

Toward Automated Algorithm Design: A Survey and Practical Guide to Meta-Black-Box-Optimization

Zeyuan Ma , Hongshu Guo , Yue-Jiao Gong , *Senior Member, IEEE*

Jun Zhang , *Fellow, IEEE* and Kay Chen TAN , *Fellow, IEEE*

Abstract—In this survey, we introduce Meta-Black-Box-Optimization (MetaBBO) as an emerging avenue within the Evolutionary Computation (EC) community, which incorporates Meta-learning approaches to assist automated algorithm design. Despite the success of MetaBBO, the current literature provides insufficient summaries of its key aspects and lacks practical guidance for implementation. To bridge this gap, we offer a comprehensive review of recent advances in MetaBBO, providing an in-depth examination of its key developments. We begin with a unified definition of the MetaBBO paradigm, followed by a systematic taxonomy of various algorithm design tasks, including algorithm selection, algorithm configuration, solution manipulation, and algorithm generation. Further, we conceptually summarize different learning methodologies behind current MetaBBO works, including reinforcement learning, supervised learning, neuroevolution, and in-context learning with Large Language Models. A comprehensive evaluation of the latest representative MetaBBO methods is then carried out, alongside an experimental analysis of their optimization performance, computational efficiency, and generalization ability. Based on the evaluation results, we meticulously identify a set of core designs that enhance the generalization and learning effectiveness of MetaBBO. Finally, we outline the vision for the field by providing insight into the latest trends and potential future directions. Relevant literature will be continuously collected and updated at <https://github.com/MetaEvo/Awesome-MetaBBO>.

Index Terms—Meta-Black-Box-Optimization, Evolutionary Computation, Black-Box-Optimization, Learning to Optimize.

I. INTRODUCTION

Optimization techniques have been central to research for decades [1], [2], with methods applied across engineering [3], economics [4], and science [5]. The optimization problems can be classified into White-Box [6] and Black-Box [7] types. White-Box problems, with transparent structures, allow efficient optimization using gradient-based algorithms like SGD [8], Adam [9], and BFGS [10]. In contrast, Black-Box Optimization (BBO) only provides objective values for solutions, making the analysis and search of the problem space even more challenging.

Zeyuan Ma, Hongshu Guo, and Yue-Jiao Gong are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (E-mail: gongyuejiao@gmail.com)

Jun Zhang is with Nankai University, Tianjin, China and Hanyang University, Seoul, South Korea. (E-mail: junzhang@ieee.org).

Kay Chen TAN is with Hong Kong Polytechnic University, Hong Kong, China. (E-mail: kaychen.tan@polyu.edu.hk).

Corresponding author: Yue-Jiao Gong

Evolutionary Computation (EC), including Evolutionary Algorithms (EAs) and Swarm Intelligence (SI), is widely recognized as an effective gradient-free approach for solving BBO problems [11]. Over the past decades, EC methods have been extensively applied to various optimization challenges [12]–[16], due to their simplicity and versatility. Though effective for solving BBO problems, traditional EC is constrained by the no-free-lunch theorem [17], which asserts that no optimization algorithm can universally outperform others across all problem types, leading to performance trade-offs depending on the problem's characteristics. In response, various human-crafted methods have been developed including offline hyper-parameter optimization [18]–[20], hyper-heuristics [21]–[23] and (self-)adaptive EC variants [24]–[31]. However, they face several limitations. 1) Limited generalization: these methods often focus on a specific set of problems, limiting their generalization due to customized designs. 2) Labor-intensive: designing adaptive mechanisms requires both deep knowledge of EC domain and the target optimization problem, making it a complex task. 3) Additional parameters: many adaptive mechanisms introduce extra hyper-parameters, which can significantly impact performance. 4) Sub-optimal performance: despite increased efforts, design biases and delays in reactive adjustments often lead to sub-optimal outcomes.

Given this, a natural question arises: can we automatically design effective BBO algorithms while minimizing the dependence on expert input? A recently emerging research topic, known as Meta-Black-Box-Optimization (MetaBBO) [32], has shown possibility of leveraging the generalization strength of Meta-learning [33] to enhance the optimization performance of BBO algorithms in the minimal expertise cost. MetaBBO follows a bi-level paradigm: the meta level typically maintains a policy that takes the low-level optimization information as input and then automatically dictates desired algorithm design for the low-level BBO optimizer. The low-level BBO process evaluates the suggested algorithm design and returns a feedback signal to the meta-level policy regarding the performance gain. The meta-objective of MetaBBO is to meta-learn a policy that maximizes the performance of the low-level BBO process, over a problem distribution. Once the training completes, the learned meta-level policy can be directly applied to address unseen optimization problems, hence reducing the need for expert knowledge to adapt BBO algorithms.

Numerous valuable ideas have been proposed and discussed in existing MetaBBO research. From the perspective

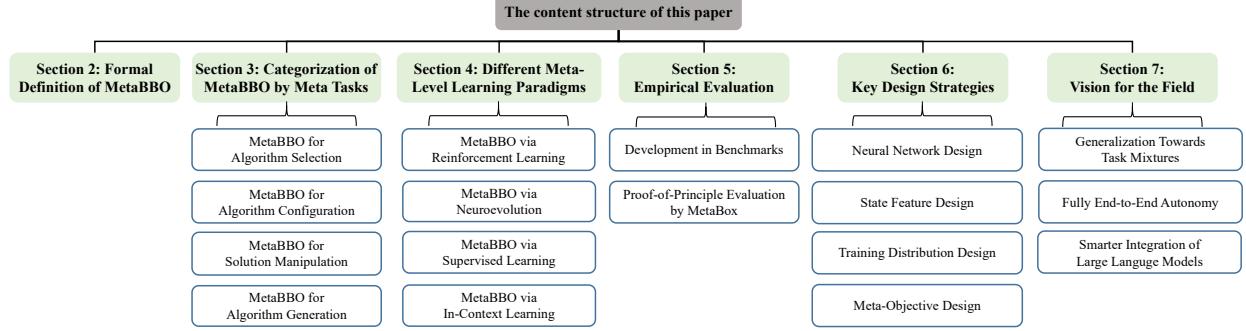


Fig. 1. Roadmap of the content structure, beginning with a concept introduction, followed by a review of existing methods across different taxonomies, a evaluation of selected methods, and a summary of key design strategies and future vision.

of algorithm design tasks (meta tasks) that the meta-level policy can address, those MetaBBO works can be categorized into four branches: 1) Algorithm Selection, where for solving the given problem, a proper BBO algorithm is selected by the meta-level policy from a pre-collected optimizer/operator pool. 2) Algorithm Configuration, where the hyper-parameters and/or operators of a BBO algorithm are adjusted by the meta-level policy to adapt for the given problem. 3) Solution Manipulation, where the meta-level policy is trained to act as a BBO algorithm to manipulate and evolve solutions. 4) Algorithm Generation, where each algorithmic component and the overall workflow are generated by the meta-level policy as a novel BBO algorithm. From the perspective of learning paradigms adopted for training the meta-level policy, different learning methods such as reinforcement learning (MetaBBO-RL) [34]–[40], auto-regressive supervised learning (MetaBBO-SL) [41]–[46], neuroevolution (MetaBBO-NE) [47]–[49], and Large Language Models (LLMs)-based in-context learning (MetaBBO-ICL) [43], [50]–[53] have been investigated in existing works. From the perspective of low-level BBO process, MetaBBO has been instantiated to various optimization scenarios such as single-objective optimization [37], [38], [40], multi-objective optimization [54], [55], multi-modal optimization [56], large scale global optimization [46], [48], [49], and multi-task optimization [57], [58]. Such an intricate combination of algorithm design tasks, learning paradigms, and low-level BBO scenarios makes it challenging for new practitioners to systematically learn, use, and develop MetaBBO methods. Unfortunately, there is still a lack of a comprehensive survey and practical guide to the advancements in MetaBBO.

While some related surveys discussed the integration of learning systems into EC algorithm designs, they have several limitations: 1) Previous surveys [59]–[61] focus on one or two algorithm design tasks, such as algorithm configuration [61] and algorithm generation [59], [60]. These surveys therefore show short in providing comprehensive review and comparison analysis on all four design tasks. 2) Some surveys [62]–[65] focus on a particular learning paradigm - RL [66]. However, in MetaBBO, various learning paradigms can be adopted, each with distinct characteristics. 3) In addition to reviewing relevant papers, existing surveys lack a practical guide that provides a comprehensive experimental evaluation

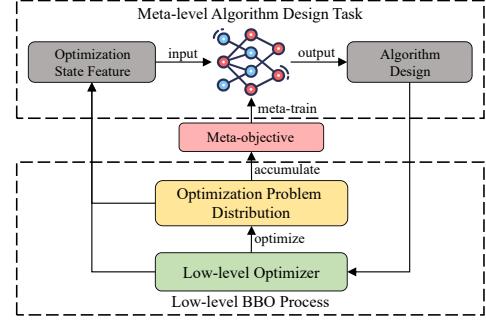


Fig. 2. A conceptual overview of the bi-level learning framework of MetaBBO, illustrating the interactions between its core components to clarify the overall workflow.

of MetaBBO methods and a summary of key design strategies, falling short in offering in-depth evaluations or actionable insights for implementing MetaBBO methods.

To address the gaps in previous surveys, this paper provides a more comprehensive coverage of the MetaBBO field. Fig. 1 offers a roadmap to help readers quickly navigate the overall content structure. We first provide a formal definition of MetaBBO in Section II. Subsequently, we identify four main algorithm design tasks in existing MetaBBO works and their working scenarios in Section III. In Section IV, we further elaborate four learning paradigms, with easy-to-follow technical details. Section V provides a proof-of-principle performance evaluation on ten representative MetaBBO methods. According to the evaluation results, Section VI provides in-depth discussion about the key design strategies in MetaBBO. Finally, we outline the vision for the MetaBBO field in Section VII. The contributions of this survey are generally summarized as follows:

- The first comprehensive survey that sorts out existing literature on MetaBBO. We provide a clear categorization of existing MetaBBO works according to four distinct meta-level tasks, along with a detailed elaboration of four different learning paradigms behind.
- A proof-of-principle evaluation is conducted to provide practical comparison between MetaBBO works, leading to an in-depth discussion over several key design strategies related to the learning effectiveness, training efficiency, and generalization.

- In the end of this paper, we mark several interesting and promising future research directions of MetaBBO, focusing different aspects such as the generalization potential, the end-to-end workflow, and the integration of LLMs.

II. DEFINITION OF METABBO

Meta-Black-Box-Optimization (MetaBBO) [32] is derived from the Meta-learning paradigm [33], [67]. One of the roots of meta-learning could date back to 1987, where Schmidhuber [68] used Genetic Programming (GP) as a learning method to learn better GP program in a self-referential way. Meta-learning is applicable to the learning of both models and algorithms [44]. For example, in [69], a prior knowledge-assisted learning method was proposed to learn synaptic learning rule that is adaptable for diverse downstream learning tasks. In [70], a novel Recurrent Neural Network (RNN) model architecture is meta-learned and subsequently used as a classifier. In contrast, in [71], a RNN model is meta-learned to serve as a gradient descent algorithm for optimizing other neural networks. The paradigm in [71] quickly becomes popular and the following works explore the possibility of such a paradigm in various white-box optimization scenarios ranging from first-order optimization [71] to combinatorial optimization [72]. To make a distinction with other applications of Learning to Learn, this research line is named by Learning to Optimize (L2O). MetaBBO draws key inspiration from the L2O, while targeting black-box optimization scenarios. In this section, we provide an overview of the abstract workflow shared by existing MetaBBO methods, explaining the motivation of the core components in MetaBBO. MetaBBO operates within a bi-level framework, as depicted in Fig. 2, and is detailed as follows.

