Dynamic Algorithm Configuration for Global Placement

Chen Lu[1,2], Ke Xue[†1,2], Ruo-Tong Chen[1,2], Yunqi Shi[1,2],
Siyuan Xu[3], Mingxuan Yuan[3], Chao Qian[†1,2], Zhi-Hua Zhou[1,2]
[1] National Key Laboratory for Novel Software Technology, Nanjing University, China
[2] School of Artificial Intelligence, Nanjing University, China
[3] Huawei Noah's Ark Lab, China

***Abstract*—Placement is a vital step in the physical design flow of very large-scale integration (VLSI) circuits. GPU-accelerated analytical placement algorithms, such as DREAMPlace, have achieved high-quality performance with dramatic speedup. The algorithm configurations of the analytical placer have a significant impact on its convergence and final performance. However, its tuning process is difficult and time-consuming. Recently, AutoDMP tries to search for optimal static algorithm configurations using Bayesian optimization, but the performance is still limited due to its static strategy, which cannot leverage information during algorithm execution. In this paper, we propose the dynamic algorithm configuration framework for DREAMPlace (DACDMP), using reinforcement learning (RL) to learn the dynamic control policy of the most critical hyperparameter, i.e., the learning rate. Moreover, to address the insufficiency of optimization, we increase the number of optimization steps in each Lagrangian relaxation problem, thereby improving the solution's optimality. DACDMP outperforms the current leading methods, i.e., DREAMPlace 4.0, AutoDMP, and Xplace. For example, compared to DREAMPlace 4.0, it achieves an average improvement of 2.75% in wirelength, 18.74% in worst negative slack (WNS), 44.60% in total negative slack (TNS), and 29.39% in the number of violation points on the ICCAD 2015 benchmark.**

***Index Terms*—Global Placement, Dynamic Algorithm Configuration, Reinforcement Learning, AI for EDA**

I. Introduction

Placement is a vital step in the physical design flow of VLSI circuits. The quality of placement has a direct and significant impact on the performance, power consumption, and area (PPA) of the final chip design, as it determines the locations of all the cells within the chip's layout area [20]. Placement typically encompasses three steps: global placement (GP), legalization (LG), and detailed placement (DP). GP plays a crucial role, which provides a rough overall layout of the cells [16]. LG aligns the cells overlap-free to the rows of the chip [25], and DP refines the layout for better quality [5].

Analytical placement [4] is one of the state-of-the-art GP methods, which regards GP as a quadratic [8, 15] or nonlinear [5, 6, 19] optimization problem. Despite the outstanding performance it achieves, analytical placement can be time-consuming. To address this, accelerating algorithms like DREAMPlace [16] and Xplace [17] are proposed, which open up the possibility to carry out GP in a data-driven manner [31]. DREAMPlace regards GP as a neural network training problem, and accelerates it using GPUs with modern deep learning toolkits. However, just as in deep learning, the configurations of such algorithms (e.g., learning rate) have a great impact on the convergence and the final performance [9, 28, 29], while tuning them is challenging and tedious [27].

Recently, AutoDMP [2] tries to automatically explore hyperparameter configurations of DREAMPlace using advanced optimization techniques [23, 33]. It achieves performance comparable to commercial tools, highlighting the importance of high-quality hyperparameter configuration for DREAMPlace. However, it is a static algorithm configuration approach, where hyperparameter settings are fixed prior to algorithm execution [1]. While in practice, there are hyperparameters (such as learning rate, density weight, smoothing factor, etc.) that need to be adjusted every step during execution. AutoDMP can only configure fixed hyperparameters or fixed hyperparameter update rules before algorithm execution. This approach fails to leverage in-execution information, significantly limiting the potential of algorithm configuration.

Among the hyperparameters of DREAMPlace requiring dynamic configuration, the learning rate is the key one calling for delicate dynamic design. As has been observed in previous works [2, 16], the learning rate is crucial for the convergence and quality of placement optimization. Moreover, following the analogy made in DREAMPlace, one optimization step in DREAMPlace can be seen as a full-batch optimization step in deep learning, which has been widely observed to have a negative impact on training loss and model accuracy, and calls for adaptive learning rate mechanisms [9, 28].

On the other hand, DREAMPlace employs a penalty method to solve the constrained optimization problem in GP by sequentially optimizing a series of Lagrangian relaxation problems with an increasing Lagrangian multiplier λ (also called "density weight"). However, the optimality of the solution at each individual λ is often overlooked, which can result in insufficient optimization and compromise the quality of the final solution.

This work tries to address the above-mentioned issues. Our contributions can be outlined as follows:

- We propose the dynamic algorithm configuration framework for DREAMPlace (called DACDMP), to dynamically configure the learning rate for DREAMPlace by reinforcement learning (RL) with automated exploration. Guided by a delicate reward design that provides the RL agent with both in-process guidance and final performance

† Corresponding author. E-mail: {xuek,qianc}@lamda.nju.edu.cn

978-3-9826741-1-7/DATE26/© 2026 EDAA

evaluation, the learned policy achieves better performance than the learning rate schedules designed by experts and found by AutoDMP. Furthermore, it provides new insights about how to dynamically configure the learning rate that experts have never thought of. Our learned policies also demonstrate certain transferability across chips.
- We highlight the importance of the solution optimality for each Lagrangian relaxation problem, and improve the final optimization performance by increasing the corresponding number of optimization steps.
- Experiments on the ISPD 2005 [22] and ICCAD 2015 [10] benchmarks clearly show the superiority of DACDMP over the leading methods, DREAMPlace 4.0 [14], AutoDMP [2], and Xplace [17]. For example, compared to DREAMPlace 4.0, DACDMP achieves an average improvement of 2.75% in wirelength, 18.74% in WNS, 44.60% in TNS, and 29.39% in the number of violation points on the ICCAD 2015 benchmark.

