

# BBOPlace-Bench: Benchmarking Black-Box Optimization for Chip Placement

Ke Xue\*, Ruo-Tong Chen\*, Rong-Xi Tan\*, Xi Lin, Yunqi Shi,  
Siyuan Xu, Mingxuan Yuan, Chao Qian, *Senior Member, IEEE*

**Abstract**—Chip placement is a vital stage in modern chip design as it has a substantial impact on the subsequent processes and the overall quality of the final chip. The use of black-box optimization (BBO) for chip placement has a history of several decades. However, early efforts were limited by immature problem formulations and inefficient algorithm designs, leading to suboptimal efficiency, quality, and scalability, compared to the more prevalent analytical methods. Recent progress in problem formulation and algorithm design has shown the effectiveness and efficiency of BBO for chip placement, proving its potential to achieve state-of-the-art results. Despite these advancements, the field lacks a unified, BBO-specific benchmark for thoroughly assessing various problem formulations and BBO algorithms. To fill this gap, we propose BBOPlace-Bench, the first benchmark designed specifically for evaluating and developing BBO algorithms for chip placement tasks. It integrates three problem formulations (with permutation, continuous, and mixed search spaces, respectively) of BBO for chip placement, and offers a modular, decoupled, and flexible framework that enables users to seamlessly implement, test, and compare their own algorithms. BBOPlace-Bench aggregates modern chip cases from representative chip cases (ISPD 2005, ICCAD 2015) and standardizes their formats, providing uniform and comprehensive information to support BBO optimization. Moreover, it integrates a wide variety of existing BBO algorithms, including simulated annealing (SA), evolutionary algorithms (EAs), and Bayesian optimization (BO), and systematically evaluates their performance across different problem formulations using key metrics (e.g., macro placement wirelength and global placement wirelength) of chip. Experimental results show that the problem formulations of mask-guided optimization and hyperparameter optimization exhibit superior performance than the sequence pair problem formulation, while EAs demonstrate better overall performance than SA and BO, especially in high-dimensional search spaces, and also achieve state-of-the-art performance compared to the mainstream chip placement methods, i.e., analytical methods and reinforcement learning methods. BBOPlace-Bench not only facilitates the development of efficient BBO-driven solutions for chip placement but also broadens the practical application scenarios (which are urgently needed) for the BBO community. The code of BBOPlace-Bench is available at <https://github.com/lamda-bbo/BBOPlace-Bench>.

## I. INTRODUCTION

Chip placement is a critical yet time-consuming step in the very large-scale integrated (VLSI) design flow, serving as a crucial process that significantly impacts the power,

The first three authors contributed equally. Chao Qian is the corresponding author (email: qianc@nju.edu.cn).

Ke Xue, Ruo-Tong Chen, Rong-Xi Tan, Xi Lin, Yunqi Shi and Chao Qian are with the National Key Laboratory for Novel Software Technology, and also with School of Artificial Intelligence, Nanjing University.

Siyuan Xu and Mingxuan Yuan are with Huawei Noah's Ark Lab.

performance, and area (PPA) metrics of the final chip [54], [55], [26]. A modern chip typically consists of numerous modules, comprising thousands of macros (i.e., individual building blocks such as memory components) and millions of standard cells (i.e., smaller fundamental elements like logic gates). Global placement (GP) aims to place all modules, which typically begins with the placement of macros and subsequently proceeds to the placement of standard cells. During this process, macro placement (MP) establishes a foundational layout for subsequent processes, such as standard cells placement and routing (connecting modules using wires), thereby playing a vital role in chip design [74]. For example, suboptimal MP results can complicate the optimal positioning of the standard cells, which may ultimately lead to unsatisfactory chip performance [78]. Furthermore, inappropriate MP can result in macro blockage within the core area, which may induce issues such as routing congestion, increased wirelength, and timing performance degradation, and thus adversely affects the overall chip performance [62], [82].

Chip placement is inherently a black-box optimization (BBO) problem, because its objective function (measuring the PPA metrics of the final chip) has no analytical form and can only be evaluated by simulator or even real manufacturing. Furthermore, the objective evaluation comes with high computational costs, e.g., several hours for exact simulation. To solve this expensive BBO task of chip placement, chip designers often rely on proxy metrics that reflect the final chip performance to guide the optimization process [9], thus significantly reducing the real evaluation cost. One important proxy metric is half-perimeter wirelength (HPWL), which provides an approximation of the routing wirelength and tightly correlates with routability, timing, and power, making it widely used to measure placement quality [66], [37], [50]. As a result, chip placement is often formulated as the problem of minimizing HPWL subject to the constraint of zero overlap among circuit components.

There are three kinds of mainstream methods for chip placement: Analytical method, reinforcement learning (RL)-based method, and BBO method. Analytical placers approximate the non-differentiable HPWL using a smooth optimization objective (e.g., weighted-average wirelength [50], [15]) and then optimize the approximate objective by gradient descent. Analytical placers can be accelerated by advanced AI hardware and software [46], [49], [44], enabling the efficient handling of the placement of millions of standard cells. Since the publication of AlphaChip in *Nature* [57], [27], RL has attracted growing attention as a promising approach for automated

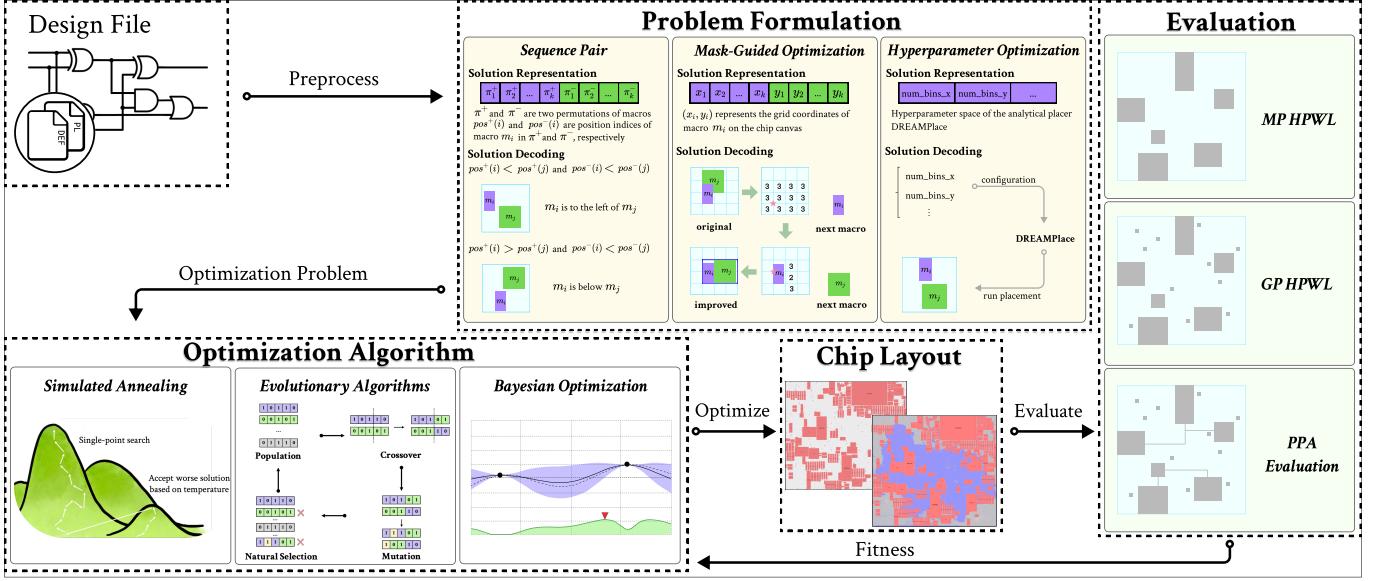


Figure 1: Illustration of BBOPlace-Bench. It decouples three core components, i.e., problem formulation, optimization algorithm, and evaluation, making it easier to test different BBO algorithms under customized settings. Three problem formulations are: 1) Sequence Pair: Using two permutations to capture positional relationships of macros. 2) Mask-Guided Optimization: Representing macros by grid coordinates, and using wire-mask-guided decoding to adjust the positions of macros to the grids with minimal incremental HPWL. 3) Hyperparameter Optimization: Optimizing the advanced analytical placer DREAMPlace’s hyperparameters (e.g., learning rate and target density) over mixed (discrete and continuous) search spaces, where the running of each configured DREAMPlace corresponds to the procedure of solution decoding, which generates a placement result of macros. Evaluation metrics include: MP-HPWL (fast proxy metric for wirelength of macros), GP-HPWL (accurate but costly metric for wirelength of macros and standard cells), and PPA (industrial chip metrics obtained via commercial tools like *Cadence Innovus*, a premier chip placement tool from the leading EDA provider *Cadence Design Systems, Inc.*).

chip placement [17], [42], [41], [82], [25], owing to its high optimization efficiency and generalization potential. In the RL framework, chip placement is formulated as a Markov decision process, where the agent places one macro at each step and the environment transitions to the next state accordingly.

The application of BBO for chip placement has a history of several decades [58], [13], [34]. Nevertheless, early attempts were hampered by immature problem formulation and inefficient algorithm design, resulting in poor placement performance, e.g., the *Nature* paper [57] claimed that “*it is very slow and difficult to parallelize, thereby failing to scale to the increasingly large and complex circuits of the 1990s and beyond*”. As the scale of chips continues increasing, these issues become progressively more severe, preventing early BBO methods from competing with advanced analytical placers [50], [15], [46], [49]. Fortunately, recent advancements in BBO problem formulation (such as WireMask-Guided BBO [68]) and efficient BBO algorithm design have highlighted the potential of BBO to achieve state-of-the-art results for chip placement, e.g., reducing wirelength by 30% and 23% compared to state-of-the-art analytical placer DREAMPlace [46] and RL placer MaskPlace [42], respectively. Despite these advancements, the field lacks a unified benchmark for thoroughly assessing various BBO problem formulations and BBO algorithms, which hinders systematic comparisons and progress, underscoring a critical need for a standardized platform.

To fill this gap, we propose BBOPlace-Bench, the first benchmark designed specifically for BBO users to tackle chip placement tasks. As illustrated in Figure 1, BBOPlace-Bench establishes a modular, decoupled framework that separates three core components—problem formulation, optimization, and evaluation—enabling users to easily test diverse BBO algorithms under customized settings. The framework supports three distinct problem formulations, each with a tailored process that maps from a genotype solution to modules’ positions: 1) Sequence pair (SP) utilizes pairs of permutation sequences to represent the relative relationships of modules, and employs the longest common subsequence algorithm for solution decoding; 2) Mask-guided optimization (MGO) uses grid coordinates for modules representation, and employs a mask-guided mechanism for solution decoding; 3) Hyperparameter optimization (HPO) optimizes the hyperparameters of the advanced analytical placer DREAMPlace [46], [29], [44] over mixed (discrete and continuous) search spaces, where the running of each configured DREAMPlace corresponds to the procedure of solution decoding, which generates a placement result of modules. BBOPlace-Bench integrates a suite of BBO algorithms spanning simulated annealing (SA) [73], [58], evolutionary algorithms (EAs) [6], [91] (including vanilla-EA, evolution strategies (ES) [32], [31], and particle swarm optimization (PSO) [38], [28]), and Bayesian optimization (BO) [67], [22], [23], which interact with the problem formulation to optimize layouts using fitness feedback. For evalua-

tion, BBOPlace-Bench standardizes two industrial chip suites (ISPD 2005 [59] and ICCAD 2015 [39]) and offers multiple metrics: MP-HPWL (i.e., HPWL of macros) for general BBO scenarios and GP-HPWL (i.e., HPWL of macros and all the standard cells) for expensive evaluation settings, alongside PPA evaluation by commercial tools. This architecture systematically decouples components to accelerate research into how problem formulation, algorithm design, and evaluation metrics interact in advancing BBO for chip placement.

To validate the efficacy and versatility of BBOPlace-Bench, we conduct comprehensive experiments on various chip cases, systematically evaluating five representative BBO algorithms (SA, Vanilla-EA, ES, PSO, BO) across the three aforementioned problem formulations. Our experimental design targets two core optimization objectives: minimizing MP-HPWL for general BBO evaluation scenario and optimizing GP-HPWL to assess performance under expensive evaluation scenario. The analysis focuses on two critical dimensions: 1) Analyzing the methodological trade-offs, including the scalability limits of permutation-based SP, the efficiency of MGO in low-dimensional spaces, and the advantages of HPO in tuning the hyperparameters of analytical placers, together with the strengths and weaknesses of each BBO algorithm (for example, EAs efficiently obtain well-performing solution quality while BO struggles in high-dimensional settings); 2) Benchmarking against state-of-the-art analytical methods (e.g., DREAMPlace [46], [44]) and RL approaches (e.g., AlphaChip [57], [27], MaskPlace [42], and EfficientPlace [25]) to quantify BBO's competitiveness in real-world chip placement tasks. These experiments confirm BBOPlace-Bench's ability to robustly assess algorithmic performance across diverse industrial benchmarks and reveal BBO's potential to outperform analytical and RL methods in both macro placement and global placement stages, laying a foundation for further advancements in BBO-driven chip design.

BBOPlace-Bench not only offers effective approaches for chip placement and facilitates the development of efficient BBO-driven solutions in this field, but also expands the application scenarios of BBO algorithms and broadens their practical use in chip design. Our contributions can be summarized as follows:

- We propose the first BBO benchmark BBOPlace-Bench for chip placement, providing an important, real-world, and challenging task for BBO algorithms. BBOPlace-Bench integrates three problem formulations (SP, MGO, and HPO) of BBO for chip placement and five representative BBO algorithms (SA, Vanilla-EA, ES, PSO and BO), and supports evaluations of three kinds of metrics (MP-HPWL, GP-HPWL and PPA) on two standard industrial chip suites (ISPD 2005 and ICCAD 2015).
- We decouple problem formulation, optimization algorithms, and evaluation, making BBOPlace-Bench convenient for researchers in the BBO community to compare their own algorithms in a clear manner.
- We conduct extensive empirical results, including MP-HPWL and GP-HPWL evaluation on ISPD 2005 and ICCAD 2015, as well as PPA evaluation on ICCAD 2015, and provide detailed discussions on different problem

formulations and BBO algorithms. The problem formulations of MGO and HPO exhibit superior performance than the SP problem formulation. For BBO algorithms, Vanilla-EA and PSO outperform SA and BO in most settings. Furthermore, compared to state-of-the-art analytical methods (i.e., DREAMPlace [46]) and RL methods (i.e., MaskPlace [42]), some combinations of BBO formulation and algorithms (e.g., Vanilla-EA under the MGO formulation and PSO under the HPO formulation) can perform better, showing the potential of BBO for chip placement.

