. Additionally, $U(s_t, s_i)$ signifies an exploratory strategy and is estimated by PUCT [32] formula as below:

$$U(s_t, s_i) = c \times P(s_t, s_i) \times \frac{\sqrt{\sum_{s_j \in child(s_t)} N(s_t, s_j)}}{1 + N(s_t, s_i)} \tag{11}$$

where c is an user-specified parameter (it is set as 1.05 in our experiment). P denotes a predicted probability to select a node and it is obtained from π_θ of the pre-trained agent. We tend to allocate a new macro group to the grid which is more favorable according to the agent. Finally, the last term divides the number of times that all edges associated to a parent node has been explored by the number of times this specific edge has been explored. Its purpose is to decrease the probability of choosing an edge if it has been chosen frequently in the past.

After s_q is selected and if it has been explored before, the selection step is called again to find s_q's child node. This process continues recursively until an unexplored node s_s is found.

2) Expansion: After s_s is selected, expansion step marks it as explored and expands all of its child nodes. Each child node corresponds to a possible grid position for placement of a macro group. Every edge between s_s and its child nodes have initial values of Q, N and W as 0. The value of P for each edge is initialized as $p_{\theta,s}$ according to π_θ of the pre-trained agent.

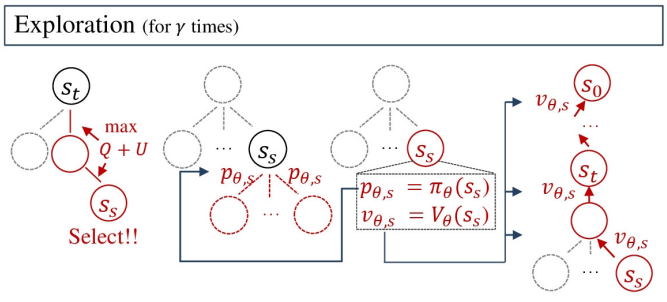
3) Evaluation: The evaluation step assesses the potential value of the placement corresponding to the selected state s_s. Traditional MCTS approach continues the search tree from s_s toward the end with random choices, which can significantly increase the evaluation time. Instead, we leverage the pre-trained agent to predict the quality of s_s. For non-terminal states, we can directly use the output of V_θ, $v_{\theta,s}$ as its evaluation. Only in terminal states do we need to perform the actual placement process to determine the exact value. This approach reduces runtime significantly by avoiding unnecessary computations in non-terminal states.

4) Backpropagation: After the result $v_{\theta,s}$ of s_s is evaluated, the backpropagation step updates the information of every edge along the path from selected node s_s to root node s_0 (denoted as $\mathbb{P}_{s,0}$). The updated information includes:

$$\begin{aligned} N(s_i, s_j) &= N(s_i, s_j) + 1, \\ W(s_i, s_j) &= W(s_i, s_j) + v_{\theta,s}, \\ Q(s_i, s_j) &= \frac{W(s_i, s_j)}{N(s_i, s_j)}. \end{aligned} \tag{12}$$

Here, (s_i, s_j) denotes every edge along $\mathbb{P}_{s,0}$. The new Q value thus represents the updated average value after this exploration and will be used in the selection function in Eq. (10) during the next exploration.

Fig. 3 shows an example of **exploration** from a target s_t, and s_t's next state s_{t+1} is found after γ times of **exploration**.



(a) *Selection* (b) *Expansion* (c) *Evaluation* (d) *Backpropagation*

Fig. 3: Flow of **exploration**. (a) Selection step picks node with the maximum $Q+U$ value by Eq. (10) iteratively until an unexplored s_s. (b) Expansion step expands all edges of s_s. (c) Evaluation step uses pre-trained agent to evaluate $p_{\theta,s}$ and $v_{\theta,s}$ of s_s. (d) Backpropagation step assigns $p_{\theta,s}$ to every expanded edge of s_s while propagating $v_{\theta,s}$ back to the root as the red arrow.

V. Our Macro Placement Algorithm

Algorithm 1 presents the pseudocode for our algorithm. Initially, the preprocessing stage divides the placement region into a set of grids and generates a coarsened netlist (Lines 1-2). During this stage, macro groups are recorded in a list M and arranged in the non-increasing order of their areas, with the area of a group is estimated by summing the areas of its components. The macro placement sequence in the RL agent are determined by M Since macro groups with larger areas are more likely to obstruct other groups, they are given higher priority to enhance placement result.