II. Background

A. Global Placement

Global placement (GP) is typically formulated as a constrained optimization problem with the half-perimeter wirelength (HPWL) as the objective function and the density as the constraint:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \text{HPWL}(\mathbf{x},\mathbf{y}) = \min_{\mathbf{x},\mathbf{y}} \sum\nolimits_{e\in E} \text{HPWL}_e(\mathbf{x},\mathbf{y}), \\ \text{s.t. } \text{D}(\mathbf{x},\mathbf{y}) \leq d_t, \end{aligned} \tag{1}$$

where $(\mathbf{x},\mathbf{y})$ denotes the 2-D positions of all the cells, E denotes the set of nets, $\text{HPWL}_e(\mathbf{x},\mathbf{y}) = (\max_{i\in e}x_i - \min_{i\in e}x_i) + (\max_{i\in e}y_i - \min_{i\in e}y_i)$, $\text{D}(\mathbf{x},\mathbf{y})$ denotes the density of each location of the layout, and d_t denotes the target density.

Analytical placement [4] represents the state-of-the-art approach for GP. A prominent variant within this paradigm is nonlinear placement [5, 6, 19], which typically employs a penalty-based methodology, solving a sequence of Lagrangian relaxation problems as follows:

$$\min_{\mathbf{x},\mathbf{y}} \sum\nolimits_{e\in E} \text{WL}_e(\mathbf{x},\mathbf{y}) + \lambda \cdot \text{D}(\mathbf{x},\mathbf{y}), \tag{2}$$

where $\text{WL}_e(\mathbf{x},\mathbf{y})$ is a smooth approximation of HPWL, $\text{D}(\mathbf{x},\mathbf{y})$ is the density penalty function, and λ is the Lagrangian multiplier (also called density weight) that controls the importance of density. ePlace [19] and RePlace [6] are two typical nonlinear placer. They compute a smooth density penalty function analogous to the potential energy of an electrostatic system, where cells are modeled as charges, and the density gradient is modeled as the electric field.

Recently, DREAMPlace [14, 16] accelerates ePlace/RePlace by treating Eq. (2) as a neural network training task and utilizing the GPU to accelerate gradient descent. DREAMPlace can achieve around 40× speedup in global placement without quality degradation compared to the previous placer. However, the hyperparameter configurations of DREAMPlace are important but difficult. AutoDMP [2] automatically explores the configurations of DREAMPlace via Bayesian optimization [7, 23]. In each iteration, it selects one configuration, executes it, and only collects the final placement result, ignoring all intermediate runtime information. Due to this static configuration approach, AutoDMP can only preset fixed hyperparameters or fixed update rules (e.g., the initial learning rate η_0 and decay factor γ in an exponential learning-rate schedule) before execution. Consequently, its ability to improve performance is fundamentally constrained. This limitation motivates the work presented in this paper.

B. Dynamic Algorithm Configuration

Dynamic algorithm configuration (DAC) has gained more and more attention recently [1], which aims to dynamically configure the hyperparameters of a target algorithm $\mathcal{A}$ during execution to boost better performance over a distribution of the problem instance domain $\mathcal{I}$. It can be formulated as

$$\arg\min_{\pi\in\Pi} \mathbb{E}_{i\sim\mathcal{I}}(c(\pi, i)),$$

where $\pi \in \Pi$ denotes a configuration policy, $i \sim \mathcal{I}$ denotes a problem instance (e.g., a circuit case in chip placement), and $c(\pi, i)$ denotes a cost function measuring the performance of the algorithm $\mathcal{A}$ with policy π for solving instance i. The DAC task can be treated as a contextual Markov decision process (MDP) $\mathcal{M}_{\mathcal{I}} := \mathcal{M}_{i\sim\mathcal{I}}$ [3], with one MDP for one instance, and all the MDPs sharing the same state and action space but having different transition and reward functions. The MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R)$ [26], where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, the transition function $T(s, a, s')$ gives the probability of transitioning from the current state s to the next state s' by performing the action a, and $R(s, a, s')$ is the reward function. DAC has been applied in various scenarios, e.g., to learn policies that can dynamically configure the mutation step-size of covariance matrix adaptation evolution strategy [24], and to adjust the learning rate in neural network training [27, 28].