We begin with the low-level BBO process. A key component at this level is the low-level optimizer \mathcal{A} . \mathcal{A} represents a flexible concept, capable of being any off-the-shelf EC algorithm, its modern variants, an algorithm pool, or a structure for creating new algorithms (rather than a specific existing one). Another crucial element is the optimization problem distribution \mathcal{P} , representing a collection of optimization problem instances to be solved. Although the size of \mathcal{P} could theoretically be infinite, facilitating Meta-learning on an infinite problem set is impossible. In practice, we instead sample a collection of N instances $\{f_1, f_2, \dots, f_N\}$ from \mathcal{P} as the training set. A meta task \mathcal{T} aims to automatically dictate an algorithm design $\omega \in \Omega$ for the low-level optimizer \mathcal{A} for each problem instance in \mathcal{P} , where Ω denotes the algorithm design space of \mathcal{A} . For instance, in a basic DE optimizer [73], its algorithm configuration (e.g., values of the two hyperparameters F and Cr that control the mutation and crossover strength) can be regarded as an algorithm design space Ω . There are various algorithm design spaces, which are discussed in detail in Section III. MetaBBO solves the meta task by learning a meta-level policy π_θ for the algorithm decision.

Formally, for a meta-level algorithm design task $\mathcal{T} :=$

$\{\mathcal{P}, \mathcal{A}, \Omega\}$, π_θ is trained to maximize the meta-objective $J(\theta)$:

$$J(\theta) = \mathbb{E}_{f \in \mathcal{P}} [\mathbf{R}(\mathcal{A}, \pi_\theta, f)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \text{perf}(\mathcal{A}, \omega_i^t, f_i) \\ \omega_i^t = \pi_\theta(s_i^t), \quad s_i^t = \text{sf}(\mathcal{A}, f_i, t) \quad (1)$$

where $\text{sf}(\cdot)$ is a state feature extraction function, which captures the optimization state information from the interplay between the optimizer \mathcal{A} and the problem instance f_i . The meta-level policy π_θ is parameterized by learnable parameters θ . It receives s_i^t as input and outputs an algorithm design ω_i^t , which is then adopted by \mathcal{A} to optimize f_i . A performance measurement function $\text{perf}(\cdot)$ is used to evaluate the performance gain obtained by this algorithm design decision. $\mathbf{R}(\cdot)$ is accumulated performance gain during the low-level optimization of a problem instance. We approximate the meta-objective $J(\theta)$ as the average performance gain across a group of N problem instances sampled from \mathcal{P} , over a certain number T of optimization steps. To summarize, MetaBBO aims to search for an optimal meta-level policy π_{θ^*} which maximizes the meta-objective $J(\theta)$.

MetaBBO vs. Hyper-Heuristics (HH): It is worthy to note that MetaBBO closely aligns with the Hyper-Heuristics (HH) paradigm introduced by Peter et al. [21] through their shared a bi-level framework for AAD tasks. Below we present two key distinctions to clarify the unique position of MetaBBO: 1) Problem Scope and Task Innovation: HH aims to select/generate heuristics within combinatorial optimization (COPs) [74], [75], while MetaBBO exclusively targets automating ‘BBO’ algorithm design. Because of its specialized focus, MetaBBO actively samples and models problem landscapes to learn latent optimization dynamics, thereby enabling novel AAD tasks like dynamic algorithm configuration and generation, which remain largely unexplored in HH research [76] due to its rigid heuristic-driven framework. 2) Learning Flexibility: Consider the meta-level method, HH primarily relies on meta-heuristics, while MetaBBO integrates diverse ML approaches (reinforcement learning, neuroevolution, supervised learning and in-context learning with LLMs). MetaBBO’s neural network-based policies enable *online adaptation* across problem classes, while HH typically operates *offline* within a fixed domain. Specifically, the neural controller of MetaBBO actively interrogates problem characteristics to synthesize algorithm specifically for the current problem landscape, which enables on-instance adjustment as well as cross-problem generalization.

III. CATEGORIZATION OF METABBO BY META TASKS

We introduce four common meta-level tasks in MetaBBO: Algorithm Selection in Section III-A, Algorithm Configuration in Section III-B, Solution Manipulation in Section III-C and Algorithm Generation in Section III-D. Generally speaking, they are organized by the size of design space, from smallest to largest. Algorithm selection deals with a small space, choosing from a few BBO optimizers, while algorithm generation explores a vast space, allowing the meta-level policy to create novel BBO optimizers in an open-ended manner. Table I presents a selection of works categorized by the meta

TABLE I

REPRESENTATIVE WORKS IN METABBO, CATEGORIZED BY DIFFERENT ALGORITHM DESIGN TASKS. WE HAVE PROVIDED AN ONLINE PAGE, WHERE MORE DETAILS OF EXISTING METABBO WORKS ARE INCLUDED.

	Algorithm	Year	Low-level Optimizer	Optimization Type	Technical Summary
Algorithm Selection	Meta-QAP [77]	2008	MMAS	CO	per-instance algorithm selection by MLP classifier for Quadratic Assignment Problem (QAP)
	Meta-TSP [78]	2011	GA	CO	per-instance algorithm selection by MLP classifier for Travelling Salesman Problem (TSP)
	Meta-MOP [79]	2019	MOEA	MOOP	per-instance algorithm selection by SVM classifier from ten multi-objective optimizers
	Meta-VRP [80]	2019	MOEA	CO	per-instance algorithm selection by MLP classifier from four multi-objective optimizers
	AR-BB [81]	2020	EAs, SI	SOP	per-instance algorithm selection by symbolic problem representation and LSTM autoregressive prediction
	ASF-ALLFV [82]	2022	EAs, SI	SOP	per-instance algorithm selection by adaptive local landscape feature and KNN classifier
	AS-LLM [83]	2024	-	SOP	per-instance algorithm selection leverage embedding layer in LLMs
	HHRL-MAR [84]	2024	SI	SOP	dynamically switch SI optimizers along the optimization process with a Q-table RL agent
	R2-RLMOEA [54]	2024	EAs	MOOP	dynamically switch 5 EA optimizers along the optimization process with an MLP RL agent
	RL-DAS [40]	2024	DE	SOP	dynamically switch 3 DE optimizers along the optimization process with an MLP RL agent
Algorithm Configuration	TransOptAS [85]	2024	EAs, SI	SOP	per-instance algorithm selection by Transformer performance predictor from single-objective optimizers
	RLMPSO [86]	2016	PSO	SOP	dynamically select PSO update rules
	RL-MOEAD [87]	2018	MOEA/D	MOOP	dynamically control the neighborhood size and the mutation operators used in MOEA/D
	QL-(S)IM-OPSO [88]	2019	PSO	SOP,MOOP	dynamically control the parameters of PSO update rule
	DE-DDQN [34]	2019	DE	SOP	mutation operator selection in DE
	DE-RLFR [89]	2019	DE	MMOOP	mutation operator selection in DE for multi-modal multi-objective problems
	LTO [90]	2020	CMA-ES	SOP	dynamically configure the mutation step-size in CMA-ES
	QLPSO [36]	2020	PSO	SOP	dynamically control the inter-particle communication topology of PSO
	LRMODE [91]	2020	DE	MOOP	incorporate landscape analysis to operator selection
	RLDE [92]	2021	DE	SOP	dynamically adjust the scaling factor F in DE
	LDE [37]	2021	DE	SOP	use LSTM to adaptively control F and CR in DE
	RLEPSO [93]	2021	PSO	SOP	dynamically adjust factors in EPSO
	qIDE [94]	2021	DE	SOP	dynamically determine parameter combinations of F and Cr in DE
	DE-DQN [39]	2021	DE	SOP	mutation operator selection
	RL-PSO [95]	2022	PSO	SOP	dynamically adjust random values in PSO update rule
	RLLPSO [96]	2022	PSO	LSOP	adaptively adjust the number of performance levels in the population.
	MADAC [97]	2022	MOEA/D	MOOP	dynamically adjust all parameters in MOEA/D by an multi-agent system
	RL-CORCO [98]	2022	DE	COP	operator selection in constrained problems
	MOEA/D-DQN [99]	2022	MOEA/D	MOOP	leverage DQN to select variation operators in MOEA
Solution Manipulation	RL-SHADE [100]	2022	DE	SOP	perform mutation operator selection in SHADE
	RL-HPSDE [35]	2022	DE	SOP	control parameter sampling method and mutation operator selection
	NRLPSO [101]	2023	PSO	SOP	dynamically adjust learning paradigms and acceleration coefficients
	Q-LSHADE [102]	2023	DE	SOP	dynamically control when to use the scheme to reduce the population.
	LADE [103]	2023	DE	SOP	leverage three LSTM models to generate three sampling distributions of key parameters in DE
	LES [48]	2023	CMA-ES	SOP	use self-attention mechanism to adjust the step size in CMA-ES
	RLAM [104]	2023	PSO	SOP	enhance the PSO convergence by using RL to control the coefficients of the PSO
	MPSORL [105]	2023	PSO	SOP	adaptively select strategy in multi-strategy PSO
	RLDMDE [106]	2023	DE	SOP	adaptively select mutation strategy of each population in multi-population DE
	RLMMDE [107]	2023	MOEA	MOOP	dynamically determine whether to perform reference point adaptation method
	MARLABC [108]	2023	ABC	SOP	dynamically select optimization strategy
	CEDE-DRL [109]	2023	DE	COP	dynamically select suitable parent population
	AMODE-DRL [110]	2023	MODE	MOOP	two RL agents, one for mutation operator selection, one for parameter tuning
	RLHDE [111]	2023	DE	SOP	use Q-learning to select mutation operators in QLSHADE and control the trigger parameters in HLSHADE
	GLEET [38]	2024	PSO,DE	SOP	dynamic hyper-parameters tuning based on exploration-exploitation tradeoff features
	RLMODE [112]	2024	DE	MOOP	dynamically control the key parameters in DE update rule
	RLNS [113]	2024	SSA,PSO,EO	MMOP	dynamically adjust the subpopulation size
	ada-smoDE [114]	2024	DE	SOP	dynamically control the key parameters in DE update rule
	PG-DE [115]	2024	DE	SOP	dynamic operator selection
	SA-DQN-DE [116]	2024	DE	MMOP	dynamically select proper local search operators
	RLEMMO [56]	2024	DE	MMOP	dynamically select DE mutation operators
	MRL-MOEA [55]	2024	MOEA	MOOP	dynamically select crossover operator in MOEA
Algorithm Generation	MSoRL [117]	2024	PSO	LSOP	automatically estimate the search potential of each particle
	UES-CMAES-RL [118]	2024	UES-CMAES	SOP	determine parameters in restart strategy by RL agent
	HF [119]	2024	DE	SOP,CO	dynamically select DE mutation operators by RL agent or manual mechanism
	MTDE-L2T [57]	2024	DE	MTOP	control information sharing in multi-population DE to solve MTOP
	ConfigX [120]	2025	DE,PSO,GA	SOP	universally control parameters and select operators for modular algorithms
	MetaDE [121]	2025	DE	SOP	a self-referential framework where DE is used to configure DE parameters
	KLEA [122]	2025	MOEA	LSMOP	dynamically switch dimension reduction strategies
	RLDE-AFL [123]	2025	DE	SOP	dynamically select DE mutation and crossover operators for each individual, and control their parameters
	SurrRLDE [124]	2025	DE	SOP	using Kan-based neural networks as surrogate models for MetaBBO
	LCC-CMAES [125]	2025	CMA-ES	LSOP	dynamically select problem decomposition operators during the cooperative co-evolution
	RNN-OI [44]	2017	-	SOP	use RNN as a BBO algorithm to output solutions iteratively
	RNN-Opt [45]	2019	-	SOP	using RNN as a algorithm to output sample distribution iteratively
	LTO-POMDP [47]	2021	-	SOP	LSTM-based optimizer to output per-dimensional distribution
	MELBA [126]	2022	-	SOP	use Transformer-based model to output sample distribution
	LGA [49]	2023	GA	SOP	use attention mechanism to imitate crossover and mutation in GA
Solution Manipulation	OPRO [127]	2023	-	SOP	use LLMs as optimizer to output solutions
	LMEA [128]	2023	-	SOP	use LLMs to select parent solutions and perform crossover and mutation to generate offspring solutions
	MOEA/D-LLM [129]	2023	MOEA/D	MOOP	use LLMs as the optimizer in MOEA/D process
	ELM [130]	2023	-	CO	use LLM agent to generate benchmark programs through evolution of existing ones
	ToLLM [131]	2023	-	SOP	prompt LLMs to generate solutions
	GLHF [46]	2024	DE	SOP	use neural network to imitate mutation and crossover in DE
	B2Opt [132]	2024	GA	SOP	use neural network to imitate operators in GA
	RIBBO [43]	2024	-	SOP	use GPT model to output optimization trajectories
	EvoLLM [51]	2024	-	SOP	imitate ES's optimization behaviour by iteratively prompting LLM
	EvoTF [42]	2024	-	SOP	use Transformer-based network to output ES's distribution parameter
Algorithm Generation	LEO [133]	2024	-	SOP	exploitation via LLM instead of crossover and mutation
	CCMO-LLM [134]	2024	-	CMOP	use LLM as the search operator within a classical CMOEA framework
	GSF [135]	2022	-	CO	generate whole BBO algorithm by using RL agent to select operators from fixed algorithmic template
	AEL [136]	2023	-	CO	use LLM to evolve algorithm source code
	EoH [52]	2023	-	CO	use LLM agent to evolve algorithm's thoughts and source code
	SYMBOL [137]	2024	-	SOP	automatically generate symbolic update rules along optimization process through LSTM
	LLaMEA [138]	2024	-	SOP	use LLM to evolve EA algorithm
	LLMOPF [139]	2024	-	MOOP	use LLM to evolve operators for multi-objective optimizer
	LLaMoCo [41]	2024	-	SOP	instruction-tuning for LLM to generate accurate algorithm code
Algorithm Generation	OptiMUS [53]	2024	-	MILP	develop multi-agent pipelines for LLM to solve MILP problem as a professional team
	LLM-EPS [140]	2024	-	-	use LLM to generate offspring codes in evolutionary program search
Algorithm Generation	ALDes [141]	2024	-	SOP	sequentially generate each component in an algorithm through auto-regressive inference

