iPO: Constant Liar Parameter Optimization for Placement with Representation and Transfer Learning

XINHUA LAI, University of Chinese Academy of Science, China
MIAO LIU, University of Chinese Academy of Science, China
XINGQUAN LI*, Pengcheng Laboratory, China
YIHANG QIU, University of Chinese Academy of Science, China
SHIJIAN CHEN, Pengcheng Laboratory, China and University of Chinese Academy of Science, China
XINHAO LI, University of Chinese Academy of Science, China
JUNGANG XU*, University of Chinese Academy of Science, China

Placement is a critical and time-consuming step in very-large-scale integration (VLSI) design flow. As placement methods continue to be researched, they introduce more parameters, making current methods for configuring parameters heavily reliant on human experience for each design. This paper proposes a novel cross-design parameter optimization method, iPO, to accelerate parameter tuning without human involvement in different placement engines (like iEDA-iPL and DREAMPlace). Specifically, we introduce a heuristic strategy called Constant Liar to accelerate parameter tuning, allowing us to optimize parameters concurrently on different machines. Our research indicates that optimizing parameters for every design is time-consuming. To address the inefficiency of parameter tuning, we propose a cross-design parameter transfer learning strategy. This strategy measures the cosine similarity between designs in collaboration with a graph embedding algorithm representing netlists and cells. Compared to DREAMPlace on ISPD2015 benchmarks, our method achieves average improvements of 9.8% in half-perimeter wirelength (HPWL) and 12.0% in route congestion. When compared to AutoDMP, iPO shows an average improvement of 11% in HPWL and 12.3% in congestion, along with a 3.49× speed-up in the number of search iterations. Furthermore, we extended our experiments to the iEDA-28nm benchmarks, showing average improvements of 4.7%, 2.7% and 2.8% in HPWL, worst negative slack (WNS) and total negative slack (TNS), respectively, compared to iEDA-iPL. Finally, our ablation studies on parallelization demonstrate that using 10 parallel processes results in approximately an 18× speed-up compared to using a single process.

CCS Concepts: • **Hardware** → **Placement**; • **Computing methodologies** → **Artificial intelligence**; **Concurrent computing methodologies**; **Search methodologies**.

Additional Key Words and Phrases: AI/ML, VLSI Placement, Design Space Exploration, Transfer Learning, Parallelization, Representation Learning

*Corresponding authors

This work was supported by the Major Key Project of PCL (No. PCL2023A03).
Authors' addresses: Xinhua Lai, laixinhua21@mails.ucas.ac.cn, University of Chinese Academy of Science, Beijing, China; Miao Liu, liumiao20@mails.ucas.ac.cn, University of Chinese Academy of Science, Beijing, China; Xingquan Li, lixq01@pcl.ac.cn, Pengcheng Laboratory, Shenzhen, China; Yihang Qiu, qiuyihang23@mails.ucas.ac.cn, University of Chinese Academy of Science, Beijing, China; Shijian Chen, chenshj@pcl.ac.cn, Pengcheng Laboratory, Shenzhen, China and University of Chinese Academy of Science, Beijing, China; Xinhao Li, lixinhao22@mails.ucas.ac.cn, University of Chinese Academy of Science, Beijing, China; Jungang Xu, xujg@ucas.ac.cn, University of Chinese Academy of Science, Beijing, China.

1084-4309/2025/XX-ARTXXX $15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Xinhua Lai, Miao Liu, Xingquan Li, Yihang Qiu, Shijian Chen, Xinhao Li, and Jungang Xu. 2025. iPO: Constant Liar Parameter Optimization for Placement with Representation and Transfer Learning. *ACM Trans. Des. Autom. Electron. Syst.* XX, X, Article XXX (XX 2025), 30 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Placement is a crucial step in the VLSI physical design flow, determining the positions of standard cells and macros on the layout. The quality of the placement results can significantly impact subsequent steps. Consequently, placement has been a focal point of research within the academic community for many years. Standard portions of the digital EDA design flow are global placement (GP), legalization (LG), and detailed placement (DP). In this paper, we focus mainly on GP.

A large number of algorithms have been proposed to address GP, which are mainly classified as partition-based placement, simulated annealing-based placement, and analytical placement. Analytical placement is now the state-of-the-art placement approach, including quadratic and nonlinear analytical placement. Specifically, nonlinear analytical placement is the main placement method, which is essentially a nonlinear optimization problem [29]. The fundamental principle of the nonlinear analytical placement method is to express a cost function and the constraints as analytical functions of cell locations, which can then be converted into a nonlinear programming problem. In the process of placement, the input is a circuit's netlist and a standard cell library. The placement engine, such as iEDA-iPL [16] and DREAMPlace [17], determines the location of cells and macros on the layout, ensuring that the overlaps between cells satisfy the density threshold.

iPL is an open-source placement tool [16], which uses the electric field density model and Weighted-Average wirelength (WAWL) model for global placement. Following the framework of analytical placement [4], many efforts have been made to accelerate global placement. DREAMPlace [17] based on ePlace/RePlAce family [5, 19–21], draws an analogy between analytical placement and deep learning (DL). By leveraging PyTorch as its API and combining both CPU and GPU acceleration, DREAMPlace achieves substantial improvements in speed and scalability.

On one hand, these placement methods mentioned above are powerful enough to perform placement tasks on VLSI designs. On the other hand, they bring more parameters that need to be configured to process more complicated placement tasks. As a result, configuring the appropriate parameters for each design typically involves experience and uncertain time [1, 2, 18]. Placement often also requires substantial human intervention to generate effective placement solutions, which makes parameter space exploration (PSE) necessary for free from manual configuration to some degree. Fig. 1 shows the 3D Pareto frontier of worst negative slack (WNS) and total negative slack (TNS), and half-perimeter wirelength (HPWL) on the design "gcd", which demonstrates that PSE can find much better parameters than manual configuration.

The motivation behind this research stems from the inherent challenges and limitations associated with traditional VLSI placement methodologies, which depend on human experience. Manual configuration often relies on common and simple numerical values, such as setting the target density to 0.9 on iEDA-iPL, which can overlook potentially optimal but uncommon values. As the complexity of VLSI designs continues to grow, manual parameter tuning becomes increasingly impractical and time-consuming [1, 30]. Manual configuration not only becomes more time-consuming and ineffective but also makes it impossible to reach the Pareto frontier of objective metrics like HPWL, WNS, and TNS. Furthermore, cross-design configuration is an impossible task with manual methods, often requiring significant time to configure appropriate parameters for each design. Therefore, there is a strong demand for a new, effective, and automated method to configure parameters across designs.

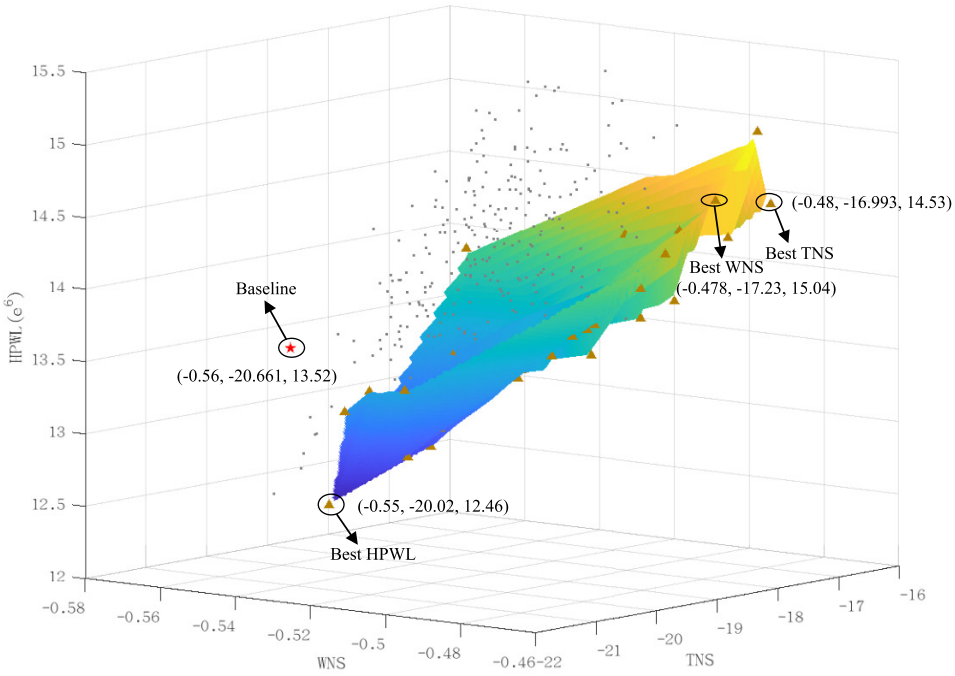


Fig. 1. The Pareto frontier of WNS (x-axis), TNS (y-axis), and HPWL (z-axis) on the design gcd from iEDA. The baseline point, marked as a red pentagon, represents the result achieved through manual configuration and is notably distant from the Pareto frontier. The point (-0.55, -20.02, 12.46) obtained through parameter space exploration (PSE) demonstrates significantly better WNS, TNS, and HPWL values compared to the baseline point. This illustrates the effectiveness of PSE in optimizing placement parameters to achieve superior performance metrics.

There are several researches in design space exploration (DSE). AutoDMP [2] was proposed to configure parameters automatically and concurrently by using multi-objective Bayesian optimizer [25], which can generate high-quality solutions. Compared to baseline results, AutoDMP also improves the macro placement quality. Unfortunately, AutoDMP never use transfer learning for transferring parameters, which may lead to unstable quality of results. Meanwhile, AutoDMP cannot configure parameters automatically on different machines and is primarily dedicated to macro placement. Another method [1] based on Reinforcement Learning (RL) has been proposed to optimize parameters, which is time-consuming with poor generalization across designs. In addition to research work in placement, Li [14] et al. proposed the PAMBOF framework for Coarse-Grained Reconfigurable Architecture (CGRA). This framework employs a deep neural network model as a surrogate to calculate the expected q-hypervolume improvement (q-EHVI) using Monte Carlo simulation. By efficiently exploring the design space, it achieves improved area and performance within a shorter runtime. While these works address specific challenges in design space exploration (DSE) within their respective fields, further research is needed to overcome time-consuming processes and limited generalization, particularly in the context of placement optimization.

To improve generalization across designs, we focus on transfer learning, which utilizes knowledge learned from one task and applies it to similar tasks. In this paper, parameters are treated as knowledge, and placement on different designs is viewed as different tasks. Thus, the key to transferring parameters from one design to another lies in the similarity between the two designs. To achieve this, we aim to study how to represent the features of the netlists and cells within a design. Ren [26] has highlighted the effectiveness of Graph Neural Networks (GNN) for EDA problems, demonstrating that GNNs can extract valuable information from circuits. GNNs primarily focus on studying relationships within graph-structured data [27]. By considering problems as networks of nodes and edges, classification [31] and representation learning [24] can be performed based on the relationships between these nodes and edges. Moreover, graphs have been applied in various research areas within EDA [6, 11, 12, 22].

Inspired by these studies, we treat cells and pins as nodes and wires as edges in a graph. Through this method, we aim to measure the similarity between the two designs, facilitating effective parameter transfer between them. In this paper, we propose iPO, which focuses on parameter optimization for placement, to address the issues of time-consuming parameter tuning and poor generalization across designs. The key contributions are summarized as follows:

(1) We introduce a heuristic strategy called Constant Liar to accelerate automated parameter tuning. This strategy optimizes parameters on different machines concurrently, significantly improving the efficiency of parameter tuning.
(2) We incorporate graph embedding technology to learn features from circuit netlists. In our method, we adopt Graph2vec with Weisfeiler-Lehman subgraph to encode cells and their neighboring cells.
(3) We propose a cluster-based parameter transfer learning strategy, which measures the similarity between two designs using cosine similarity and transfers parameters across designs through a modified K-Means clustering algorithm.
(4) In our experiments, compared to DREAMPlace on ISPD2015 benchmarks, our method achieves average improvements of 9.8% in HPWL and 12.0% in congestion. Additionally, compared to AutoDMP, iPO shows an average improvement of 10% in HPWL and 12.3% in congestion, along with a 3.3× speed-up in the number of search iterations (#s).
(5) Our framework is easy to extend, and can be easily applied to other placement engines (like DREAMPlace and iEDA-iPL) with simple configuration.

The subsequent sections of this paper are organized as follows. Section 2 primarily introduces the placement problem, and parameter optimization methods involved. Section 3 introduces our proposed framework. In Section 4, we present and analyze the experimental results. Finally, Section 5 summarizes our work.