III. Method

To address the challenging task of hyperparameter tuning in DREAMPlace, we apply DAC to dynamically configure the key hyperparameter, i.e., the learning rate. We reconstruct the DREAMPlace algorithm as an MDP, which can serve as an environment interacting with the RL agent. In each gradient-descent step, it provides the agent with the current state information, receives the action given by the agent, tunes the hyperparameters accordingly, and sends back a reward signal. For each chip case, we train the RL agent using Deep Q-Network (DQN) [21] to learn a dynamic learning rate configuration policy. Note that although the policy is learned for one single chip case, it shows certain transferability, i.e., achieves good performance on unseen chip cases (as shown in Section IV-G), attributed to our generalizable state and reward design. Moreover, optimization insufficiency occurs when DREAMPlace only optimizes each Lagrangian relaxation problem for one step. Thus, we increase the number of optimization steps for each Lagrangian relaxation problem to enhance the optimality of the corresponding solution, which improves the optimality of the final solution. The overall pipeline of the proposed DACDMP in one gradient-descent step is shown in Fig. 1.

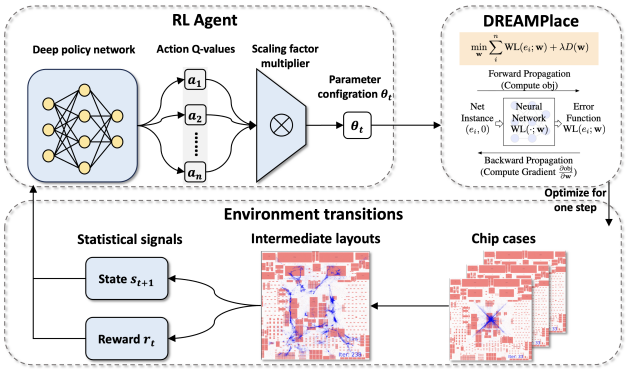


Fig. 1: Overall pipeline of DACDMP.

TABLE I: State features at execution step t in our environment.

Index	Feature	Notes
0	t	Current number of iterations
1	$\log f_t$	Log of current objective function value
2	$\log \text{lr}_{t-1}$	Log of previous learning rate
3	$\log \|\|\text{grad}_t\|\|$	Log of current gradient norm
4	$\log \text{hpwl}_t$	Log of current HPWL value
5	overflow_t	Current overflow
6	$\log \lambda_t$	Log of current Lagrangian multiplier λ
7	$\log \gamma_t$	Log of current smoothing factor γ

A. State

The state is to provide the RL agent with the current information of the algorithm execution and statistics of the current chip layout, based on which the agent adjusts the hyperparameter configuration for the next step.

Following the state design principles established in prior DAC works [27, 30], we reconstruct the DREAMPlace algorithm to provide fundamental runtime information, including the current iteration number, objective function value, previous learning rate, and the norm of the current gradient. However, DREAMPlace involves greater complexity than typical numerical optimization tasks. For example, it dynamically adjusts the Lagrangian multiplier λ and the smoothing factor γ of the HPWL model. To capture this algorithmic structure, we further introduce several GP-specific features: the current HPWL, overflow value, Lagrangian multiplier λ, and smoothing factor γ. These additions provide the agent with a detailed view of the Lagrangian relaxation problem throughout its execution. As some state features may be numerically large, we normalize them using their logarithmic transformation. The features are listed in Table I, where the first four rows are optimization-relevant features, and the last four rows are GP-relevant features.

B. Action

The action taken by the RL agent is to control the configuration (i.e., learning rate in this work) of DREAMPlace every execution step. Due to the sensitivity of learning rate tuning, as well as the instability of the output of neural networks [28], it is infeasible to directly use the action output of the agent as the new learning rate applied. Therefore, we use the agent's action output as an amending factor, and the new learning rate is given by multiplying the amending factor by the old learning rate. In our setting, the amending factor is chosen from 0.99, 0.995, 1, and 1.005, which means that the agent can both increase and decrease the current learning rate, as well as keep it unchanged. Thus, the space action is a four-dimensional discrete space in DACDMP.

C. Reward

The reward is to provide guidance for the agent to take good action and finally learn a well-performing dynamic configuration policy. The quality of the reward function has a great impact on the learning process of the agent. One simple and direct design is to give the agent a reward of the HPWL decrement every step [13]. However, this design will guide the agent to only focus on HPWL optimization, ignoring the density constraints. Moreover, this design cannot guarantee assigning the largest cumulative reward to the best final performance after LG and DP, because there may be a mismatch between GP and DP results, i.e., some policies achieve better HPWL results in GP, but fail to maintain superiority after LG and DP. Note that good GP results may still contain overlaps and nonalignment, and the HPWL may get worse after LG fixes these overlaps and nonalignment.

Inspired by the above domain knowledge, we design a reward consisting of both in-process guidance and final performance evaluation, which improves the agent's searching efficiency with the guidance of every step and guarantees assigning the largest cumulative reward to the best final performance after LG and DP by considering the final performance evaluation. Note that integrating domain knowledge in the process of utilizing data has been a promising direction (e.g., abductive learning [32]) in artificial intelligence. Particularly, our reward design consists of two parts.

1) In-process guidance reward: We guide the agent with the reward as: $r_t = -\Delta_t\text{hpwl} - \omega \cdot \Delta_t\text{overflow}$, where t is the current step, $\Delta_t\text{hpwl}$ and $\Delta_t\text{overflow}$ are the change of HPWL and overflow between steps t and $t-1$, respectively, and ω is the balancing factor.