- We hope our flexible framework not only facilitates solving the critical chip placement problem but also provides a real-world, important benchmark urgently needed for the BBO community. We hope it can promote the research of BBO algorithms (since BBOPlace-Bench involves several challenging optimization scenarios, e.g., high-dimensional optimization and expensive optimization), and expand the influence of the BBO field.

## II. BACKGROUND

### A. Black-Box Optimization

Black-box optimization (BBO) refers to the process of optimizing an objective function whose mathematical formulation and internal structure are unknown or inaccessible, relying solely on input-output evaluations to iteratively search for optimal solutions [5], [23], [88]. BBO algorithms mainly fall into three categories: SA, EA, and BO. SA [73], [58] is a stochastic optimization algorithm that performs search through iteratively mutating the candidate solution. A key characteristic of SA is its ability to accept worse solutions with a probability that is dynamically controlled by a predefined cooling schedule (referred to as temperature), which enables the algorithm to escape from local optima and gradually approach the global optimum. However, traditional SA has inherent drawbacks, such as slow convergence rate, which primarily stems from its single-point search mechanism that explores the solution space sequentially. In contrast, EAs [6], [91] are population-based BBO algorithms, which leverage a set of candidate solutions (i.e., population) to search in parallel by simulating natural evolutionary processes with reproduction (including mutation and crossover operators for generating offspring solutions) and natural selection (where better-performing solutions are more likely to survive in the generation), thereby enhancing exploration efficiency and mitigating limitations of single-point search strategies. There are many popular variants of EAs, including ES [32], [31] and PSO [38], [28], with their main difference lying in the reproduction operators used. Since the reproduction operators usually have global search capability and can generate any solution in the search space, EAs can converge to the global optimum [65] and also have certain theoretical guarantees [91]. BO [67], [22], [23] is a widely used sample-efficient method for expensive BBO problems. At each iteration, BO fits a surrogate model, typically Gaussian process [64], to approximate the objective function, and maximizes an acquisition function (e.g., probability of improvement [40], expected improvement [36] or upper confidence bound [71]) to determine the next query point, automatically balancing exploration and exploitation.

Despite there are various practical applications of BBO, *real-world* tasks addressed in academic research remain limited, primarily focusing on problems such as hyperparameter optimization of machine learning algorithms [4], [87], and robotic control [75]. This paper aims to formulate the important and challenging problem of chip placement within a framework that is conducive to BBO, thereby expanding the application scope of BBO. The framework we provide is user-friendly and facilitates the integration of various advanced BBO algorithms.

### B. Chip Placement

Chip placement is a vital stage in modern chip design. The main input of chip placement is a netlist  $\mathcal{N} = (V, E)$ , where  $V$  denotes the information (i.e., height and width) about all modules (macros and standard cells) of the chip, and  $E$  is a hyper-graph comprised of nets  $e_i \in E$ , which encompasses multiple modules and denotes their inter-connectivity. Given a netlist, a fixed canvas layout, and a library of modules, a chip placement method is expected to determine the appropriate physical locations of movable modules on the canvas such that the objective (e.g., total wirelength) of measuring the chip performance can be optimized. Given  $k$  modules to be placed  $\{m_i\}_{i=1}^k$ , a legal chip placement solution  $s = \{(x_i, y_i)\}_{i=1}^k \in \mathbb{R}^{2k}$  includes the positions of all the modules, where  $x_i$  and  $y_i$  denote the horizontal and vertical physical coordinates of module  $m_i$ , respectively. As we mentioned before, a typical formulation of chip placement is to minimize the total HPWL of all the nets while satisfying the cell density constraint, which is formulated as follows:

$$\min_s \text{HPWL}(s) = \min_s \sum_{e \in E} \text{HPWL}_e(s), \quad (1)$$

$$\text{s.t. } D(s) \leq \epsilon,$$

where  $D$  denotes the density (such as rectangular uniform wire density [70]), and  $\epsilon$  is a threshold of density.

$\text{HPWL}_e$  in Eq. (1) is the HPWL of net  $e$ , which is defined as:

$$\text{HPWL}_e(s) = (\max_{P \in e} P_x - \min_{P \in e} P_x) + (\max_{P \in e} P_y - \min_{P \in e} P_y), \quad (2)$$

where  $P \in e$  denotes the pin of a module in net  $e$ , which serves as input and output interfaces for the module and should be connected using wires.  $P_x$  and  $P_y$  denote the horizontal and vertical coordinates of pin  $P$ , respectively. Intuitively,  $\text{HPWL}_e$  is calculated as the half perimeter of the rectangle that encloses all the pins in net  $e$ . For a better understanding of HPWL for BBO researchers, we provide an example illustration of 2D chip canvas in Figure 2, where  $m_i$  and  $P_{(i,j)}$  denote the  $i$ -th module and its  $j$ -th pin, respectively. Net  $e_1$  (colored in green) connects modules  $m_1$ ,  $m_2$  and  $m_3$  using wires through pins  $P_{(1,1)}$ ,  $P_{(2,1)}$  and  $P_{(3,2)}$ , while net  $e_2$  (colored in purple) connects modules  $m_2$ ,  $m_3$  and  $m_4$  using wires through pins  $P_{(2,2)}$ ,  $P_{(3,1)}$  and  $P_{(4,1)}$ . Solid boxes in green and purple are the bounding boxes for the two nets. Based on Eqs. (1) and (2), the HPWL can be computed as  $w_1 + h_1 + w_2 + h_2 = 4 + 5 + 5 + 5 = 19$ .

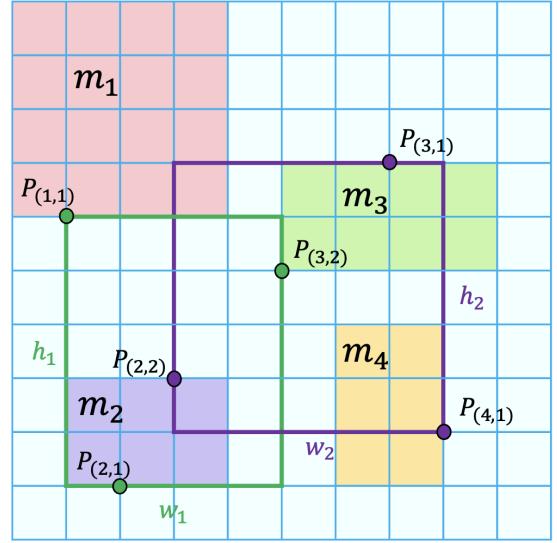


Figure 2: Example illustration of calculating HPWL. There are four modules  $\{m_i\}_{i=1}^4$  on the chip canvas, where  $P_{(i,j)}$  denotes the  $j$ -th pin of module  $m_i$ . There are two nets based on these modules, where net  $e_1$  (colored in green) connects modules  $m_1$ ,  $m_2$  and  $m_3$  using wires through pins  $P_{(1,1)}$ ,  $P_{(2,1)}$  and  $P_{(3,2)}$ , and net  $e_2$  (colored in purple) connects modules  $m_2$ ,  $m_3$  and  $m_4$  using wires through pins  $P_{(2,2)}$ ,  $P_{(3,1)}$  and  $P_{(4,1)}$ . HPWL $_e$  is calculated as the half perimeter of the rectangle that encloses all the pins in net  $e$ ; thus, the HPWL value of the current placement result is  $w_1 + h_1 + w_2 + h_2 = 4 + 5 + 5 + 5 = 19$ .

The calculation of HPWL is non-differentiable and the optimization of Eq. (1) is NP-complete [50]. Solving such a problem is extremely challenging. There are three mainstream methods for chip placement, i.e., analytical methods, reinforcement learning-based methods, and black-box optimization methods, which will be introduced in the next section.

### C. Existing Methods and Benchmarks

a) *Analytical Methods*: Analytical methods reformulate the optimization objective in Eq. (1) with a smooth version and use the weighted average model for approximating the gradient. They typically place macros and standard cells simultaneously, and can be roughly categorized into the following two kinds of approaches [12]. Quadratic placement [33], [45] iterates between an unconstrained quadratic programming phase to minimize wirelength and a heuristic spreading phase to remove overlaps. Nonlinear placement [14], [50], [15] formulates a nonlinear optimization problem and tries to directly solve it with gradient descent methods, which can achieve better solution quality but restrict to low efficiency. Recently, there has been extensive attention on GPU-accelerated nonlinear placement methods. For example, DREAMPlace [46], [44] transforms the non-linear placement problem of optimizing Eq. (1) into a neural network training problem and solves it by gradient descent with GPU acceleration, enabling ultra-high parallelism and producing state-of-the-art analytical placement

quality. However, the existing analytical approaches still suffer from the following two drawbacks: 1) Analytical placers optimize complex objectives by approximating gradients, a process prone to getting stuck in local optima and even encountering failure cases [42], [83]; 2) Advanced analytical placers involve numerous hyperparameters that require tuning, and optimizing them effectively remains non-trivial [1].

*b) Reinforcement Learning-based Methods:* RL has become a popular approach for chip placement since the *Nature*'s publication of AlphaChip [57], [27] in 2021, which solves chip placement using RL for the first time. AlphaChip uses a two-stage pipeline: An RL agent places the macros one by one in the first stage, and a placement tool handles the remaining standard cells in the second stage. For the first stage, AlphaChip divides the chip canvas into discrete grids, then sorts the macros according to some rules (for example, prioritizing the larger macros), and finally learns to place one macro per time step until all are placed. The detailed Markov decision process (MDP) is formulated as follows: States encode key placement information, such as netlist metadata; Actions are valid grid coordinates for placing the current macro without violating constraints; Rewards are zero for all intermediate actions and a negatively weighted sum of proxy wirelength, congestion, and density for the final action that leads to a whole placement. Since then, many works on RL for chip placement have been studied to address AlphaChip's limitations, such as enhancing state-representation capabilities [17], [16], constructing dense rewards [42], [82], incorporating offline training [41], and leveraging efficient planning [25], with recent works showing competitive or better performance compared with analytical placers. However, RL-based placement methods still face challenges in generalizing to different circuit designs and require substantial computational resources for training, which may limit their practical applicability in industrial settings.

*c) Black-box Optimization Methods:* BBO has been used for chip placement for three decades. Earlier approaches, such as the sequence pair (SP [58], [60]), suffer from poor scalability due to an inefficient rectangular packing formulation, which requires compact placements that do not conform to advanced chip-design standards, and suffers from enormous search space as well as complex transformation from BBO solutions to legal placement results. Recently, some BBO methods have made significant progress by changing the search space. AutoDMP [1] improves DREAMPlace by using Bayesian optimization to explore the hyperparameter configuration space and shows remarkable performance on multiple benchmarks. WireMask-BBO [68] adopts a wire-mask-guided greedy mapping from genotype solutions to phenotype solutions, and can be equipped with any BBO algorithm, demonstrating superior performance over packing-based, RL, and analytical methods. In this paper, our proposed BBOPlace-Bench integrate these BBO problem formulation approaches into a unified benchmark for easier comparison and development of BBO algorithms for chip placement.

*d) Benchmarks:* Recently, there have been some benchmarks for AI in EDA. Competitions in the EDA field, such as ISPD contest [59] and ICCAD contest [39], provide datasets

with basic chip information that include fundamental netlist information for several chips. However, they do not offer placement methods, and researchers need to extract the data themselves and implement their own methods. CircuitNet [10], [86] focuses on providing multi-modal data for chip performance prediction tasks, enhancing the capability through the use of diverse data modalities. ChiPBench [81] emphasizes the entire end-to-end EDA workflow, providing complete files for each case and necessary design kits, thereby offering a comprehensive dataset that supports all stages of design. In contrast, our proposed BBOPlace-Bench aims to provide a unified and user-friendly benchmark that is specifically designed for BBO in chip placement research, allowing researchers in the BBO community to focus solely on BBO algorithm design and encouraging the expansion of BBO applications in the emerging and critical field of chip placement.

### III. BBOPLACE-BENCH

In this section, we introduce our proposed BBO benchmark, BBOPlace-Bench, for chip placement. We first introduce how to bridge existing chip placement cases, i.e., ISPD 2005 [59] and ICCAD 2015 [39], with BBO in Section III-A. Then, we introduce the three major components in our benchmark, i.e., problem formulation, optimization algorithm, and evaluation in Sections III-B, III-C, and III-D, respectively. We finally introduce the BBO user-friendly interfaces and give a code example in Section III-E. Figure 1 gives an illustration of our BBOPlace-Bench.

#### A. Preprocessing: Bridge Chip Placement and BBO

With the fast development of EDA, datasets for chip design have undergone significant changes in structure and format. Early datasets, such as ISPD 2005 [59], use a simplified *Bookshelf* format, which is, however, not suitable for real-world chip design and manufacturing due to missing the significant amount of important information. In contrast, later datasets such as ICCAD 2015 [39] are much more complex, which offer *LEF/DEF* format along with other necessary files, including essential information for subsequent design stages. To address the challenges posed by different dataset formats, we provide interfaces that are compatible with both *Bookshelf* and *LEF/DEF* formats and capable of processing them. Based on this, we extract the essential information required for the placement stage, which makes it easier to implement the components of problem formulation and evaluation for BBO.

#### B. Problem Formulations

This section presents three problem-formulation approaches for BBO in chip placement, i.e., how to map a BBO solution (genotype) to a valid chip placement solution (phenotype).