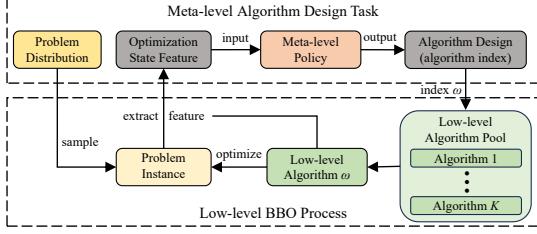


Fig. 3. Conceptual workflow of MetaBBO for Algorithm Selection.

tasks, along with their references, publication years, low-level optimizers, targeted problem types¹, and technical summaries.

A. Algorithm Selection

Algorithm Selection (AS) has been discussed for decades [142], [143]. The goal of AS is to select the most suitable algorithm from the algorithm pool according to the target task. The motivation of AS is that optimization behaviors and preferred scenarios vary with the algorithms, resulting in a notable performance difference [144]. Initially, AS is performed by human experts, who suggest algorithms based on their knowledge, which is labor-intensive and requires extensive expertise. To alleviate this dependency, researchers seek to develop more automated approaches.

1) Formulation: We examine the common AS paradigm. In the low-level BBO process, the component $\mathcal{A} = \{\mathcal{A}_1, \dots, \mathcal{A}_K\}$ represents an algorithm pool \mathcal{A} with K candidate BBO algorithms. The algorithm design space $\Omega = \{1, 2, \dots, K\}$ is the selective space involving all indexes of the candidate algorithms, where $\omega \in \Omega$ denotes an index of a candidate from \mathcal{A} . For each problem instance f_i in the training set, the goal of AS is to output an algorithm decision ω_i^t for f_i at each optimization step t . As illustrated in Fig. 3, MetaBBO automates this task by maintaining a learnable meta-level policy π_θ with parameters θ , which takes a state feature s_i^t obtained by $sf(\cdot)$ describing the optimization state of this optimization step, and then outputs ω_i^t . The selected candidate algorithm $\mathcal{A}[\omega_i^t]$ is used to optimize f_i in the low-level BBO process. Its performance on f_i serves as the performance measurement in Eq. (1). MetaBBO aims to find an optimal meta-level policy that suggests a best-performing algorithm in \mathcal{A} for each f_i at each optimization step t automatically. Suppose the optimization horizon of the low-level BBO process is T , the meta-objective $J(\theta)$ of AS is calculated as:

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \text{perf}(\mathcal{A}[\omega_i^t], f_i) \quad (2)$$

After training, π_θ is expected to select well-matched candidate algorithms from \mathcal{A} for unseen problems.

¹We use SOP, MOOP, COP, CMOP, MMOP, MMOOP, LSOP, LS-MOOP, MILP and CO to denote single-objective optimization, multi-objective optimization, constrained optimization, constrained multi-objective optimization, multi-modal optimization, multi-modal multi-objective optimization, large-scale optimization, large-scale multi-objective optimization, mixed integer linear programming and combinatorial optimization, respectively.

2) Related Works: First, per-instance AS is widely adopted in the literature, where a single algorithm is selected for the entire optimization progress for each specific problem, meaning that ω_i^t remains time-invariant. A straightforward approach to learning an effective meta-level policy for the AS task is to form a logical association between the attributes of f_i and the algorithm selection decision ω_i that corresponds to them. Since typically the number of candidate algorithms in the pool \mathcal{A} is finite, many early-stage MetaBBO for AS researches transformed the meta-level learning process to a classification task [77]–[83], [145], [146]. In their methodologies, the state feature extraction function $sf(\cdot)$ in Eq. (1) extracts problem characteristics s_i of f_i , which is significant enough to distinguish f_i with the other problem instances. A benchmarking process is employed to identify the top-performing candidate algorithm for f_i . The identified algorithm is then used as the classification label. The meta-level policy π_θ is regarded as a classifier and hence meta-trained to achieve maximum prediction accuracy. The state feature extraction mechanism $sf(\cdot)$ in these works can be very different according to the target optimization problem types. Meta-QAP [77], Meta-TSP [78] and Meta-VRP [80] construct an information collection termed as meta data for combinatorial optimization problems, which maintains the nodes information, edge connections in the graph and constraints of a problem instance. For continuous single-objective optimization problems, exploratory landscape analysis (ELA) techniques are adopted in [79], [82], [145], [146], which profiles the objective space characteristics of a problem instance such as the convexity, peaks and valleys. While for multi-objective optimization, the options are comparatively limited, with representative works as the decomposition-based landscape features [147], [148]. These works mainly apply basic classification models such as Support Vector Machine (SVM), K-Nearest Neighbors (KNN) and Multi-Layer Perceptron (MLP) for the label prediction. In contrast, to achieve in-depth data mining of the relationship between the problem structures and the optimizer performance, the study in [81] uses symbolic regression techniques to recover the mathematical equation of the given problem and then leverages a Long Short-Term Memory (LSTM) [149] to auto-regressively predict the desired candidate algorithm. The study in [150] introduces time-series of fitnesses obtained from the first few iterations of an algorithm as trajectory feature inputs and employs ML methods such as Random Forest for algorithm selection. Ana et al. [151], [152] propose per-run AS that conducts a warm-starting strategy to enhance the trajectory-based ELA sampling. TransOptAS [85] explores the possibility of constructing a performance indicator based solely on the raw objective values to eliminate the computation cost for computing $sf(\cdot)$. It leverages a Transformer [153]-styled architecture that takes a batch of sampled objective values as input and outputs the performance of the candidate algorithms through supervision under the benchmark results. AS-LLM [83] leverages pre-trained LLM embeddings to extract features from the candidate algorithms and the target optimization problem, then selects the best algorithm by feature similarities.

Several latest MetaBBO works explored the possibility of

extending per-instance AS to dynamic AS during the low-level BBO process [40], [54], [84]. Concretely, the meta-level algorithm design task in this paradigm turns to flexibly suggest one candidate algorithm to optimize f_i for each optimization step t . The dynamic AS is regarded as Markov Decision Process in the mentioned MetaBBO works and hence can be maximized by using RL to meta-learn an optimal policy. The optimization state feature in RL-DAS [40] includes not only the problem properties but also the dynamic optimization state information to support such flexible algorithm switch which help RL-DAS achieve at most 13% performance improvement over the advanced DE variants in its algorithm pool.

Moreover, the fundamental nature of AS highlights the significance of constructing the algorithm pool \mathcal{A} , which necessitates a comprehensive understanding of the target problem distribution and effective BBO algorithms. An effective pool should include a variety of BBO algorithms to tackle problems with diverse characteristics. Consequently, there is considerable interest among researchers in the automatic construction of algorithm portfolios. This involves leveraging data-driven approaches, such as Hydra [154], AutoFolio [155], and PAAAS [156], which offer extensive analysis and valuable insights for developing the algorithm pool \mathcal{A} .

3) Challenges: While past research has made progress in AS, several technical challenges persist:

- For per-instance AS, labeling the training set is expensive due to the exhaustive search needed to find the optimal algorithm for each instance. Limited candidates and problem instances lead to generalization issues. In dynamic AS, the increased methodological complexity challenges the learning effectiveness of RL methods.
- The algorithm design space in MetaBBO for AS is coarse-grained, limited by the performance of individual algorithms without tuning their configurations. In the next section, we introduce algorithm configuration tasks, which offer larger and more fine-grained design spaces.

B. Algorithm Configuration

Algorithm configuration (AC) is a key task in optimization, since almost all BBO algorithms possess hyper-parameters [157] and optional operators [158] that affect performance. To automate the AC task, various adaptive and self-adaptive BBO algorithms have been developed in the past decades [159]. Algorithms like JADE [25] and APSO [26] leverage historical optimization data to compute informative decision statistics such as the potential of the hyper-parameter values and the success rates of the optional operators [160]–[162]. However, as discussed in the introduction, these approaches suffer from design bias, limited generalization, and high labor costs.

1) Formulation: As shown in Fig. 4, MetaBBO overcomes the limitations of manual AC techniques by using meta-learning to develop a meta-level configuration policy. This policy dynamically adjusts a BBO algorithm throughout the lower-level BBO procedure. More formally: in the low-level BBO process, the optimizer \mathcal{A} represents the BBO algorithm to be configured. The algorithm design space Ω is hence the configuration space of \mathcal{A} . The size of Ω can be either infinite (with continuous hyper-parameters) or finite (with discrete

hyper-parameters or several optional operators). MetaBBO dictates AC in a dynamic manner: given a problem instance f_i , at each optimization step t of the low-level BBO process, a state feature s_i^t is obtained by $sf(\cdot)$ to describe the state of this optimization step. The meta-level policy $\pi_\theta(s_i^t)$ outputs the algorithm design ω_i^t , which sets the configuration of \mathcal{A} as $\mathcal{A}.set(\omega_i^t)$. Then the algorithm is used to optimize f_i for the current optimization step. Suppose the optimization horizon of the low-level BBO process is T , the meta-objective $J(\theta)$ of MetaBBO for AC is formulated as

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \text{perf}(\mathcal{A}.set(\omega_i^t), f_i) \quad (3)$$

MetaBBO for AC improves on human-crafted adaptive methods by meta-learning the configuration policy through optimizing the meta-objective in Eq. (3), removing the need for labor-intensive, expert-driven designs. The bi-level meta-learning paradigm also enhances generalization, as the policy can be trained on a large set of problem instances, distilling configuration strategies that can be applied to new problems.

2) Related Works: Typically, the AC studies involve a two-step process: initially choosing an algorithm template and subsequently adjusting internal components or parameters. In this paper, we categorize existing MetaBBO for AC research into three distinct subgroups based on the second step. The first sub-category is adaptive operator selection (AOS), where several optional operators are flexibly selected by the meta-level policy. The second is hyper-parameter optimization (HPO), where the hyper-parameter values are controlled by the meta-level policy. The last is the combination of AOS and HPO, where Ω is a complex configuration space including both hyper-parameters and operators. Next we introduce related works in these sub-categories.

a) Adaptive Operator Selection: The works in this line aims to dynamically switch the operators of the low-level BBO algorithms during the optimization process. The majority of AOS methods still focus on DE algorithms [34], [39], [91], [98]–[100], [106], [111], [163], due to their strong performance and the availability of various operators for selection. These works share similar methodologies: a mutation operator pool is maintained, involving representative mutation operators such as DE/rand/2, DE/best/2, DE/current-to-rand/1, DE/current-to-best/1 and DE/current-to-pbest/1. In order to address different types of problems, the technical differences in these works revolve around the tailored state feature extraction design and the operator pool. The state feature extraction in these works can be divided into two main strategies: discrete representation and continuous representation.