2 PRELIMINARIES

2.1 Analytical Placement

In this section, we primarily introduce the theory of analytical placement and some key parameters. analytical placement, as described in [4], is currently the mainstream placement method and involves parameters that need to be configured. The analytical placement problem is treated as a nonlinear optimization problem, as shown in Eq. (1).

$$\min_{\mathbf{x},\mathbf{y}} \sum_{e \in E} \mathrm{WL}(e; \mathbf{x}, \mathbf{y}) \qquad \text{s.t. } \mathcal{D}(\mathbf{x}, \mathbf{y}) \leq \hat{\mathcal{D}} \tag{1}$$

where $\mathrm{WL}(\cdot;\cdot)$ denotes the wirelength function of a net instance $e \in E$, $\mathcal{D}(\cdot)$ denotes the density constraints, and $\hat{\mathcal{D}}$ is the target density specified by the user. If the density constraints are satisfied for all bins, it indicates that the cells are sufficiently spread out with adequate spacing.

To solve this problem, Eq. (1) can be transformed into a Lagrangian unconstrained optimization problem, where the solutions satisfy the constraint conditions and converge to an optimal solution, as shown in Eq. (2).

$$\min_{\mathbf{x},\mathbf{y}} \quad \sum_{e \in E} \mathrm{WL}(e;\mathbf{x},\mathbf{y}) + \lambda \mathcal{D}(\mathbf{x},\mathbf{y}) \tag{2}$$

where λ is the penalty factor [8]. In iEDA-iPL [16] and DREAMPlace [17], λ is called the density penalty coefficient or density weight. Specifically, The non-differentiable HPWL function ($\mathrm{WL}(\cdot;\cdot)$) is estimated by the Weighted-average wirelength (WA) model [9].

$$\mathrm{WA}_e = \frac{\sum_{i \in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{i \in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{-\frac{x_i}{\gamma}}} \tag{3}$$

where γ is a parameter to control the smoothness and accuracy of the approximation to HPWL. The smaller the value of γ, the more accurate but less smooth the HPWL approximation becomes.

In iPL and DREAMplace, the GP constraints correlate with the electrostatic equilibrium's system state. Specifically, they are members of ePlace/RePlAce family [5, 19], where density penalty is modeled as potential energy, and density gradient is modeled as the electric field. According to Poisson's equation from the charge density distribution [19, 21], the electric potential and the field can be calculated by Eq. (4)

$$\begin{cases} \nabla \cdot \nabla \psi(x,y) & = -\rho(x,y), \\ \hat{\mathbf{n}} \cdot \nabla \psi(x,y) & = \mathbf{0}, \quad (x,y) \in \partial \mathcal{R} \\ \iint_{\mathcal{R}} \rho(x,y) & = \iint_{\mathcal{R}} \psi(x,y) = 0 \end{cases} \tag{4}$$

where $\mathcal{R}$ denotes the placement region, $\partial \mathcal{R}$ denotes the boundary of the region, $\hat{\mathbf{n}}$ denotes the outer normal vector of the region, ρ denotes the charge density, and the ψ denotes the electric potential. The numerical solution of the electric potential and field distribution can be obtained by solving Poisson's equation.

In analytical placement, target density $\hat{\mathcal{D}}$, density weight λ, and the smoothness coefficient γ can be configured manually, except for these parameters, we can select HPWL smoothness model for our placement. And there are other parameters can be configured, which will be introduced in Section 3.2.

2.2 Neural Graph Embedding Models

In this section, we introduce the basic theory of graph embedding technology used in our paper. In the field of Natural Language Processing (NLP), word2vec [23] uses a simple and efficient feed-forward neural network architecture called "Skipgram" to learn distributed representations of words. Given an sequence of words $\{w_1, w_2, \cdots, w_T\}$, the target word representation w_t can be learned by the Skipgram model, the model aims to maximize the log-likelihood of the context words given the center word. This is represented as:

$$\sum_{t=1}^{T} \log P(w_{t-c}, \ldots, w_{t+c} | w_t) \tag{5}$$

where w_t is the center word, and $w_{t-c}, \ldots, w_{t+c}$ are the context words within a window of size $2c$ around w_t. The probability $P(w_{t-c}, \ldots, w_{t+c}|w_t)$ is defined as:

$$P(w_{t-c}, \ldots, w_{t+c}|w_t) = \prod_{-c \le j \le c, j \neq 0} P(w_{t+j}|w_t) \tag{6}$$

Each $P(w_{t+j}|w_t)$ is computed using the softmax function:

$$P(w_{t+j}|w_t) = \frac{\exp(\vec{w}_t \cdot \vec{w}'_{t+j})}{\sum_{w \in V} \exp(\vec{w}_t \cdot \vec{w})} \tag{7}$$

where $\vec{w}_t$ is the embedding vector for the center word w_t, $\vec{w}'_{t+j}$ is the embedding vector for a context word w_{t+j}, V is the vocabulary of all words.

Based on the Skipgram model, Le and Mikolov proposed Doc2vec [13], an extension of word2vec that transitions from learning embeddings of words to those of word sequences. This model is capable of learning representations for arbitrary-length word sequences, such as sentences, paragraphs, and even entire documents. Specifically, given a set of documents $D = \{d_1, d_2, \ldots, d_N\}$ and a sequence of words $c(d_i) = \{w_1, w_2, \ldots, w_{l_i}\}$ sampled from document $d_i \in D$, Doc2vec uses Skipgram to learns δ-dimensional embeddings of the document $d_i \in D$ and each word w_j sampled from $c(d_i)$. This results in $\tilde{d}_i \in \mathbb{R}^\delta$ and $\tilde{w}_j \in \mathbb{R}^\delta$ respectively. The model operates by considering a word $w_j \in c(d_i)$ as occurring in the context of document d_i and aims to maximize the following log likelihood:

$$\sum_{j=1}^{l_i} \log P(w_j|d_i) \tag{8}$$

where the probability $P(w_j|d_i)$ is defined as:

$$P(w_j|d_i) = \frac{\exp(\vec{d}_i \cdot \vec{w}_j)}{\sum_{w \in V} \exp(\vec{d}_i \cdot \vec{w})} \tag{9}$$

where V represents the vocabulary of all words across all documents in D. Graph2vec [24] treated graphs as an analogy to documents. In Graph2vec, graphs are composed of rooted subgraphs, which are analogous to words from a special language. This approach extends document embedding models to learn graph embeddings. In our method, we use this method to learn graph embedding of netlists and cells.

2.3 Expected Improvement

Before we introduce Expected Improvement (EI), we first introduce Ordinary Kriging (OK) for better understanding of EI. Ordinary Kriging is a popular Kriging metamodel, which provides a mean prediction model and allows for the quantification of prediction accuracy at each point [7], which can be seen as a parameter point. The mean and variance at a point $\mathbf{x}$ are given by Eq. (10) and Eq. (11) respectively:

$$m_{\mathrm{OK}}(\mathbf{x}) = \left[\mathbf{c}(\mathbf{x}) + \left(1 - \frac{\mathbf{c}(\mathbf{x})^T \Sigma^{-1} \mathbf{1}_n}{\mathbf{1}_n^T \Sigma^{-1} \mathbf{1}_n}\right) \mathbf{1}_n\right]^T \Sigma^{-1} \mathbf{Y} \tag{10}$$

$$s^2_{\mathrm{OK}}(\mathbf{x}) = \sigma^2 - \mathbf{c}(\mathbf{x})^T \Sigma^{-1} \mathbf{c}(\mathbf{x}) + \frac{(1 - \mathbf{1}_n^T \Sigma^{-1} \mathbf{c}(\mathbf{x}))^2}{\mathbf{1}_n^T \Sigma^{-1} \mathbf{1}_n} \tag{11}$$

where $\mathbf{c}(\mathbf{x}) := \left[c\left(Y(\mathbf{x}), Y(\mathbf{x}^{(1)})\right), \ldots, c\left(Y(\mathbf{x}), Y(\mathbf{x}^{(n)})\right)\right]$ represents the covariance vector between location $\mathbf{x}$ and the sample points, Σ is the covariance matrix of the sample points, $\mathbf{Y}$ is the vector of observed values at the sample points, $\mathbf{1}$ is a vector of all ones, and σ^2 is the variance vector of

the sample points. We then introduce the theory of Expected Improvement (EI), which is used to optimize parameters. EI [28] is an alternative method used to maximize the expected value of improvement in Eq. (12):

$$\begin{aligned} \mathrm{EI}(\mathbf{x}) &= \mathbb{E}[(\min(Y(\mathbf{X}) - Y(\mathbf{x}))^{+} | Y(\mathbf{X}) = \mathbf{Y}] \\ &= \mathbb{E}[\max\{0, \min(Y(\mathbf{X})) - Y(\mathbf{x})\} | Y(\mathbf{X}) = \mathbf{Y}] \end{aligned} \tag{12}$$

where $\mathbf{X}$ represents the history set of points, specifically comprising all configurations generated during previous evaluations. This method also considers the magnitude of improvement. EI measures the expected amount of improvement when sampling at point $\mathbf{x}$. In practice, if $Y(\mathbf{x})$ is higher than $\min(\mathbf{Y})$, the improvement is 0; otherwise, the improvement is $(\min(\mathbf{Y}) - Y(\mathbf{x}))$. Based on OK, knowing the conditional distribution of $Y(\mathbf{x})$, EI can be computed in Eq. (13):

$$\begin{aligned} \mathrm{EI}(\mathbf{x}) = (\min(\mathbf{Y}) - m_{\mathrm{OK}}(\mathbf{x}))\Phi\left(\frac{\min(\mathbf{Y}) - m_{\mathrm{OK}}(\mathbf{x})}{s_{\mathrm{OK}}(\mathbf{x})}\right) \\ + s_{\mathrm{OK}}(\mathbf{x})\phi\left(\frac{\min(\mathbf{Y}) - m_{\mathrm{OK}}(\mathbf{x})}{s_{\mathrm{OK}}(\mathbf{x})}\right) \end{aligned} \tag{13}$$

where Φ denotes the cumulative distribution function (CDF) of the standard normal distribution $N(0, 1)$, ϕ represents its probability density function (PDF).

2.4 Tree-structured Parzen Estimator (TPE)

The aforementioned method can be practically used to interpolate the parameter sample point of the placement engine in the placement process. Here, we will focus on the Tree-structured Parzen Estimator (TPE) [3]. This algorithm is also a parameter configuration method based on expected improvement. However, unlike the previously introduced OK-based algorithm, we need to set a threshold y^* internally to divide two layout density functions $l(x)$ and $g(x)$, thereby defining a conditional probability $p(x|y)$ as follows:

$$p(x|y) = \begin{cases} l(x), & y < y^* \\ g(x), & y \geq y^* \end{cases} \tag{14}$$

where $l(x)$ is the function for all observed samples $\{x^{(i)}\}$ whose corresponding metric values $f(x^{(i)})$ are less than y^*, and the density function for the remaining samples is $g(x)$. Assuming the probability that a parameter x corresponds to a metric value y less than y^* is denoted as $p(y < y^*) = \gamma'$. The expected improvement using the TPE algorithm can be expressed as:

$$\begin{aligned} \mathrm{EI}_{y^*}(\mathbf{x}) &= \mathbb{E}\left[y^* - Y(\mathbf{x}) | Y(\mathbf{X}) = \mathbf{Y}\right] \\ &= \int_{-\infty}^{y^*} (y^* - y) p(y|\mathbf{x}) dy \\ &= \int_{-\infty}^{y^*} (y^* - y) \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} dy \end{aligned} \tag{15}$$

then we can further refine and differentiate EI in Eq. (16) as follows:

$$\begin{aligned}\mathrm{EI}_{y^*}(\mathbf{x}) &= \frac{y^* \int_{-\infty}^{y^*} p(\mathbf{x}|y)p(y)dy - \int_{-\infty}^{y^*} yp(\mathbf{x}|y)p(y)dy}{p(\mathbf{x})} \\ &= \frac{y^* l(x)\gamma' - l(x)\int_{-\infty}^{y^*} yp(y)dy}{\int_{-\infty}^{y^*} p(\mathbf{x}|y)p(y)dy + \int_{y^*}^{+\infty} p(\mathbf{x}|y)p(y)dy} \\ &= \frac{y^* l(\mathbf{x})\gamma' - l(\mathbf{x})\int_{-\infty}^{y^*} yp(y)dy}{\gamma' l(\mathbf{x}) + (1-\gamma')g(\mathbf{x})} \propto \left(\gamma' + \frac{g(\mathbf{x})}{l(\mathbf{x})}(1-\gamma')\right)^{-1}\end{aligned} \tag{16}$$

so that to maximize EI we use high probability under $l(\mathbf{x})$, and low probability under $g(\mathbf{x})$, in Eq. (16), we will get the point $\mathbf{x}$ with the greatest EI value.