*1) Sequence Pair (SP):* SP is a traditional combinatorial way of encoding solutions in chip placement [58], which represents a chip placement solution by a pair of permutations  $s = (\pi^+, \pi^-)$  of  $k$  modules  $\{m_i\}_{i=1}^k$ . SP extracts relative relationships between modules from the permutations and then determines their coordinates on the chip canvas by the

**Algorithm 1** Sequence pair formulation for chip placement

---

**Input** Netlist  $\mathcal{N} = (V, E)$ , set of modules  $\{m_i\}_{i=1}^k$  to be placed, SP solution  $s = (\pi^+, \pi^-)$   
**Output** Modules' coordinates  $\{(x_i, y_i)\}_{i=1}^k$

- 1: Initialize empty horizontal order array  $H$  and vertical order array  $V$
- 2: Compute position indices  $pos^+(i)$  in  $\pi^+$  and  $pos^-(i)$  in  $\pi^-$  for each module  $m_i$
- 3: **for** each module pair  $(m_i, m_j)$  **do**
- 4:   **if**  $pos^+(i) < pos^+(j)$  **and**  $pos^-(i) < pos^-(j)$  **then**
- 5:      $m_i$  is to the left of  $m_j$ , add  $(m_i, m_j)$  to  $H$
- 6:   **else if**  $pos^+(i) > pos^+(j)$  **and**  $pos^-(i) < pos^-(j)$  **then**
- 7:      $m_i$  is below  $m_j$ , add  $(m_i, m_j)$  to  $V$
- 8: Determine horizontal and vertical orders using longest common subsequence on  $H$  and  $V$
- 9: Calculate  $x$ -coordinates based on horizontal order and  $y$ -coordinates based on vertical order
- 10: Adjust coordinates to ensure no overlaps and minimal area

**Return** Modules' coordinates  $\{(x_i, y_i)\}_{i=1}^k$

---

longest common subsequence (LCS) algorithm. Detailed SP problem formulation flow is shown in Algorithm 1. For any two modules  $m_i$  and  $m_j$ , their relative positions (i.e., left, right, down, up) are determined by comparing their indices in  $\pi^+$  and  $\pi^-$ . Specifically, for each module  $m_i$ , we compute its position indices  $pos^+(i)$  in  $\pi^+$  and  $pos^-(i)$  in  $\pi^-$ . The relative relationships are established as follows: If  $pos^+(i) < pos^+(j)$  and  $pos^-(i) < pos^-(j)$ ,  $m_i$  is to the left of  $m_j$ ; If  $pos^+(i) > pos^+(j)$  and  $pos^-(i) < pos^-(j)$ ,  $m_i$  is below  $m_j$ . These relationships are stored in the horizontal order array  $H$  and vertical order array  $V$ . Then, the LCS algorithm is applied on  $H$  and  $V$  to determine the horizontal and vertical orders, from which  $x$  and  $y$  coordinates of the modules are calculated. Finally, these coordinates are adjusted to ensure no overlaps and minimal area, yielding the final placement solution  $\{(x_i, y_i)\}_{i=1}^k$ , where  $(x_i, y_i)$  is the physical coordinate of module  $m_i$  on the chip canvas. SP provides a structured combinatorial framework for encoding module placements, leveraging permutation-based relationships to ensure minimal area placement with no further adjustments needed. Its formulation is intuitive for modeling relative positions, making it suitable for problems where positional constraints dominate. However, the permutation search space grows exponentially with the number of modules. Additionally, extracting the relative relationships for all module pairs requires  $O(k^2)$  time complexity due to the double-loop structure, leading to high computational costs and poor scalability for large designs [57].

2) *Mask-Guided Optimization (MGO)*: Since the chip canvas is two-dimensional, it is natural to treat it as a grid and represent solutions using coordinates. Under this grid representation, a chip placement solution  $s$  is defined by the coordinates of all modules  $\{m_i\}_{i=1}^k$ , i.e.,  $s = \{(x_i, y_i)\}_{i=1}^k$ , where  $(x_i, y_i)$  denotes the coordinates of the module  $m_i$  on the chip canvas. However, if optimizing in this search space directly, it is difficult to efficiently find a solution that

**Algorithm 2** Mask-guided optimization formulation for chip placement

---

**Input** Netlist  $\mathcal{N} = (V, E)$ , set of modules  $\{m_i\}_{i=1}^k$  to be placed, MGO solution  $s = \{(x_i, y_i)\}_{i=1}^k$   
**Output** Modules' coordinates  $\{(x'_i, y'_i)\}_{i=1}^k$

- 1: Initialize chip canvas as  $n \times n$  grids
- 2: Order macros  $\{m_i\}_{i=1}^k$  in descending order of their areas of all the connected modules
- 3: **for** each macro  $m_i$  in the ordered list **do**
- 4:   Generate wire mask  $W_i$  that records incremental HPWL by placing  $m_i$  in each grid
- 5:   Exclude grids causing overlap or exceeding canvas boundaries from  $W_i$
- 6:   Find the best grids  $W_i^*$  with minimum incremental HPWL
- 7:   Select the grid  $(x'_i, y'_i)$  within  $W_i^*$  that is nearest to  $m_i$ 's original position  $(x_i, y_i)$

**Return** Modules' coordinates  $\{(x'_i, y'_i)\}_{i=1}^k$

---

has a small HPWL value and satisfies the non-overlapping constraint [3], [42]. To improve the optimization efficiency and ensure constraint satisfaction, Shi et al. [68] proposed a wire-mask-guided greedy procedure, which converts *any given solution* into a valid, high-quality placement result. Detailed algorithm flow is provided in Algorithm 2. For each module  $m_i$ , we first compute the total area of all modules that connect to it in its nets; modules are then permuted in descending order based on these areas to determine the adjustment priority, as modules with larger connected areas are more critical to placement quality. When placing each module in this order, a wire mask [42] is generated, which records the incremental HPWL caused by placing the current module to each candidate grid. Grids that would cause overlap or exceed the canvas boundary are excluded, making MGO formulation efficiently to satisfy the non-overlapping constraint. The module is then moved to the grid that minimizes the incremental HPWL; in case of ties, the grid closest to the original position of the module (that stored in the MGO solution) is selected. This sequential, greedy adjustment ensures both small HPWL and non-overlapping placement, balancing quality and legality effectively.

3) *Hyperparameter Optimization (HPO)*: Although the analytical placers are currently the mainstream methods for chip placement, they usually involve numerous hyperparameters that must be tuned carefully, and inappropriate adjustments can significantly degrade performance [1]. BBO algorithms have proven to be efficient methods for HPO and achieve excellent performance on various tasks. Therefore, an effective problem formulation of BBO for chip placement [1] is to use BBO to tune the hyperparameters of the state-of-the-art analytical placer, e.g., DREAMPlace [46], [29], [44]. In the HPO formulation, we set the search space for chip placement as the hyperparameter space  $\Theta$  of DREAMPlace, with details shown in Table I. The first part in Table I consists of the general placement configurations, and the second part includes the configurations at each iteration of DREAMPlace. The

Table I: Search space of HPO in BBOPlace-Bench.

HPO search space	Type	Range
GP_num_bins_x	discrete	{1024, 2048}
GP_num_bins_y	discrete	{1024, 2048}
GP_optimizer	discrete	{"adam", "nesterov"}
GP_wirelength	discrete	{"weighted_average", "logsumexp"}
GP_learning_rate	continuous	[0.001, 0.01]
GP_Lambda_density_weight_iteration	continuous	[1, 3]
GP_Sub_iteration	continuous	[1, 3]
GP_learning_rate_decay	continuous	[0.99, 1.0]
stop_overflow	continuous	[0.06, 0.1]
target_density	continuous	[0.8, 1.2]
RePIAce_LOWER_PCOF	continuous	[0.9, 0.99]
RePIAce_UPPER_PCOF	continuous	[1.02, 1.15]
RePIAce_ref_hpwl	continuous	[150000, 550000]
density_weight	continuous	[1e-6, 1e-4]
gamma	continuous	[1, 4]

**Algorithm 3** Hyperparameter optimization formulation for chip placement

**Input** Netlist  $\mathcal{N} = (V, E)$ , set of modules  $\{m_i\}_{i=1}^k$  to be placed, analytical placer  $\mathcal{P}$ , HPO solution  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_N) \in \Theta$

**Output** Modules' coordinates  $\{(x_i, y_i)\}_{i=1}^k$

- 1: Set analytical placer (e.g., DREAMPlace)  $\mathcal{P}_\theta$  with the HPO solution  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_N)$
- 2: Run analytical placement process:  $s \leftarrow \mathcal{P}_\theta(\mathcal{N}, \{m_i\}_{i=1}^k)$

**Return** Modules' coordinates  $s = \{(x_i, y_i)\}_{i=1}^k$

mixed search space requires specialized operators for handling both discrete and continuous variables. The detailed HPO formulation is shown in Algorithm 3, where running a configured DREAMPlace corresponds to the procedure of solution decoding, which generates a placement result of modules.

Note that the three BBO formulations in BBOPlace-Bench involve different search spaces: permutation, continuous, and mixed (both continuous and discrete) search spaces, which thus are helpful for developing various types of BBO algorithms. The search space size of SP and MGO is related to the number of macros, while the search space for HPO is fixed, i.e., the hyperparameter space of the analytical placer DREAMPlace. As the number of macros can be specified arbitrarily in our BBOPlace-Benchmark, the SP and MGO formulations can also be used for research on high-dimensional BBO, which is a recent popular topic in BBO.

### C. Optimization Algorithms

We use the following three typical kinds of BBO algorithms in our BBOPlace-Bench:

- SA [73] is a classic approach in chip placement [58]. By mimicking the cooling process of metals, it effectively explores the search space, balancing exploration and exploitation to minimize an objective function. Its ability to escape from local minima makes it particularly valuable in optimizing complex layouts.
- EA is a population-based search framework [6]. In this paper, we implement three main types of EA: Vanilla-EA, ES, and PSO.
  - Vanilla-EA is a basic EA, which updates a set of solutions (i.e., population) by iterative crossover,

mutation, and selection. We implement various operators to handle different types of search spaces. Details can be found in Section IV-A.

- ES is a representative method used in the field of continuous BBO. We integrate pycma<sup>1</sup>, a popular implementation of CMA-ES [31] in Python, into our benchmark. It not only provides a basic implementation of CMA-ES but also includes numerous advanced features suitable for high-dimensional optimization and many other scenarios.
- PSO [38] is a population-based optimization technique inspired by social behavior of bird flocking or fish schooling. It maintains a swarm of particles that explore the search space by following their own best known position as well as the swarm's best known position. PSO is particularly effective for continuous optimization problems and has been successfully applied to various engineering problems.
- BO [23] is a sample-efficient method for expensive BBO problems. We integrate one of the most popular BO frameworks, BoTorch [7]<sup>2</sup>, into our benchmark. BoTorch leverages GPUs for efficient Gaussian process fitting and inference, and includes a wide range of advanced BO algorithms, e.g., TurBO [21] and SAASBO [20].

### D. Evaluation

As an important part of the EDA process, chip placement has many evaluation approaches. In our benchmark, we employ the following three methods.

a) *Macro Placement HPWL*: The chip placement problem can be divided into two successive stages [1]: Macro placement (MP) and standard cell placement. That is, macros are first placed, and then the positions of standard cells are determined. MP heavily influences the subsequent placement of standard cells, and poor MP might make it challenging to place standard cells optimally, leading to an unsatisfactory chip performance. Thus, MP-HPWL, i.e., the HPWL value of all the macros, is an important metric for evaluating the quality of chip placement.

b) *Global Placement HPWL*: GP-HPWL is the HPWL value of all the macros and standard cells. Compared to MP-HPWL, GP-HPWL is more closely related to the final chip performance but the computation takes much more time to place the standard cells. In BBOPlace-Bench, after obtaining the positions of macros through different problem formulations and optimization algorithms, if GP-HPWL evaluation is required, we will fix the already placed macros and then place the standard cells by DREAMPlace [46] to obtain GP-HPWL. GP-HPWL considers the total wirelength of both macros and standard cells, typically on a scale that is two orders of magnitude larger than MP-HPWL, providing a better estimation of the final performance of the chip. As the number of macros is much smaller than the number of standard cells (several hundreds vs. millions), MP-HPWL is cheaper to evaluate and more suitable as a surrogate metric for

<sup>1</sup><https://github.com/CMA-ES/pycma>

<sup>2</sup><https://github.com/pytorch/botorch>

BBO algorithms. In our BBOPlace-Bench framework, the GP-HPWL interface can be called independently, in which case the problem can be treated as an expensive BBO problem. The detailed time overhead comparison of MP-HPWL and GP-HPWL is shown in Section IV-E.

c) *PPA Evaluation:* The entire chip-design process is lengthy and complex. Proxy metrics, like MP-HPWL and GP-HPWL, may not accurately reflect the final chip performance, namely the power, performance, and area (PPA) metrics. In this paper, once the global placement results (i.e., the positions of all macros and standard cells) are obtained, we also utilize the commercial tool *Cadence Innovus* to proceed the subsequent stages (including routing, RC extraction, clock tree synthesis, etc) and evaluate the PPA metrics, including routed wirelength (wirelength of all the modules after routing), routed vertical and horizontal congestion overflow (congestion that exceeds the limits in vertical and horizontal directions), worst negative slack (the maximum amount by which the timing fails to meet its required constraint), total negative slack (the sum of all negative slacks across all timing paths that fail to meet their constraints), and the number of violation paths (the number of paths that violate timing constraints). Although these metrics are difficult to obtain and expensive to evaluate, they are crucial in chip design and are used to comprehensively assess chip quality.

#### E. BBO User-Friendly Interfaces

Our proposed BBOPlace-Bench has easy-to-use interfaces, making it very easy to set up the execution of both built-in algorithms and user-customized algorithms. A simple example of running CMA-ES under the MGO formulation on the chip case adaptec1 is shown in Code Example 1. It runs the optimization process by setting parameters in lines 5–13 (e.g., `benchmark = adaptec1`, `placer = mgo`), initializing an evaluator (in lines 15–20) and optimizer (in lines 22–28), and then iteratively generating and evaluating solutions in lines 30–35. We also provide a visualization interface that conveniently displays the placement of modules, allowing for an intuitive assessment of the results, as shown in Figure 4.

```

1 import argparse
2 import numpy as np
3 from src.evaluator import Evaluator
4
5 # Set parameters
6 parser = argparse.ArgumentParser()
7 parser.add_argument("--benchmark", type=str, default="adaptec1")
8 parser.add_argument("--eval_gp_hpwl", action="store_true", default=False)
9 parser.add_argument("--placer", type=str, choices=["sp", "mgo", "hpo"], default="mgo")
10 parser.add_argument("--sigma", type=float, default=0.5)
11 parser.add_argument("--pop_size", type=int, default=20)
12 parser.add_argument("--seed", type=int, default=42)
13 args = parser.parse_args()
14
15 # Initialize the evaluator
16 evaluator = Evaluator(args)
17 dim = evaluator.n_dim
18 xl = evaluator.xl.tolist()
19 xu = evaluator.xu.tolist()
20 x0 = np.random.uniform(low=xl, high=xu, size=dim)
21 # Initialize CMA-ES
22 cmaes = cma.CMAEvolutionStrategy(
23     x0,
24     args.sigma,
25     {'popsize': args.pop_size,
26      'bounds': [xl, xu]}
27 )
28
29 # Run CMA-ES for optimization
30 while not cmaes.stop():
31     solutions = cmaes.ask()
32     fitness_values = evaluator.evaluate(solutions)
33     cmaes.tell(solutions, fitness_values)
34     print(f"Generation {cmaes.countiter}: Best
35         fitness = {min(fitness_values):.6f}")

```

Code Example 1: Run CMA-ES [31] under the problem formulation of mask-guided optimization on the chip case adaptec1.