For discrete state representation, the study in [163] first computes the diversity variation and the performance improvement between two consecutive optimization steps as an effective profile of the optimization dynamics. These two indicators, being continuous variables, are then divided into five distinct levels each. According to the discretized state feature, a Q-table policy is constructed to select one operator from an operator pool with three candidates. RLHDE [111] uses the relative density in the solution space and the objective

space against the initial population and objective values to indicate the convergence trend and the performance improvement. The values of the two density indicators are discretized into five and four levels respectively, constituting 20 different optimization states. The operators pool in RLHDE involves six mutation operators, which improve the diversity of the optimization behaviours. RL-CORCO [98] addresses constrained multi-objective optimization by enhancing the CORCO algorithm through multiple Q-table policies. In the algorithm, each sub-population maintains a Q-table, where rows represent nine states indicating different levels of objective improvement and constraint violation, and columns represent two mutation operators. The policy selects the appropriate mutation operator to optimize the solution as effectively as possible.

Compared to discrete features, continuous state feature extraction enables finer state modeling, providing unique representations for optimization states and leading to smarter decisions by the meta-level policy. For instance, DE-DDQN [34] proposes a very comprehensive optimization state extraction function, which computes a total of 99 features: the first 19 features describe the optimization progress and the properties of the target optimization problems, while the rest 80 are statistics describing the optimization potential of the four mutation operators in the operator pool. An MLP neural network-based meta-level policy generates Q-values for the candidate mutation operators and the one with maximal Q-value is chosen for the next optimization step. Following DE-DDQN, DEDQN [39] and MOEA/D-DQN [99] also construct MLP policies. DEDQN indicates that the features in DE-DDQN show certain redundancy and might fall short in capturing the local landscape features. To address this, DEDQN proposes a feature extraction mechanism inspired from classical fitness landscape analysis [164]. By using random walk sampling, DEDQN computes ruggedness and fitness distance correlations in the local landscape. Results show that landscape features are effective for MetaBBO methods to generalize across problem types, i.e., from synthetic problems to realistic problems [32]. For addressing multi-objective optimization problem, MOEA/D-DQN embeds the information of the reference vectors in MOEA/D into the state extraction. KLEA [122] further explores effective RL policy that could adaptively select desired dimension reduction strategies to enhance MOEA/D in large scale problem instances. To tackle multi-modal optimization problem, RLEMMO [56] first clusters solutions to compute the neighborhood information. The optimization state is then constructed by concatenating optimization progress, distributional properties and neighborhood information.

Besides the selection of DE modules [165], other BBO algorithms such as PSO and CMA-ES also have their own modular frameworks (i.e., PSO-X [166] for PSO and modCMA [167] for CMA-ES), and operator selection methods [86], [105]. Furthermore, a novel genetic BBO algorithm modularization system covering diverse algorithm modules from DE, PSO and GA is developed in [120], where a Transformer-based agent, named ConfigX, is proposed to meta-learn a universal configuration policy through multitask reinforcement learning across a designed joint optimization task space.

b) Hyper-parameter Optimization: Several early attempts meta-learn a configuration policy that dictates a single hyper-parameter setting throughout the entire process of solving a problem instance [168], [169]. Now, most MetaBBO for AC approaches follow the dynamic AC paradigm in Eq. (3), offering a flexible exploration-exploitation tradeoff to further improve the optimization performance. Since different BBO algorithms have distinct hyper-parameters, existing MetaBBO for AC works customize their methods to explore the intricate relationships between the hyper-parameters and the resulting exploration-exploitation tradeoff in each specific algorithm.

Since DE is known to be highly sensitive to hyperparameter settings, particularly the scaling factor F and the crossover probability Cr , many efforts have focused on meta-tuning DE. RLDE [92] propose a simple Q-table policy to adjust F when optimizing the power generation efficiency in solar energy system. It uses a Boolean indicator as the optimization state feature: indicating whether the solution quality is improved between two optimization steps. The algorithm design space, represented as $\delta F \in \{-0.1, 0, 0.1\}$, indicates the variation in F for the subsequent optimization step. Following RLDE, QLDE [94] extends the algorithm design space to five combinations of the parameter values. RLMODE [112] further proposes a specific state extraction function for constrained multi-objective optimization, which divides the state feature into eight possible situations, according to the solution feasibility and the dominance relationship. The algorithm design space is three combinations of different F and Cr settings to represent different exploration-exploitation tradeoffs. The Q-table agent is updated by first selecting the most promising combination and then observing the resulting performance improvement. For more fine-grained parameter control, LDE [37] firstly considers using recurrent neural network (i.e., LSTM) as the meta-level policy, which extracts hidden state feature for separate optimization step and outputs the values for F and Cr from a continuous range $[0, 1]$. The same authors subsequently propose LADE [103] as an extension of LDE. Compared to LDE, LADE aims to control more hyper-parameters including not only the mutation strength and crossover rate but also the update weights. All parameters are represented as matrix operations. Instead of using one LSTM for controlling all parameters, LADE's meta-level policy comprises three LSTM networks for controlling these parameters respectively. LADE shows more robust learning effectiveness than LDE. A recent study, L2T [57], employs the MetaBBO framework to regulate the setting of DE parameters and the likelihood of knowledge transfer within the multitask optimization working scenarios. The state feature is represented by the rate of successful transfers and the enhancement in sub-population performance. Despite adapting F and Cr , the control of population size is considered in Q-LSHADE [102]. The algorithm design space is the decay rate of the linear population size reduction in LSHADE [170], which can take values from $\{0, 0.2\}$. Q-LSHADE also meta-learns a Q-table policy by the feedback indicating the performance improvement. There are also several MetaBBO works which facilitate hyper-parameter optimization on other algorithms, such as PSO [36], [38], [88], [93], [95], [96], [104], [113], ES [48], [90], and the

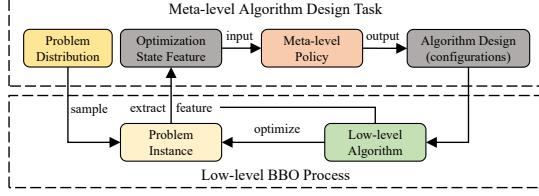


Fig. 4. Conceptual workflow of MetaBBO for Algorithm Configuration.

Firefly algorithm [171]. In addition, a recent work GLEET [38] proposes a general learning paradigm which show generic HPO ability for both DE and PSO. Due to the space limitation, other related works are summarized in Table I.

c) Hybrid Control: Some MetaBBO works explore other AC perspectives [115], [119]. In particular, the combination of HPO and AOS has gained significant attention [35], [87], [97], [101], [110], [111], since learning a meta-level policy in $\Omega_{\text{HPO+AOS}}$ would probably result in a better AC policy than learning them separately. Nevertheless, this poses a significant challenge as learning from an expanded algorithm design space necessitates more intricate learning strategies and model frameworks. Thoughtful design is essential to guarantee effective learning.

3) Challenges: Despite their success, existing MetaBBO works for AC still face some challenges.

- A certain proportion of existing methods use a very limited set of training problems. In particular, some only train their meta-level policies on a specific optimization problem instance, raising doubts about the true generalization performance of the resulting policies.
- MetaBBO for AC works operate on the basis of predefined low-level BBO algorithms. Hence, the performance of these methods is closely tied to the original BBO algorithm. Furthermore, the inherent algorithm structures, optimization logic, and design biases significantly restrict the algorithm design space. Can we further expand the algorithm design space and step out this boundary? In the next two subsections, we introduce two novel categories of MetaBBO works that offer potential solutions.

C. Solution Manipulation

So far, we have introduced two basic categories of MetaBBO: AS and AC. An intuitive observation is that within the MetaBBO framework for AS/AC tasks, the low-level BBO procedure necessitates a BBO algorithm as the foundational optimizer, which comes with a defined algorithm design space (e.g., algorithm pool or configuration space). This leads to two limitations. First, it requires expert knowledge to select an appropriate BBO algorithm, otherwise the meta-level policy's learning effectiveness and overall performance may suffer. Second, managing both the meta-level policy and the low-level BBO optimizer simultaneously incurs certain computational costs. To address these limitations, several MetaBBO works have explored the potential of directly using the meta-level policy for solution manipulation. This approach integrates meta-level training and low-level optimization into a single entity, eliminating the need for a predefined BBO algorithm.

In this framework, the meta-level policy itself functions as an optimization algorithm, directly manipulating candidate solutions throughout the optimization process. We illustrate this MetaBBO workflow in Fig. 5, referring to it as MetaBBO for solution manipulation (SM).

1) Formulation: To formulate the process of solution manipulation in MetaBBO, some clarifications have to be made. First, MetaBBO for SM integrates the functions of meta-level policy and the low-level BBO algorithm into a single parameterized agent π_θ , removing the need for a traditionally perceived BBO algorithm. Therefore, the meta-level policy π_θ , typically a neural network, inherently serves as the BBO algorithm. In this case, the algorithm design space Ω turns to the parameter space of the policy, where each algorithm design ω in this space corresponds to the values of the neural network parameters θ . Given a problem instance f_i , at each optimization step t , the optimization state feature s_i^t is first computed by $\text{sf}(\cdot)$. According to s_i^t , the policy (acts as the BBO algorithm) π_θ optimizes f_i for one optimization step, e.g., reproducing the candidate solutions. The performance improvement is hence measured as $\text{perf}(\pi_\theta(s_i^t), f_i)$. Suppose the optimization horizon of the low-level BBO process is T , the meta-objective of MetaBBO for SM is formulated as

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \text{perf}(\pi_\theta(s_i^t), f_i) \quad (4)$$

Through maximizing $J(\theta)$ over N problem instances in the training set, a neural network-based BBO algorithm is obtained, functioning similarly to human-crafted BBO algorithms: iteratively optimizes the problem instances. Next, we next introduce representative MetaBBO works for SM.

2) Related Works: An intuitive way of resembling the iterative optimization behaviour by neural networks is considering temporal network structure such as recurrent neural networks [44], [45], [47], which enable MetaBBO to directly adjust candidate solutions over sequential steps. The corresponding mathematical formulation is quite straightforward:

$$X^t, h^t = \pi_\theta(X^{t-1}, Y^{t-1}, h^{t-1}), \quad Y^t = f_i(X^t) \quad (5)$$

where π_θ is an RNN/LSTM, h^t is the hidden state. This paradigm is first adopted in RNN-OI [44], which meta-learns an LSTM to reproduce candidate solutions. For each f_i in the training problem set, RNN-OI randomly initializes a solution X^0 , obtains the corresponding objective values Y^0 , and then optimizes f_i by iteratively inferring the next-step solution. To meta-learn a well-performing π_θ , the observed improvement per step is computed as the $\text{perf}(\cdot)$ function. Once trained, the LSTM serves as a BBO algorithm and iteratively optimizes the target optimization problem following Eq. (5). Due to the end-to-end inferring process, RNN-OI is shown to run 10^4 times faster compared to hand-crafted algorithms such as Spearmint [172]. Following RNN-OI, similar works include RNN-Opt [45] improving RNN-OI through input normalization and constraint-dependent loss function, LTO-POMDP [47] using neuroevolution to learn the network parameters, MELBA [126] improving the long sequence modelling of RNN/LSTM by introducing Transformer structure,

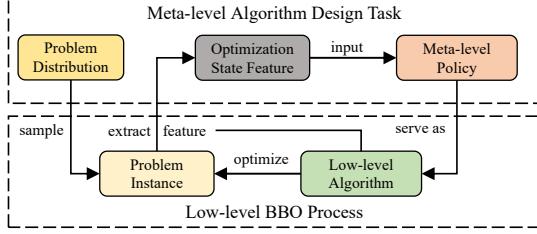


Fig. 5. Conceptual workflow of MetaBBO for Solution Manipulation.

and RIBBO [43] leveraging efficient and generic behaviour cloning framework to learn an optimizer that resembles the given teacher optimizer.