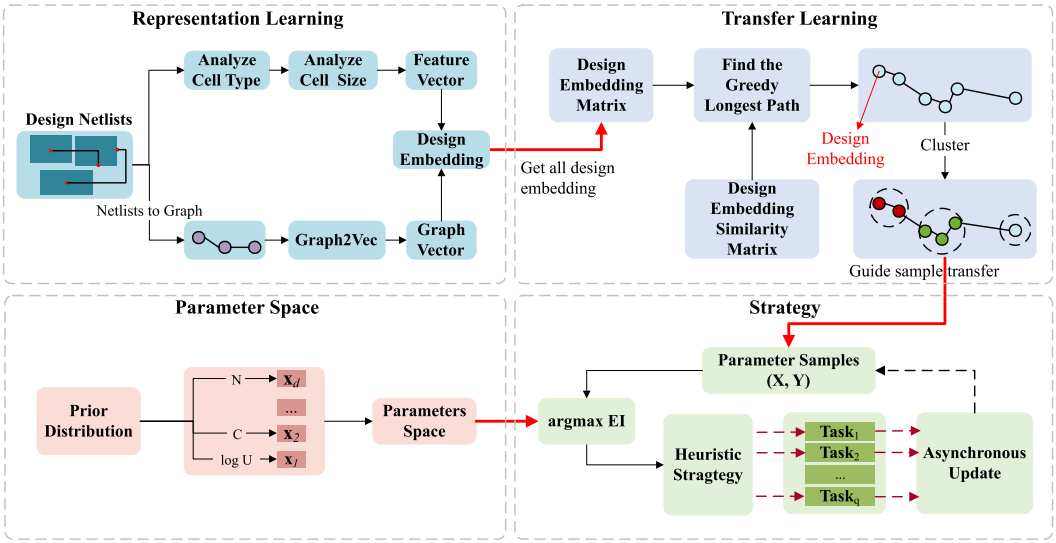


Fig. 2. The iPO framework includes four parts: Representation Learning, Parameter Space, Strategy, and Transfer Learning. In Representation Learning, the input is design netlists files, and the output is design embeddings that encode netlist graphs and design properties. The Parameter Space part takes parameter configuration files as input and defines a parameter space with a prior distribution to maximize Expected Improvement (EI), producing the parameter space as output. The Strategy part uses the transfer learning path and parameter space as input, generating q pairs of (parameters, metrics) for each design as output. In Transfer Learning, the input consists of design embeddings and a similarity matrix, and the output is a transfer learning path that clusters and guides parameter sampling across designs. Overall, the framework's input is design netlists and parameter configuration files, and the output is q (parameter, metric) pairs.

3 iPO FRAMEWORK

In this section, we introduce our proposed framework iPO. The overall flow of iPO is depicted in Fig.2, which is easy to be extended to different placement engines, such as iEDA-iPL, DREAMPlace and run on different machines concurrently. The framework also takes transfer learning technology into account for addressing low-effective cross-design parameter tuning. The entire framework of iPO consists of four main components: Representation Learning, Parameter Space, Strategy, and Transfer Learning.

- **Representation Learning**: This component focuses on creating a comprehensive feature vector. We concatenate the feature vector of cell type and size to a graph vector obtained through the Graph2vec algorithm. This combined vector effectively represents the netlist structure, capturing essential information about the cells and their connections.
- **Parameter Space**: we define the parameter space with prior distribution to maximize the Expected Improvement (EI). This space includes various parameters necessary for the placement tasks, establishing the foundation for efficient optimization.
- **Strategy**: This component deals with parameter tuning and parallelization. It involves generating parameter samples for each design and executing placement tasks concurrently on different processes or machines. This parallel approach accelerates the parameter tuning process, enhancing efficiency and effectiveness.
- **Transfer Learning**: The component focuses on transferring parameters to other designs. We use a cluster-based parameter transfer learning strategy that measures the similarity between designs using cosine similarity. Parameters are then transferred across designs through a modified K-Means clustering algorithm, ensuring the effective reuse of optimized parameters for similar designs.

3.1 Representation Learning

The objective of placement is to arrange all the macro cells and standard cells on a chip. These cells are logically connected by the netlist-defined connections. From a graph perspective, the cells in the netlist and their logical connections form nodes and edges.

3.1.1 Graph Based on Netlist. As illustrated in the top left of Fig. 3, the diagram represents a Johnson Ring Counter circuit, which comprises four Flip-Flops and two logic gates. We treat the Flip-Flops and logic gates (referred to as cells) as nodes in a graph, and the connections between nodes as edges. In Fig. 3, we treat the Flip-Flops and logic gates as nodes and construct a graph, which encapsulates the structure information of the circuit and contains the information of the number of nets and cells. The overall process of graph construction is depicted in Fig. 3.

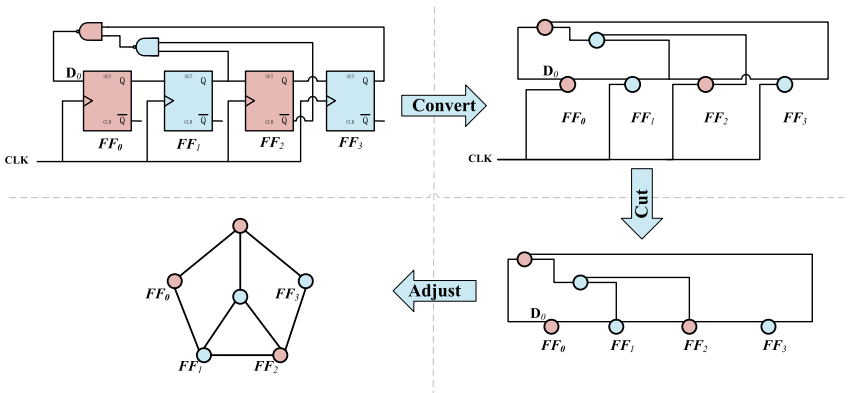


Fig. 3. The graph connections are based on netlist. Firstly, we convert Flip-Flops and logic gates into nodes (represented in blue). Next, we remove unrelated connections. After this cutting process, we adjust the relative positions of the nodes.

3.1.2 Netlist Representation Learning. we considers the relationships between nodes and edges from both cell and net perspectives. Graphs are treated analogously to documents, composed

Algorithm 1 Netlist Representation Learning Algorithm

Input: different designs netlist files $\mathbb{N} = \{N_1, N_2, \ldots, N_n\}$; number of dimensions (default 128) δ; design feature size ω

Output: embedding matrix for different designs $\mathbf{V} = \{v_1, v_2, \ldots, v_n\}$

function CircuitGraphLearning($\mathbb{N}, \delta, \omega$)
  $\mathbb{G}_c \leftarrow constructGraphByCells(\mathbb{N})$ ▷ $\mathbb{G}_c = \left\{G_c^{(1)}, G_c^{(2)}, \ldots, G_c^{(n)}\right\}$
  $D \leftarrow \text{maxDegree}(\mathbb{G}_c)$ ▷ the maximum number of neighbouring
  $\mathbf{V}_{n\times\delta} = \text{Graph2vec}(\mathbb{G}_c, \delta, D)$ ▷ get a $n \times \delta$ graph embedding of different circuit designs
  $\mathbf{U}_{n\times 1} = [\phi_1, \phi_2, \cdots, \phi_n]$ ▷ initialize a $n \times 1$ vector
  $M \leftarrow getCellTypeNum(\mathbb{N})$ ▷ get the number of all cell types from all circuit designs $\mathbb{N}$
  for $i \leftarrow 1, n$ **do**
    $w = (0.0^1, 0.0^2, \ldots, 0.0^M)$ ▷ $w_m (m \in [0, M])$ denotes the sum of width of the type of m-th cell
    $w_* = (0.0^1, 0.0^2, \ldots, 0.0^M)$ ▷ $w_m^* (m \in [0, M])$ denotes the maximum width of the m-th cell type
    $s = (0^1, 0^2, \ldots, 0^M)$ ▷ $w_m^* (m \in [0, M])$ denotes the number of the type of m-th cell
    $C_i \leftarrow getAllCells(N_i)$ ▷ get cells set from design N_i
    for each $c_j \in C_i$ **do**
      $w^j += c_j.width$ ▷ plus the width of cell c_j to w_j
      $s^j += 1$ ▷ plus 1 to s_j
      if $w_*^j < w^j$ **then**
        $w_*^j = w^j$ ▷ update the maximum width of the m-th cell type
      end if
    end for
    $\mathbf{U}_i = \frac{w}{s \cdot w^*}$ ▷ vector $\mathbf{U}_i$ contains information of cell type and size
  end for
  $\mathbf{U}'_{n\times\omega} = PCA(\mathbf{U}_{n\times M}, \omega)$ ▷ reduce the dimension of $\mathbf{U}$ to $n \times \omega$ using principal components analysis
  $\mathbf{V}_{n\times(\delta+\omega)} = \mathbf{V}_{n\times\delta} \oplus \mathbf{U}'_{n\times\omega}$ ▷ concatenate graph embedding matrix $\mathbf{V}$ concatenate to $\mathbf{U}'$
  return $\mathbf{V}$
end function

of rooted subgraphs that are analogous to words from a special language. This method extends document embedding models to learn graph embeddings of a circuit.

We introduce the Graph2vec algorithm (shown in Algorithm 1) to represent different netlist structures as embedding vectors. As discussed above in Section 3.1.1, we can determine connections between two cells and know the number of nets and cells from the graph. Therefore, the embedding vector encapsulates the information of the netlist structure, including the number of cells and nets. To contains more information about the cells, we consider cell size and type.

A graph representation learning algorithm is used to represent each graph as a embedding vector, which contains information on sizes and types of all cells. Specifically, as shown in line 9 of *Graph2vec* of Algorithm 2, the *GetWLSubgraph* function follows the well-known Weisfeiler-Lehman relabeling process to extract subgraphs.

3.2 Parameter Space

3.2.1 DREAMPlace Parameter Analysis. We use 13 parameters to form the effective parameter space. These parameters are summarized in Table 1.

3.2.1.1 Original Parameters. The last nine parameters in Table 1 represent the original parameters from DREAMPlace. In machine learning tasks, the parameters related to gradient descent are considered as hyperparameters, which significantly affects the results. Taking inspiration from this

Algorithm 2 Netlist Graph2vec Algorithm

Input: Netlist graph set of different designs $\mathbb{G} = \left\{G^{(1)}, G^{(2)}, \ldots, G^{(n)}\right\}$; number of dimensions δ, maximum degree of rooted subgraphs D, number of epochs T
Output: embedding matrix for different designs $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$

function GRAPH2VEC($\mathbb{G}$, δ, D, T)
  Initialize $n \times \delta$ embedding matrix $\mathbf{V}$
  for $i \leftarrow 1, T$ **do**
    $\mathbb{B} = Shuffle(\mathbb{G})$
    for $G^{(i)} \in \mathbb{B}$ **do**
      $\mathbf{N}_i \leftarrow traverse(G^{(i)})$ ▷ set of all nodes in $G^{(i)}$
      for $c \in \mathbf{N}_i$ **do**
        for $d \leftarrow 0, D$ **do** ▷ use Weisfeiler-Lehman kernel to encode subgraph rooted at c
          $sg_c^{(d)} = GetWLSubgraph(c, G^{(i)}, d)$ ▷ $sg_c^{(d)}$ encode a d-degree subgraph rooted at c in $G^{(i)}$
          $J(\mathbf{V}) = -\log P(sg_c^{(d)} | G^{(i)})$ ▷ Doc2vec algorithm, $sg_c^{(d)}$ seen as word, $G^{(i)}$ seen as document, compute log-likelihood defined in Eq. (8)
          $\mathbf{V} = \mathbf{V} - \alpha \frac{\partial J}{\partial \mathbf{V}}$ ▷ update $\mathbf{V}$ using backpropagation
        end for
      end for
    end for
  end for
end function

Input: Node which acts as the root of the subgraph c; Graph $\mathbf{G}$, Degree of neighbours d
Output: Rooted subgraph of degree d around node c $sg_c^{(d)}$

function GETWLSUBGRAPH(c, G, d)
  $sg_c^{(d)} = \{\}$
  if d=0 **then**
    $sg_c^{(d)} \leftarrow Label(c)$ ▷ get label of node c
  else
    $\mathbb{C}_c \leftarrow \{c' | (c, c') \in E\}$ ▷ neighbour nodes set, E denotes edges set of G
    $\mathbf{M}_c^{(d)} \leftarrow \{GetWLSubgraph(c', G, d-1) | c' \in \mathbb{C}_c\}$
    $sg_c^{(d)} \leftarrow sg_c^{(d)} \cup GetWLSubgraph(c, G, d-1) \oplus sort(\mathbf{M}_c^{(d)})$
  end if
  return $sg_c^{(d)}$
end function

technology in machine learning, we take the learning rate, gradient descent (GD) method, and the remaining seven parameters as hyperparameters.