## IV. EXPERIMENT

In this section, we first introduce the experimental settings, including dataset preprocessing and parameter configurations for different BBO algorithms in Section IV-A. We then present the results on ISPD 2005 [59], comparing the performance of various problem formulations and BBO algorithms in terms of MP-HPWL and GP-HPWL, and comparing them with advanced RL and analytical methods in Section IV-B. Subsequently, we report the findings on ICCAD 2015 [39], analyzing HPWL results and conducting PPA evaluations to assess real-world chip performance in Section IV-C. Following that, we investigate the influence with different number of macros (i.e., different search space dimensions) on optimization efficiency in Section IV-D. Additionally, we provide a runtime analysis to compare the computational costs of different algorithms and problem formulations in Section IV-E. Finally, we show the visualization results of chip layouts to offer intuitive insights into the placement solutions generated by various methods in Section IV-F.

#### A. Experimental Settings

In the experiments, we compare different BBO algorithms under three problem formulations in BBOPlace-Bench. The main chip cases are ISPD 2005 [59] (adaptec and bigblue series) and ICCAD 2015 [39] (superblue series), with detailed information provided in Table II. An important setting for BBO in chip placement is the number of macros, which corresponds to the search space dimension for SP and MGO. For ISPD 2005, we use the number of macros specified in the dataset. For ICCAD 2015, as no macros are specified, we default to defining the 512 largest modules (by area) as macros. Due to the difficulty of handling mixed spaces, we set the search space of HPO to be continuous by converting the categorical variables to class logits in our experiments.

For a fair comparison, the hyperparameters of the compared algorithms in the following experiments are set as same as possible, e.g., the population size is set to 50 for most experiments. We introduce the detailed hyperparameters of each algorithm as follows:

```

1 import argparse
2 import numpy as np
3 from src.evaluator import Evaluator
4
5 # Set parameters
6 parser = argparse.ArgumentParser()
7 parser.add_argument("--benchmark", type=str, default="adaptec1")
8 parser.add_argument("--eval_gp_hpwl", action="store_true", default=False)
9 parser.add_argument("--placer", type=str, choices=["sp", "mgo", "hpo"], default="mgo")
10 parser.add_argument("--sigma", type=float, default=0.5)
11 parser.add_argument("--pop_size", type=int, default=20)
12 parser.add_argument("--seed", type=int, default=42)
13 args = parser.parse_args()
14
15 # Initialize the evaluator
16 evaluator = Evaluator(args)
17 dim = evaluator.n_dim
18 xl = evaluator.xl.tolist()
19 xu = evaluator.xu.tolist()

```

Table II: Detailed statistics of the chips, where # Macros, # Cells, # Nets, and # Pins denote the number of macros, standard cells, nets, and pins contained by a chip, respectively.

Chips	# Macros	# Cells	# Nets	# Pins
adaptec1	543	210,904	221,142	944,053
adaptec2	566	254,457	266,009	1,069,482
adaptec3	723	450,927	466,758	1,875,039
adaptec4	1,329	494,716	515,951	1,912,420
bigblue1	560	277,604	284,479	1,144,691
bigblue3	1,298	1,095,514	1,123,170	3,833,218
superblue1	512	1,209,716	1215710	3,767,494
superblue3	512	1,213,253	1,224,979	3,905,321
superblue4	512	795,645	802,513	2,497,940
superblue5	512	1,086,888	1,100,825	3,246,878
superblue7	512	1,931,639	1,933,945	6,372,094
superblue10	512	1,876,103	1,898,119	5,560,506
superblue16	512	981,559	999,902	3,013,268
superblue18	512	768,068	771,542	2,559,143

a) *SA*: We set the initial temperature to 100, with a decay rate of 0.99 and an update frequency of 100 steps. Mutation operators vary for different formulations. For SP, we apply the *inversion* operator, which randomly selects a start point and an end point from a permutation, and then inverts the sequence between them. For MGO, we apply the *shuffle* operator to shuffle the positions of several randomly selected macros. For HPO, we apply the *random resetting* operator to randomly select one element and assign a new value within the search space.

b) *Vanilla-EA*: As a population-based search algorithm, Vanilla-EA uses not only mutation to perturb a single solution, but also crossover to recombine two solutions. We use the same mutation operators as SA. For crossover operators, we use *order crossover* for the SP formulation, where two parent permutations exchange segments while preserving the relative order and positions of the remaining elements, and use *uniform crossover* for MGO and HPO formulations, where each element of the offspring solution comes from either

parent with equal probability.

c) *ES*: For MGO and HPO formulations, we set the initial value of sigma to 0.5. For other hyperparameters, we use the default settings in pycma<sup>3</sup>.

d) *PSO*: We integrate PyPop7 [19]<sup>4</sup> to implement PSO. For MGO and HPO formulations, we set the cognition learning rate, social learning rate and maximal ratio of velocities to 2.0, 2.0 and 0.2, respectively.

e) *BO*: We use BoTorch [7]<sup>5</sup> to implement our Bayesian optimization algorithm, with the default Matern-5/2 kernel function and expected improvement acquisition function.

Note that for the SP problem formulation, given its permutation search space, we only run SA and Vanilla-EA. In our experiments, the evaluation budget for optimizing MP-HPWL is set to 10,000, while that for optimizing GP-HPWL is set to 200 due to significantly longer evaluation time (detailed runtime comparisons are provided in Table IX of Section IV-E). All experiments are run with five seeds, which is a common setting in chip placement [42], [82].

## B. Results on ISPD 2005

a) *MP-HPWL Comparisons*: We first compare algorithms in BBOPlace-Bench on ISPD cases in terms of MP-HPWL. In addition to these methods, we include three representative RL methods<sup>6</sup> (i.e., AlphaChip [57], [27], MaskPlace [42], and EfficientPlace [25]) and one state-of-the-art analytical method DREAMPlace [46] for comparison. The results are shown in Table III. The performance of SP formulation is limited due to its exponentially growing permutation search space, yielding the poorest results with an average rank of 15–16. In contrast, the MGO and HPO formulations both

<sup>3</sup><https://github.com/CMA-ES/pycma>

<sup>4</sup><https://github.com/Evolutionary-Intelligence/pypop>

<sup>5</sup><https://github.com/pytorch/botorch>

<sup>6</sup>The results for MaskPlace come from our training, while the results for AlphaChip and EfficientPlace come from [25].

Table III: MP-HPWL values ( $\times 10^5$ ) obtained by compared methods on the six chip cases of ISPD 2005. Each result consists of the mean and standard deviation of five runs. The results of three RL methods (i.e., AlphaChip [57], MaskPlace [42], EfficientPlace [25]) are from [25]. The best and runner-up methods on each chip case are **bolded** and underlined, respectively. The symbols ‘≈’ and ‘-’ indicate that the result is almost equivalent and inferior to the best methods, respectively, according to the Wilcoxon rank-sum test with significance level 0.05.

Formulation	Algorithm	adaptec1	adaptec2	adaptec3	adaptec4	bigblue1	bigblue3	Average Rank
SP	SA	76.80±3.41 -	604.18±12.16 -	655.61±20.30 -	699.85±1.72 -	31.73±0.60 -	939.84±32.19 -	16
	Vanilla-EA	41.80±3.30 -	442.77±13.71 -	486.91±10.24 -	559.43±6.73 -	20.04±0.59 -	554.93±23.21 -	15
MGO	SA	6.32±0.05 -	83.61±5.82 -	64.05±0.73 -	65.53±0.72 -	2.44±0.02 -	67.51±3.41 -	8.67
	Vanilla-EA	<b>5.80±0.03</b>	61.46±4.47 -	56.13±0.81 -	56.79±0.80 -	2.30±0.03 -	52.40±2.30 -	4.83
	ES	6.98±0.48 -	103.66±21.81 -	66.95±2.04 -	68.67±5.35 -	2.43±0.03 -	75.66±15.96 -	9.5
	PSO	<u>5.80±0.02</u> ≈	58.78±4.31 -	57.64±0.50 -	59.56±0.39 -	2.35±0.03 -	56.04±3.01 -	5.83
	BO	6.38±0.04 -	83.25±3.44 -	63.08±1.16 -	64.34±0.86 -	2.46±0.02 -	65.19±2.01 -	8.33
HPO	SA	7.89±0.12 -	34.30±1.82 -	<u>53.07±0.89</u> ≈	43.33±0.23 -	3.45±0.07 -	42.44±1.66 -	5.33
	Vanilla-EA	7.55±0.11 -	32.06±0.47 ≈	<b>52.70±0.89</b>	42.77±0.54 ≈	3.35±0.06 -	<b>40.03±0.72</b>	<b>3.5</b>
	ES	8.15±0.12 -	33.00±0.31 -	53.57±0.73 ≈	43.94±1.11 -	3.40±0.04 -	41.69±0.00 -	5.67
	PSO	7.99±0.25 -	<b>31.73±0.70</b>	53.07±0.07 ≈	<b>42.49±0.08</b>	3.39±0.01 -	42.57±1.57 -	4.5
	BO	9.33±0.76 -	37.41±1.82 -	56.27±1.59 -	47.70±1.22 -	3.69±0.04 -	46.16±3.51 -	7.5
RL	AlphaChip [57]	30.01±2.98 -	351.71±38.20 -	358.18±13.95 -	151.42±13 -	10.58±1.29 -	357.48±47.83 -	13.83
	MaskPlace [42]	7.62±0.67 -	75.16±4.97 -	100.24±13.54 -	87.99±3.25 -	3.04±0.06 -	90.04±4.83 -	10.33
	EfficientPlace [25]	5.94±0.04 -	46.79±1.60 -	56.35±0.99 -	58.47±1.61 -	<b>2.14±0.01</b>	58.38±0.54 -	5.5
Analytical	DREAMPlace [46]	21.20±9.13 -	40.05±4.09 -	62.94±0.45 -	345.37±39.96 -	5.40±0.46 -	93.02±6.57 -	11.33

Table IV: GP-HPWL values ( $\times 10^7$ ) obtained by compared methods on the six chip cases of ISPD 2005. Each result consists of the mean and standard deviation of five runs. The results of two RL methods (i.e., MaskPlace [42] and EfficientPlace [25]) are from [25]. The best and runner-up methods on each chip case are **bolded** and underlined, respectively. The symbols ‘ $\approx$ ’ and ‘-’ indicate that the result is almost equivalent and inferior to the best methods, respectively, according to the Wilcoxon rank-sum test with significance level 0.05.

Formulation	Algorithm	adaptec1	adaptec2	adaptec3	adaptec4	bigblue1	bigblue3	Average Rank
SP	SA	$11.87 \pm 0.28$ -	$18.29 \pm 0.19$ -	$31.51 \pm 0.31$ -	$33.50 \pm 0.32$ -	$11.54 \pm 0.12$ -	$58.34 \pm 0.96$ -	14.83
	Vanilla-EA	$11.41 \pm 0.18$ -	$17.37 \pm 0.26$ -	$30.00 \pm 0.36$ -	$33.47 \pm 0.25$ -	$11.31 \pm 0.11$ -	$51.98 \pm 1.62$ -	13.67
MGO	SA	$8.93 \pm 0.09$ -	$12.08 \pm 0.50$ -	$20.30 \pm 0.47$ -	$21.62 \pm 0.26$ -	$9.42 \pm 0.06$ -	$45.57 \pm 0.56$ -	9.83
	Vanilla-EA	$8.49 \pm 0.08$ -	$11.05 \pm 0.26$ -	$18.45 \pm 0.23$ -	$19.80 \pm 0.73$ -	$9.29 \pm 0.05$ -	$40.43 \pm 0.57$ -	7.67
	ES	$9.33 \pm 0.36$ -	$13.39 \pm 0.58$ -	$21.85 \pm 1.24$ -	$23.01 \pm 0.35$ -	$9.70 \pm 0.15$ -	$47.31 \pm 2.21$ -	11.83
	PSO	$8.65 \pm 0.07$ -	$12.27 \pm 0.16$ -	$18.01 \pm 0.45$ -	$21.35 \pm 0.68$ -	$9.32 \pm 0.07$ -	$44.52 \pm 1.04$ -	8.5
	BO	$9.01 \pm 0.20$ -	$12.36 \pm 0.48$ -	$20.16 \pm 0.27$ -	$21.44 \pm 0.35$ -	$9.45 \pm 0.03$ -	$45.45 \pm 1.31$ -	10
HPO	SA	$6.10 \pm 0.06 \approx$	$6.95 \pm 0.12 \approx$	$12.84 \pm 0.10$ -	$12.32 \pm 0.09$ -	$8.10 \pm 0.05 \approx$	$25.36 \pm 0.77$ -	4.67
	Vanilla-EA	<b><math>6.05 \pm 0.03</math></b>	$6.82 \pm 0.08 \approx$	$12.73 \pm 0.11 \approx$	<u><math>12.12 \pm 0.08 \approx</math></u>	$8.06 \pm 0.03 \approx$	<b><math>24.09 \pm 0.19</math></b>	<b>2</b>
	ES	$6.09 \pm 0.03 \approx$	$6.87 \pm 0.14 \approx$	<b><math>12.63 \pm 0.08</math></b>	$12.21 \pm 0.04$ -	$8.11 \pm 0.03$ -	$24.72 \pm 0.60 \approx$	3.17
	PSO	$6.08 \pm 0.02 \approx$	$7.00 \pm 0.12 \approx$	$12.64 \pm 0.08 \approx$	$12.18 \pm 0.15 \approx$	$8.05 \pm 0.02 \approx$	$24.51 \pm 0.40 \approx$	2.67
	BO	$6.09 \pm 0.04 \approx$	<b><math>6.72 \pm 0.12</math></b>	$12.80 \pm 0.12 \approx$	<b><math>12.06 \pm 0.04</math></b>	<b><math>8.04 \pm 0.03</math></b>	$25.05 \pm 0.33$ -	2.33
RL	MaskPlace [42]	$10.86 \pm 0.18$ -	$12.98 \pm 0.58$ -	$26.14 \pm 0.07$ -	$26.14 \pm 0.07$ -	$10.64 \pm 0.01$ -	$54.98 \pm 1.06$ -	12.83
	EfficientPlace [25]	$7.20 \pm 0.12$ -	$9.20 \pm 0.61$ -	$16.49 \pm 1.07$ -	$14.70 \pm 0.25$ -	$8.67 \pm 0.10$ -	$28.48 \pm 0.96$ -	6.17
Analytical	DREAMPlace [46]	$9.62 \pm 0.78$ -	$12.45 \pm 3.31$ -	$17.18 \pm 0.51$ -	$40.04 \pm 3.87$ -	$8.30 \pm 0.06$ -	$38.22 \pm 1.48$ -	9.67

demonstrate superior performance. Among BBO algorithms, Vanilla-EA consistently outperforms the others: MGO-Vanilla-EA (i.e., using the MGO problem formulation with Vanilla-EA as the optimizer) achieves an average rank of 4.83, while HPO-Vanilla-EA ranks even higher at 3.5, demonstrating its superiority and aligning with previous research findings [68]. PSO shows competitive performance (average rank 5.83 for the MGO formulation and 4.5 for the HPO formulation), whereas BO struggles in high-dimensional spaces, e.g., resulting in a lower ranking 8.33 for the MGO formulation where the highest dimension is more than 1000. It has been well known that the performance of BO in high-dimensional spaces may need the assistance of additional techniques [8]. Notably, top BBO methods, e.g., HPO-Vanilla-EA (average rank 3.5), HPO-PSO (average rank 4.5), and MGO-Vanilla-EA (average rank 4.83), surpass the state-of-the-art RL method EfficientPlace [25] (average rank 5.5) and the analytical placer DREAMPlace [46] (average rank 11.33), highlighting the competitiveness of BBO in MP tasks.