Nevertheless, the above works still suffer from generalization limitation and interpretability issues. On the one hand, the optimization state features only include the raw population information, which makes the policy easily overfits to the training problems. On the other hand, the learned policies in these works shift toward “black-box” systems, which hinders further analysis on what they have learned. In the last two years, several more interpretable MetaBBO for SM works are proposed to address these issues [42], [46], [49], [132]. These works propose using higher-level features as a substitute for the raw features to achieve generalizable state features across diverse problems. Typically, these features include the distributional characteristics of the solution space and the objective space, the rank of objective values, and the temporal features reflecting the optimization dynamics. They have proposed several novel architecture designs to make the meta-level policy explicitly resembles representative EC algorithms such GA [49], [132], DE [46], and ES [42]. For instance, LGA [49] designs two attention-based neural network modules to act as the selection and mutation rate adaption mechanisms in GA. The parameterized selection module applies cross-attention between the parent population and the child population, and the obtained attention score matrix is used as the selection probability. The parameterized mutation rate adaption module applies self-attention within the child population, and the obtained attention scores is used as the mutation rate variation strength. B2Opt [132] improves LGA by proposing a novel, fully end-to-end network architecture which resembles all algorithmic components in GA, including crossover, mutation, selection. For example, the selection module within B2Opt utilizes a method similar to the residual connection in Transformer, facilitating the use of matrix operations for selecting populations. By meta-training the proposed meta-level policies on the training problem set, these MetaBBO for SM works show competitive optimization performance. In particular, their meta-level policies are trained with low dimensional synthetic problems (≤ 10) yet could be directly generalized for solving high dimensional continuous control problems (> 500) [49], e.g., neuroevolution [173].

With the emergence of LLMs, their ability to understand the reasoning in natural language outlines a novel opportunity for SM. Related works in this line widely leverage the In-Context Learning (ICL) [174] to prompt with general LLMs iteratively as an analog to BBO algorithms to reproduce solutions. A

pioneer work is OPRO [127], which first provides LLMs a context of the problem formulation and historical optimization trajectory described in natural language. It then prompts LLMs to suggest better solutions based on the provided context. This idea soon becomes popular and spreads to multiple optimization scenarios such as program search [130] combinatorial optimization [128], multi-objective optimization [129], [134], large scale optimization problem [51], [133] and prompt optimization [50]. The eye-catching advantage of LLM-based SM is that it requires minimal expertise - users only need to describe the optimization problem in nature language, and LLMs handle the rest.

3) **Challenges:** As a novel direction, MetaBBO for SM is promising due to the end-to-end manner. However, several technical challenges remain:

- Approaches like RNN-Opt directly learn to manipulate candidate solutions without following a specific algorithm structure. While this provides flexibility, these methods often lack transparency and clear understanding of their inner workings. Additionally, due to the complexity of BBO tasks, exploring strong neural networks capable of handling diverse, complex problems remains a challenge.
- In contrast, methods like LGA closely mimic the structure and components of existing EAs, making the process more transparent. However, because these methods resemble existing algorithms, their performance might be inherently constrained by the limits of the original ones.
- MetaBBO approaches that use LLMs, while reducing the need for manual algorithm design, face significant computational overhead. The iterative interactions with LLMs generate large volumes of tokens, leading to inefficiencies in both time and cost.
- Finally, MetaBBO for SM treats the policy itself as the optimizer, targeting at learning the optimal mapping from current landscape to next candidate positions. However, this remains a highly challenging task for continuous BBO tasks. The possible landscapes are diverse and infinite. As a result, so far, it is very challenging to build and train a model that can effectively handle these complexities in practice. In the next section, we will explore the “algorithm generation” approach, which leverages the meta-level policy as an algorithm discoverer, namely, using learning to create new algorithmic workflows, update rules, and implementations.

D. Algorithm Generation

Besides MetaBBO for SM, an interesting research question comes out: whether learning-based systems such as MetaBBO could automatically create (generate) new BBO algorithms with competitive optimization performance and minimal expertise requirement? To this end, MetaBBO for algorithm generation (AG) presents a different methodology: meta learning a parameterized policy that could discover novel algorithms accordingly without the human-expert prior, of which the workflow is illustrated in Fig. 6. The difference between AG and SM is that the meta-level policy in SM plays both the role of the meta-level policy and the low-level optimizer, while

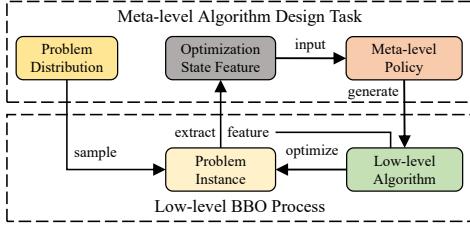


Fig. 6. Conceptual workflow of MetaBBO for Algorithm Generation.

the meta-level policy in AG is trained to output a complete optimizer which is used then in the low-level BBO process.

1) **Formulation:** MetaBBO for AG works construct an algorithm representation space Ω as its design space. For example, Ω can be a algorithm workflow space, a mathematical expression space or a programming language space, reflecting the way humans express algorithms - through modular algorithm workflows, symbolic mathematical expressions or programming language syntax. For a problem instance f_i , a concrete algorithm design ω_i^t is output by the meta-level policy π_θ , according to the optimization state feature s_i^t . The $sf(\cdot)$ function, in this case, can incorporate landscape features, symbolic representations, or natural language descriptions of f_i . The generated ω_i^t can be a complete workflow, a mathematical expression or a functional program that represents a novel BBO algorithm \mathcal{A} . the meta-objective of MetaBBO for AG is to meta learn a policy π_θ capable of generating well-performing algorithms:

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \text{perf}(\omega_i^t, f_i), \quad \omega_i^t = \pi_\theta(s_i^t) \quad (6)$$

where $\text{perf}(\omega_i^t, f_i)$ is the one-step optimization performance gain of the generated algorithm on f_i . Through training the policy across a problem set, the policy is expected to automatically generate flexible and even novel BBO algorithms to address various optimization problems. Besides, note that MetaBBO for AG could work with varying granularity: a) generating a universal algorithm for all problems [52], b) generating customized algorithms for each problem [141], and c) generating flexible optimization rules that adapt to each step of the optimization process and each specific problem [135], [137]. In Eq. (6), we demonstrate the case c). In contrast, in the case a), a single algorithm ω is generated to serve as ω_i^t in Eq. (6). In case b), a problem-specific ω_i is generated to serve as ω_i^t for each optimization step in solving f_i .

2) **Related Works:** Creating a comprehensive algorithm representation space Ω is crucial for the meta-level policy to produce innovative and efficient BBO algorithms. Current MetaBBO methodologies for AG can be categorized into three types based on their formulation of algorithm representation space Ω : algorithm workflow composition, mathematical expressions, or natural/programming languages.

First, we introduce the works that perform algorithm workflow composition. A very early-stage work is conducted by Schmidhuber [68] in 1987, where GP components are represented by the computer program space. Following such idea, GP is further applied to create improved EA variation operators [175], [176], evolve EA selection heuristics [177]

and generate complete algorithm template [178]. At the meta-level, a GP is used to evolve low-level GP programs in a self-referential way. In the latest literature, GSF [135] first defines an algorithm template for EAs, then uses RL to fill each part of the template with operators from a predefined operator pool. ALDes [141] overcomes the limitation of using fixed-length template through autoregressive learning. It first tokenizes the common algorithmic components and the corresponding configuration parameters in EAs, as well as the execution workflows such as loop and condition. Then, the algorithm generation task turns into a sequence generation task of the tokens. Concretely, ALDes prepares three types of operators: four “selection for evolution” operators, six evolution operators and five “selection for replacement” operators, each is associated with some hyper-parameters. Given the property of the target optimization problem, a Transformer-style policy auto-regressively selects a desired operator and configures its hyper-parameter from the candidate pool of each operator type.

Second, we introduce the works that leverage mathematical expression to formulate Ω . The motivation behind this line is that the design space of GSF and ALDes is highly dependent on manual engineering, which may limit the exploration of more novel algorithm structures. SYMBOL [137] addresses this issue by breaking down the update equations of BBO algorithms into atomic mathematical operators and operands. SYMBOL constructs a token set of common mathematical symbols used in EAs, such as $\{+, -, \times, x, x^*, x^-, x_i^*, \Delta x, x_r, c\}$. It then designs an LSTM-based policy which is capable of auto-regressively generating a sequence of these mathematical symbols. The generated sequence can be parsed into update equations for optimizing the low-level optimization problem. SYMBOL generates flexible update rules for each optimization step and each problem instance, bringing in certain self-adaptation capabilities.

Third, we introduce works that leverage natural language and programming language to define Ω . All works in this line leverage LLMs as their meta-level policies [41], [52], [53], [136], [138], [139]. The differences lie in the learning methodologies, the generation workflows and the target problem types. OptiMUS [53] leverages modular-structured LLM agents to formulate and solve (mixed integer) linear programming problems. There are four agents in OptiMUS: formulator, programmer, evaluator, and manager, which constitute an optimization expert team and automate the algorithm generation task through their cooperation. To enable more general-purpose algorithm generation, AEL [136] and EoH [52] are inspired by the evolution capability of large models [130], prompting LLMs to perform mutation and crossover operations on code implementations of previous algorithms. After evolution, the best-so-far algorithm generated shows at most 24% performance margin over human-crafted heuristics on Traveling Salesman Problems. Subsequent works such as LLaMEA [138] and LLMOpt [139] generalize this paradigm to continuous BBO scenarios, and LLaMEA is shown to be capable of generating a more complex algorithm that is competitive with CMA-ES within 1% score gap. Despite the above works, LLaMoCo [41] offers a novel perspective: instruction-tuning the general LLMs to act as an expert-

level optimization programmer. LLaMoCo allows users to describe their specific optimization problems in Python/LaTeX formulation, then it outputs the complete Python implementation of a desired optimizer for solving the given problems. To achieve this, a large-scale benchmarking is conducted to attain thousands of problem-solver pairs as the expert-level optimization knowledge. This knowledge is then injected into LLMs through instruction tuning. The experimental results in LLaMoCo demonstrates that a small model (e.g., codeGen-350M) could generate superior algorithm program to larger models which are not fine-tuned by LLaMoCo (e.g., GPT-4), underscoring that domain specific knowledge might be the key for LLMs to understand, reasoning and solve problems.

3) **Challenges:** MetaBBO for AG works operate in a more expressive algorithm design space. The experimental results in some of these works demonstrate that the generated algorithms are on par with or even superior to human-crafted ones. The generated BBO algorithms can not only address optimization problems, but also be further analysed by human experts for novel insights in developing optimization techniques. Nevertheless, there are still several bottlenecks in existing works:

- As an early-stage research avenue, related works in this area are still limited. More studies are expected to further unleash the potential of MetaBBO for AG.
- For symbolic system-based generation frameworks such as ALDes and SYMBOL, the token sets are relatively small, which leads to limited representation capability. How to construct a comprehensive and expressive token set tailored for BBO algorithm, and how to ensure the learning effectiveness in the enlarged algorithm design space need further investigation.
- For LLM-assisted MetaBBO for AG, although the workflow promises an efficient development pipeline, the computational resources required to obtain a competitive BBO algorithm are substantial. Besides, these works rely heavily on the prompt engineering, since LLMs are sensitive to the prompts they receive.

IV. DIFFERENT LEARNING PARADIGMS AT META LEVEL

In this section, we introduce four key learning paradigms behind the majority of existing MetaBBO works: MetaBBO with reinforcement learning (MetaBBO-RL), MetaBBO with supervised learning (MetaBBO-SL), MetaBBO with neuroevolution (MetaBBO-NE) and MetaBBO with in-context learning (MetaBBO-ICL).