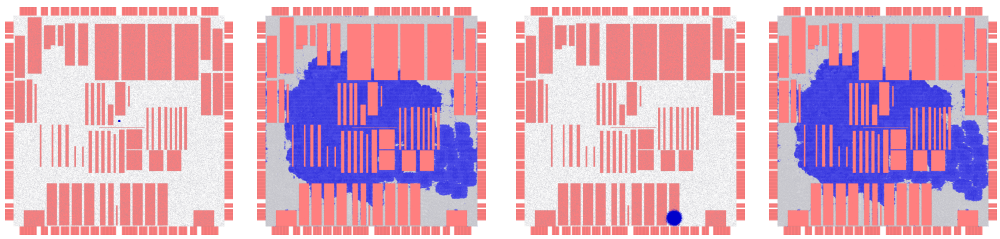
3.2.1.2 Initial Cell Positions Parameters. In DREAMPlace, analytical placement is analogous to training a neural network, which is inherently a nonlinear optimization problem. Just as effective weight parameter initialization in deep learning accelerates model convergence, initializing cell positions in analytical placement plays a similar role. By leveraging this analogy, we aim to initialize cell positions closer to their converged positions to enhance the convergence speed of placement optimization. This approach is expected to reduce placement convergence time significantly.

DREAMPlace initializes all cells at the center of the placement. This methods brings inflexible initialization to cell positions and hinders better quality from being generated. We define four parameters related to initial cell positions, they are x_{loc}, y_{loc}, x_{scale} and y_{scale}. x_{loc} and y_{loc} are the

Table 1. Parameter List of DREAMPlace.

Parameter	Description	*Prior Distribution*
x_{loc}	x location	$U(0.001, 1.0)$
y_{loc}	y location	$U(0.001, 1.0)$
x_{scale}	x location scale	$U(0.001, 1.0)$
y_{scale}	y location scale	$U(0.001, 1.0)$
$bins_{num}$	number of bins	{64, 128, 256, 512, 1024, 2048}
$HPWL_{model}$	HPWL smooth model	{weighted_average, logsumexp}
GD_{method}	Gradient descent method on GP	{nesterov, adam, sgd, sgd_momentum, sgd_nesterov}
α	GP learning rate	$U(0.001, 0.05)$
λ	density weight	$U(e^{-6}, 2e^{-4})$
d_{target}	target density	$U(0.3, 1.0)$
γ	gamma	$U(0.01, 0.02)$
GP_{ratio}	GP noise ratio	$U(0.01, 0.05)$
τ	stop overflow	$U(0.05, 0.15)$

U denotes uniform distribution.



(a) (b) (c) (d)

Fig. 4. Different placement results with original initialization and our initialization. (a) and (c) show the original initial cells positions and our initial cells positions respectively, blue dots is cells. (b) shows the original final placement where iteration is 610. (d) shows the final placement of cells position parameters where iteration is 520.

x and y positions center respectively, x_{scale} and y_{scale} are the x and y positions scale, which denote the standard deviation in normal distribution and affect the results of initial cell positions. x and y coordinates of initial cell positions are generated by Eq. (17) and Eq. (18) respectively:

$$f(x) = \frac{1}{\sqrt{2\pi}x_{scale}} \exp\left(-\frac{(x - x_{loc})^2}{2x_{scale}^2}\right) \tag{17}$$

$$f(y) = \frac{1}{\sqrt{2\pi}y_{scale}} \exp\left(-\frac{(y - y_{loc})^2}{2y_{scale}^2}\right) \tag{18}$$

Therefore, setting different values to these parameters can change the initial cells positions on the layout. Fig. 4 shows the effect of different initial locations on the final placement.

Table 2. Parameter List of iEDA-Placement in the stage of global placement.

Parameter	Description	*Prior Distribution*
init_wirelength_coef	initial wirelength coefficient	$U(0.1, 0.5)$
min_wirelength_force_bar	minimum wirelength force bar	$U(-500.0, -50.0)$
target_density	target density	$U(0.8, 1.0)$
bin_cnt	the number of bin on vertical and horizontal direction	{16, 32, 64, 128, 256,512,1024}
max_backtrack	the maximum number of backtracks	$U(5, 50)$
init_density_penalty	initial density penalty coefficient	$U(0.0, 0.001)$
target_overflow	target overflow	$U(0.0, 0.2)$
initial_prev_coordi_update_coef	coefficient for initial perturbation of coordinates	$U(50.0, 1000.0)$
min_precondition	minimum precondition	$U(1.0, 10.0)$
min_phi_coef	minimum phi coefficient	$U(0.75, 1.25)$
max_phi_coef	maximum phi coefficient	$U(0.75, 1.25)$

U denotes uniform distribution.

3.2.2 iEDA Placement Parameters Analysis. In the stage of global placement of iEDA flow [15], there are 11 parameters that need to be configured. We use AiEDA [10] (AI library for EDA) to run netlist-GDS flow. To optimize these parameters, we perform parameter tuning. The parameters are summarized in Table 2, where *bin_cnt_x* and *bin_cnt_y* are set to be equal and represented by a single parameter, *bin_cnt*, which can take values within a specified range. We utilize a prior distribution to define the other parameters, subsequently creating a parameter space for the iEDA placement. To evaluate parameter sensitivity, we employ Random Forest (RF) - a well-established method for assessing individual parameter impacts on performance metrics. Using 2000 historical samples from the gcd design, we analyze each parameter's influence on HPWL, TNS, and WNS. The importance analysis (Fig. 5) reveals that *max_phi_coef* exhibits the strongest correlation with all three target metrics. These insights enable more strategic parameter selection and tuning refinement to enhance placement quality.

3.3 Strategy

3.3.1 Sequential Model-based Optimization. In the stage of placement, based on EI, the Tree-structured Parzen estimator (TPE) is designed to optimize objective function by Algorithm 3. In our method, $\mathbf{x}$ is treated as parameters vector which need to be configured, $y(\mathbf{x})$ is seen as a metric for measuring wirelength, timing and congestion. We can search a group of parameters by using Algorithm 3, then we configure parameters for placement. With these parameters, y is evaluated by placement engine.

3.3.2 Constant Liar Strategy. In Sequential Model-based Optimization (SMBO), we sample parameter point concurrently to accelerate optimization intuitively. But this may result in the next group of parameters being generated in the same Expected Improvement state (see Fig. 7 (a)), which will cause a waste of computing resources without performance improvement. To address this issue, we use q-points expected improvement (q-EI) [7] to accelerate parameter tuning. The q-EI can yield several points at each iteration, which suits the parallelization well. The q-EI maximization of

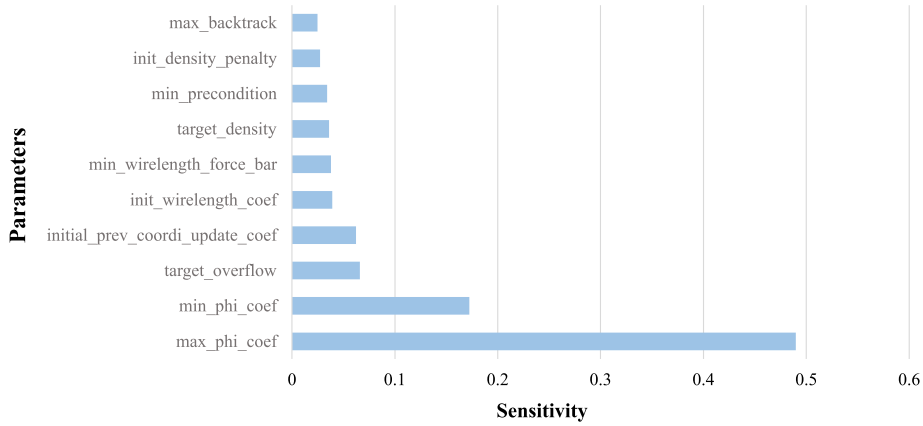


Fig. 5. Parameter sensitivity to objective metrics (HPWL, WNS, and TNS) in the design gcd.

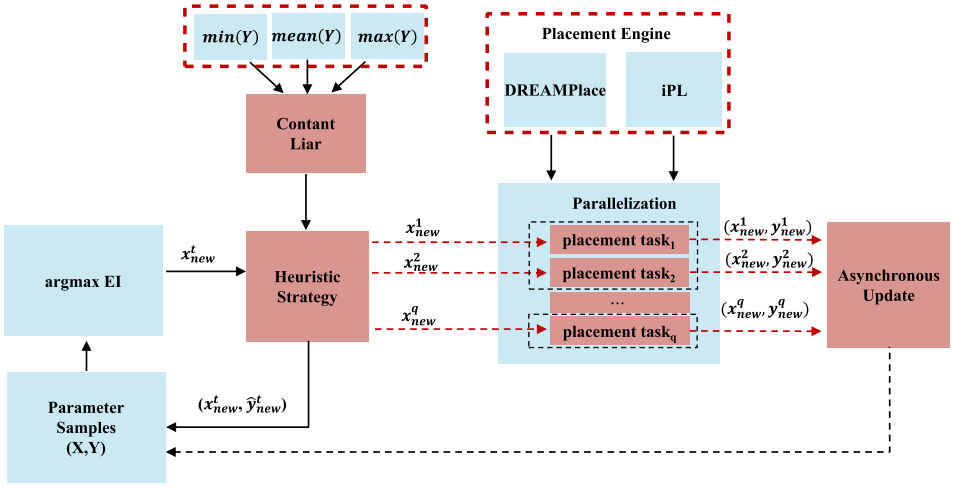


Fig. 6. The overall flow of parameter tuning. When $q = 1$, the sample x^t_{new} is directly taken by the parallelization module where only one placement task is running, indicating that the Heuristic Strategy module is disabled. The update module updates the real metric value for the sample with the "lie" constant L. The parallelization module performs the placement tasks concurrently; placement task 1 and task 2 run on different processes, while task q is executed on another machine.

multiple parameter groups is shown as

$$(\mathbf{x}^{n+1}, \mathbf{x}^{n+2}, \ldots, \mathbf{x}^{n+q}) = argmax_{\mathbf{x}\in D^q}[EI(\mathbf{x})] \tag{19}$$

where q is the number of parameter sets generated at each iteration, and D is parameter space. $\mathbf{x}^{n+m}$ denotes the mth group of parameters generated by parameter space in nth iteration. $\mathbf{X}$ denotes the set of parameters that have been generated. Given the computational complexity of a direct q-EI maximization, we use Constant Liar strategy (as shown in Algorithm 4) to approximate q-EI maximization [7]. In Constant Liar strategy, we use a “lie” constant to generate multiple groups of parameters, which reduces the high computational cost. As shown in Fig. 6, the parallelization

Algorithm 3 Placement Tuning Algorithm for Placement

Input: Parameter set $\mathbf{X}$, value set $\mathbf{Y}$, threshold y^*, the number of epoch T ;
Output: new parameter set $\mathbf{X}$, new value set $\mathbf{Y}$

function FINDMAXEI($\mathbf{X}$, $\mathbf{Y}$, y^*, T, q)
  for $i \leftarrow 1, T$ **do**
    $\hat{\mathbf{x}}_{1\times m} = argmax_{\mathbf{x}} EI_{y^*}(\mathbf{x})$ ▷ m is the number of configuration parameter, argmax denotes Eq. (16)
    $y \leftarrow PlacementEngine(\mathbf{x})$ ▷ Get y from placement engines like iEDA-iPL, DREAMPlace
    Adjust $p(y|\mathbf{x})$ by y^*
    $\mathbf{X}_{i\times m} = \mathbf{X}_{(n+i-1)\times m} \cup \{\mathbf{x}\}$
    $\mathbf{Y}_{i\times 1} = \mathbf{Y}_{(n+i-1)\times 1} \cup \{y\}$
  end for
end function

module runs placement tasks on different processes or machines concurrently, applying different parameters to the same circuit design.

In Algorithm 4, $\mathbf{X}$ denotes the set of parameter generated, $\mathbf{Y}$ denotes the HPWL value set corresponding to current $\mathbf{X}$, q denotes the number of parameter sets generated at each iteration. Fig. 7 (a) shows the point distribution, where points are concentrated in AutoDMP, and many points have the same EI state. This reduces the exploration ability of the parameter space. Fig. 7 (b) shows the point distribution in iPO, where points are scattered more effectively, enabling better exploration of the parameter space. To further analyze the distributions, we compare the standard deviation as a percentage of the mean for both approaches. This metric is displayed in the top-right corner of each subfigure (denoted as lr for learning rate and dw for density weight) for both the first and second iterations. In AutoDMP, the percentages are 168.96% in the first iteration and 127.45% in the second iteration. These values indicate that, in AutoDMP, a few points are sampled far away from the rest, while the majority of points are clustered in adjacent areas.