b) *GP-HPWL Comparisons:* The experimental results of GP-HPWL on the ISPD cases are reported in Table IV. Note that AlphaChip [57], [27] is not included in the table for comparison due to lack of data. Overall, the HPO problem formulation leads to the best performance, which is attributable to its capability of tuning the hyperparameters of the analytical placer DREAMPlace, which simultaneously accounts for both macros and standard cells. In contrast, the SP and MGO formulations focus exclusively on macros and disregard standard-cell information, thereby limiting their capacity to optimize GP-HPWL. As can be observed in Table IV, when combined with different BBO algorithms (SA, Vanilla-EA, ES, PSO, and BO), HPO achieves substantially higher rankings than both MGO and SP, with HPO-Vanilla-EA attaining the best performance (average rank 2). SP formulation performs the worst due to its less effective search-space design. Among the BBO variants, Vanilla-EA and PSO retain their advantages

within both HPO and MGO formulations; BO under the HPO formulation is the runner-up algorithm in terms of GP-HPWL evaluation, as optimizing the expensive GP-HPWL objective allows much fewer evaluations than optimizing MP-HPWL, which is advantageous for BO. Under GP-HPWL evaluation, the best BBO method (i.e., HPO-Vanilla-EA, average rank 2.0) still outperforms RL approaches (e.g., EfficientPlace [25], average rank 6.17) as well as analytical baselines (e.g., DREAMPlace [46], average rank 9.67), highlighting the robustness of BBO in optimizing this more critical placement metric.

### C. Results on ICCAD 2015

a) *HPWL Comparisons:* The results of MP-HPWL and GP-HPWL comparisons on ICCAD 2015 are shown in Tables V and VI. Note that AlphaChip [57], [27] and EfficientPlace [25] are not included for comparison due to lack of data. We can observe from Table V that the MGO formulation dominates on MP-HPWL due to its advantage in the dataset’s low-dimensional search space (fixed 512 macros), where MGO-PSO achieves the lowest MP-HPWL with the best average rank of 1.12, followed closely by MGO-Vanilla-EA with average rank of 1.75. HPO variants are much worse than MGO, and the SP formulation remains consistently the worst performance. Similar to the GP-HPWL results on ISPD, Table VI shows that the HPO formulation continues to lead in GP-HPWL on ICCAD. To further illustrate the performance of different methods, we plot the convergence curve of GP-HPWL under the HPO formulation, as shown in Figure 3. Compared to the baselines MaskPlace [42] and DREAMPlace [46], HPO methods exhibit an early advantage and progressively achieve better solutions as the number of evaluations increases. Compared to the mixed placement of DREAMPlace, which places macros and standard cells simultaneously in a single stage, our HPO formulation under GP-HPWL evaluation adopts a two-stage placement and shows a clear early advantage by considering the critical macros at first: It first places the

Table V: MP-HPWL values ( $\times 10^5$ ) obtained by compared methods on the eight cases of ICCAD 2015. Each result consists of the mean and standard deviation of five runs. The best and runner-up methods on each chip case are **bolded** and underlined, respectively. The symbols ‘ $\approx$ ’ and ‘-’ indicate that the result is almost equivalent and inferior to the best methods, respectively, according to the Wilcoxon rank-sum test with significance level 0.05.

Formulation	Algorithm	superblue1	superblue3	superblue4	superblue5	superblue7	superblue10	superblue16	superblue18	Average Rank
SP	SA	12.74 $\pm$ 0.32 -	30.94 $\pm$ 0.36 -	20.81 $\pm$ 0.59 -	55.20 $\pm$ 1.19 -	24.67 $\pm$ 0.31 -	11.09 $\pm$ 0.67 -	29.75 $\pm$ 0.43 -	6.40 $\pm$ 0.22 -	14
	Vanilla-EA	5.27 $\pm$ 0.28 -	13.34 $\pm$ 1.04 -	11.33 $\pm$ 0.64 -	31.65 $\pm$ 2.44 -	14.15 $\pm$ 0.75 -	2.31 $\pm$ 0.19 -	14.59 $\pm$ 1.26 -	2.66 $\pm$ 0.13 -	
MGO	SA	0.62 $\pm$ 0.01 -	1.70 $\pm$ 0.03 -	1.12 $\pm$ 0.02 -	4.16 $\pm$ 0.07 -	1.81 $\pm$ 0.03 -	<u>0.55<math>\pm</math>0.00</u> -	1.21 $\pm$ 0.04 -	0.53 $\pm$ 0.01 -	3.38
	Vanilla-EA	<u>0.59<math>\pm</math>0.00</u> -	1.55 $\pm$ 0.01 -	<u>0.95<math>\pm</math>0.01</u> $\approx$	3.84 $\pm$ 0.03 -	1.72 $\pm$ 0.02 -	<b>0.54<math>\pm</math>0.00</b> -	<b>0.95<math>\pm</math>0.01</b> -	0.49 $\pm$ 0.00 -	
	ES	0.66 $\pm$ 0.04 -	1.80 $\pm$ 0.10 -	1.20 $\pm$ 0.09 -	4.78 $\pm$ 0.37 -	1.92 $\pm$ 0.10 -	<b>0.54<math>\pm</math>0.00</b> -	1.23 $\pm$ 0.08 -	0.53 $\pm$ 0.02 -	4.25
	PSO	<b>0.58<math>\pm</math>0.00</b> -	<b>1.52<math>\pm</math>0.00</b> -	<b>0.94<math>\pm</math>0.01</b> -	<b>3.76<math>\pm</math>0.00</b> -	<b>1.69<math>\pm</math>0.02</b> -	<b>0.54<math>\pm</math>0.00</b> -	<u>0.96<math>\pm</math>0.01</u> $\approx$	<b>0.48<math>\pm</math>0.00</b> -	
	BO	0.63 $\pm$ 0.01 -	1.71 $\pm$ 0.01 -	1.12 $\pm$ 0.02 -	4.13 $\pm$ 0.05 -	1.84 $\pm$ 0.03 -	<u>0.55<math>\pm</math>0.00</u> -	1.23 $\pm$ 0.02 -	0.52 $\pm$ 0.01 -	
HPO	SA	2.29 $\pm$ 0.12 -	4.86 $\pm$ 0.17 -	2.38 $\pm$ 0.08 -	10.45 $\pm$ 0.23 -	3.44 $\pm$ 0.09 -	1.88 $\pm$ 0.03 -	3.76 $\pm$ 0.24 -	1.56 $\pm$ 0.15 -	9.5
	Vanilla-EA	2.07 $\pm$ 0.14 -	4.29 $\pm$ 0.19 -	2.35 $\pm$ 0.10 -	9.98 $\pm$ 0.13 -	3.19 $\pm$ 0.03 -	1.59 $\pm$ 0.05 -	3.40 $\pm$ 0.12 -	1.43 $\pm$ 0.09 -	
	ES	2.22 $\pm$ 0.09 -	4.83 $\pm$ 0.24 -	2.32 $\pm$ 0.11 -	10.37 $\pm$ 0.38 -	3.34 $\pm$ 0.07 -	1.69 $\pm$ 0.10 -	3.90 $\pm$ 0.31 -	1.54 $\pm$ 0.15 -	8.25
	PSO	2.24 $\pm$ 0.05 -	4.56 $\pm$ 0.42 -	2.34 $\pm$ 0.09 -	9.93 $\pm$ 0.07 -	3.56 $\pm$ 0.05 -	2.14 $\pm$ 0.18 -	3.35 $\pm$ 0.03 -	1.63 $\pm$ 0.19 -	
	BO	2.60 $\pm$ 0.07 -	5.93 $\pm$ 0.29 -	2.66 $\pm$ 0.13 -	11.77 $\pm$ 0.24 -	3.93 $\pm$ 0.09 -	2.74 $\pm$ 0.34 -	4.51 $\pm$ 0.49 -	2.01 $\pm$ 0.10 -	
RL	MaskPlace [42]	2.24 $\pm$ 0.91 -	4.17 $\pm$ 0.22 -	1.50 $\pm$ 0.15 -	9.95 $\pm$ 0.11 -	2.45 $\pm$ 0.02 -	0.99 $\pm$ 0.23 -	2.38 $\pm$ 1.29 -	1.06 $\pm$ 0.22 -	6.38
Analytical	DREAMPlace [46]	2.76 $\pm$ 0.06 -	8.49 $\pm$ 0.36 -	2.93 $\pm$ 0.03 -	12.63 $\pm$ 0.39 -	4.28 $\pm$ 0.04 -	4.35 $\pm$ 0.06 -	5.11 $\pm$ 0.10 -	2.28 $\pm$ 0.02 -	12.12

Table VI: GP-HPWL values ( $\times 10^7$ ) obtained by compared methods on the eight cases of ICCAD 2015. Each result consists of the mean and standard deviation of five runs. The best and runner-up methods on each chip case are **bolded** and underlined, respectively. The symbols ‘ $\approx$ ’ and ‘-’ indicate that the result is almost equivalent and inferior to the best methods, respectively, according to the Wilcoxon rank-sum test with significance level 0.05.

Formulation	Algorithm	superblue1	superblue3	superblue4	superblue5	superblue7	superblue10	superblue16	superblue18	Average Rank
SP	SA	86.12 $\pm$ 1.92 -	88.37 $\pm$ 4.59 -	60.33 $\pm$ 2.35 -	109.15 $\pm$ 5.22 -	100.24 $\pm$ 1.62 -	102.07 $\pm$ 2.31 -	67.29 $\pm$ 1.17 -	30.29 $\pm$ 0.09 -	13.25
	Vanilla-EA	82.03 $\pm$ 1.82 -	82.14 $\pm$ 4.01 -	56.52 $\pm$ 2.81 -	100.47 $\pm$ 4.81 -	96.41 $\pm$ 1.99 -	98.26 $\pm$ 2.60 -	65.61 $\pm$ 1.45 -	29.60 $\pm$ 0.52 -	
MGO	SA	62.39 $\pm$ 0.85 -	72.63 $\pm$ 0.65 -	44.58 $\pm$ 0.66 -	81.21 $\pm$ 1.69 -	84.52 $\pm$ 0.98 -	94.07 $\pm$ 0.61 -	50.39 $\pm$ 0.55 -	28.75 $\pm$ 0.49 -	8.63
	Vanilla-EA	55.35 $\pm$ 1.24 -	62.62 $\pm$ 1.24 -	40.24 $\pm$ 0.65 -	72.15 $\pm$ 1.16 -	75.79 $\pm$ 1.47 -	88.69 $\pm$ 1.35 -	46.71 $\pm$ 0.77 -	26.96 $\pm$ 0.15 -	
	ES	66.12 $\pm$ 2.00 -	73.51 $\pm$ 1.10 -	45.58 $\pm$ 1.13 -	84.09 $\pm$ 0.90 -	85.87 $\pm$ 2.03 -	97.19 $\pm$ 1.31 -	51.47 $\pm$ 0.48 -	30.03 $\pm$ 0.37 -	10.38
	PSO	56.65 $\pm$ 1.15 -	66.53 $\pm$ 1.48 -	41.09 $\pm$ 0.73 -	78.00 $\pm$ 3.13 -	79.91 $\pm$ 2.74 -	90.90 $\pm$ 2.04 -	47.96 $\pm$ 0.62 -	26.98 $\pm$ 0.11 -	
	BO	61.49 $\pm$ 1.09 -	72.65 $\pm$ 0.99 -	44.32 $\pm$ 0.23 -	81.64 $\pm$ 0.90 -	85.94 $\pm$ 0.33 -	92.12 $\pm$ 1.52 -	50.81 $\pm$ 0.19 -	29.06 $\pm$ 0.44 -	
HPO	SA	37.51 $\pm$ 0.73 -	42.93 $\pm$ 0.05 -	28.86 $\pm$ 0.28 -	40.54 $\pm$ 0.13 -	54.26 $\pm$ 0.49 -	69.13 $\pm$ 0.27 -	37.10 $\pm$ 0.28 -	22.17 $\pm$ 0.18 -	4.38
	Vanilla-EA	36.91 $\pm$ 0.88 -	42.38 $\pm$ 0.31 -	28.15 $\pm$ 0.24 -	<b>40.03<math>\pm</math>0.27</b> -	53.22 $\pm$ 0.13 -	<u>68.65<math>\pm</math>0.16</u> $\approx$	36.85 $\pm$ 0.22 -	21.90 $\pm$ 0.07 -	
	ES	38.04 $\pm$ 0.14 -	42.57 $\pm$ 0.39 -	28.25 $\pm$ 0.18 -	40.27 $\pm$ 0.28 -	54.55 $\pm$ 0.75 -	68.79 $\pm$ 0.21 -	37.02 $\pm$ 0.16 -	22.00 $\pm$ 0.06 -	3.50
	PSO	37.02 $\pm$ 0.66 -	<b>42.34<math>\pm</math>0.41</b> -	<b>28.05<math>\pm</math>0.09</b> -	40.21 $\pm$ 0.08 -	<b>53.17<math>\pm</math>0.15</b> -	<b>68.54<math>\pm</math>0.21</b> -	<b>36.71<math>\pm</math>0.21</b> -	<b>21.87<math>\pm</math>0.05</b> -	
	BO	<b>36.64<math>\pm</math>0.25</b> -	44.92 $\pm$ 1.07 -	28.29 $\pm$ 0.11 -	40.43 $\pm$ 0.11 -	53.96 $\pm$ 1.06 -	70.01 $\pm$ 0.76 -	37.10 $\pm$ 0.02 -	22.12 $\pm$ 0.02 -	
RL	MaskPlace [42]	73.59 $\pm$ 1.68 -	81.12 $\pm$ 4.57 -	50.59 $\pm$ 3.49 -	91.50 $\pm$ 3.00 -	95.77 $\pm$ 0.43 -	102.89 $\pm$ 4.75 -	55.12 $\pm$ 0.96 -	32.74 $\pm$ 0.90 -	12
Analytical	DREAMPlace [46]	69.92 $\pm$ 2.73 -	88.53 $\pm$ 3.18 -	44.58 $\pm$ 5.21 -	77.28 $\pm$ 5.00 -	107.20 $\pm$ 2.17 -	110.60 $\pm$ 4.54 -	67.58 $\pm$ 4.50 -	25.68 $\pm$ 0.28 -	11.13