2) Final performance evaluation reward: As the agent's final goal is to maximize the expected cumulative reward, the final reward should be negatively correlated with the final HPWL after LG and DP. While it is simple to design linear rewards (e.g., the negative of HPWL or improvement ratio to a certain baseline), we find that it is harder to improve the same absolute value when HPWL gets smaller. Therefore, we assign a superlinear reward to the agent, which gives more reward when the same improvement value is made to a smaller HPWL. The final performance evaluation reward r' is as follows:

$$r' = \begin{cases} \frac{\alpha}{\text{hpwl}'/\text{hpwl}_0 - \beta} & \text{if overflow}' \leq \text{overflow}_{\text{target}}, \\ -50 & \text{otherwise}, \end{cases}$$

where hpwl' ($\text{overflow}'$) is the final HPWL (overflow) value achieved after GP, LG, and DP; hpwl_0 is the HPWL value

of the initial layout, which serves as the scaling factor as the HPWL value varies numerically in different chip cases; α is a weighting factor to balance the in-process guidance reward and final performance evaluation reward, which is set to 1 by default; β is a bias that makes the denominator closer to 0 lead to a more superlinear reward, which is set to 0.2 by default; $\text{overflow}_{\text{target}}$ is the target overflow threshold, and GP stops if the target overflow is achieved and HPWL does not improve anymore. If the target overflow is not ultimately achieved, a penalty of -50 will be given.

D. Policy Optimization

Any efficient RL algorithm can be applied in DACDMP. To demonstrate the effectiveness of our method, we apply the classic RL algorithm DQN [21] in this work, which is a relatively simple off-policy value-based RL algorithm and is easy to use.

Specifically, DQN learns an optimal policy by modeling an action-value function $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a]$, where π is the policy, γ is the discount factor, and r_t is the immediate reward obtained at timestep t. DQN finds the optimal action-value function $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$ by the well-known Bellman update formula:

$$Q(s,a) \leftarrow Q(s,a) + \eta_{\text{DQN}} \cdot \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right],$$

where s, a is the current state and action, s', a' is the next state and action, $R = R(s,a,s')$ is the reward obtained after the transition from the current state s to the next state s' by performing action a, and η_{DQN} is the update step size.

E. Augmentation in Lagrangian Relaxation Optimization

Global placement typically uses the penalty method to solve the constrained optimization problem in Eq. (1), optimizing a sequence of Lagrangian relaxation problems with increasing Lagrangian multiplier λ. Some works explore on how to configure λ every iteration: ePlace [19], RePlace [6] and DREAMPlace [14, 16] use heuristics based on $\Delta_t$hpwl, while Kirby et al. [12] updates λ by RL. However, the optimality of the solution for each individual λ is neglected, which may damage the overall optimality of the final solution, especially when using deep learning toolkits' optimizer in DREAMPlace. DREAMPlace only applies gradient descent for a single step in each Lagrangian relaxation problem, which is often not sufficient to find an optimal solution to the current Lagrangian relaxation problem.

We investigate the importance of the solution optimality in each Lagrangian relaxation problem by increasing the number of optimization steps for each Lagrangian relaxation problem, where the number of optimization steps for one individual λ is denoted as N_λ. Note that for each Lagrangian relaxation problem, the agent configures the learning rate of DREAMPlace and optimizes for N_λ steps with the learning rate unchanged. Our empirical results in Section IV-F will show that an appropriate increase in the number of optimization steps can generate better-quality solutions.

IV. Experiments

A. Experimental Settings

Our experiments are conducted on two open-source benchmarks: ISPD 2005 [22] and ICCAD 2015 [10]. Our DACDMP framework is developed based on DREAMPlace 4.0 [14], which uses the widely used Adam [11] optimizer in PyTorch. We compare the performance of the following methods in the global placement phase:

- DREAMPlace 4.0 [14]: A leading open-source nonlinear analytical placement method.
- AutoDMP [2]: A static algorithm configuration method based on Bayesian optimization for DREAMPlace, searching for the initial learning rate and decay factor for the Adam optimizer. For a fair comparison, AutoDMP uses the same searching budget as the training budget of DACDMP.
- Xplace [17]: Another recent leading open-source analytical placement method, which uses the state-aware hyperparameter scheduling and the neural network enhancement for density gradient prediction.

For a fair comparison, all methods use their own default LG and DP processes (i.e., no external detailed placement engine applied). All the evaluations are conducted on a Linux server equipped with a 32-core CPU and a GPU.

B. ISPD 2005 Results

Table II demonstrates the detailed comparison of HPWL values achieved after GP, LG, and DP between our proposed DACDMP and three compared methods on the ISPD 2005 benchmark. Our proposed DACDMP achieves the best HPWL result in every single case, with an average improvement of 2.64% over 8 cases compared to DREAMPlace 4.0. Replicate experiments indicate that the HPWL improvement does not come from random disturbances. The superiority of our method over AutoDMP demonstrates the effectiveness of dynamically configuring the learning rate for DREAMPlace. Moreover, our method produces less variance on HPWL than AutoDMP because of its dynamic self-adjusting policy, which can be more robust when coping with noise. It is worth noting that a marginal improvement in HPWL can have a huge impact on the final PPA performance, which can be found in Section IV-C.