Algorithm 4 *q-EI* Constant Liar Algorithm for Placement

Input: The parameter set $\mathbf{X}$, the metric value set $\mathbf{Y}$, the number of parameters q, the "lie" constant (can be set to mean($\mathbf{Y}$), max($\mathbf{Y}$), or min($\mathbf{Y}$)) L, threshold y^* ;
Output: The updated parameter set $\mathbf{X}$, the updated metric value set $\mathbf{Y}$

procedure GREEDYQCL($\mathbf{X}$, $\mathbf{Y}$, L, y^*, q)
  for $i \leftarrow 1, q$ **do**
    $\mathbf{x}^{(n+i)}|_{1\times m} = argmax_{\mathbf{x}} EI_{y^*}(\mathbf{x})$ ▷ m is the number of configuration parameter, argmax denotes Eq. (16)
    $\mathbf{X}_{(n+i)\times m} = \mathbf{X}_{(n+i-1)\times m} \cup \{\mathbf{x}^{(n+i)}\}$
    $\mathbf{Y}_{(n+i)\times 1} = \mathbf{Y}_{(n+i-1)\times 1} \cup \{L\}$
  end for
  for $i \leftarrow 1, q$ **do**
    $y^{(n+1)} \leftarrow PlacementEngine(\mathbf{x}^{(n+1)})$
    $\mathbf{Y}^{(n+1)} \leftarrow y^{(n+1)}$
  end for
  return $\mathbf{X}, \mathbf{Y}$
end procedure

3.3.2.1 Samples. In the overall framework (shown in Fig. 6), the Samples module is primarily responsible for maintaining all sample tuples ($\mathbf{X}$, $\mathbf{Y}$). Initially, the Heuristic Strategy module transfers

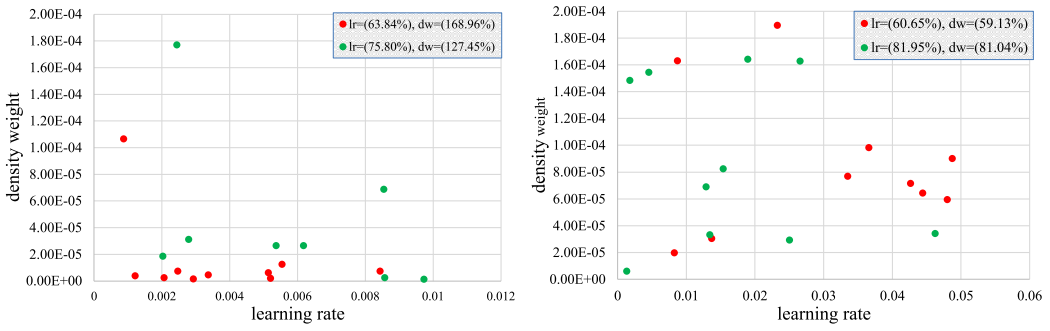


Fig. 7. Two parameters, density weight and GP learning rate, follow a uniform distribution. Red points represent the first group of parameters generated by the parameter space, while green points represent the second group.

the sample tuple (x_{new}, $\hat{y}_{new}$= L) to the Samples module. Here, $\hat{y}_{new}$ will be replaced by the real y_{new} after placement. As the sample size increases, we aim for the parameter distribution to progressively align more closely with the real distribution.

3.3.3 Placement Engine Adapting. During parameter optimization, we observed that for certain parameter groups, convergence in DREAMPlace terminates prematurely on the circuit *mgc_fft_b* (as shown in Fig. 8). This premature termination is caused by a fixed overflow threshold in DREAMPlace, which is designed to handle abnormal placement cases. However, in our experiments, this threshold mistakenly interrupts the placement process. To address this issue, we propose a self-adaptive overflow threshold to achieve smoother termination. This adaptive method allows for larger overflow values during the early stages of placement, and gradually tightens the threshold in the later stages to refine the placement results. The self-adaptive convergence adjustment is governed by the following equation:

$$threshold = \mu(1-\theta) + \tau\theta, \ \ \theta = S(\nu t - c) \tag{20}$$

where $c = 6$, $\mu = 0.5$, $\nu = 0.01$, t is current iteration and τ is last overflow. $S(x) = (1+e^{-x})^{-1}$ denotes the sigmoid function. Convergence finishes when current overflow rate of increase is greater than or equal to $threshold$ by Eq. (20).

iPL [16] aims to determine cell positions that comply with design rules and contribute to routing, timing convergence, and power consumption. The goal is to minimize wirelength, timing, and congestion. iPL can be executed using Python and TCL scripts. To integrate seamlessly with iPL, we have developed a scalable interface for our framework. This ensures that iPO is embedded within iPL without introducing any extraneous modules.

3.4 Transfer Learning

3.4.1 Transfer Learning Path. In the field of machine learning, transfer learning involves transferring labeled data, knowledge structures, or model parameters from one completed task to another similar but distinct task [32]. In placement, one might directly apply previously optimized parameters to a different circuit, but this approach does not guarantee effectiveness. To address this, our framework incorporates transfer learning to facilitate efficient and adaptive parameter transfer.

In our framework, we firstly use cosine similarity to measure similarity two designs. By cosine similarity, a transfer learning path S is generated as presented in Algorithm 5. S is a sequence of

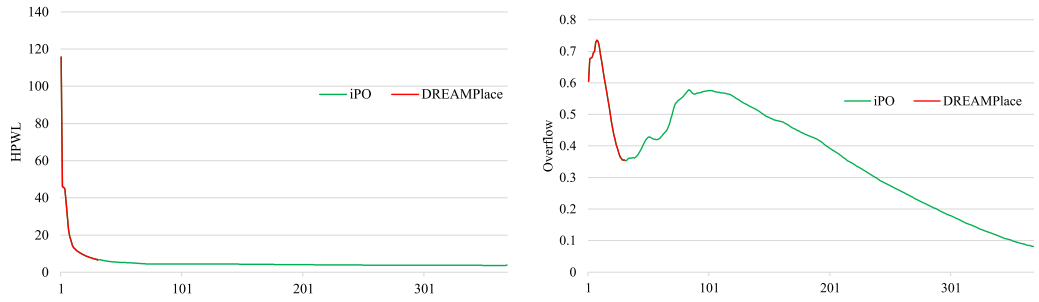


Fig. 8. Different curves are observed with and without self-adaptive convergence adjustment. In (a), HPWL decreases gradually, but convergence finishes too early on DREAMPlace for some groups of parameters. In contrast, (b) shows that convergence finishes normally in iPO when using Eq. (20).

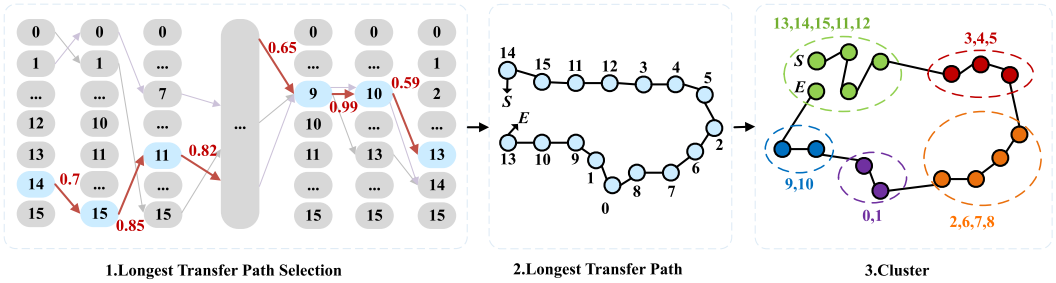


Fig. 9. The transfer learning flow. In the first step, all designs are interconnected, necessitating an analysis of the longest path for our transfer task. In the second step, we identify the longest path from all possible transfer paths. In the third step, designs are grouped into clusters based on the identified paths.

points, where each point represents an embedding corresponding to a specific design. In Fig. 9, the bold red arrow denotes the sequence of S, digits in red denotes cosine similarity.

3.4.2 Cluster Analysis. In Algorithm 5, the cosine similarity mentioned in lines 3 and 8 is specifically expressed by the cosine similarity formula as shown in Eq. (21):

$$\text{sim}(\mathbf{v}_1, \mathbf{v}_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| * \|\mathbf{v}_2\|} \tag{21}$$

where $\mathbf{v}_1$ and $\mathbf{v}_2$ denotes point vector in dimension space. Based on Eq. (21), cosine distance are calculated as:

$$\text{dist}_{\text{cosine}}(\mathbf{v}_1, \mathbf{v}_2) = 1 - \text{sim}(\mathbf{v}_1, \mathbf{v}_2) \tag{22}$$

Eq. (22) is used in function **KMeansByCosine** of Algorithm 6. The silhouette coefficient evaluates clustering quality by measuring how similar an object is to its own cluster compared to other clusters. In the line 7 of Algorithm 6, the silhouette coefficient is computed using cosine distance. Let a point $i \in C_I$, then the average distance of this point to other points in the same cluster C_I is measured by $a(i)$, as shown in Eq. (23):

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} \text{dist}_{\text{cosine}}(\mathbf{v}_i, \mathbf{v}_j) \tag{23}$$

Algorithm 5 Transfer Learning Path Generation Algorithm

Input: different circuit netlist files $\mathbb{N} = \{N_1, N_2, \ldots, N_n\}$
Output: Transfer Learning path **S**

function TranferRouteGenerate($\mathbb{N}$)
  V = CircuitGraphLearning($\mathbb{N}$, 128) ▷ Algorithm 1
  Calculate the cosine similarity between two designs to obtain the similarity matrix $\mathbf{D}_{n\times n}$
  L = {}, $\mathbb{S}$ = {}
  for $i \leftarrow 1, n$ **do**
    $l_i = 0, S_i = \{v_i\}$ ▷ l_i denotes transfer path length, S_i denotes transfer path
    vi = $\{0, 0, \cdots, 0\}$ ▷ $1 \times n$ vector, 0 denotes design is not visited by transfer path
    l_i, S_i = calcCosineLength(v_i, l_i, S_i, **vi**, **D**) ▷ Calculate the path length from the initial point v_i to the adjacent point with the highest cosine similarity
    $\mathbf{L_{i\times 1}} = \mathbf{L} \cup \{l_i\}, \mathbb{S}_{i\times n} = \mathbb{S} \cup \{S_i\}$
  end for
  $l_{\max} = \max\{\mathbf{L}_{n\times 1}\}$
  $\mathbf{S}_{1\times n} = \mathbb{S}_{\max}$
  return S
end function

Input: initial design v, transfer path length l, transfer path S, visit flag vector **vi**, similarity matrix **D**
Output: transfer path length l, transfer path S.

function calcCosineLength(v, l, S, **vi**, **D**)
  if all design are visited **then** ▷ all elements of **vi** is 1
    return l, S
  end if
  $v_{max} = v$
  for $i \leftarrow 1, n$ **do**
    if v_i are not visited **then** ▷ **vi**[i] = 0
      if (v, v_i).length > (v, v_{max}).length **then** ▷ compare current edge with current longest edge
        $v_{max} = v_i$
      end if
    end if
    if the longest edge (v, v_{max}) is found **then** ▷ v_i denotes the i-5h design
      mark v_{max} as visited, $S = S \cup \{v_{max}\}$, $l = l + (v, v_{max})$.length
    end if
  end for
  return calcCosineLength(v_{max}, l, S, **vi**, **D**)
end function

The average distance of this point to points in other clusters C_J is measured by $b(i)$, as shown in Eq. (24):

$$b(i) = \min_{J\neq I} \frac{1}{|C_J|} \sum_{j\in C_J} \mathrm{dist}_{\mathrm{cosine}}(\mathbf{v}_i, \mathbf{v}_j) \tag{24}$$

Then the silhouette coefficient of this point is $s(i)$, as shown in Eq. (25):

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{if } |C_I| > 1 \tag{25}$$

The silhouette coefficient for this classification k is given by Eq. (26):

$$SC = \max_{k\in C} \widetilde{s}(k) \tag{26}$$

where $\widetilde{s}(k)$ represents the average $s(i)$ value of all points when they are clustered into k, and C represents the collection of all clusters resulting from the clustering process. In Fig. 10, we analyze

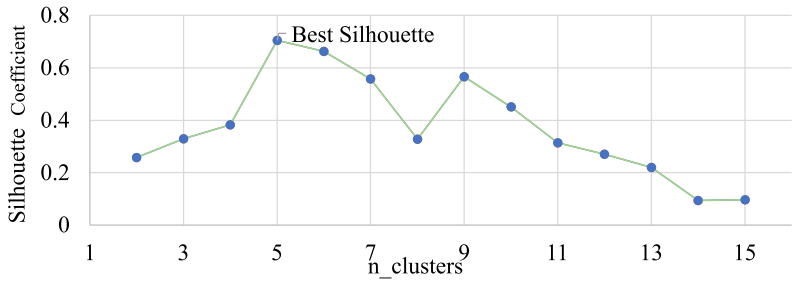


Fig. 10. Silhouette coefficient for K-Means clustering when $n_{cluster}$ (number of clusters) = $\{2, 3, \cdots, 15\}$.

the silhouette coefficient for K-Means clustering on the ISPD2015 benchmarks, which indicates that the silhouette coefficient is maximized when $n_{cluster} = 5$. The mapping from ID to design name is shown in Table 3. As shown in Fig. 9, we cluster designs along the transfer path when $n_{\text{clusters}} = 5$.