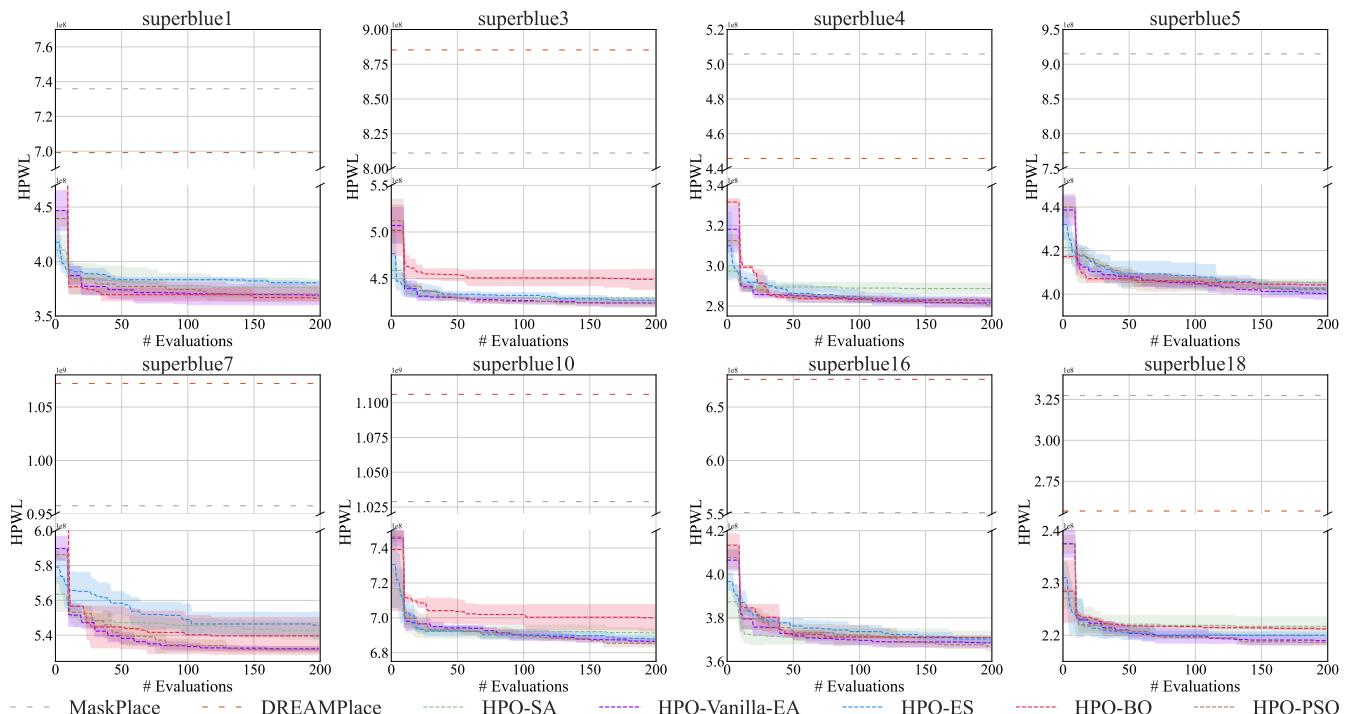


Figure 3: GP-HPWL vs. number of evaluations of different methods on ICCAD 2015.

Table VII: PPA metrics obtained by compared methods under the HPO formulation setup on ICCAD 2015 benchmarks. Among these metrics, rWL (m) is the routed wirelength, rO-H (%) and rO-V (%) represent the routed horizontal and vertical congestion overflow, respectively, WNS (ns) is the worst negative slack, TNS ( $1e5\mu s$ ) is the total negative slack, and NVP (1e4) is the number of violation paths. The best and runner-up methods are **bolded** and underlined, respectively.

Benchmark	Formulation	Algorithm	rWL (m) ↓	rO-H (%) ↓	rO-V (%) ↓	WNS (ns) ↑	TNS ( $1e5\mu s$ ) ↑	NVP (1e4) ↓
superblue1	HPO	SA	75.06	<b>0.00</b>	<b>0.00</b>	-21.23	-0.16	0.60
		Vanilla-EA	<b>74.90</b>	<b>0.00</b>	<b>0.00</b>	-21.82	<b>-0.10</b>	<b>0.50</b>
		ES	77.55	<b>0.00</b>	<b>0.00</b>	<u>-20.35</u>	-0.19	0.73
		PSO	75.23	<u>0.01</u>	<b>0.00</b>	-25.04	<u>-0.11</u>	<b>0.50</b>
		BO	80.33	0.08	<u>0.02</u>	<b>-16.91</b>	-0.16	0.75
superblue3	HPO	SA	88.51	0.29	0.01	<b>-23.77</b>	<b>-0.11</b>	0.35
		Vanilla-EA	<u>86.66</u>	0.13	<u>0.01</u>	<u>-23.85</u>	<b>-0.11</b>	<u>0.35</u>
		ES	87.97	<b>0.04</b>	<b>0.00</b>	-26.70	-0.13	<b>0.34</b>
		PSO	<b>86.41</b>	<u>0.12</u>	0.01	-26.63	<b>-0.11</b>	0.38
		BO	91.24	0.44	<u>0.01</u>	-27.32	-0.17	0.46
superblue4	HPO	SA	58.61	0.04	0.01	-25.42	-0.28	0.59
		Vanilla-EA	<u>57.97</u>	<u>0.03</u>	<b>0.00</b>	<u>-19.17</u>	-0.29	0.61
		ES	58.60	<u>0.03</u>	0.03	-26.96	-0.29	0.60
		PSO	<b>57.51</b>	0.05	<b>0.00</b>	<b>-15.04</b>	<b>-0.26</b>	<b>0.58</b>
		BO	59.23	<b>0.02</b>	<b>0.00</b>	-19.34	-0.30	0.61
superblue5	HPO	SA	82.58	<u>0.11</u>	0.02	-60.39	-0.17	0.45
		Vanilla-EA	<b>81.42</b>	0.15	<b>0.01</b>	-46.08	<b>-0.14</b>	0.51
		ES	82.90	0.26	<u>0.02</u>	<u>-45.90</u>	-0.17	0.57
		PSO	<u>81.43</u>	0.29	<u>0.01</u>	-52.35	<u>-0.16</u>	<b>0.42</b>
		BO	84.85	<b>0.10</b>	<u>0.01</u>	<b>-44.16</b>	-0.18	0.50
superblue7	HPO	SA	116.37	<u>0.01</u>	<b>0.00</b>	-13.81	-0.14	0.81
		Vanilla-EA	<u>113.17</u>	<u>0.01</u>	<b>0.00</b>	-13.88	<b>-0.12</b>	<b>0.71</b>
		ES	115.10	<b>0.00</b>	<b>0.00</b>	-13.26	-0.14	0.81
		PSO	<b>111.84</b>	<b>0.00</b>	<b>0.00</b>	<b>-11.90</b>	<b>-0.12</b>	0.80
		BO	116.68	<u>0.01</u>	<b>0.00</b>	-12.30	-0.14	0.84
superblue10	HPO	SA	140.73	<b>0.01</b>	<b>0.00</b>	-26.93	<u>-0.77</u>	<b>1.12</b>
		Vanilla-EA	<u>139.07</u>	<b>0.01</b>	<b>0.00</b>	<u>-22.70</u>	-0.78	1.20
		ES	140.56	<u>0.02</u>	<b>0.00</b>	<b>-20.28</b>	<b>-0.66</b>	<u>1.15</u>
		PSO	<b>138.67</b>	0.06	<u>0.01</u>	-25.62	-0.91	1.29
		BO	140.35	<b>0.01</b>	<b>0.00</b>	-24.61	-0.88	1.29
superblue16	HPO	SA	74.99	<b>0.03</b>	<b>0.00</b>	<u>-16.08</u>	<b>-0.30</b>	<b>1.33</b>
		Vanilla-EA	74.23	0.21	<u>0.01</u>	-16.37	-0.36	1.52
		ES	74.00	0.16	<u>0.01</u>	-16.14	<u>-0.31</u>	<u>1.44</u>
		PSO	<b>73.52</b>	0.17	<u>0.01</u>	-16.55	-0.32	1.55
		BO	75.02	<u>0.06</u>	<b>0.00</b>	<b>-15.27</b>	<b>-0.30</b>	<u>1.33</u>
superblue18	HPO	SA	46.44	<b>0.02</b>	<b>0.00</b>	<b>-8.46</b>	<b>-0.04</b>	<b>0.21</b>
		Vanilla-EA	<b>45.36</b>	0.07	<u>0.01</u>	<u>-10.58</u>	-0.07	0.32
		ES	45.96	<u>0.05</u>	<u>0.01</u>	-13.73	<u>-0.07</u>	<u>0.28</u>
		PSO	<u>45.57</u>	<u>0.04</u>	<b>0.00</b>	-12.70	-0.15	0.53
		BO	46.44	0.09	<u>0.01</u>	-12.75	-0.09	0.30

macros by running DREAMPlace, and then places the standard cells by running DREAMPlace again to perform GP-HPWL evaluation. MaskPlace performs the worst on the GP-HPWL metric because it only considers information of macros. The performance of BO is worse than that of Vanilla-EA and PSO on most cases, no matter if the MGO or HPO formulation is used. This trend contradicts the common expectation that BO outperforms Vanilla-EA in sample efficiency, which underscores the need for more targeted chip-placement techniques that take advantage of BO's ability to effectively use historical optimization experience.

b) *PPA Comparisons*: Since GP-HPWL is more closely aligned with the final chip performance and the HPO formulation shows superior performance on the ICCAD 2015 dataset, we focus our PPA evaluations exclusively on the solutions derived from the HPO formulation of optimizing GP-HPWL to assess real-world manufacturing metrics. For each method, we

select the best chip placement result from multiple runs based on GP-HPWL for PPA evaluation. The PPA evaluation results of HPO formulation are shown in Table VII, where rWL (m) is the routed wirelength, rO-H (%) and rO-V (%) represent the routed horizontal and vertical congestion overflow, respectively, WNS (ns) is the worst negative slack, TNS ( $1e5\mu s$ ) is the total negative slack, and NVP (1e4) is the number of violation paths. For WNS and TNS, the larger the better, while for the other metrics, the smaller the better. As shown in Table VII, no single BBO algorithm dominates across all performance metrics, which highlights the complexity of optimizing real-world chip performance. Similar to the GP-HPWL results in Table VI, PSO and Vanilla-EA generally yield superior routed wirelength outcomes: PSO performs the best in five cases, while Vanilla-EA does so in three cases. The values of rO-H and rO-V are all very small ( $\leq 0.3\%$  and  $\leq 0.03\%$ , respectively), suggesting that all algorithms keep

Table VIII: Influence analysis of the number of macros: GP-HPWL values ( $\times 10^7$ ) obtained by the two best-performing methods (i.e., Vanilla-EA and PSO) on ICCAD 2015 under the MGO formulation. Each result consists of the mean and standard deviation of five runs. The best and runner-up settings are **bolded** and underlined, respectively.

Algorithm	Number of macros	superblue1	superblue3	superblue4	superblue5	superblue7	superblue10	superblue16	superblue18
MGO-Vanilla-EA	128	<b>52.07</b> $\pm$ 0.37	<b>59.89</b> $\pm$ 0.79	37.81 $\pm$ 0.32	<b>64.08</b> $\pm$ 1.18	<b>70.54</b> $\pm$ 1.72	<b>87.89</b> $\pm$ 1.39	46.08 $\pm$ 0.42	<b>25.72</b> $\pm$ 0.09
	256	53.91 $\pm$ 0.52	61.74 $\pm$ 0.14	<u>39.04</u> $\pm$ 0.33	70.66 $\pm$ 0.33	75.83 $\pm$ 1.58	88.73 $\pm$ 0.52	<b>45.99</b> $\pm$ 0.72	25.96 $\pm$ 0.24
	512	55.35 $\pm$ 1.24	62.62 $\pm$ 1.24	40.24 $\pm$ 0.65	72.15 $\pm$ 1.16	75.79 $\pm$ 1.47	88.69 $\pm$ 1.35	46.71 $\pm$ 0.77	26.96 $\pm$ 0.15
	1024	60.27 $\pm$ 0.68	68.29 $\pm$ 0.48	45.77 $\pm$ 0.27	72.99 $\pm$ 1.00	78.70 $\pm$ 1.06	89.72 $\pm$ 1.26	48.33 $\pm$ 1.28	27.59 $\pm$ 0.20
MGO-PSO	128	<u>52.86</u> $\pm$ 0.17	63.31 $\pm$ 0.59	<b>36.73</b> $\pm$ 0.04	65.57 $\pm$ 4.47	72.29 $\pm$ 1.67	89.76 $\pm$ 1.33	46.87 $\pm$ 1.07	26.93 $\pm$ 0.26
	256	<u>55.58</u> $\pm$ 0.43	66.41 $\pm$ 1.37	38.91 $\pm$ 0.03	<u>77.99</u> $\pm$ 2.84	<u>83.04</u> $\pm$ 1.63	88.56 $\pm$ 3.20	46.19 $\pm$ 0.14	26.76 $\pm$ 0.18
	512	56.65 $\pm$ 1.15	66.53 $\pm$ 1.48	41.09 $\pm$ 0.73	78.00 $\pm$ 3.13	79.91 $\pm$ 2.74	<u>90.90</u> $\pm$ 2.04	47.96 $\pm$ 0.62	26.98 $\pm$ 0.11
	1024	63.84 $\pm$ 0.51	72.93 $\pm$ 2.02	44.73 $\pm$ 0.21	78.88 $\pm$ 0.78	<u>85.17</u> $\pm$ 0.96	88.74 $\pm$ 3.06	48.41 $\pm$ 0.59	27.72 $\pm$ 0.09

congestion relatively low. WNS measures the most timing-critical path in the chip, while TNS measures the overall severity of timing issues. For better analysis, we calculate the Pearson correlation coefficients: The correlation between rWL and WNS is -0.10, and that between rWL and TNS is -0.65. The value of -0.10 implies a weak relationship between wirelength and WNS. For instance, SA and BO, despite having less favorable GP-HPWL results, achieve the best WNS in two and three cases, respectively. As for TNS, Vanilla-EA and PSO have the best results, which are closely related to their excellent performance in rWL. The strong correlation between NVP and TNS (with Pearson correlation coefficient -0.68) indicates that fewer NVP leads to better TNS, which is consistent with previous research [82]. In summary, these results suggest that for chip placement, BBO needs further progress in surrogate modeling and multi-objective optimization to meet PPA requirements.