Algorithm 1 Our Placement Flow

Input: lists of macro and cells M and placement region R, full netlist N
Output: legal placement result L

1: Partition a placement region R into grids ▷ Start preprocessing
2: Divide a netlist into macro groups and cell groups, where macro groups are sorted and recorded in M
3: **while** halting condition is not met **do** ▷ Start pre-training
4: **for** every macro groups M_t in M **do**
5: Place M_t by π_θ, store $p_{\theta,t}$ and $v_{\theta,t}$
6: **end for**
7: Legalize macros and place cells
8: Estimate reward r_n and set each step reward $r_t = r_n$
9: Update agent according to $p_{\theta,t}$, $v_{\theta,t}$, and r_t every 30 episodes
10: **end while**
11: Initialize a searching tree T ▷ Start MCTS
12: **for** every macro groups M_t in M **do**
13: Do explorations for γ times and place M_t
14: **end for**
15: Allocate macro groups into L by tracing back the searching path in T
16: Legalize macros and place cells on L
17: **return** L

To enhance the placement strategy in MCTS, an RL agent is trained in advance (Lines 3-10). In each episode (Lines 4-6), every macro group $M_t \in M$ is sequentially placed according to π_θ, with the probability $p_{\theta,t}$ of choosing an action and the estimated value $v_{\theta,t}$ being recorded. Once all macro groups are placed, the exact macro locations are determined (see Sec II-B), and cells are subsequently placed by DREAMPlace [25] to estimate wirelength. The reward of the final step, r_n, is then estimated and used as the reward for each intermediate step. Finally, the agent is trained using the Actor-Critic scheme (Line 9) based on $p_{\theta,t}$, $v_{\theta,t}$, and r_t (see Sec. III-D).

The agent training process can be halted at any time specified by the user and integrated into the MCTS process. During MCTS (Lines 11-15), the pre-trained agent is used to optimize macro group allocation. We initialize a search tree T (Line 11) and sequentially place each macro group M_t. During each iteration, we conduct γ exploration attempts (Lines 12-14) until the optimal grid for placing M_t is identified (see Sec. IV-B). The macro group allocation in L is obtained by tracing back the search path in T (Line 15). Finally, macro legalization and cell placement are performed in the same way as in the agent training process, which then serves as our ultimate placement on L (Lines 15-16).

VI. Experiment

Our algorithm was implemented in Python and ran on a Linux workstation equipped with an Intel® Xeon® Silver 4110 2.1GHz CPU and 400GB of memory. Additionally, we trained our RL agent on an NVIDIA Tesla T4 GPU with 16GB of memory.

Our experiment is divided into four parts. First, we show that the training process for handling macro group allocation converges more quickly after adjusting the reward value through our tuning function compared to using an intuitive value. In the second part, we demonstrate the effectiveness of macro group allocation using MCTS, even though its agent is obtained in the early training stages. In the last two parts, we compare our placer with previous works according to industrial [24] [26] and academic [2] [3] benchmarks, respectively.

TABLE II: Comparisons of heuristic SE-base Macro Placer [26] and analytical DREAMPlace [25] with our approach.

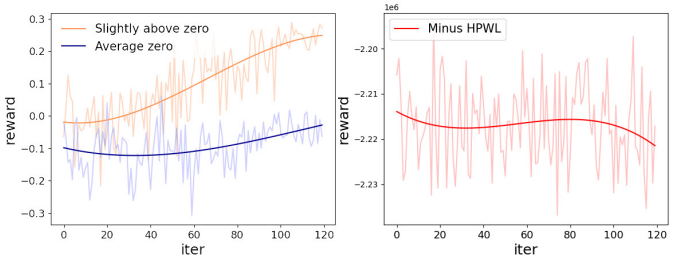
Cir.	# Mov. Macros	# Prep. Macros	# I/O Pads	# Stand. Cells	# Nets	[26]	[25]	Ours
						WL(10^7 μm)		
Cir1	30	13	130	157K	181K	1.12	1.24	1.14
Cir2	71	47	365	1098K	1126K	6.55	7.14	6.33
Cir3	55	15	219	232K	235K	1.28	1.77	1.14
Cir4	38	15	169	321K	327K	1.70	2.30	1.60
Cir5	32	12	351	347K	352K	0.81	0.85	0.93
Cir6	66	3	481	209K	217K	1.07	1.34	0.75
Nor.	-	-	-	-	-	1.05	1.23	1