C. ICCAD 2015 Results

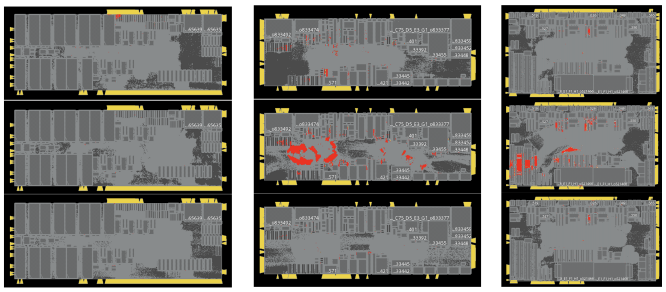
Table III shows the comprehensive comparison of PPA metrics between our DACDMP and two recent state-of-the-art open-source analytical placement methods on the ICCAD 2015 benchmark. We test the methods by applying them to perform GP, LG, and DP, and then completing the subsequent processes and obtaining the final PPA results using EarlyGlobalRoute. The PPA metrics include the routed wirelength (rWL), the routed horizontal and vertical congestion overflows (rO-H, rO-V), the worst and total negative slacks (WNS, TNS), and the number of violation points (NVP). As shown in Table III, our proposed DACDMP achieves the shortest rWL and the smallest NVP in every single case, and achieves the best overflow and timing performance in most cases. DACDMP demonstrates significantly superior performance in many cases.

TABLE II: HPWL ($\times 10^6$) results after GP+LG+DP on the ISPD 2005 benchmark. Each HPWL result consists of the mean and standard deviation of three independent runs. The best results are in bold.

Design	DREAMPlace 4.0 [14]	AutoDMP [2]		Xplace [17]		DACDMP (ours)	
	HPWL	HPWL	% impro	HPWL	% impro	HPWL	% impro
adaptec1	74.17 ± 0.12	73.73 ± 0.05	(0.59)	73.10 ± 0.01	(1.45)	**72.22 ± 0.01**	**(2.63)**
adaptec2	83.54 ± 0.02	83.10 ± 0.11	(0.52)	81.57 ± 0.03	(2.36)	**81.44 ± 0.06**	**(2.51)**
adaptec3	194.17 ± 0.04	192.57 ± 0.10	(0.82)	193.70 ± 0.05	(0.24)	**188.90 ± 0.05**	**(2.71)**
adaptec4	176.60 ± 0.16	174.44 ± 0.07	(1.22)	173.45 ± 0.03	(1.78)	**172.02 ± 0.02**	**(2.59)**
bigblue1	90.52 ± 0.26	89.95 ± 0.01	(0.63)	89.00 ± 0.07	(1.68)	**88.65 ± 0.00**	**(2.05)**
bigblue2	137.72 ± 0.04	137.18 ± 0.23	(0.39)	136.48 ± 0.02	(0.90)	**136.30 ± 0.06**	**(1.03)**
bigblue3	315.52 ± 0.32	313.20 ± 0.80	(0.73)	302.39 ± 0.08	(4.16)	**296.19 ± 0.08**	**(6.13)**
bigblue4	748.26 ± 0.29	745.31 ± 0.61	(0.40)	741.05 ± 0.02	(0.96)	**737.14 ± 0.01**	**(1.49)**
Average Ratio	1.000	0.993	(0.66)	0.983	(1.69)	**0.974**	**(2.64)**

TABLE III: PPA results achieved by different methods on the ICCAD 2015 benchmark. The best results are in bold.

Design	DREAMPlace 4.0 [14]						Xplace [17]						DACDMP (ours)					
	rWL↓ (m)	rO-H↓ (%)	rO-V↓ (%)	WNS↑ (ns)	TNS↑ (μs)	NVP↓ (#)	rWL↓ (m)	rO-H↓ (%)	rO-V↓ (%)	WNS↑ (ns)	TNS↑ (μs)	NVP↓ (#)	rWL↓ (m)	rO-H↓ (%)	rO-V↓ (%)	WNS↑ (ns)	TNS↑ (μs)	NVP↓ (#)
superblue1	85.36	0.06	**0.00**	-20.65	-15.19	5897	82.32	**0.00**	**0.00**	-26.49	-23.22	7652	**81.61**	**0.00**	**0.00**	**-11.67**	**-9.64**	**5082**
superblue3	101.25	0.07	0.01	-34.86	-16.04	4877	128.50	0.77	2.94	-26.84	-14.74	4508	**91.59**	**0.01**	**0.00**	**-20.14**	**-9.41**	**3522**
superblue4	63.09	0.05	**0.00**	**-14.80**	**-21.93**	6041	80.62	1.56	0.42	-25.70	-34.86	7525	**61.99**	**0.04**	**0.00**	-15.18	-22.92	**5759**
superblue5	93.86	**0.00**	**0.00**	**-29.51**	-15.91	4544	115.34	0.26	0.09	-34.70	-20.38	5678	**92.84**	0.03	**0.00**	-30.40	**-13.23**	**3240**
superblue7	121.31	**0.00**	**0.00**	-20.18	-22.21	10144	123.83	**0.00**	**0.00**	-28.63	-21.02	8838	**120.74**	**0.00**	**0.00**	**-17.16**	**-16.16**	**7004**
superblue10	180.58	**0.02**	**0.02**	-36.14	-110.00	9949	181.43	0.03	**0.02**	-33.64	-111.00	10394	**179.78**	0.04	**0.02**	**-32.65**	**-102.00**	**9037**
superblue16	83.99	0.05	**0.01**	**-15.53**	-41.21	16890	101.70	0.35	0.39	-32.09	-47.01	17634	**83.29**	**0.04**	**0.01**	-15.88	**-26.71**	**12052**
superblue18	48.80	0.03	0.01	-12.91	-23.32	5820	47.66	0.06	**0.00**	-17.22	-9.45	3069	**47.46**	**0.02**	0.01	**-11.34**	**-4.66**	**2443**
Average Ratio	1.03	1.56	1.25	1.23	1.80	1.42	1.15	16.83	96.50	1.58	1.65	1.37	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**