A. MetaBBO-RL

In MetaBBO-RL, the meta-level algorithm design task is modeled as an MDP [66], [179], where the environment is the low-level BBO process for a given problem instance f . The optimization state feature space for s , the algorithm design space Ω , and the performance metric $\text{perf}(\cdot)$ serve as the MDP's state space, action space, and reward function, respectively. As a result, the meta-objective defined in Eq. (1) becomes the expected accumulated reward. While various RL techniques can be applied, the choice must be made by considering the characteristics of the state and action spaces, which we categorize into three main types below.

1) **Discrete State & Discrete Action:** Tabular Q-learning [180] and SARSA [66] are value-based RL techniques that maintain a Q-table to iteratively update state-action values based on interactions with the environment. These methods have a notable benefit in their straightforward Q-table structures, which facilitates efficient convergence and reliable effectiveness. However, they are confined to MDPs with discrete (finite) state and action spaces. Many MetaBBO-RL works adopt these methods for their simplicity. In such works, optimization states and algorithm designs are pre-defined to form the rows and columns of the Q-table. At each optimization step t in the low-level BBO process, the meta-level policy suggests an algorithm design ω^t according to s^t and the Q table. Then, a transition $< s^t, \omega^t, \text{perf}(s^t, \omega^t, f), s^{t+1} >$ is obtained and the Q-table is updated as

$$Q(s^t, \omega^t) = \text{perf}(s^t, \omega^t, f) + \gamma \max_{\omega \in \Omega} Q(s^{t+1}, \omega) \quad (7)$$

An example of this approach is the QLPSO algorithm [36], which dynamically adjusts the particle swarm topology. In QLPSO, the optimization states are $\{L2, L4, L8, L10\}$, representing different neighborhood size features of particles, with corresponding actions to either maintain or change the neighborhood size. Performance improvements resulting from successful topology adjustments are rewarded. Other works using similar methods include RLNS [113], QFA [171], qlDE [94], RLMPSO [86], DE-RLFR [89], QL-(S)M-OPSO [88], MARL-wCMA [163], LRMODE [91], RLEA-SSC [181], RLDE [92], RL-CORCO [98], RL-SHADE [100].

2) **Continuous State & Discrete Action:** While Tabular Q-learning and SARSA are effective for discrete state spaces, some MetaBBO scenarios require continuous optimization states for finer algorithm design. In such cases, the MDP involves an infinite state space, making the Q-table structure incompatible. To address this, neural network-based Q-agents, such as DQN [182] and DDQN [183], are employed to handle continuous state features. The Q-agent is updated by minimizing the estimation error between the target and predicted Q-functions as

$$\text{Loss}(\theta) = \frac{1}{2} \left[Q_\theta(s^t, \omega^t) - \left(\text{perf}(s^t, \omega^t, f) + \gamma \max_{\omega \in \Omega} Q_\theta(s^{t+1}, \omega) \right)^2 \right] \quad (8)$$

A representative example is DEDQN [39], where the optimization state is represented by four continuous FLA indicator features derived from a random walking strategy. The meta-level policy is an MLP Q-agent with three hidden layers. During the low-level BBO process, the Q-agent outputs Q-values for three candidate DE mutation operators and selects one for the current optimization step. The performance improvement after this step serves as a reward. The transition obtained is used to update the Q-agent by Eq. (8). Other works employing similar methods include R2-RLMOEA [54], DE-DDQN [34], MADAC [97], MOEA/D-DQN [99], CEDE-DRL [109], SA-DQN-DE [116], UES-CMAES-RL [118], HF [119].

3) **Continuous State & Continuous Action:** Building on the success of RL techniques in continuous control [184], some MetaBBO-RL works adopt policy gradient-based methods (e.g., REINFORCE [185], A2C [186], PPO [187]) to handle both continuous states and algorithm designs. These allow for more flexible control of optimization behavior in the low-level BBO process, possibly improving performance. In this case, a policy neural network π_θ is used to output a probability distribution over the algorithm design space based on the optimization state. The gradient $\nabla_\theta J(\theta)$ used to update π_θ is computed as

$$\nabla_\theta J(\theta) = -\nabla_\theta \log \pi_\theta(\omega^t | s^t) \left(\sum_{t'=t}^T \gamma^{t'-t} \text{perf}(s^{t'}, \omega^{t'}, f) \right) \quad (9)$$

We illustrate the method with the representative work GLEET [38]. In GLEET, the optimization state is represented by a structured feature set, including low-level information such as solution/objective space density and performance improvement indicators. A Transformer-style policy network (3 layers) outputs the posterior Gaussian distribution for each parameter of each individual. The concrete parameter values are then sampled from this distributions for the current optimization step, and the corresponding reward is assigned. After completing an optimization episode (T steps), the policy network π_θ is updated by summing the gradients from each step, as shown in Eq. (9). Other MetaBBO-RL works employing similar methodologies include LTO [90], RLEPSO [93], LDE [37], RL-PSO [95], MELBA [126], MOEADRL [188], LADE [103], RLAM [104], AMODE-DRL [110], PG-DE [115], GLEET [38], RLEMMO [56], RL-DAS [40], SYMBOL [137].

B. MetaBBO-NE

Neuroevolution [189] is a machine learning subfield where neural networks are evolved using EC methods rather than updated by gradient descent. In [7], ES is demonstrated as a scalable alternative to RL for MDPs, especially when actions have long-lasting effects. This inspired the development of MetaBBO methods using EC to evolve the policies, referred to as MetaBBO-NE. In MetaBBO-NE, the meta-level maintains a population of policies $\{\pi_{\theta_1}, \dots, \pi_{\theta_K}\}$, with each policy π_{θ_k} being used to guide the algorithm design task for a training problem set. The fitness of each policy is the average performance gain across the problem instances in the training set. An EC method, such as ES, is employed to iteratively update the the meta-level policies, and after several generations, the optimal policy π_{θ^*} is obtained.

Representative works in MetaBBO-NE include LTO-POMDP [47] and LGA [49]. For example, in LGA, a population of attention-based neural networks is maintained at the meta-level, where each network functions as a neural GA to manipulate solutions. The OpenAI-ES [7] is then used to evolve $K = 32$ such networks over ten 10-dimensional synthetic functions from the COCO benchmark [190].

C. MetaBBO-SL

MetaBBO works using the supervised learning paradigm are closely related to the meta task of solution manipulation (SM). As we described in Eq. (4), SM aims to learn a parameterized meta-level policy π_θ as the low-level optimizer. The optimization process proceeds by iteratively calling π_θ to optimize the current (population of) solution(s). A key difference between MetaBBO-SL and MetaBBO-RL is that MetaBBO-SL meta-trains policies using direct gradient descent on an explicit supervising objective. This resembles regret minimization [191] of the target optimization problem's objective function. To illustrate this, let us examine the recent work GLHF [46], which proposes an end-to-end MetaBBO method mimicking a DE algorithm. GLHF unifies the DE mutation and crossover operations as matrix operations and designs π_θ as two customized network modules, LMM and LCM, to simulate matrix-based mutation and crossover. The Gumbel-Softmax function is used in the crossover module to make it differentiable. Given a solution population X^t at the optimization step t when optimizing a problem f , the π_θ in GLHF optimizes X^t to generate offspring population: $X^{t+1} = \pi_\theta(X^t)$. The explicit supervising objective in this case is the objective value $f(X^{t+1})$, which serves as a regret function to minimize. Then the gradient used to update the policy at step t is computed as

$$\nabla_\theta J(\theta) \propto \frac{\partial f(X^{t+1})}{\partial \pi_\theta} \cdot \frac{\partial \pi_\theta}{\partial \theta} \quad (10)$$

Minimizing this regret-based objective trains the meta-level policy for effective optimization on the target problem. However, the differentiability of f is a requirement, which may not hold for “black-box” problems. Other works in this line include RNN-OI [44], RNN-Opt [45], B2Opt [132], EvoTF [42], LEO [133], RIBBO [43], NAP [192]. Notably, RIBBO [43] and EvoTF [42] use supervised imitation learning to meta-train their policies to mimic a teacher BBO algorithm. For instance, RIBBO uses a GPT architecture to imitate optimization trajectory from diverse existing BBO algorithms. It tokenizes each of the collected optimization trajectories into a target token sequence $\{R^1, X^1, Y^1, \dots, R^T, X^T, Y^T\}$, where R^t , X^t and Y^t are the regret-based explicit supervising objective, population positions and objective values respectively. Through behavior cloning, RIBBO trains the GPT to mimic these trajectories, resulting in a generalized optimization behavior and outperforming baselines on 3 out of 5 problems.

D. MetaBBO-ICL

In-Context Learning (ICL) [174] is a popular paradigm in LLM research, which prompts LLMs with a structured text collection: a *task description*, several *in-context examples*, and a concrete *task instruction*. This structured prompt enables LLMs to reason effectively based on the provided context, without requiring gradient descent or parameter updates. MetaBBO-ICL is closely related to two meta tasks: Solution Manipulation (SM) and Algorithm Generation (AG). The main distinction between existing works lies in how they construct effective in-context prompts.

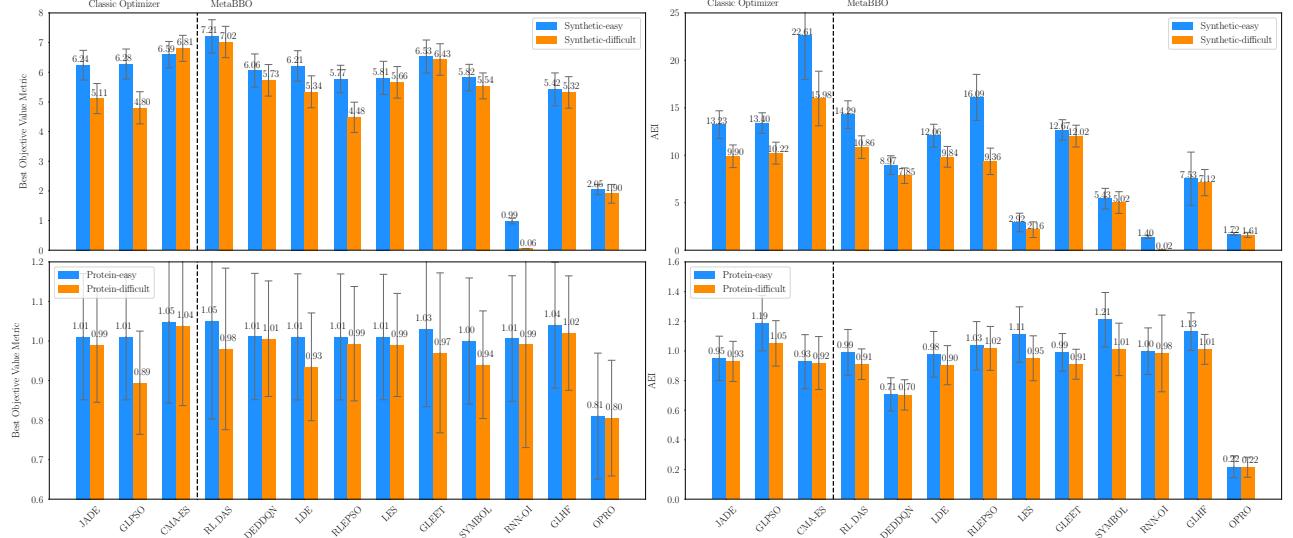


Fig. 7. Performance comparisons. **Top Left:** Best objective values on synthetic testsuites. **Top Right:** AEI scores on synthetic testsuites. **Bottom Left:** Best objective values on protein docking testsuites. **Bottom Right:** AEI scores on protein docking testsuites.

For the SM task, OPRO [127] introduces optimization via iterative prompting. In each iteration, the *task description* is tailored to the specific problem, including the objective function and constraints in natural language. The *in-context examples* consist of prior optimization trajectories, and the *task instruction* asks the LLM to find a solution better than the previous best. However, this approach faces challenges due to the limited optimization expertise of general LLMs, which are not typically trained with optimization knowledge in mind [41]. Recent studies creatively suggest guiding LLMs to mimic certain EAs [128], which involves directing LLMs to execute mutation, crossover, and elitism strategies on the specified *in-context examples*.