A. Reward function

This section explores the impact of different reward functions on the training convergence of RL. Fig. 4 illustrates the training progress of RL on the ibm10 [2] [3] netlist using various reward functions: Eq (9), Eq (9) without the variable α, and an intuitive function $-\mathcal{W}$, where $\mathcal{W}$ denotes the wirelength of a placement.

In Eq (9), every reward value remains slightly above zero, but it approaches zero after eliminating α from the equation. The x-axis in Fig. 4 represents the number of training iterations in RL, where each iteration equivalent to one episode. The y-axis depicts the corresponding reward values. Due to large differences in reward value scaling, results based on Eq (9) and the intuitive function are presented separately in Fig. 4 (a) and Fig. 4 (b), respectively. Note that the 50 episodes before training are not included in the figure (see Sec. III-E).

In these figures, the orange (blue) line represents the scenarios where reward values are slightly above (close to) zero, while the red line indicates results of the intuitive method. Notably, in Fig. 4 (a), the orange line rises more rapidly compared to the blue one. Also, the red line's results do not converge throughout the entire observation period. This suggests that the agent may perceive all actions as inadequate if it consistently receives negative rewards.



(a) Average equals and slightly above zero (b) Minus HPWL

Fig. 4: Convergence speed of RL in ibm10 based on different reward functions. (a) Our reward function. (b) An intuitive function as $-\mathcal{W}$.

B. Effectiveness of post-optimization by MCTS

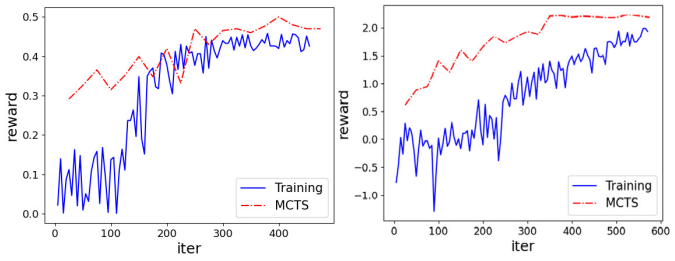
This section demonstrates that our post-optimization method using MCTS can yield promising results even when the agent's training is suboptimal or incomplete. To demonstrate this, we use a partially trained agent at every 35 iterations during the training process to guide the MCTS. We then compare the results obtained through this approach with those from the original RL results.

Fig. 5 (a) and (b) show the outcomes of RL and MCTS at various training stages for ibm01 and ibm06, where MCTS is applied based on the agent of RL at that state. The red dashed line represents the MCTS results, while the blue solid line represents the RL training results. Two key observations can

TABLE III: Comparisons of HPWL ($\times 10^7$) on ICCAD04 Benchmarks

	ibm01	ibm02	ibm03	ibm04	ibm06	ibm07	ibm08	ibm09	ibm10	ibm11	ibm12	ibm13	ibm14	ibm15	ibm16	ibm17	ibm18	Nor.
# Macros	246	280	290	608	178	507	309	253	786	373	651	424	614	393	458	760	285	-
# Standard Cells	12K	19K	22K	26K	32K	45K	51K	53K	68K	70K	70K	83K	146K	161K	183K	184K	210K	-
# Nets	14K	19K	27K	31K	34K	48K	50K	60K	75K	81K	77K	99K	152K	186K	190K	189K	201K	-
CT [27]	0.33	0.62	0.93	0.97	0.70	1.30	1.56	1.56	5.47	2.16	4.81	2.96	5.12	5.45	6.86	8.18	4.31	1.39
MaskPlace [19]	0.24	0.47	0.71	0.78	0.55	0.95	1.20	1.23	3.67	2.02	3.97	2.46	3.02	4.57	5.84	6.43	3.98	1.10
RePlAce [10]	0.22	0.52	0.63	0.71	0.57	0.98	1.11	1.19	3.94	1.65	3.04	2.18	3.39	4.43	5.15	6.27	3.97	1.01
Our Methodology	0.20	0.48	0.67	0.73	0.57	0.96	1.13	1.17	2.77	1.76	3.38	2.15	3.28	4.37	4.91	5.96	3.84	1.00