(a) superblue1 (b) superblue3 (c) superblue4

Fig. 2: Placement layouts on three ICCAD 2015 benchmarks. The three rows from top to bottom are the results of DREAMPlace 4.0 [14], Xplace [17], and DACDMP, respectively. The red points denote the congestion critical regions.

Compared to DREAMPlace 4.0, DACDMP achieves an average improvement of 2.75% in wirelength, 18.74% in WNS, 44.60% in TNS, and 29.39% in NVP; while compared to Xplace, it achieves an average improvement of 13.18% in wirelength, 36.88% in WNS, 39.46% in TNS, and 27.11% in NVP. We visualize the placement layouts in Fig. 2. We compare the layouts and congestions on three ICCAD 2015 benchmarks. Our proposed DACDMP evidently demonstrates more regular cell placement layouts and fewer congestion issues.

D. Visualization of the Learned Policies

Fig. 3 plots the learning rate curves (showing how the learning rate changes during the optimization process) of our policies (learned by DACDMP) and other commonly used learning rate schedules (constant, exponential decay, cosine decay, and step decay) designed by experts, on the two chip designs of the ISPD 2005 benchmark. It is clear that the learned policies vary across different cases, and are much dissimilar to the expert-designed schedules. One interesting pattern we can observe from the learned policies is that we may need to increase or keep the learning rate in the early optimization stages, rather than simply decreasing it.

E. Runtime Analysis

Table IV provides the total RL exploration time of DACDMP to find the best-performing policies on each case. DACDMP demonstrates certain time efficiency in automated RL exploration attributed to our effective reward guidance. Fig. 4 plots the GP+LG+DP runtime comparison between the proposed DACDMP with $N_\lambda = 1$ and the original DREAMPlace 4.0. Intuitively, DACDMP requires some extra time to configure the learning rate every timestep through neural network inference. However, in many cases, especially large cases (e.g., superblue7, superblue10, etc.), DACDMP spends less time than the original DREAMPlace 4.0, which is because the learning rate schedule produced by DACDMP leads to faster convergence of global placement.

F. Comparison with Different Numbers of Optimization Steps

Table V demonstrates the HPWL results achieved by DACDMP after GP, LG, and DP with different values of N_λ. The results show that in general, with larger N_λ, the solution is more sufficiently optimized and better performance is achieved,

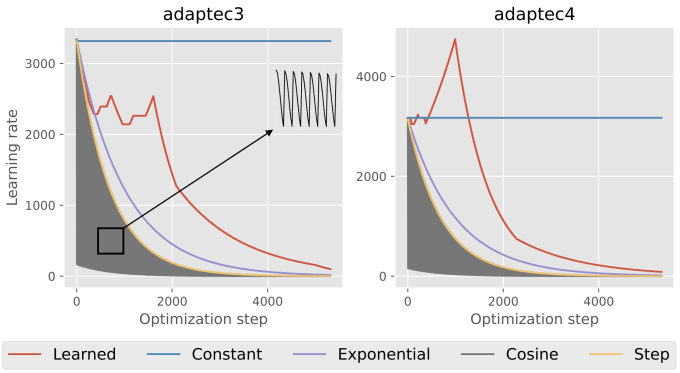


Fig. 3: Comparison of the learning rate curves obtained by our learned policies (red) and different common schedules.

TABLE IV: The total RL training time of DACDMP to find the best policies on each case in the main experiments.

Benchmark	Design	Exploration time (h)	Design	Exploration time (h)
ISPD 2005	adaptec1	1.39	bigblue1	4.51
	adaptec2	1.99	bigblue2	0.46
	adaptec3	0.59	bigblue3	7.94
	adaptec4	0.71	bigblue4	4.18
ICCAD 2015	superblue1	2.07	superblue7	15.84
	superblue3	1.72	superblue10	10.93
	superblue4	2.14	superblue16	4.85
	superblue5	8.54	superblue18	9.61

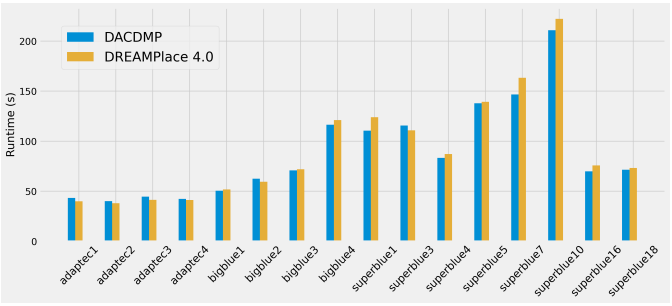


Fig. 4: GP+LG+DP runtime comparison between DACDMP and DREAMPlace 4.0 for ISPD2005/ICCAD2015 benchmarks.

but at the cost of more optimization steps. We conclude that an appropriate increase in the number of optimization steps can generate better-quality solutions, i.e., too few steps lead to insufficient optimization, while too many steps merely increase optimization costs without significant improvement. It can also be inferred from the results that the best N_λ varies among different cases due to the different levels of optimization complexity. In the eight test cases, the best N_λ is 7 or 10.