#### D. Influence with Different Number of Macros

As we mentioned before, an important setting of the MGO formulation is the number of macros, which defines the problem dimension, because a solution under the MGO formulation records the coordinates of all macros directly. We examine this setting on ICCAD 2015 designs, by selecting the two best algorithms (Vanilla-EA and PSO) to assess the influence of the number of macros. As shown in Table VIII, the overall results deteriorate as the number of macros increases, which is because the optimization difficulty increases with the number of macros, while the number of evaluations is kept fixed. When the number of macros is small, the MGO formulation prioritizes the more important macros (i.e., those with larger connected area) under a limited number of evaluation budget, yielding better GP-HPWL values. In certain instances, such as superblue10 and superblue16, adjusting a greater number of macros occasionally yields superior results, which may be because the given evaluation budget is still sufficient for the enlarged search space. Overall, our BBOPlace-Bench offers flexible problem formulation choices, providing researchers with a controllable test-bed for investigating high-dimensional BBO challenges.

#### E. Runtime Analysis

Due to the differences in problem formulation and optimization algorithms, the runtime varies significantly across

different methods. Here, we present the average runtime of each iteration on the chip case adaptec3, as shown in Table IX. The runtime is partitioned into two components: Optimization time is the runtime of algorithm execution, and evaluation time is the runtime of MP-HPWL evaluation or GP-HPWL evaluation. Since the optimization time of each iteration of BO and ES increases with the number of iterations, we calculate the average optimization time of each iteration of all the algorithms after their completion. The per-iteration optimization time of BO ( $\approx 1\text{-}9\text{s}$ ) and ES ( $\approx 0.2\text{--}1.0\text{s}$ ) is one to two orders of magnitude higher than that of Vanilla-EA ( $\approx 0.03\text{--}0.7\text{s}$ ) and SA ( $\approx 0.01\text{--}0.7\text{s}$ ), reflecting the cubic complexity of Gaussian process training and inference in BO and the covariance-matrix adaptation overhead in ES. PSO exhibits the lowest optimization cost ( $<0.04\text{s}$ ) owing to its lightweight velocity updates.

Table IX: Optimization and evaluation time of each iteration of different methods on the chip case adaptec3, where the optimization time is the runtime of algorithm execution and the evaluation time is the runtime of MP-HPWL evaluation or GP-HPWL evaluation.

Evaluation Settings		Optimization Time	Evaluation Time
MP-HPWL	SP	SA Vanilla-EA	0.0271 0.0296
	MGO	SA Vanilla-EA ES PSO BO	0.0514 0.1751 1.0171 0.0021 5.3156
	HPO	SA Vanilla-EA ES PSO BO	0.0182 0.0494 0.3777 0.0118 1.1098
	SP	SA Vanilla-EA	0.1496 0.6337
	MGO	SA Vanilla-EA ES PSO BO	0.7083 0.6855 1.0578 0.0390 2.6328
GP-HPWL	HPO	SA Vanilla-EA ES PSO BO	0.1755 0.4483 0.2507 0.0028 9.1910

For evaluation time, the SP and MGO formulations require significantly less time under MP-HPWL evaluation compared

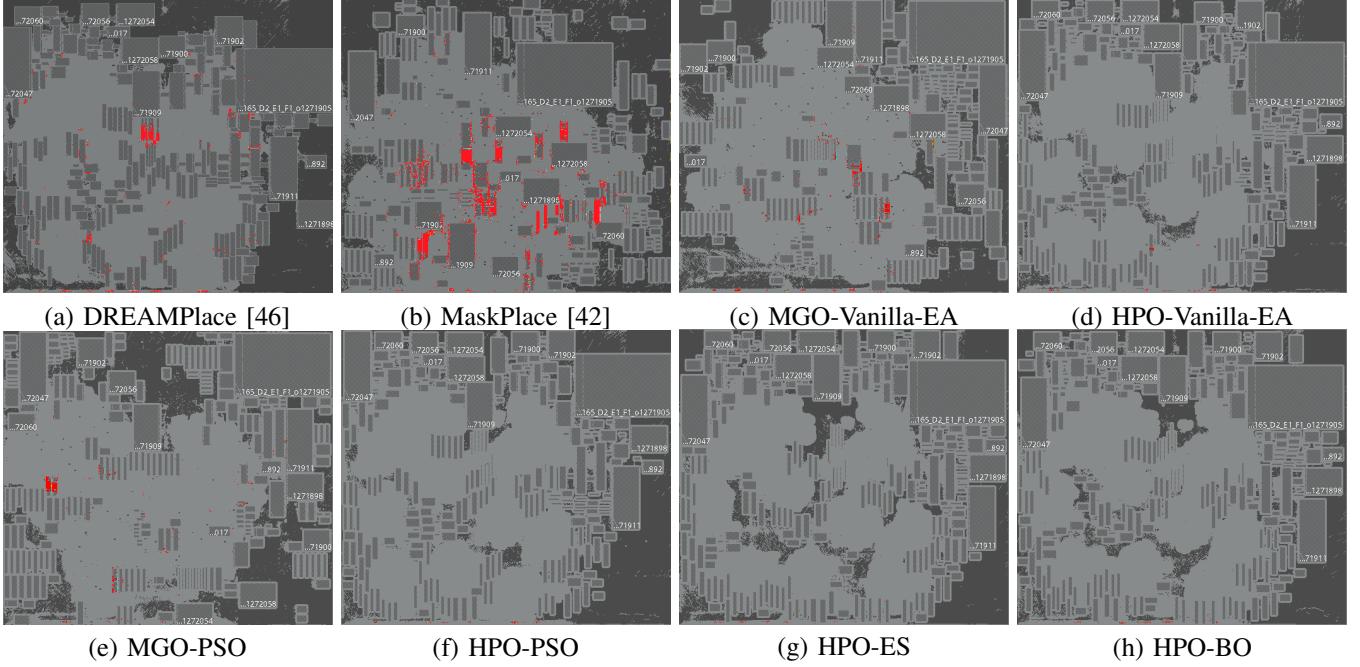


Figure 4: Placement layouts and congestions of different methods on the chip case superblue7 of ICCAD 2015. The congestion results are obtained by the commercial tool *Cadence Innovus*, where the red points indicate the congestion critical regions.

to the HPO formulation, which is due to the significant overhead of DREAMPlace for the placement of all modules in the HPO formulation. The evaluation time of SP and MGO increases considerably under GP-HPWL evaluation, because it necessitates utilizing DREAMPlace to place standard cells following the macro placement stage. Nevertheless, the resulting evaluation time of SP and MGO formulations remain less than that of HPO formulation, since their macro placement steps are inherently faster. BBOPlace-Bench is designed to support flexible problem formulations and evaluation settings, allowing researchers to examine BBO algorithms at different computational costs.

#### F. Visualization Analysis

To provide an intuitive comparison, we present the visualization results of various methods evaluated on ICCAD 2015, as shown in Figure 4. Our proposed BBOPlace-Bench provides a convenient visualization interface that assists users unfamiliar with chip placement in understanding the output results of their BBO algorithms. Compared with the advanced analytical placer DREAMPlace [46] and the RL placer MaskPlace [42], BBO methods equipped with different problem formulations and optimization algorithms demonstrate superior placement results, with fewer critical congestion points. It can also be observed that different problem formulations lead to distinct placement styles, even when the same BBO algorithm is used, such as Figure 4 (c) versus (d).

## V. CONCLUSIONS AND DISCUSSIONS

In this paper, we propose BBOPlace-Bench, which is the first BBO benchmark for chip placement. BBOPlace-Bench decouples problem formulation, optimization algorithm, and

evaluation, offering a flexible framework that allows users to easily implement and test their BBO algorithms, with the hope of facilitating the application of BBO, as well as better solving the significant problem of chip placement. One limitation of this paper is that we used the commercial software *Cadence Innovus* for PPA evaluation. We plan to integrate open-source EDA tools (e.g., OpenROAD [2] and iEDA [43]) into our benchmark to facilitate comprehensive performance evaluation for users without the commercial license. We also plan to incorporate more advanced chip designs, which can broaden the set of available test cases. One interesting future work is to apply fitness landscape analysis [56], [51], [76] to unravel the intrinsic properties of chip placement problems, which can deepen our understanding and thereby guide the development of more targeted BBO algorithms tailored to address the specific challenges of this domain.

Based on our empirical findings, several promising directions emerge for advancing BBO for chip placement, each addressing critical challenges in the field:

- **Multi-objective optimization.** Beyond the wirelength minimization explored in this work, chip design actually requires the simultaneous optimization of several possibly conflicting objectives [11], including routing congestion, dynamic power consumption, timing slack, and area utilization, all of which collectively determine the final chip quality. Applying and developing multi-objective BBO algorithms [18], [89], [84] that can balance these trade-offs while maintaining computational efficiency remains a pressing need for chip placement.
- **Expensive optimization.** Many real-world applications of BBO involve expensive objective function evaluations, with chip design serving as a representative example.

Developing accurate surrogate models [35], [79] for GP-HPWL evaluation or even PPA metrics [90], [24] to reduce reliance on expensive full evaluations is a promising future direction. Advanced algorithms for expensive optimization such as trust region search [21], [80] can be further considered.

- **High-dimensional optimization.** Chip placement inherently involves high-dimensional search spaces (e.g., coordinates and orientations for thousands of macros). Besides applying general dimensionality reduction techniques [8], [61], [69], future work could focus on methods that exploit problem-specific structures to mitigate the high dimensional issue, such as macro groupings and hierarchical dependencies between placement stages.
- **Learning-enhanced optimization.** Leveraging learning approaches to automate BBO algorithms [53], such as optimization by learning from offline datasets [77], [63], automated operator selection and configuration [85], [52], [30], or automated BBO algorithm design [48], [47], [72], could enable robust algorithms that generalize across diverse placement cases.

#### ACKNOWLEDGMENT

We sincerely thank Prof. Miqing Li for reading the manuscript and providing helpful comments. This work was supported by the National Science and Technology Major Project (2022ZD0116600), the National Science Foundation of China (62276124, 624B2069), the Fundamental Research Funds for the Central Universities (14380020), and the Young Elite Scientists Sponsorship Program by CAST for PhD Students. Chao Qian is the corresponding author.

#### REFERENCES

- [1] A. Agnesina, P. Rajvanshi, T. Yang, G. Pradipta, A. Jiao, B. Keller, B. Khailany, and H. Ren, “AutoDMP: Automated DREAMPlace-based macro placement,” in *Proceedings of the 27th International Symposium on Physical Design*, Virtual, 2023, pp. 149–157.
- [2] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Saptnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu, “Toward an open-source digital flow: First learnings from the OpenROAD project,” in *Proceedings of the 56th Annual Design Automation Conference*, Las Vegas, NV, 2019, pp. 1–4.
- [3] S. Alupoaei and S. Katkoori, “Net-based force-directed macrocell placement for wirelength optimization,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10, no. 6, pp. 824–835, 2002.
- [4] S. P. Arango, H. S. Jomaa, M. Wistuba, and J. Grabocka, “HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML,” *arXiv:2106.06257*, 2021.
- [5] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*. Springer, 2017.
- [6] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.
- [7] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, “BoTorch: A framework for efficient Monte-Carlo Bayesian optimization,” in *Advances in Neural Information Processing Systems 33*, Virtual, 2020, pp. 21 524–21 538.
- [8] M. Binois and N. Wycoff, “A survey on high-dimensional Gaussian process modeling with application to Bayesian optimization,” *ACM Transactions on Evolutionary Learning and Optimization*, vol. 2, no. 2, pp. 1–26, 2022.
- [9] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zeilikovsky, “On wirelength estimations for row-based placement,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 9, pp. 1265–1278, 1999.
- [10] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, “CircuitNet: An open-source dataset for machine learning in VLSI CAD applications with improved domain-specific evaluation metric and learning strategies,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 5034–5047, 2023.
- [11] F.-C. Chang, Y.-W. Tseng, Y.-W. Yu, S.-R. Lee, A. Cioba, I.-L. Tseng, D.-s. Shiu, J.-W. Hsu, C.-Y. Wang, C.-Y. Yang *et al.*, “Flexible multiple-objective reinforcement learning for chip placement,” *arXiv:2204.06407*, 2022.
- [12] Y. Chang, Z. Jiang, and T. Chen, “Essential issues in analytical placement algorithms,” *IPSJ Transactions on System LSI Design Methodology*, vol. 2, pp. 145–166, 2009.
- [13] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, “B\*-trees: A new representation for non-slicing floorplans,” in *Proceedings of the 37th Annual Design Automation Conference*, Los Angeles, CA, 2000, pp. 458–463.
- [14] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, “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*, vol. 27, no. 7, pp. 1228–1240, 2008.
- [15] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, “RePIAcE: Advancing solution quality and routability validation in global placement,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2018.
- [16] R. Cheng, X. Lyu, Y. Li, J. Ye, J. Hao, and J. Yan, “The policy-gradient placement and generative routing neural networks for chip design,” in *Advances in Neural Information Processing Systems 35*, New Orleans, LA, 2022, pp. 26 350–26 362.
- [17] R. Cheng and J. Yan, “On joint learning for solving placement and routing in chip design,” in *Advances in Neural Information Processing Systems 34*, Virtual, 2021, pp. 16 508–16 519.
- [18] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.
- [19] Q. Duan, G. Zhou, C. Shao, and Others, “PyPop7: A pure-Python library for population-based black-box optimization,” *Journal of Machine Learning Research*, vol. 25, no. 296, pp. 1–28, 2024.
- [20] D. Eriksson and M. Jankowiak, “High-dimensional bayesian optimization with sparse axis-aligned subspaces,” in *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence*, Virtual Event, 2021, pp. 493–503.
- [21] D. Eriksson, M. Pearce, J. R. Gardner, R. Turner, and M. Poloczek, “Scalable global optimization via local Bayesian optimization,” in *Advances in Neural Information Processing Systems 32*, Vancouver, Canada, 2019, pp. 5497–5508.
- [22] P. I. Frazier, “A tutorial on Bayesian optimization,” *arXiv:1807.02811*, 2018.
- [23] R. Garnett, *Bayesian Optimization*. Cambridge University Press, 2023.
- [24] Z. Geng, J. Wang, Z. Liu, S. Xu, Z. Tang, S. Kai, M. Yuan, J. Hao, and F. Wu, “LaMPlace: Learning to optimize cross-stage metrics in macro placement,” in *Proceedings of the 13th International Conference on Learning Representations*, Singapore, 2025.
- [25] Z. Geng, J. Wang, Z. Liu, S. Xu, Z. Tang, M. Yuan, J. Hao, Y. Zhang, and F. Wu, “Reinforcement learning within tree search for fast macro placement,” in *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 2024, pp. 15 402–15 417.
- [26] A. Goldie, A. Mirhoseini, and J. Dean, “That chip has sailed: A critique of unfounded skepticism around AI for chip design,” *arxiv:2411.10053*, 2024.
- [27] A. Goldie, A. Mirhoseini, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nova *et al.*, “Addendum: A graph placement methodology for fast chip design,” *Nature*, vol. 634, no. 8034, pp. E10–E11, 2024.
- [28] Y.-J. Gong, J.-J. Li, Y. Zhou, Y. Li, H. S.-H. Chung, Y.-H. Shi, and J. Zhang, “Genetic learning particle swarm optimization,” *IEEE Transactions on Cybernetics*, vol. 46, no. 10, pp. 2277–2290, 2015.
- [29] J. Gu, Z. Jiang, Y. Lin, and D. Z. Pan, “DREAMPlace 3.0: Multi-electrostatics based robust VLSI placement with region constraints,” in *Proceedings of the 33rd IEEE/ACM International Conference on Computer-Aided Design*, San Diego, CA, 2020, pp. 1–9.
- [30] H. Guo, Z. Ma, J. Chen, Y. Ma, Z. Cao, X. Zhang, and Y.-J. Gong, “ConfigX: Modular configuration for evolutionary algorithms via multitask