be made from the figure: First, the MCTS post-optimization consistently outperforms RL at every training stage, demonstrating its effectiveness. Additionally, it shows that satisfactory results can be achieved using the MCTS technique even when the training in RL has not yet complete. For instance, MCTS can achieves a reward of nearly 0.3 in the early-stage which is significantly higher than the 0.1 reward achieved by RL at the same stage in Fig. 5 (a) according to ibm01. More, the value is close to the final reward 0.4 achieved by RL. A similar result can be observed in Fig. 5 (b) for the ibm06.



(a) ibm01 (b) ibm06

Fig. 5: Rewards of MCTS at any training stages compared to those obtained by RL. (a) ibm01. (b) ibm06

C. Comparison with Previous Works in Industrial Benchmarks

This experiment compares our approach with the state-of-the-art macro placers including SE-base Macro Placer [26] and DREAMPlace [25] according to industrial benchmarks [24] and [26]. These benchmarks contain preplaced macros and design hierarchy. Columns 2-6 of Table II show the information of the benchmarks, which include numbers of movable and preplaced macros, number of pads and standard cells, and number of nets, respectively.

The comparison results are shown in the right part of Table II, where columns 7 and 8 respectively display the results of SE-based Macro Placer and DREAMPlace while our result is shown column 9. Note that the *deterministic* flag in DREAMPlace was enabled to ensure stable results. Due to parser errors in DREAMPlace which cannot read multiple LEFs files, we do not show the results of Cir.7-8. Our placer macro outperforms SE-based Macro Placer and DREAMPlace. Specifically, the wirelengths of SE-based Macro Placer and DREAMPlace are 5% and 23% larger than ours, respectively (see columns 7-9). We guess that DREAMPlace obtains worse result than ours since they do not consider design hierarchy. Although SE-based Macro Placer also considers design hierarchy during placement, our approach which combines the advantages of a smart pre-trained agent of RL and solution space exploration scheme of MCTS can outperform its random search scheme.

D. Comparison with Previous Works in Academic Benchmarks

We evaluate our methodology against leading placers, including AI-based approaches such as CT [27] MaskPlace [19], as well as analytical methods like RePlAce [10], using the ICCAD04 Mixed-size Placement benchmarks [2] [3]. We exclude comparisons with [4] [24] [26] [31], due to their use of proprietary industrial benchmarks, which are unavailable to us.

TABLE IV: The runtime (in minute) of each benchmark.

Cir.	Runtime (m)	Cir.	Runtime (m)	Cir.	Runtime (m)
ibm01	27.07	ibm07	66.10	ibm013	36.71
ibm02	34.8	ibm08	48.51	ibm14	55.48
ibm03	28.16	ibm09	40.33	ibm15	22.07
ibm04	82.43	ibm10	91.7	ibm16	25.4
ibm05	-	ibm11	47.88	ibm17	79.42
ibm06	18.29	ibm12	50.02	ibm18	30.01

Additionally, [13] randomly converts 50 macros into movable ones, preventing us from making fair comparisons. Details of the benchmarks are listed in the rows 2-4 in Table III, where the number of macros (standard cells) ranges from 178 to 786 (12K to 210K) while the number of nets ranges from 12K to 210K. Notably, these benchmarks do not contain design hierarchy information and lack preplaced macros. Moreover, ibm05 is not included in this analysis as it does not contain any macros.

The wirelength comparison results are presented in Table III, with CT, MaskPlace, RePlAce listed in rows 5-7 and our approach in row 8. Note that the results of CT and RePlAce are sourced from [12]. Our method demonstrates a significant advantage, with CT's wirelength being 39% longer than ours. Unlike CT, which relies solely on RL for macro placement, our approach incorporates MCTS guided by an agent trained through RL. While MaskPlace also utilizes RL for macro placement, it surpasses CT by employing a wiremask technique that approximates the increase in wirelength for each occupied grid after placing each macro. However, MaskPlace's wirelength remains 10% longer than ours, underscoring the effectiveness of our approach. While both CT and MaskPlace train their agents on individual macros, our agent is trained on macro groups, offering a more holistic view. Additionally, we use MCTS to explore optimal solutions based on this pre-trained agent. Furthermore, our method outperforms the analytical approach of RePlAce [10]. Although the improvement is only 1%, it marks a significant milestone, as it is the first instance where an AI-based method has outperformed an analytical approach in public academic benchmarks.