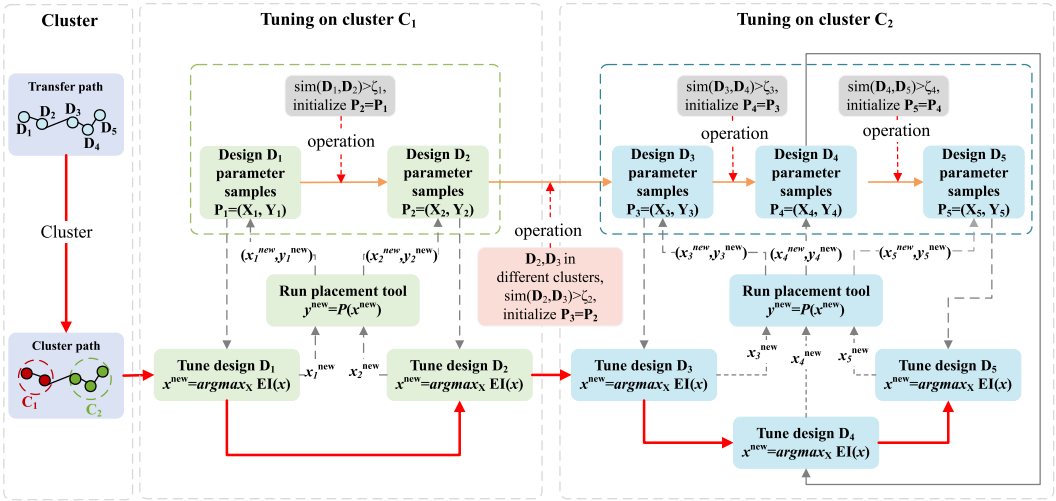


Fig. 11. The sample transfer strategy on our parameter transfer path is illustrated. The bold red arrow represents the overall flow of sample transfer and tuning design. sim($\mathbf{D}_1, \mathbf{D}_2$) is cosine similarity. In some cases, designs within the same cluster may be separated in the transfer path, we share the parameter samples of this cluster to facilitate parameter tuning.

3.4.3 Parameter Transfer Strategy. In Section 3.4.2 above, we cluster designs into multiple clusters. To fully utilize parameter samples from different designs and improve the efficiency of parameter tuning, we implement the sample transfer strategy depicted in Fig. 11. Specifically, the first step, **Cluster**, outlines the flow of Algorithm 6. After clustering, we obtain a cluster path that guides the parameter tuning process effectively. For example, in the **Tuning on Cluster** $\mathbf{C}_1$ part, we tune parameters for design $\mathbf{D}_1$ using the argmax EI function, which predicts the new parameters x_1^{new}. The corresponding value y_1^{new} is obtained from the placement tool by evaluating x_1^{new}, and

Algorithm 6 Transfer Nodes Cluster Algorithm

Input: Transfer path with one node $\mathbf{S}_{n\times 1}$, number of iteration T.
Output: Transfer learning path $\mathbf{T} = \{T_1, T_2, \ldots, T_{k'}\}$ (k' denotes the number clusters), where $T_i (i \in [0, k'])$ denotes the set of points of a certain cluster.

function TransferCluster(S)
  $\mathbb{C} = \{\}$
  for $k \leftarrow 2, K$ **do**
    $\mathbf{C}_k$ = KMeansByCosine(S, k, T) ▷ K-Means using cosine distance
    $\mathbb{C} = \mathbb{C} \cup \{\mathbf{C}_k\}$
  end for
  $\mathbf{T} = FindBestSC(\mathbb{C})$ ▷ choose cluster with best silhouette coefficient
  return T
end function

Input: Transfer path with one node $\mathbf{S}_{n\times 1}$, number of cluster k, number of iteration T.
Output: Clusters C.

function KMeansByCosine(S, k, T)
  $\mathbf{U}_{k\times\delta} \leftarrow RandomSelect(\mathbf{S}, k)$ ▷ randomly select k points from S
  $\mathbf{C}_{k\times\delta} \leftarrow \mathbf{U} = \{\{\mathbf{v}_1\}, \{\mathbf{v}_2\}, \ldots, \{\mathbf{v}_k\}\}$
  for $t \leftarrow 0, T$ **do**
    for $\mathbf{v}_i \in \mathbf{S} - \mathbf{C}$ **do**
      $d_i = 0, k = -1$
      for $\mathbf{u}_j \in \mathbf{U}$ **do**
        if $d_i < \text{sim}(\mathbf{v}_i, \mathbf{u}_j)$ **then**
          $d_i \leftarrow \text{sim}(\mathbf{v}_i, \mathbf{u}_j)$
          $k \leftarrow j$
        end if
      end for
      $\mathbf{C}^{(k)} = \mathbf{C}^{(k)} \cup \{\mathbf{v}_i\}$
    end for
    $\mathbf{U} \leftarrow \{\}$
    for $k \leftarrow 0, |\mathbf{C}|$ **do**
      $\mathbf{U}_{k|_{1\times\delta}} \leftarrow mean({\mathbf{C}^{(k)}}_{c'\times\delta})$ ▷ c' represents the number of elements in the k-th cluster
    end for
  end for
  return C
end function

the sample pair $(x_1^{\text{new}}, y_1^{\text{new}})$ is added to the sample set $\mathbf{P}_1$. Before tuning design $\mathbf{D}_2$, the similarity between $\mathbf{D}_1$ and $\mathbf{D}_2$ is calculated (as shown in the gray module at the top of the **Tuning on Cluster** $\mathbf{C}_1$ section). If the similarity exceeds the threshold ζ (default: 0.6), $\mathbf{P}_2$ is initialized with $\mathbf{P}_1$; otherwise, it is initialized as an empty sample set. After completing the tuning of $\mathbf{D}_2$, the process moves to tuning $\mathbf{D}_3$, which belongs to a different cluster, $\mathbf{C}_2$. This transition requires the operations outlined in the pink module, as illustrated in the center of the figure. Subsequently, tuning continues with $\mathbf{D}_3$, $\mathbf{D}_4$, and $\mathbf{D}_5$. The entire tuning process concludes after successfully completing the tuning of $\mathbf{D}_5$.

Table 3. Detail Characteristics of the ISPD 2015 and iEDA 28nm Benchmarks

the ISPD 2015 Benchmarks					
Design	#macros	#cells	#nets	#fence regions	% area utilization
mgc_des_perf_a	4	108K	115K	4	71.70
mgc_des_perf_b	0	113K	113K	12	49.70
mgc_edit_dist_a	6	127K	134K	1	61.60
mgc_fft_2	0	32K	32K	0	49.90
mgc_fft_a	6	31K	32K	0	74.00
mgc_fft_b	6	31K	32K	0	74.00
mgc_matrix_mult_a	5	150K	154K	0	76.70
mgc_matrix_mult_b	7	146K	152K	3	72.60
mgc_matrix_mult_c	7	146K	152K	3	77.31
mgc_pci_bridge32_a	4	30K	34K	3	40.80
mgc_pci_bridge32_b	6	29K	33K	3	50.60
mgc_superblue11_a	1458	926K	936K	4	73.00
mgc_superblue12	89	1287K	1293K	0	57.00
mgc_superblue14	340	612K	620K	0	77.61
mgc_superblue16_a	419	680K	697K	2	73.90
mgc_superblue19	286	506K	512K	0	80.70

the iEDA 28nm Benchmarks					
Design	#macros	#cells	#nets	#pins	% area utilization
apb4_archinfo	87	389	378	1K	68.08
apb4_clint	161	1K	1K	3K	61.84
apb4_i2c	141	785	722	2K	63.20
apb4_ps2	96	512	494	2K	66.52
apb4_pwm	161	1K	885	3K	55.47
apb4_rng	67	193	202	577	75.56
apb4_timer	141	718	686	2K	62.73
apb4_wdg	161	1K	1K	3K	58.29
s1238	74	348	289	1K	62.75
s13207	143	720	640	2K	58.18
s1488	64	379	324	1K	75.50
s15850	240	2K	2K	7K	55.46
s38417	484	6K	6K	20K	55.71
s713	46	134	124	343	46.57
s9234	109	653	581	2K	65.10
gcd	77	295	268	890	61.84

4 EXPERIMENTS

4.1 Experimental Settings

Our framework is implemented in Python, supporting placement engines such as DREAMPlace and iEDA-iPL as demonstrated in this paper. In our experiments, the framework runs on Machine A (2*Intel Xeon Gold 6338 CPU with 4*A10), B (2*Intel Xeon CPU E5-2698 v4 with 4*V100), and Machine C (160*Intel(R) Xeon(R) Platinum 8380 CPU). Specifically, we conducted our experiments on ISPD2015 benchmarks on Machine A and B with 5 parallel processes respectively, and iEDA

28nm benchmarks on Machine C with a single process. The detailed characteristics of the iEDA 28nm and ISPD2015 benchmarks are summarized in Table 3.

Since there may be significant variations in the hardware environment across physical machines, we have explored a fair method for measuring the performance of methods on different machines. The fair indicator is the number of iterations, which isn't affected by hardware changes. As shown in Fig. 12, the iterations consist of both GP, LG, and DP. Specifically, LG and DP occur during the final iteration, while GP is performed in the preceding iterations.

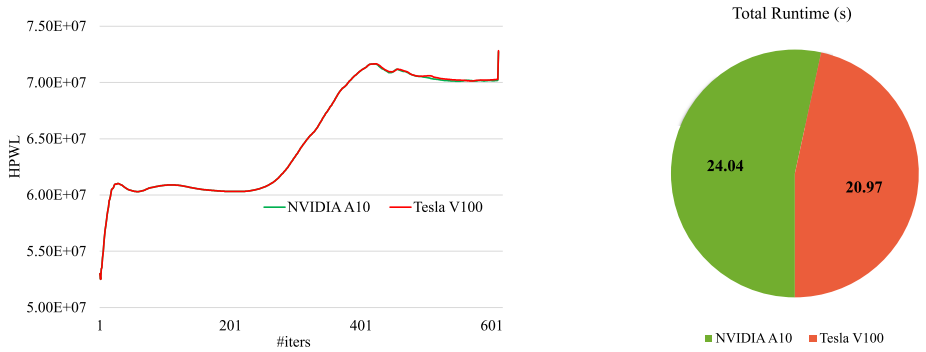


Fig. 12. The number of iterations and total time of placement on different machines with the same parameters. (a) The number of iterations is the same on the A10 and V100 machines, and the iteration curves are also identical. (b) However, the total runtime on the A10 machine is longer than that on the V100 machine

4.2 Automated Placement Evaluation

4.2.1 Parameter Tuning Results Analysis on ISPD2015 Benchmarks. In general, we need to repeatedly configure parameters and run the placement engine for every new design manually, which is a time-consuming and inevitable step. In this section, we conduct our experiment on the ISPD2015 benchmarks. As shown in Table 4, we apply our proposed method to tune the parameters for these benchmarks without human intervention. The transfer learning path for parameter tuning is illustrated in Fig. 9. This path effectively guides the transfer of previously tuned historical parameter sets to new designs.

Specifically, in the first step, iPO tunes the initial design *mgc_superblue16_a* in the transfer path for 2000 iterations. The best metric for this design is identified at the 1532nd iteration. Next, for a new design *mgc_superblue19*, which belongs to the same cluster as *mgc_superblue16_a*, the *cosine similarity* between these two designs is 0.7. As shown in Fig. 11, since this *similarity* exceeds the threshold of 0.6, the historical parameter samples from *mgc_superblue16_a* are directly transferred to initialize tuning for *mgc_superblue19*. iPO then tunes *mgc_superblue19* for only 100 iterations, where the best metric is identified at the 32nd iteration. Following the processes illustrated in Fig. 11, all designs in the ISPD2015 benchmarks are efficiently tuned by our framework, showcasing the effectiveness of transfer learning combined with parameter optimization.

Table 4, we observe that the parameter tuning search iterations for the first design reach 1532. However, for all subsequent designs, the search iterations are significantly reduced, remaining below 30 in most cases except for 84 on *mgc_superblue11_a*. This highlights the effectiveness of our parameter transfer strategy. By performing extensive parameter tuning on the first design, our

Table 4. Automated Placement Evaluation on the ISPD 2015 benchmarks.