For the AG task, the core idea is using LLMs to understand and evolve optimizer programs. Note that evolving programs is not a novel concept. This topic traces back to Genetic Programming (GP) method, which performs evolution of computer program within the code space in a self-referential way: evolve evolution algorithms. Leveraging the semantic reasoning ability of CodeLLMs for program evolution, initial works such as Funsearch [193] and EUREKA [194] discover competitive heuristic program and reward design respectively. Following these works, in MetaBBO-ICL, researchers begin to discuss the possibility of evolving competitive optimization program with CodeLLMs. A representative work in this area is EoH [52], where the *task description* includes both the optimization problem formulation and a concrete algorithm design task. The LLM is asked to first describe a new heuristic and then implement it in Python. The *in-context examples* consist of previously suggested programs, while the *task instruction* provides five evolution instructions, each with varying levels of code refinement. Other related works include AEL [136], LLAMEA [138], and LLLOPT [139].

In this section, we have introduced four key learning paradigms widely adopted in existing MetaBBO works. To summarize, each learning paradigm has its own advantages

and downsides. By the aid of powerful LLMs, MetaBBO-ICL spends least efforts in developing and designing. In contrast, MetaBBO-RL requires certain expertise in designing effective RL systems, while achieving significant policy improvement. As for the final performance, it is empirically observed that MetaBBO-RL and MetaBBO-SL generally achieve superior optimization results (see Fig. 7). As for the research interests for the four learning paradigms, we found MetaBBO-RL is continuously catching researchers' attention, while MetaBBO-ICL rises quickly (see Table I).

V. EMPIRICAL EVALUATION

A. Development in Benchmarks

For benchmarking BBO optimizers, many well-known test-suites have been extensively studied and developed [195], [196]. With the ongoing development of BBO, the corresponding benchmarks aim to 1) propose more diverse benchmark problems in synthetic [81], [197]–[202] and realistic [173], [203], [204] scenarios; and 2) automate the benchmarking process through a software platform [190], [205]. These traditional BBO benchmarks can serve as evaluation tools for MetaBBO methods. However, compared with traditional EC algorithms, the system structure of MetaBBO is more intricate. Its bi-level learning paradigm involves a meta-level policy, a low-level optimizer, the training/testing logic of the entire system, and the interfaces between the meta and lower levels. This complexity creates a gap between the conventional BBO benchmarks and MetaBBO methods. To address this compatibility issue, a recent work termed MetaBox [32] proposes the first benchmark platform specifically for developing and evaluating MetaBBO methods. It provides three different single-objective numerical problem collections (Synthetic-10D, Noisy-Synthetic-10D, Protein-Docking-12D), along with two different train-test split modes (easy and difficult), which benefits MetaBBO's training under different problem distributions and difficulties. In the next subsection, we pro-

vide a proof-of-principle evaluation of several representative MetaBBO methods using MetaBox.

B. Proof-of-Principle Evaluation by MetaBox

In this section, we use MetaBox [32] to evaluate the performance of three traditional EC algorithms and ten representative MetaBBO methods. For traditional EC algorithms, we empirically select three representative algorithms: JADE [25], GLPSO [27] and CMA-ES [28] from three mainstream traditional BBO classes: DE [73], PSO [206] and ES [207], respectively. Further, we include the MetaBBO methods covering all four meta-tasks and all four learning paradigms:

- **AS:** RL-DAS [40] (RL-based dynamic selection);
- **AC:** DE-DDQN [34] (RL-based AOS); LDE [37]/RLEPSO [93]/GLEET [38] (RL-based HPO); LES [48] (NE-based HPO);
- **AG:** SYMBOL [137] (RL-based symbolic synthesis);
- **SM:** RNN-OI [44]/GLHF [46] (early/recent SL-based trajectory prediction); OPRO [127] (ICL-based linguistic optimization).

All experiments follows the protocols in MetaBox. Due to the space limitation We leave the detailed baseline selection criteria and experimental setup in Appendix I.A & I.B respectively.

The AEI score in MetaBox [32] evaluates the overall optimization performance of a MetaBBO method by aggregating three key metrics: final optimization results, FEs consumed, and runtime complexity, using an exponential average, larger is better. The left side of Fig. 7 presents the final optimization accuracy of all baselines on Synthetic BBOB (top) and Realistic Protein Docking (bottom) testsuites, while the right side presents their respective AEI scores. The results show that: 1) When considering only the final accuracy, MetaBBO methods such as RL-DAS, LDE, and GLEET achieve comparable or even superior performance to traditional BBO optimizers, while some other MetaBBO methods still perform inferiorly compared to traditional BBO methods. This indicates that while MetaBBO methods show potential, as an emerging topic, there is still significant room for improvement. 2) Different evaluation metrics yield different conclusions regarding performance. When considering both optimization performance and computational overhead, traditional BBO optimizers such as CMA-ES achieve a significantly better trade-off, as shown on the right side of Fig. 7. This highlights a potential limitation of MetaBBO methods: they typically involve additional computation during the meta-level decision-making process. 3) Comparing the performance on the Synthetic BBOB and Protein Docking test suites, we observe that the performance gap between MetaBBO methods and traditional BBO optimizers narrows on both evaluation metrics as the problem type shifts from the relatively simpler synthetic set to the more challenging realistic protein docking set. This suggests that MetaBBO is promising for solving complex optimization problems. 4) MetaBBO-RL methods (including RL-DAS, DE-DDQN, LDE, RLEPSO, SYMBOL) outperform MetaBBO-NE methods (LES), MetaBBO-SL methods (RNN-OI) and MetaBBO-ICL methods (OPRO). This observation highlights an important future direction for the MetaBBO

domain: analyzing the theoretical performance bounds of different MetaBBO methods. 5) The AEI of OPRO (MetaBBO-ICL method) is significantly lower, this might indicate that iteratively optimization paradigm through in-context prompting LLMs is severely challenged by the efficiency issue.

Besides the above algorithmic performances, one should also examine a MetaBBO method's learning capability. As a learning system, it is expected that a MetaBBO method should show certain generalization ability on unseen problem instances/distributions. To this end, we have tested the MetaBBO baselines on MetaBox for their Meta Generalization Decay (MGD) Meta Transfer Efficiency (MTE), two indicators proposed in MetaBox to measure a MetaBBO method's learning ability. Detailed results are provided in Appendix II.

VI. KEY DESIGN STRATEGIES

A. Neural Network Design

Four common neural network architectures are frequently adopted: 1) MLP, 2) RNN and LSTM, 3) temporal dependency Transformer, and 4) spatial dependency Transformer, as illustrated in Fig. 8 from left to right. The basic MLP (leftmost in Fig. 8) is widely used in existing works due to its simplicity and efficiency in training and inference. However, the MLP is limited in analyzing the temporal and data batch dependencies within the low-level BBO process. We next introduce novel designs that address these limitations.

1) **Temporal Dependency Architectures:** The low-level BBO process involves iterative optimization over T generations. A basic MLP-based policy may struggle to effectively leverage historical information along the optimization trajectory. Then, an intuitive solution is to introduce architectures that support temporal sequence modeling. To this end, works such as RNN-OI [44], RNN-Opt [45], and LTO-POMDP [47] introduce RNNs and LSTMs [149], which integrate historical optimization information into hidden representations and combine it with the current optimization state (shown in the second part of Fig. 8). While these approaches improve learning effectiveness by incorporating historical information, training on long horizons (often involving hundreds of generations) using RNN/LSTM can be challenging due to the inherent issues of gradient vanishing or explosion. Subsequent works, such as MELBA [126], RIBBO [43], and EvoTF [42], address this limitation by leveraging Transformer architectures for better long-sequence modeling. The common workflow in these works is illustrated in the third part of Fig. 8, where a trajectory of historical optimization states is processed by the Transformer to inform the next step in algorithm design.

2) **Spatial Dependency Architectures:** In addition to temporal properties, a key characteristic of EC is its population-based search manner. Recent MetaBBO methods tailor algorithmic components for each individual in the population, maximizing flexibility for low-level optimization. As illustrated in the rightmost part of Fig. 8, works such as LGA [49], LES [48], B2Opt [132], GLEET [38], RLEMDO [56], and GLHF [46] construct optimization state features as a collection of individual optimization states and leverage the Transformer's attention mechanism to enhance information

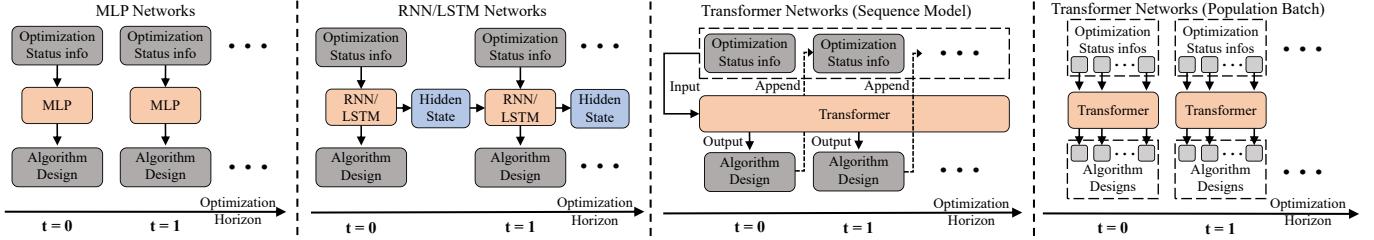


Fig. 8. The workflow of different neural networks used in existing MetaBBO works: MLP, RNN/LSTM and Transformer architectures.

sharing across the population of individuals. For instance, GLEET [38] proposes a novel Transformer-style network that includes a “fully informed encoder” and an “exploration-exploitation decoder”. The encoder promotes information sharing by applying self-attention to the state features of all individuals. The decoder then decodes the hyperparameter values for each individual specifically. Besides mining the spatial dependency in solution space, a recent work termed as TabPFN [208] leverages attention mechanism and bayesian prior to discover the spatial dependency in problem instance space, which improves the performance in per-instance algorithm selection [209].

B. State Feature Design

A key component for MetaBBO’s generalization across diverse optimization problems is the state feature extraction function $sf(\cdot)$. We identify three types of features: a) *Problem identification features*, which captures the landscape properties of the target problem. b) *Population profiling features*, which describes the distribution of solutions in the low-level BBO process. c) *Optimization progress features*, which tracks improvements in the solution evaluations at each step. Next, we introduce common practices for preparing these features.

1) **Problem Identification Features:** To identify the target optimization problem, the Exploratory Landscape Analysis (ELA) framework [210] is widely used for single-objective optimization problems. ELA includes six groups of metrics, such as local search, skewness of the objective space, and approximated curvature (both first and second-order), which provide a comprehensive summary of the problem’s landscape properties. To compute ELA features, a large number of points are sampled from the BBO problem and used to compute the features. For example, linear and quadratic models are fitted to the sampled points and their objective values, and the resulting model parameters have been shown to be useful for differentiating different problems. For multi-objective optimization, the features can be obtained by decomposing the problem into single-objective sub-problems and conducting single-objective feature analysis techniques [147].

2) **Population Profiling Features:** In an optimization problem, the solution population can converge to different regions of the fitness landscape. MetaBBO aims to dynamically adapt algorithm designs to help the low-level optimizer adjust to these diverse regions. To analyze the distribution of the population, Fitness Landscape Analysis (FLA) [164] is commonly used, providing various indicators, such as Fitness Distance Correlation [211], Ruggedness of Information Entropy [212],

Auto-Correlation Function [213], Dispersion [214], Negative Slope Coefficient [215], and Average Neutral Ratio [216]. Some of these indicators measure local landscape properties based on the population’s location in the fitness space. Population profiling features complement problem identification features, providing the meta-level policy with a more accurate optimization state for specific optimization steps.