G. Analysis on Transferability and Generalization

Table VI shows the HPWL results after GP+LG+DP on the ISPD 2005 benchmark, achieved by applying the policy learned solely on the bigblue1 case with $N_\lambda = 7$ (denoted as DACDMP-bigblue1). The superiority of DACDMP-bigblue1 over DREAMPlace 4.0 demonstrates certain transferability of

TABLE V: HPWL ($\times 10^6$) results achieved by DACDMP (with different values of N_λ) after GP+LG+DP on the ISPD 2005 benchmark. Each HPWL result consists of the mean of three independent runs. The best results are in bold.

Design	$N_\lambda = 1$	$N_\lambda = 3$	$N_\lambda = 5$	$N_\lambda = 7$	$N_\lambda = 10$
adaptec1	72.95	72.40	72.32	72.23	**72.22**
adaptec2	81.88	81.76	81.56	81.51	**81.44**
adaptec3	192.41	190.05	189.78	**188.90**	189.07
adaptec4	173.69	172.05	172.12	**172.02**	172.38
bigblue1	89.33	88.82	88.69	**88.65**	88.67
bigblue2	137.64	136.64	136.48	**136.30**	136.55
bigblue3	329.38	302.79	297.19	296.33	**296.19**
bigblue4	749.68	738.74	737.96	737.46	**737.14**

TABLE VI: HPWL ($\times 10^6$) results achieved by DACDMP-bigblue1 (trained solely on the bigblue1 case) after GP+LG+DP on the ISPD 2005 benchmark.

Design	DREAMPlace 4.0 [14] HPWL	DACDMP-bigblue1 HPWL	DACDMP-bigblue1 % impro
adaptec1	74.17	**72.52**	**(2.23)**
adaptec2	83.54	**81.96**	**(1.89)**
adaptec3	194.17	**191.90**	**(1.17)**
adaptec4	176.60	**173.61**	**(1.69)**
bigblue1	90.52	**88.65**	**(2.07)**
bigblue2	137.72	**136.64**	**(0.79)**
bigblue3	315.52	**308.17**	**(2.33)**
bigblue4	748.26	**738.70**	**(1.28)**
Average Ratio	1.000	**0.983**	**(1.68)**

the learned policy, as DACDMP-bigblue1 (trained solely on bigblue1) can optimize the cases it has never seen. Our design of state and reward captures certain commonalities among various cases, and thus some similar and important patterns of optimization processes of different cases.

V. Conclusion

In this paper, we propose DACDMP to dynamically configure the most vital hyperparameter, learning rate, of the leading placement method DREAMPlace. Experiments on the ISPD 2005 and ICCAD 2015 benchmarks demonstrate the superior performance of the proposed DACDMP on PPA. DACDMP also shows runtime efficiency and certain transferability across various chip designs. Moreover, DACDMP can be applied to other placement algorithms and extended to collaboratively adjust multiple important hyperparameters by using advanced multi-agent RL algorithms [30, 18], which opens up the potential for AI-empowered algorithm configuration in the field of chip placement.

Acknowledgment

The authors thank anonymous reviewers for their insightful and valuable comments. This work was supported by the National Science and Technology Major Project (2022ZD0116600), the Jiangsu Science Foundation Leading-edge Technology Program (BK20232003), the Fundamental Research Funds for the Central Universities (14380020), and the National Science Foundation of China (62276124, 624B2069).

REFERENCES

[1] Steven Adriaensen, André Biedenkapp, Gresa Shala, Noor Awad, Theresa Eimer, Marius Lindauer, et al. "Automated dynamic algorithm configuration". In: *Journal of Artificial Intelligence Research* 75 (2022), pp. 1633–1699.

[2] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, et al. "Autodmp: Automated dreamplace-based macro placement". In: *Proceedings of the 2023 International Symposium on Physical Design*. Virtual Event, 2023, pp. 149–157.

[3] André Biedenkapp, H. Furkan Bozkurt, Theresa Eimer, Frank Hutter, and Marius Thomas Lindauer. "Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework". In: *Proceedings of the 2020 European Conference on Artificial Intelligence*. Santiago de Compostela, Spain, 2020, pp. 427–434.

[4] Yao-Wen Chang, Zhe-Wei Jiang, and Tung-Chieh Chen. "Essential issues in analytical placement algorithms". In: *IPSJ Transactions on System LSI Design Methodology* 2 (2009), pp. 145–166.

[5] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.7 (2008), pp. 1228–1240.

[6] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. "Replace: Advancing solution quality and routability validation in global placement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.9 (2018), pp. 1717–1730.