Design	DREAMPlace					AutoDMP						iPO					
	HPWL (e6)	Cong. (e-4)	GP (s)	TR (s)	#iters	HPWL (e6)	Cong. (e-4)	GP (s)	TR (s)	#iters	#s	HPWL (e6)	Cong. (e-4)	GP (s)	TR (s)	#iters	#s
mgc_des_perf_a	11.13	4.11	20.5	30.1	947	10.37	3.78	32.6	40.9	874	1261	**9.24**	**3.43**	**13.3**	**22.7**	**747**	**15**
mgc_des_perf_b	9.10	7.60	49.9	60.2	1001	8.13	6.66	44.4	49.7	949	157	**7.39**	**6.01**	**36.6**	**50.5**	**849**	**8**
mgc_edit_dist_a	21.30	10.03	12.6	23.2	1001	19.88	9.28	12.2	17.4	1014	828	**18.23**	**8.68**	**6.4**	**16.4**	**628**	**6**
mgc_fft_2	1.93	5.96	5.7	11.7	587	2.18	6.72	4.1	6.8	582	21	**1.66**	**5.15**	**1.8**	**7.8**	**457**	**20**
mgc_fft_a	3.14	1.88	3.4	9.3	593	3.01	1.79	3.6	6.5	543	43	**2.79**	**1.69**	**1.7**	**7.9**	**404**	**9**
mgc_fft_b	4.23	2.44	3.3	9.1	580	4.28	2.45	6.0	8.6	934	18	**3.93**	**2.31**	**1.6**	**7.5**	**336**	**3**
mgc_matrix_mult_a	15.18	2.25	**3.9**	16.5	632	16.00	2.38	7.9	13.2	1184	1	**14.40**	**2.15**	4.3	**14.1**	**606**	5
mgc_matrix_mult_b	15.78	2.31	18.8	30.5	1001	16.05	2.39	27.9	33.8	914	438	**15.52**	**2.23**	**9.2**	**20.3**	**653**	27
mgc_matrix_mult_c	15.27	2.21	18.7	30.7	1001	15.35	2.25	13.8	19.2	719	734	**14.68**	**2.13**	**10.5**	**21.4**	**673**	**2**
mgc_pci_bridge32_a	1.98	4.48	23.0	25.0	1001	1.91	4.35	35.8	38.5	1072	305	**1.64**	**3.70**	**11.2**	**18.0**	**678**	**15**
mgc_pci_bridge32_b	3.20	1.88	19.5	26.4	1001	2.70	1.59	15.2	17.7	859	708	**2.76**	**1.60**	**6.8**	**13.1**	**484**	**5**
mgc_superblue11_a	350.80	16.48	**29.7**	111.2	963	370.09	17.72	28.5	85.8	906	161	**330.80**	**15.63**	44.8	**96.7**	**805**	**84**
mgc_superblue12	257.34	33.39	12.9	83.5	1001	267.37	33.85	26.5	71.2	1960	38	**232.32**	**29.22**	**8.4**	**61.6**	**607**	**18**
mgc_superblue14	229.99	30.43	5.5	46.4	627	217.72	29.74	20.3	43.7	1804	482	**205.06**	**26.87**	**4.0**	**32.7**	**486**	7
mgc_superblue16_a	268.42	36.24	18.2	64.7	955	260.46	34.41	24.1	51.8	891	**1001**	**239.67**	**31.60**	**15.2**	**49.3**	**816**	1532
mgc_superblue19	156.12	23.46	5.1	37.9	603	151.65	23.71	11.3	29.2	978	53	**131.79**	**20.59**	**4.3**	**28.0**	**455**	**35**
avg. impr. (%)	-	-	-	-	-	-0.2	1.1	-25.4	13.4	-20.0	-	9.8	12.0	28.2	24.1	28.2	326.1

Table 5. Experimental results on the iEDA 28nm benchmarks are presented using iEDA.

Design	iEDA-iPL				iPO				
	HPWL (e6)	WNS (ns)	TNS (ns)	TR (s)	HPWL (e6)	WNS (ns)	TNS (ns)	TR (s)	#s
apb4_archinfo	3.72	**0.340**	0.000	182	**3.60**	0.343	0.000	**154**	**74**
apb4_clint	11.34	−0.401	−6.473	294	**10.84**	**−0.395**	**−6.188**	340	**89**
apb4_i2c	7.27	−0.235	−7.393	240	**6.73**	**−0.224**	**−6.986**	**169**	**94**
apb4_ps2	4.74	−0.028	−0.108	199	**4.47**	**−0.018**	**−0.027**	170	**32**
apb4_pwm	11.03	−0.314	−10.516	**265**	**9.98**	**−0.304**	**−10.083**	289	**85**
apb4_rng	2.23	0.187	0.000	171	**2.13**	**0.192**	0.000	193	**3**
apb4_timer	7.55	−0.370	−14.244	**248**	**7.22**	**−0.362**	**−14.066**	256	**84**
apb4_wdg	10.66	−0.456	−16.112	423	**10.15**	**−0.456**	**−15.495**	**356**	**4**
s1238	3.07	0.004	0.000	123	**2.87**	**0.006**	0.000	**97**	**13**
s13207	6.06	−0.064	−0.303	173	**5.89**	**−0.051**	**−0.225**	172	**10**
s1488	4.80	−0.012	−0.013	217	**4.28**	**−0.004**	**−0.007**	205	**282**
s15850	23.22	**−0.333**	−30.340	697	**22.35**	−0.342	**−29.750**	**695**	**76**
s38417	71.88	−0.384	−149.884	8095	**68.95**	**−0.380**	**−147.754**	8049	**16**
s713	1.16	−0.033	−0.065	170	**1.07**	**−0.028**	**−0.061**	**163**	**12**
s9234	6.51	−0.234	−12.127	227	**6.10**	−0.241	**−11.642**	**214**	2
gcd	13.52	−0.556	−20.105	217	**13.33**	**−0.548**	**−17.947**	**216**	292
avg. impr. (%)	-	-	-	-	4.7	2.7	2.8	1.7	-

method effectively transfers prior experience to other designs, drastically reducing the computational effort required. Additionally, many occurrences of the number 1001 are marked in red, which indicates that the placement does not converge. In our method, none are marked in red, while in DREAMPlace, there are 7 occurrences, which shows that our method can improve convergence.

The results demonstrate iPO has an average improvement of 9.8% in HPWL, as well as average improvements of 12.0%, 28.2%, 24.1%, and 28.2% in congestion, GP, total runtime (TR), and the number of iterations (#iters), respectively, when compared to DREAMPlace. These results clearly indicate that our proposed automated tuning method not only improves the quality of the placement (in terms of HPWL and congestion) but also significantly reduces the number of iterations required for parameter tuning on subsequent designs. The reduction in iterations is particularly noteworthy as it highlights the efficiency of our parameter transfer strategy, which effectively leverages prior tuning experiences from one design to expedite the tuning process for subsequent designs. In addition to comparing iPO with DREAMPlace (baseline), we also evaluated it against AutoDMP. Compared to DREAMPlace, AutoDMP achieves average improvements of 1.1% and 13.4% in congestion and TR, respectively, without any degradation in HPWL. Compared to AutoDMP, iPO shows an average improvement of 11% in HPWL, along with improvements of 12.3%, 74.2%, 14.0%, and 67.1% in congestion, GP, TR, and the number of iterations (#iters), respectively. Furthermore, iPO achieves 3.49× speed-up in #s, demonstrating the effectiveness of our method in efficiently searching for optimal parameters.

4.2.2 iEDA 28nm Benchmarks. In this section, we conduct our experiments on the iEDA 28nm Benchmarks. We utilize our method to optimize HPWL, WNS, and TNS. The results, as summarized in Table 5, demonstrate significant improvements across all metrics when compared to iEDA-iPL.

The results show an average improvement of 4.7% in HPWL and 2.8% in WNS. These enhancements illustrate the effectiveness of our automated tuning method in optimizing critical parameters for VLSI placement. In addition to HPWL and WNS, we also focus on optimizing TNS, a crucial timing metric in VLSI design. Table 5 presents the comparative results for TNS optimization. From

Table 5, it is evident that our method consistently improves TNS across all designs, with an average improvement of 2.8%. This substantial improvement underscores the efficacy of our automated tuning approach in addressing critical timing metrics in VLSI placement.

The experimental results on the iEDA 28nm Benchmarks validate the effectiveness of our automated tuning method. The significant improvements in HPWL, WNS, and TNS highlight the potential of our approach in optimizing VLSI placement parameters. By leveraging automated tuning, our method reduces the dependency on manual intervention, thereby enhancing the overall efficiency and performance of the placement process.

4.3 Ablation Study

4.3.1 Ablation Study on the ISPD2015 Benchmarks Using Different Transfer Strategies. In this section, we present the results of ablation studies aimed at evaluating the effectiveness of various transfer strategies on the ISPD2015 benchmarks. The studies investigate how different approaches to transferring knowledge and parameters between designs impact performance metrics such as HPWL, congestion, and #s. The design ID denotes the id in Table 3.

Table 6. Experimental Results of Ablation Study on the ISPD2015 Benchmarks Using Different Strategies

Design ID	DREAMplace					iPO (Automated Tuning for Each Design)					
	HPWL (e6)	Cong. (e-4)	GP (s)	TR (s)	#iters	HPWL (e6)	Cong. (e-4)	GP (s)	TR (s)	#iters	#s
0	11.13	4.11	20.49	30.09	947	**8.43**	**3.12**	17.16	**22.47**	849	668
2	21.30	10.03	12.60	23.25	1001	**17.29**	**8.12**	9.83	20.06	878	219
3	1.93	5.96	5.70	11.69	587	**1.62**	**5.00**	3.53	**5.72**	**417**	2244
9	1.98	4.48	22.97	24.95	1001	**1.61**	**3.59**	14.60	**16.92**	703	14 059
14	268.42	36.24	18.16	64.66	955	**239.67**	**31.60**	**15.19**	**49.34**	**816**	**1532**
avg. impr. (%)	-	-	-	-	-	13.5	18.3	32.5	35.1	22.6	-
	iPO (Direct Parameter Transfer)					iPO (Cluster-Based Transfer)					
0	**8.43**	**3.12**	17.16	22.47	849	9.24	3.43	**13.34**	22.67	**747**	**15**
2	17.37	8.26	12.60	23.25	1001	18.23	8.68	**6.38**	**16.36**	**628**	**6**
3	1.62	5.04	5.70	11.69	495	1.66	5.15	**1.76**	7.81	457	**20**
9	1.74	3.94	22.97	24.95	988	1.64	3.70	**11.15**	18.05	**678**	**15**
14	241.30	31.86	**12.98**	60.00	816	**239.67**	**31.60**	**15.19**	**49.34**	**816**	**1532**
avg. impr. (%)	12.7	16.5	11.9	8.6	8.2	12.7	15.7	67.1	35.4	35.0	1079.0

We conduct our experiments on the ISPD2015 benchmarks, employing several strategies to optimize the process of parameter tuning. The strategies explored include:

(1) **Direct Parameter Transfer**: Utilizing parameters from firstly optimized design without modifications.
(2) **Automated Tuning for Each Design**: The automated tuning strategy provides the best overall performance. By dynamically adjusting tuning parameters based on the specific characteristics of each design, it is expected to achieve the best performance in most cases.
(3) **Cluster-Based Transfer**: Firstly, filtering parameters based on cosine similarity thresholds between designs. Then, transferring parameters between designs within the same cluster, using the transfer strategy shown in Fig. 11.

Table 6 summarizes the results of our ablation studies. The table highlights the performance metrics achieved using different transfer strategies. From the results in Table 6, it is evident that different strategies yield varying performance outcomes.

(1) **Direct Parameter Transfer**: This method achieves reasonable performance but shows limited improvements in congestion and HPWL compared to other strategies.
(2) **Automated Tuning for Each Design**: This demonstrates improved performance in all metric except for #s . Specifically, we observe a notable 3.8% reduction in congestion compared to the strategy **Cluster-Based Transfer**, suggesting that this strategy effectively filters parameters to better match the design characteristics of the target benchmarks.
(3) **Cluster-Based Transfer**: This strategy yields more balanced improvements across all metrics. In metric #s, this achieves an 11.8× speed-up compared to the strategy Automated Tuning for Each Design without HPWL downgrade. In addition, we observe a notable 12.7%, 15.7%, and 35.4% reduction in HPWL, congestion, and TR compared to DREAMPlace. The cluster-based approach appears to leverage the similarity between designs more effectively, leading to relatively better parameter adaptation and optimization.

The ablation studies highlight the impact of different strategies on the process of parameter tuning. The **automated tuning for each design** and **cluster-based transfer** strategies outperform the **direct parameter transfer** in all metrics, demonstrating their effectiveness in enhancing parameter tuning efficiency.