Finally, we present the runtime of our MCTS in TableIV. The agent that guides the MCTS is trained until convergence, with training times ranging from 3 to 10 hours, depending on the number of macros to be placed. According to the table, the runtime of MCTS correlates with the number of macros in the benchmarks. It takes up to 92 minutes to determine macro locations for ibm10, which has the largest number of macros, while it requires 18 minutes for ibm06, which has the fewest macros. These results indicate that our approach is both efficient and effective.

VII. Conclusion

This paper proposes an effective model-based RL approach by integrating RL and MCTS for the macro placement in VLSI designs. Our method leverages the advantages of MCTS and RL. Moreover, in order to handle large-scale designs, we have transformed macro placement problem into macro group allocation problem. The experimental results have demonstrated that our approach yields better results than previous works.

REFERENCES

[1] Synopsys. Inc. *IC Compiler* Accessed: October 5, 2021. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html

[2] S. N. Adya and I. L. Markov, "Consistent placement of macro-blocks using floorplanning and standard-cell placement," in *Proc. of ISPD*, pp. 12-17, 2002.

[3] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa and I. L. Markov, "Unification of partitioning, placement and floorplanning," in *Proc. of ICCAD*, pp. 550-557, 2004.

[4] A. Agnesina, P. Rajvanshi, T. Yang, G. Pradipta, A. Jiao, B. Keller, R. Khailany and H. Ren, "AutoDMP: Automated DREAMPlace-based Macro Placement," in *Proc. of ISPD*, pp. 149-157, 2023.

[5] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and T.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Trans. of TCAD*, vol. 27, no. 7, pp. 1228-1240, 2008.

[6] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and T.-Y. Liu, "MP-trees: A packing-based macro placement algorithm for modern mixed-size designs," *IEEE Trans. of TCAD*, vol. 27, no. 9, pp. 1621-1634, 2008.

[7] Y.-F. Chen. C.-C. Huang, C.-H Chiou, Y.-W. Chang, and C.-J. Wang, "Routability-driven blockage-aware macro placement," in *Proc. of DAC*, pp. 1-6, 2014.

[8] C.-H. Chiou, C.-H. Chang, S.-T. Chen, and Y.-W. Chang, "Circular-contour-based obstacle-aware macro placement," in *Proc. of ASP-DAC*, pp. 172-177, 2016.

[9] C.-H. Chang, Y.-W. Chang, and T.-C. Chen, "A novel damped-wave framework for macro placement," in *Proc. of ICCAD*, pp. 504-511, 2017.

[10] C.-K. Cheng, A. B. Kahng, I. Kang and L. Wang, "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement",*IEEE Trans. of TCAD*, vol. 38, no. 9, pp. 1717-1730, 2018.

[11] F.-C. Chang, Y.-W. Tseng, Y.-W. Yu, S.-R. Lee, A. Cioba, I.-L. Tseng, D.-S. Shiu, J.-W. Hsu, C.-Y. Wang, C.-Y. Yang, R.-C. Wang, Y.-W. Chang, T.-C. Chen, T.-C. Chen, "Flexible chip placement via reinforcement learning," in *Proc. of DAC*, pp. 1392-1393, 2022.

[12] C.-K. Cheng , A. B. Kahng , S. Kundu , Y. Wang , Z. Wang, "Assessment of Reinforcement Learning for Macro Placement," in *Proc. of ISPD*, pp. 158–166, 2023.

[13] H. Gu, J. Gu, K. Peng, Z. Ziran, N. Xu, X. Geng, J. Yang, "LAMPlace: Legalization-Aided Reinforcement Learning-Based Macro Placement for Mixed-Size Designs With Preplaced Blocks," *IEEE Trans. of TCSII*, vol. 71, no. 8, pp. 3770-3774, 2024.