- reinforcement learning,” in *Proceedings of the 39th AAAI Conference on Artificial Intelligence*, Philadelphia, PA, 2025, pp. 26 982–26 990.
- [31] N. Hansen, “The CMA evolution strategy: A tutorial,” *arXiv:1604.00772*, 2016.
- [32] N. Hansen, D. V. Arnold, and A. Auger, “Evolution strategies,” in *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 871–898.
- [33] X. He, T. Huang, L. Xiao, H. Tian, and E. F. Y. Young, “Ripple: A robust and effective routability-driven placer,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1546–1556, 2013.
- [34] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu, “Corner block list: An effective and efficient topological representation of non-slicing floorplan,” in *Proceedings of the 13th IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, 2000, pp. 8–12.
- [35] Y. Jin, H. Wang, T. Chugh, D. Guo, and K. Miettinen, “Data-driven evolutionary optimization: An overview and case studies,” *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 442–458, 2019.
- [36] D. R. Jones, M. Schonlau, and W. J. Welch, “Efficient global optimization of expensive black-box functions,” *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.
- [37] A. B. Kahng and S. Reda, “A tale of two nets: Studies of wirelength progression in physical design,” in *Proceedings of the 7th International Workshop on System-level Interconnect Prediction*, Munich, Germany, 2006, pp. 17–24.
- [38] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of 3rd International Conference on Neural Networks*, Perth, Australia, 1995, pp. 1942–1948.
- [39] M. Kim, J. Hu, J. Li, and N. Viswanathan, “ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite,” in *Proceedings of the 28th IEEE/ACM International Conference on Computer-Aided Design*, Austin, TX, 2015, pp. 921–926.
- [40] H. J. Kushner, “A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise,” *Journal of Basic Engineering*, vol. 86, no. 1, pp. 97–106, 1964.
- [41] Y. Lai, J. Liu, Z. Tang, B. Wang, J. Hao, and P. Luo, “ChiPFormer: Transferable chip placement via offline decision transformer,” in *Proceedings of the 40th International Conference on Machine Learning*, Honolulu, HI, 2023, pp. 18 346–18 364.
- [42] Y. Lai, Y. Mu, and P. Luo, “MaskPlace: Fast chip placement via reinforced visual representation learning,” in *Advances in Neural Information Processing Systems 35*, New Orleans, LA, 2022, pp. 24 019–24 030.
- [43] X. Li, Z. Huang, S. Tao, Z. Huang, C. Zhuang, H. Wang, Y. Li, Y. Qiu, G. Luo, H. Li, H. Shen, M. Chen, D. Bu, W. Zhu, Y. Cai, X. Xiong, Y. Jiang, Y. Heng, P. Zhang, B. Yu, B. Xie, and Y. Bao, “iEDA: An open-source infrastructure of EDA,” in *Proceedings of the 29th Asia and South Pacific Design Automation Conference*, Incheon, Korea, 2024, pp. 77–82.
- [44] P. Liao, D. Guo, Z. Guo, S. Liu, Y. Lin, and B. Yu, “DREAMPlace 4.0: Timing-driven placement with momentum-based net weighting and Lagrangian-based refinement,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 10, pp. 3374–3387, 2023.
- [45] T. Lin, C. C. N. Chu, and G. Wu, “POLAR 3.0: An ultrafast global placement engine,” in *Proceedings of the 28th IEEE/ACM International Conference on Computer-Aided Design*, Austin, TX, 2015, pp. 520–527.
- [46] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, “DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 748–761, 2020.
- [47] F. Liu, T. Xialiang, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang, “Evolution of heuristics: Towards efficient automatic algorithm design using large language model,” in *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 2024, pp. 32 201–32 223.
- [48] F. Liu, Y. Yao, P. Guo, Z. Yang, Z. Zhao, X. Lin, X. Tong, M. Yuan, Z. Lu, Z. Wang *et al.*, “A systematic survey on large language models for algorithm design,” *arXiv:2410.14716*, 2024.
- [49] L. Liu, B. Fu, S. Lin, J. Liu, E. F. Y. Young, and M. D. F. Wong, “Xplace: An extremely fast and extensible placement framework,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 6, pp. 1872–1885, 2024.
- [50] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, “ePlace: Electrostatics-based placement using fast Fourier transform and Nesterov’s method,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 20, no. 2, pp. 1–34, 2015.
- [51] Z. Ma, J. Chen, H. Guo, and Y.-J. Gong, “Neural exploratory landscape analysis for meta-black-box-optimization,” in *Proceedings of the 13th International Conference on Learning Representations*, Singapore, 2025.
- [52] Z. Ma, H. Guo, J. Chen, Z. Li, G. Peng, Y.-J. Gong, Y. Ma, and Z. Cao, “MetaBox: A benchmark platform for meta-black-box optimization with reinforcement learning,” in *Advances in Neural Information Processing Systems 36*, New Orleans, LA, 2023, pp. 10 775–10 795.
- [53] Z. Ma, H. Guo, Y.-J. Gong, J. Zhang, and K. C. Tan, “Toward automated algorithm design: A survey and practical guide to meta-black-box-optimization,” *IEEE Transactions on Evolutionary Computation*, 2025.
- [54] D. MacMillen, R. Camposano, D. Hill, and T. W. Williams, “An industrial view of electronic design automation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1428–1448, 2000.
- [55] I. L. Markov, J. Hu, and M.-C. Kim, “Progress and challenges in VLSI placement research,” in *Proceedings of the 25th IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 2012, pp. 275–282.
- [56] O. Mersmann, B. Bischi, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, “Exploratory landscape analysis,” in *Proceedings of the 13th Genetic and Evolutionary Computation Conference*, Dublin, Ireland, 2011, pp. 829–836.
- [57] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, “A graph placement methodology for fast chip design,” *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.
- [58] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “VLSI module placement based on rectangle-packing by the sequence-pair,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518–1524, 1996.
- [59] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, “The ISPD 2005 placement contest and benchmark suite,” in *Proceedings of the 9th International Symposium on Physical Design*, San Francisco, CA, 2005, pp. 216–220.
- [60] C. Oh, R. Bondesan, D. Kianfar, R. Ahmed, R. Khurana, P. Agarwal, R. Lepert, M. Sriram, and M. Welling, “Bayesian optimization for macro placement,” *arXiv:2207.08398*, 2022.
- [61] M. N. Omidvar, X. Li, and X. Yao, “A review of population-based metaheuristics for large-scale black-box global optimization—Part I,” *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 5, pp. 802–822, 2022.
- [62] Y. Pu, T. Chen, Z. He, J. Qin, C. Bai, H. Zheng, Y. Lin, and B. Yu, “IncreMacro: Incremental macro placement refinement,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 8, pp. 3222–3235, 2025.
- [63] H. Qian, Y. Zhu, X. Shu, X. An, Y. Wen, S. Liu, H. Lu, A. Zhou, K. Tang, and Y. Yu, “SOO-Bench: Benchmarks for evaluating the stability of offline black-box optimization,” in *Proceedings of the 13th International Conference on Learning Representations*, Singapore, 2025.
- [64] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
- [65] G. Rudolph, “Finite Markov chain results in evolutionary computation: A tour d’horizon,” *Fundamenta Informaticae*, vol. 35, no. 1–4, pp. 67–89, 1998.
- [66] K. Shahooor and P. Mazumder, “VLSI cell placement techniques,” *ACM Computing Surveys*, vol. 23, no. 2, pp. 143–220, 1991.
- [67] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, “Taking the human out of the loop: A review of Bayesian optimization,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.
- [68] Y. Shi, K. Xue, L. Song, and C. Qian, “Macro placement by wire-mask-guided black-box optimization,” in *Advances in Neural Information Processing Systems 36*, New Orleans, LA, 2023, pp. 6825–6843.
- [69] L. Song, K. Xue, X. Huang, and C. Qian, “Monte Carlo tree search based variable selection for high dimensional Bayesian optimization,” in *Advances in Neural Information Processing Systems 35*, New Orleans, LA, 2022.
- [70] P. Spindler and F. M. Johannes, “Fast and accurate routing demand estimation for efficient routability-driven placement,” in *Proceedings of the 14th Conference on Design, Automation & Test in Europe*, Nice, France, 2007, pp. 1–6.
- [71] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, “Information-theoretic regret bounds for Gaussian process optimization in the bandit setting,” *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 3250–3265, 2012.

- [72] N. v. Stein and T. Bäck, "LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics," *IEEE Transactions on Evolutionary Computation*, vol. 29, no. 2, pp. 331–345, 2025.
- [73] H. H. Szu and R. L. Hartley, "Nonconvex optimization by fast simulated annealing," *Proceedings of IEEE*, vol. 75, no. 11, pp. 1538–1540, 1987.
- [74] M. Tang and X. Yao, "A memetic algorithm for VLSI floorplanning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 37, no. 1, pp. 62–69, 2007.
- [75] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proceedings of the 35th IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, 2012, pp. 5026–5033.
- [76] H. Tong, M. Li, J. Liu, and X. Yao, "How do dynamic events change the fitness landscape of traveling salesman problems?" *IEEE Transactions on Evolutionary Computation*, vol. 29, no. 5, pp. 1463–1474, 2025.
- [77] B. Trabucco, X. Geng, A. Kumar, and S. Levine, "Design-Bench: Benchmarks for data-driven offline model-based optimization," in *Proceedings of the 39th International Conference on Machine Learning*, Baltimore, MD, 2022, pp. 21658–21676.
- [78] D. Vashisht, H. Rampal, H. Liao, Y. Lu, D. Shanbhag, E. Fallon, and L. B. Kara, "Placement in integrated circuits using cyclic reinforcement learning and simulated annealing," *arXiv:2011.07577*, 2020.
- [79] H. Wang, L. Jiao, and X. Yao, "Two\_Arch2: An improved two-archive algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 524–541, 2015.
- [80] L. Wang, R. Fonseca, and Y. Tian, "Learning search space partition for black-box optimization using Monte Carlo tree search," in *Advances in Neural Information Processing Systems 33*, Virtual, 2020, pp. 19511–19522.
- [81] Z. Wang, Z. Geng, Z. Tu, J. Wang, Y. Qian, Z. Xu, Z. Liu, S. Xu, Z. Tang, S. Kai *et al.*, "Benchmarking end-to-end performance of AI-based chip placement algorithms," in *Advances in Neural Information Processing Systems 38*, San Diego, CA, 2025.
- [82] K. Xue, R.-T. Chen, X. Lin, Y. Shi, S. Kai, S. Xu, and C. Qian, "Reinforcement learning policy as macro regulator rather than macro placer," in *Advances in Neural Information Processing Systems 38*, Vancouver, Canada, 2024, pp. 140565–140588.
- [83] K. Xue, X. Lin, Y. Shi, S. Kai, S. Xu, and C. Qian, "Escaping local optima in global placement," *arXiv:2402.18311*, 2024.
- [84] K. Xue, R.-X. Tan, X. Huang, and C. Qian, "Offline multi-objective optimization," in *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 2024, pp. 55595–55624.
- [85] K. Xue, J. Xu, L. Yuan, M. Li, C. Qian, Z. Zhang, and Y. Yu, "Multi-agent dynamic algorithm configuration," in *Advances in Neural Information Processing Systems 35*, New Orleans, LA, 2022, pp. 20147–20161.
- [86] J. Xun, Z. Chai, Y. Zhao, Y. Lin, R. Wang, and R. Huang, "CircuitNet 2.0: An advanced dataset for promoting machine learning innovations in realistic chip design environment," in *Proceedings of the 12th International Conference on Learning Representations*, Vienna, Austria, 2024.
- [87] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, CA, 2019, pp. 7105–7114.
- [88] Y. Yu, H. Qian, and Y.-Q. Hu, *Derivative-Free Optimization: Theoretical Foundations, Algorithms, and Applications*. Springer, 2025.
- [89] X. Zhang, R. Cheng, Y. Tian, and Y. Jin, *Evolutionary Large-Scale Multi-Objective Optimization and Applications*. John Wiley & Sons, 2024.
- [90] Z. Zheng, S. Huang, J. Zhong, Z. Shi, G. Dai, N. Xu, and Q. Xu, "DeepGate4: Efficient and effective representation learning for circuit design at scale," *arxiv:2502.01681*, 2025.
- [91] Z.-H. Zhou, Y. Yu, and C. Qian, *Evolutionary Learning: Advances in Theories and Algorithms*. Springer, 2019.