[7] Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[8] Xu He, Tao Huang, Linfu Xiao, Haitong Tian, and Evangeline F. Y. Young. "Ripple: A robust and effective routability-driven placer". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.10 (2013), pp. 1546–1556.

[9] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima". In: *arXiv:1609.04836* (2016).

[10] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite". In: *Proceedings of the 2015 International Conference on Computer-Aided Design*. Austin, TX, 2015, pp. 921–926.

[11] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv:1412.6980* (2014).

[12] Robert Kirby, Kolby Nottingham, Rajarshi Roy, Saad Godil, and Bryan Catanzaro. "Guiding global placement with reinforcement learning". In: *arXiv:2109.02631* (2021).

[13] Yao Lai, Yao Mu, and Ping Luo. "MaskPlace: Fast chip placement via reinforced visual representation learning". In: *Advances in Neural Information Processing Systems 35*. New Orleans, LA, 2022, pp. 24019–24030.

[14] Peiyu Liao, Dawei Guo, Zizheng Guo, Siting Liu, Yibo Lin, and Bei Yu. "DREAMPlace 4.0: Timing-driven placement with momentum-based net weighting and lagrangian-based refinement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.10 (2023), pp. 3374–3387.

[15] Tao Lin, Chris C. N. Chu, and Gang Wu. "POLAR 3.0: An ultrafast global placement engine". In: *Proceedings of the 2015 International Conference on Computer-Aided Design*. Austin, TX, 2015, pp. 520–527.

[16] Yibo Lin, Zixuan Jiang, Jiaqi Gu, Wuxi Li, Shounak Dhar, Haoxing Ren, et al. "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern VLSI placement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.4 (2020), pp. 748–761.

[17] Lixin Liu, Bangqi Fu, Shiju Lin, Jinwei Liu, Evangeline F. Y. Young, and Martin D. F. Wong. "Xplace: An extremely fast and extensible placement framework". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.6 (2024), pp. 1872–1885.

[18] Chen Lu, Ke Xue, Lei Yuan, Yao Wang, Yaoyuan Wang, Sheng Fu, et al. "Sequential multi-agent dynamic algorithm configuration". In: *Advances in Neural Information Processing Systems 39*. San Diego, CA, 2025.

[19] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, et al. "ePlace: Electrostatics-based placement using fast Fourier transform and Nesterov's method". In: *ACM Transactions on Design Automation of Electronic Systems* 20.2 (2015), pp. 1–34.

[20] Igor L Markov, Jin Hu, and Myung-Chul Kim. "Progress and challenges in VLSI placement research". In: *Proceedings of the 25th International Conference on Computer-Aided Design*. San Jose, CA, 2012, pp. 275–282.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[22] Gi-Joon Nam, Charles J Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. "The ISPD2005 placement contest and benchmark suite". In: *Proceedings of the 2005 International Symposium on Physical Design*. San Francisco, CA, 2005, pp. 216–220.

[23] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, Masahiro Nomura, and Masaki Onishi. "Multiobjective tree-structured parzen estimator". In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 1209–1250.

[24] Gresa Shala, André Biedenkapp, Noor Awad, Steven Adriaensen, Marius Lindauer, and Frank Hutter. "Learning step-size adaptation in CMA-ES". In: *Proceedings of the 16th International Conference on Parallel Problem Solving from Nature*. Leiden, The Netherlands, 2020, pp. 691–706.

[25] Peter Spindler, Ulf Schlichtmann, and Frank M Johannes. "Abacus: Fast legalization of standard cell circuits with minimal movement". In: *Proceedings of the 2008 International Symposium on Physical Design*. Portland, OR, 2008, pp. 47–53.

[26] Richard S Sutton, Andrew G Barto, et al. *Reinforcement Learning: An Introduction*. MIT press Cambridge, 1998.

[27] Long Wen, Xinyu Li, and Liang Gao. "A new reinforcement learning based learning rate scheduler for convolutional neural network in fault classification". In: *IEEE Transactions on Industrial Electronics* 68.12 (2020), pp. 12890–12900.

[28] Zhen Xu, Andrew M Dai, Jonas Kemp, and Luke Metz. "Learning an adaptive learning rate schedule". In: *arXiv:1909.09712* (2019).

[29] Ke Xue, Xi Lin, Yunqi Shi, Shixiong Kai, Siyuan Xu, and Chao Qian. "Escaping local optima in global placement". In: *arXiv:2402.18311* (2024).

[30] Ke Xue, Jiacheng Xu, Lei Yuan, Miqing Li, Chao Qian, Zongzhang Zhang, et al. "Multi-agent dynamic algorithm configuration". In: *Advances in Neural Information Processing Systems 35*. New Orleans, LA, 2022, pp. 20147–20161.

[31] Junchi Yan, Xianglong Lyu, Ruoyu Cheng, and Yibo Lin. "Towards machine learning for placement and routing in chip design: A methodological overview". In: *arXiv:2202.13564* (2022).

[32] Zhi-Hua Zhou and Yu-Xuan Huang. "Abductive learning". In: *Neuro-Symbolic Artificial Intelligence: The State of the Art*. IOS Press, 2021, pp. 353–369.

[33] Zhi-Hua Zhou, Yang Yu, and Chao Qian. *Evolutionary Learning: Advances in Theories and Algorithms*. Springer, 2019.