[14] Z. He, Y. Ma, L. Zhang, P. Liao, N. Wong, B. Yu and M. D.F. Wong, "Learn to Floorplan through Acquisition of Effective Local Search Heuristics," in *Proc. of ICCD*, pp. 324-331, 2020.

[15] M. -K. Hsu *et al.*, "NTUplace4h: A Novel Routability-Driven Placement Algorithm for Hierarchical Mixed-Size Circuit Designs," *IEEE Trans. of TCAD*, vol. 33, no. 12, pp. 1914-1927, 2014.

[16] M. -C. Kim, D. -J. Lee and I. L. Markov, "SimPL: An effective placement algorithm," in *Proc. of ICCAD*, pp. 649-656, 2010.

[17] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement learning: A survey," *JAIR*, vol. 4, no. 1, p.p. 237-285, 1996.

[18] L. Kocsis, C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Proc. of ECML*, pp.282-293, 2006.

[19] Y. Lai, Y. Mu, P. Luo, "MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning," in *Proc. of NeurIPS*, pp. 24019-24030, 2022.

[20] J.-M. Lin, Y.-W. Chang, and S.-P. Lin, "Corner sequence: A P-admissible floorplan representation with a worst case linear-time packing scheme," *IEEE Trans. of TVLSI*, vol. 11, no. 4, pp. 679-686, 2003.

[21] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany and I. Nedelchev, "POLAR: Placement based on novel rough legalization and refinement," in *Proc. of ICCAD*, pp. 357-362, 2013.

[22] J. Lu *et al.*, "ePlace-MS: Electrostatics-Based Placement for Mixed-Size Circuits," *IEEE Trans. of TCAD*, vol. 34, no. 5, pp. 685-698, 2015.

[23] J.-M. Lin, S.-T. Li, and Y.-T. Wang, "Routability-driven Mixed-size Placement Prototyping Approach Considering Design Hierarchy and Indirect Connectivity Between Macros," in *Proc. of DAC*, pp. 1-6, 2019.

[24] J.-M. Lin, Y.-L. Deng, Y.-C. Yang, J.-J. Chen, and Y.-C. Chen, "Novel Macro Placement Approach based on Simulated Evolution Algorithm," in *Proc. of ICCAD*, pp. 1-7, 2019.

[25] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," in *Proc. of DAC*, pp. 1-6, 2019.

[26] J.-M. Lin, Y.-L. Deng, Y.-C. Yang, J.-J. Chen and P.-C. Lu, "Dataflow-Aware Macro Placement Based on Simulated Evolution Algorithm for Mixed-Size Designs," *IEEE Trans. of TVLSI*, vol. 29, no. 5, pp. 973-984, 2021.

[27] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[28] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajatani, "Rectangle packing based module placement," in *Proc. of ICCAD*, pp. 472–479, 1995.

[29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T .P.Lillicrap, D. Silver, K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *Proc. of ICML*, pp. 1928-1937, 2016.

[30] M. van Otterlo and M. Wiering, *Reinforcement Learning: State-of-the-Art*. Berlin, Heidelberg: Springer, 2012.

[31] Y. Pu, T. Chem, Z. He, C. Bai, H. Zheng, Y. Lin, B. Yu, "IncreMacro: Incremental Macro Placement Refinement," in *Proc. of ISPD*, pp. 169-176, 2024.

[32] C. D. Rosin, "Multi-armed bandits with episode context, "*Ann. Math. Artif. Intell.*, vol. 61, pp. 203–230, 2011.

[33] D. Silver, J. Schrittwieser, K. Simonyan *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, p.p. 354–359, 2017.

[34] X. Tang, R. Tian, and Martin D. F. Wong, "Optimal redistribution of white space for wire length minimization," in *Proc. of ASP-DAC*, pp. 412-417, 2005.

[35] N. Viswanathan and C. C. . -N. Chu, "FastPlace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model," *IEEE Trans. of TCAD*, vol. 24, no. 5, pp. 722-733, 2005.

[36] M.-C. Wu and Y.-W. Chang, "Placement with alignment and performance constraints using the B*-tree representation," in *Proc. of ICCAD*, pp. 568-571, 2004.