Table 7. Ablation Study Results on ISPD2015 Benchmarks with Different Parallel Processes

Design	Parallel Process (es)								
	1			5			10		
	HPWL (e6)	Cong. (e-4)	ST (s)	HPWL (e6)	Cong. (e-4)	ST (s)	HPWL (e6)	Cong. (e-4)	ST (s)
mgc_des_perf_a	8.86	3.20	14 169.05	**8.63**	**3.15**	2392.80	8.72	3.21	**1284.11**
mgc_edit_dist_a	18.95	9.07	9948.93	**17.68**	**8.17**	1429.79	18.20	8.74	**762.48**
mgc_fft_2	1.67	5.12	1674.59	**1.62**	**5.04**	857.99	1.62	**5.04**	**428.57**
mgc_pci_bridge32_a	1.66	3.70	9942.65	1.67	3.81	1346.57	**1.66**	**3.76**	**965.97**
mgc_superblue16_a	243.43	32.12	19 803.23	**243.31**	**32.03**	4673.93	246.96	32.49	**2673.93**
avg. impr. (%)	-	-	-	0.6	1.9	419.0	−0.9	0.0	808.2

4.3.2 Ablation Studies on the ISPD2015 Benchmarks Using Different Parallel Processes. In this section, we present the experimental results of our ablation studies conducted on the ISPD2015 benchmarks after 100 parameter tuning iterations. The study investigates the impact of varying the number of parallel processes on the performance of our proposed method.

From the results summarized in Table 7, we observe that increasing the number of parallel processes leads to significant improvements in search time (ST) reduction. Specifically, with 10 parallel processes, we achieve 9.1× speed-up in ST (the columns are summed then compared), compared to a single process. Additionally, we achieve a 1.75× speed-up in ST with only slight performance downgrades of 1.5% in HPWL and 1.9% in congestion compared to 5 parallel processes. These findings highlight the importance of leveraging higher parallel processes to optimize the parameter tuning for the placement process in VLSI design. Interestingly, we found that concurrency 5 achieves slightly better Quality of Results (QoR), which may be partially influenced by the random initialization of parameter samples. In conclusion, the ablation studies demonstrate that our method benefits significantly from increased concurrency, leading to better optimization results on the ISPD2015 benchmarks.

4.3.3 Sensitivity Analysis on Different Vocabulary Size. We conducted an experiment to compare objective metrics using four different vocabulary sets: "V-10", "V-D", "V-S", and "V-L".

- **"V-D"**: Default vocabulary (size 12M) with rooted subgraph maximum degree $D = 2$.
- **"V-10"**: 90% random sample removal (1.2M retained) from the default vocabulary.
- **"V-S"**: Vocabulary reduction (size 8M) with $D = 1$.
- **"V-L"**: Vocabulary expansion (size 20M) with $D = 6$.

As shown in Fig. 13, we evaluate average #s, congestion, and HPWL across *mgc_superblue11_a*, *mgc_superblue12*, *mgc_superblue14*, *mgc_superblue16_a*, and *mgc_superblue19*. Compared to "V-D", "V-10" and "V-S" exhibit a 50.24% and 24.3% increase in #s. "V-L" achieves performance parity with "V-D". Notably, HPWL and congestion metrics remain stable across all four vocabulary sizes. These results confirm that vocabulary expansion enhances parameters optimization efficiency below 12M entries, with diminishing returns observed beyond this critical threshold.

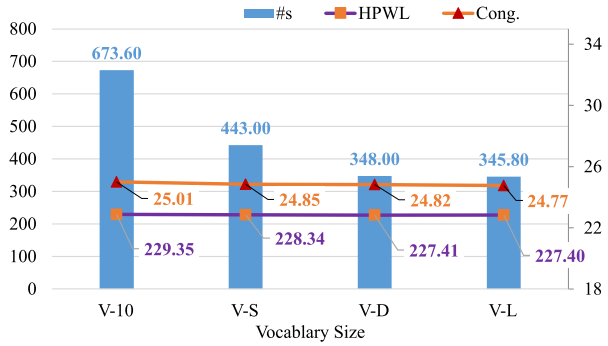


Fig. 13. Average metrics (#s, HPWL and congestion) comparison across different vocabulary sizes.

4.3.4 Convergence Analysis. We conduct an experiment to compare convergence efficiency using six different initialization methods: "0", "R-5", "R-10", "R-20", "R-50", and "S-10".

- **"0" (Empty Initialization)**: The initial parameter sample set of iPO is empty.
- **"R-5"**: The parameter sample set is initialized with 5 randomly selected samples.
- **"R-10"**: The parameter sample set is initialized with 10 random samples.
- **"R-20"**: The parameter sample set is initialized with 20 random samples.
- **"R-50"**: The parameter sample set is initialized with 50 random samples.
- **"S-10"**: The parameter sample set is initialized with one random sample, and then 9 additional samples are generated by copying the first sample.

As shown in Fig. 14, this experiment aims to analyze the impact of different initialization strategies on the convergence speed and optimization performance of iPO. We compare these methods based on key metrics such as iteration count, HPWL improvement, and runtime efficiency. The experimental results for "0" and "R-10" have similar results for HPWL and congestion, because method "0" will fail iPO in the first, then will take default strategy ("R-10"). Compared to "R-10", both "R-50" and "R-20" exhibit performance degradation in HPWL and congestion. This indicates that increased randomness undermines the effectiveness of the EI sampling criterion, leading to a partial loss of directed search capability in the optimization process.

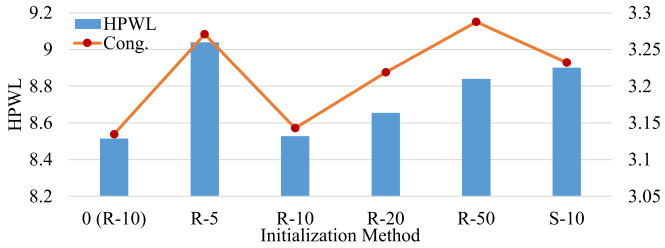


Fig. 14. Metrics (HPWL, congestion) comparison to different sample initialization on design *mgc_des_perf_a*.

5 CONCLUSION

In this paper, we propose an intelligent parameter optimization method, iPO, which integrates transfer learning and parallel automated parameter tuning together. Experimental results on the ISPD2015 benchmarks show substantial improvements in HPWL and congestion compared to DREAMPlace and AutoDMP, while results on the iEDA 28nm benchmarks demonstrate notable improvements in HPWL, TNS, and WNS. These findings highlight the effectiveness of cross-design parameter transfer across various metrics and benchmarks. Additionally, for further exploration, our method achieves significant speedups in #s compared to AutoDMP, which also shows the effectiveness of introducing the Constant Liar strategy.

Future work will focus on refining these strategies further and exploring their application to a broader range of benchmarks and design scenarios. Our proposed method may lose effectiveness for extremely large designs due to the inherently time-consuming placement process, significantly prolonging each DSE iteration. For example, 100 DSE iterations would take 200 hours (2 hours/iteration), drastically reducing optimization efficiency. To address this, we will also explore more efficient techniques specifically tailored for ultra-large designs in the future.

REFERENCES

[1] Anthony Agnesina, Kyungwook Chang, and Sung Kyu Lim. 2020. VLSI placement parameter optimization using deep reinforcement learning. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[2] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. 2023. AutoDMP: Automated DREAMPlace-based macro placement. In *Proceedings of the ACM International Symposium on Physical Design*. 149–157.

[3] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Proceedings of the Advances in Neural Information Processing Systems*. 2546–2554.

[4] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. 2008. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 7 (July 2008), 1228–1240.

[5] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. 2019. RePlAce: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 9 (Sept. 2019), 1717–1730.

[6] Ruoyu Cheng and Junchi Yan. 2021. On joint learning for solving placement and routing in chip design. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 16508–16519.

[7] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. 2010. Kriging is well-suited to parallelize optimization. *Computational Intelligence in Expensive Optimization Problems* 2 (2010), 131–162.

[8] Albert William Tucker Harold William Kuhn. 1950. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. 481–492.

[9] Meng-Kai Hsu, Valeriy Balabanov, and Yao-Wen Chang. 2013. TSV-Aware analytical placement for 3-D IC designs based on a novel weighted-average wirelength model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2013).

[10] Zhipeng Huang, Zengrong Huang, Simin Tao, Shijian Chen, Zhisheng Zeng, Liwei Ni, Chunan Zhuang, Weiguo Li, Xueyan Zhao, He Liu, et al. 2024. AiEDA: an open-source AI-native EDA library. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 794–795.

[11] Zixuan Jiang, Ebrahim Songhori, Shen Wang, Anna Goldie, Azalia Mirhoseini, Joe Jiang, Young-Joon Lee, and David Z. Pan. 2021. Delving into macro placement with reinforcement learning. In *Proceedings of the ACM/IEEE 3rd Workshop on Machine Learning for CAD*. 1–3.

[12] Yao Lai, Yao Mu, and Ping Luo. 2022. MaskPlace: Fast chip placement via reinforced visual representation learning. *arXiv preprint* arXiv:2211.13382 (2022).

[13] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1188–1196.

[14] Bing Li, Wendi Sun, Xiaobing Ni, Kaixuan He, Qi Xu, Song Chen, and Yi Kang. [n. d.]. Parallel multi-objective bayesian optimization framework for CGRA microarchitecture. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition* (2024-03). 1–6.

[15] Xingquan Li, Zengrong Huang, Simin Tao, Zhipeng Huang, Chunan Zhuang, Hao Wang, Yifan Li, Yihang Qiu, Guojie Luo, Huawei Li, Haihua Shen, Mingyu Chen, Dongbo Bu, Wenxing Zhu, Ye Cai, Xiaoming Xiong, Ying Jiang, Yi Heng, Peng Zhang, Bei Yu, Biwei Xie, and Yungang Bao. 2024. iEDA: An Open-source infrastructure of EDA. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 77–82.

[16] Xingquan Li, Simin Tao, Shijian Chen, Zhisheng Zeng, Zhipeng Huang, Hongxi Wu, Weiguo Li, Zengrong Huang, Liwei Ni, Xueyan Zhao, He Liu, Shuaiying Long, Ruizhi Liu, Xiaoze Lin, Bo Yang, Fuxing Huang, Zonglin Yang, Yihang Qiu, Zheqing Shao, Jikang Liu, Yuyao Liang, Biwei Xie, Yungang Bao, and Bei Yu. 2024. iPD: An open-source intelligent physical design toolchain. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 83–88.

[17] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2019. DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In *Proceedings of the ACM/IEEE Design Automation Conference*. 1–6.

[18] Yibo Lin, David Z. Pan, Haoxing Ren, and Brucek Khailany. 2020. DREAMPlace 2.0: Open-source GPU-accelerated global and detailed placement for large-scale VLSI designs. In *Proceedings of the China Semiconductor Technology International Conference*. 1–4.

[19] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis J-.H. Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2014. ePlace: Electrostatics based placement using Nesterov's method. In *Proceedings of the ACM/IEEE Design Automation Conference*. 1–6.

[20] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method. *ACM Transactions on Design Automation of Electronic Systems* 20, 2 (March 2015), 1–34.

[21] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace-MS: Electrostatics-based placement for mixed-size circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 5 (May 2015), 685–698.

[22] Yi-Chen Lu, Sai Pentapati, and Sung Kyu Lim. 2020. VLSI placement optimization using graph neural networks. In *Proceedings of the Advances in Neural Information Processing Systems*. 6–12.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint* arXiv:1301.3781 (2013).

[24] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. Graph2vec: Learning distributed representations of graphs. *arXiv preprint* arXiv:1707.05005 (2017).

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

[26] Haoxing Ren, Siddhartha Nath, Yanqing Zhang, Hao Chen, and Mingjie Liu. 2022. Why are graph neural networks effective for EDA problems?: (Invited paper). In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, San Diego California, 1–8.

[27] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. 2020. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4938–4947.

[28] Matthias Schonlau. 1997. *Computer experiments and global optimization*. Ph. D. Dissertation. University of Waterloo.

[29] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng. 2009. *Electronic Design Automation: Synthesis, Verification, and Test*.

[30] Zhiyao Xie, Guan-Qi Fang, Yu-Hung Huang, Haoxing Ren, Yanqing Zhang, Brucek Khailany, Shao-Yun Fang, Jiang Hu, Yiran Chen, and Erick Carvajal Barboza. 2020. FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning. In *Proceedings of the Asia and South Pacific Design Automation Conference*.

[31] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. 2023. Learning on large-scale text-attributed graphs via variational inference. arXiv:2210.14709 (2023).
[32] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *arXiv preprint* arXiv:1911.02685 (2020).

Received 20 November 2024; revised 12 March 2025; accepted 1 May 2025