3) **Optimization Progress Features:** Optimization progress features further complement ELA and FLA features by providing the meta-level policy with additional information on objective evaluation-related properties, such as the consumed FEs, and the distribution of the current population along with the objective values. These features track the improvement and convergence of the population. Interestingly, recent MetaBBO works like DE-DDQN [34], RLEPSO [93], and GLEET [38] have found that optimization progress features alone can be sufficient for learning a generalizable meta-level policy. A key reason is that computing ELA/FLA features consumes additional FEs, which reduces the learning steps available for the meta-level policy, thus degrading both learning effectiveness and final optimization performance.

C. Training Distribution Design

The training problem set is crucial for learning a generalizable meta-level policy, with diversity being a key factor. Early works like RNN-OI [44] were trained on a limited set of instances from the CoCo-BBOB test suite. As shown in Fig. 7, a narrow training set leads to poor generalization. To enhance the diversity of the training set, two main methodologies are commonly used in existing MetaBBO approaches.

1) **Augmenting Existing Benchmarks:** Standard BBO benchmarks include the CoCo-BBOB [196], [217] and CEC BBOB-Competition [195], [218] testsuites, which contain approximately 20–30 synthetic functions with various properties like multimodality, non-separability, and non-convexity. Most MetaBBO works augment these testsuites by mathematical transformations: given a D-dimensional function instance $f(x) : \mathbb{R}^D \rightarrow \mathbb{R}$, it can be transformed to a new instance $f'(x) = f(M^T(x - o))$, where $M \in \mathbb{R}^{D \times D}$ is a rotation matrix and $o \in \mathbb{R}^D$ is an offset to the optimal. For example, recent works like GLEET [38] and RL-DAS [40] apply random combinations of shifts and rotations on the CEC2021 test suite [195], generating thousands of synthetic instances and significantly improving the generalization performance of the learned meta-level policy.

2) **Constructing New Benchmarks:** While augmenting existing standard synthetic functions with shift and rotation

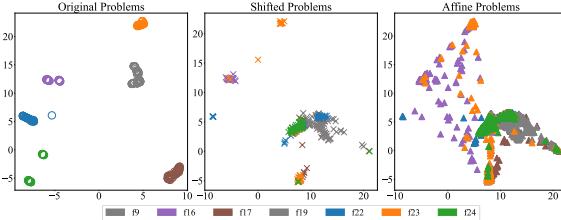


Fig. 9. The projected 2D ELA distributions of **Left:** the original BBOB problems; **Middle:** the BBOB problems with shifted optimum; **Right:** the BBOB problems generated by MA-BBOB [202].

transformations improves generalization, there is still room for greater diversity in the problem set. To illustrate this, we show the 2D projection of the ELA distribution for some CoCo-BBOB problem instances and their transformed counterparts in the left and middle of Fig. 9. The results reveal that the transformations introduce some diversity, but the improvement is still limited. There also exist a few studies that focus on the sensitivity of the landscape features to represent the benchmark functions, such as Škvorec et al. [219] reveal the importance of sampling methods in the invariance of landscape features, and Prager et al. [220] analyze the the sensitivity of landscape features to absolute objective values. More effective approaches are expected to generate novel benchmarks. The recent work MA-BBOB [202] demonstrates that affine combinations of existing synthetic functions can create more diverse instances. This is shown in the right part of Fig. 9, where the instances generated by MA-BBOB covers wider feature space.

D. Meta-Objective Design

The meta-objective $J(\theta)$ in MetaBBO represents the expected accumulated performance gain $\text{perf}(\cdot)$ over the problems in the training set. In existing MetaBBO works, $\text{perf}(\cdot)$ is typically tied to the objective values of the solution population, guiding the meta-level policy toward improved optimization performance. An intuitive approach is to use an indicator function: if performance improves between two optimization steps, a positive reward is given; otherwise, a negative or zero reward is assigned. This approach is widely used in early MetaBBO works such as DE-DDQN [34], QLPSO [36], and MARLwCMA [163].

1) **Scale Normalization:** Nevertheless, this basic approach can pose challenges when aiming to precisely assess performance improvements, which in turn could affect the adaptability of the learned policy. An alternative method involves computing $\text{perf}(\cdot)$ directly by determining the reduction in the objective value, expressed as $\Delta f^t = f^{*,t-1} - f^{*,t}$. However, directly using this absolute objective value descent may lead to unstable learning due to differing objective value scales across various optimization problems. To mitigate this issue, recent MetaBBO works apply normalization to the objective descent:

$$\text{perf}(\cdot, t) = \frac{f^{*,t-1} - f^{*,t}}{f^{*,1} - f^*} \quad (11)$$

where $f^{*,1}$ denotes the objective value of the best solution in the initialized population, and f^* represents the optimum of f . In practice, f^* is unknown because f is a black-box

function. However, it can be approximated by an efficient BBO algorithm running in advance.

2) **Sparse Reward Handling:** The difficulty of the low-level BBO process increases over time. Initially, the objective value may decrease rapidly, but later, the descent slows as convergence approaches, often resulting in a sparse reward issue in learning systems. This can mislead the learning of the meta-level policy, causing it to favor sub-optimal algorithm designs that focus primarily on the early stages of optimization. To address this, recent works introduce an adaptive performance metric with a scale factor $\lambda(t)$ to Eq. (11) to amplify the performance improvement in the later optimization stages:

$$\text{perf}(\cdot, t) = \lambda(t) \times \frac{f^{*,t-1} - f^{*,t}}{f^{*,1} - f^*} \quad (12)$$

where the scale factor $\lambda(t)$ is a incremental function of the optimization step t . For instance, in MADAC [97], $\lambda(t)$ is $2f^* - f^{*,t-1} - f^{*,t}$. However, ablation studies in RL-DAS [40], GLEET [38], RIBBO [43], and GLHF [46] suggest that using the unscaled, exact performance improvement metric without a scaling factor may be more effective. This underscores the variability in scaling methods' effectiveness across different MetaBBO tasks, warranting further investigation.

VII. VISION FOR THE FIELD

A. Generalization Toward Task Mixtures

A promising direction is the generalization toward a mixture of tasks. While MetaBBO works have explored various aspects of model generalization, the evaluation and analysis we provide in previous sections outline potential improvement through advanced learning techniques, e.g., transfer learning [221] and multitask learning [222].

First, existing works often focus on algorithm design for specific optimizers. For instance, methods like LDE [37] and GLHF [46] are designed for algorithm configuration or imitation tasks, but primarily with basic DE. This narrow focus might lead to uncertain performance when applying these methods to other optimizers. A more effective approach would be to create a higher-level framework that defines MetaBBO tasks across multiple optimizers, establishing a multitask design space. Developing a universal modularization paradigm for various optimizers could allow training a meta-level policy that generalizes well across tasks.

Additionally, existing works focus exclusively on a single specific problem type. Separate policies are trained for each task type, leading to increased complexity. This outlines an opportunity to develop a unified agent capable of engaging in automatic algorithm design that adapts to various problem types. This method not only streamlines the optimization process but also aligns more closely with real-world scenarios, where practitioners frequently encounter a diverse array of problems. To overcome the limitations of existing methods, a universal problem representation system is essential to bolster the generalization across diverse problem domains.

B. Fully End-to-End Autonomy

The main motivation behind MetaBBO is to reduce the labor-intensive need for expert consultation by offering a

general optimization framework. However, existing MetaBBO approaches still introduce design elements that rely on expert knowledge to enhance performance. This reliance typically involves: 1) the low-level optimization state s , often hand-crafted as a feature vector to represent problem properties or optimization progress; and 2) the meta-objective, which is mostly developer-defined, introducing subjectivity. While initial efforts have been made to automate feature extraction using neural networks [123], [223], [224] and employ model-based RL to learn the meta-objective objectively [108], further systematic studies are needed. Besides, MetaBBO focuses on designing algorithms in isolation, assuming the optimization problem is predefined and ready for evaluation. In reality, the initial step often involves formulating the problem, either through manual model construction [202] or data-driven methods [124], [225]–[227]. This disconnect reveals a major gap in the optimization process. A more integrated approach would involve objective formulation learning, automatic feature extraction and customized algorithm design. Developing a cohesive pipeline for these steps offers a promising direction for advancing optimization and improving problem-solving in practical applications.

C. Smarter Integration of LLMs

Although existing MetaBBO-ICL methods have shown possibility of leveraging general LLMs to assist algorithm design tasks, they still face challenges considering computational efficiency, code quality, interpretability, quality & diversity control and reasoning ability for complex optimization tasks. It has been observed MetaBBO-ICL methods (e.g., OPRO [127] and LMEA [128]) have to consume 100-200k tokens during the iterative in-context conversation with LLMs to achieve certain optimization performance [41]. Such efficiency issue is addressed by instruction-tuning general LLMs in [41]. However, the results in this work show that the error rate of the output code is 5% - 10%, which degrades the final performance. Considering the interpretability, since works such as OPRO implicitly prompt LLMs to learn the intent for optimization performance improvement, the logic behind the LLMs reasoning workflow is still black-box to us, which hinders the feedback loop between LLMs and human experts. Besides, existing MetaBBO-ICL methods merely focus on the quality & diversity control within the optimization process. Future works must allow explicit commands for such exploration-exploitation tradeoff [228]. Last but not least, recent researches such as [229], [230] indicate the fragility of mathematical reasoning in existing LLMs, which further questions those LLM-assisted MetaBBO approaches.

This suggests two promising directions: First is the automated MetaBBO workflow search, leveraging LLMs for designing MetaBBO workflow through code generation and function search. Designing a learning system like MetaBBO is inherently challenging, as it requires considerable expertise. By providing LLMs with foundational principles of MetaBBO, the chain of thought within the models may uncover novel paradigms. Second, enhancing the semantic understanding of LLMs regarding optimization processes, terminologies, programming logics, problem descriptions would significantly

elevate their expertise. To achieve this, an interesting direction is to develop symbolic language tailored to optimization domain, establishing a comprehensive grammar system and accumulating sufficient use cases to train a foundation model specifically for optimization. Third, since LLM-based MetaBBO approaches implicitly requires (massive) pre-training, an imminent future work for MetaBBO is to construct fair benchmarking standards and platforms that could compare MetaBBO approaches with traditional BBO methods fairly in both computational cost and optimization performance.

VIII. CONCLUSION

In this survey, we provide a comprehensive review of recent advancements in MetaBBO. As a novel research avenue within the BBO and EC communities, MetaBBO offers a promising paradigm for automated algorithm design. Through a bi-level data-driven learning framework, MetaBBO is capable of meta-learning effective neural network-based meta-level policies. These policies assist with algorithm selection and algorithm configuration for a given low-level optimizer, as well as to imitate or generate optimizers with certain flexibility.

Our review begins with the mathematical definition of MetaBBO, clarifying its bi-level control workflow. Next, we systematically explore four main algorithm design tasks where MetaBBO excels: AS, AC, SM, and AG. Following the discussion of these tasks, we examine four methodologies of training MetaBBO: SL, RL, NE, and ICL. We hope these two parts will provide readers with a clear roadmap to quickly locate their interested MetaBBO methods. Furthermore, we provide comprehensive benchmarking on latest MetaBBO approaches considering their computational efficiency, optimization performance and learning capability, revealing current works' significance and limitations. Subsequent in-depth analysis on some core designs of MetaBBO: the neural network architecture, optimization state feature extraction mechanism, training problem distribution, and meta-objective design. These insights offer practical guidelines for researchers and practitioners aiming to develop more effective and efficient MetaBBO methods. At last, we propose several interesting and open-ended future directions for MetaBBO research, encouraging further exploration and innovation in this promising field.

REFERENCES

- [1] Y. Jin and J. Branke, “Evolutionary optimization in uncertain environments-a survey,” *TEC*, 2005.
- [2] S. Sun, Z. Cao, H. Zhu, and J. Zhao, “A survey of optimization methods from machine learning perspective,” *TC*, 2019.
- [3] M. H. Yar, V. Rahmati, and H. R. D. Oskouei, “A survey on evolutionary computation: Methods and their applications in engineering,” *Mod. Appl. Sci.*, 2016.
- [4] A. Ponsich, A. L. Jaimes, and C. A. C. Coello, “A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications,” *TEC*, 2012.
- [5] A. M. Gopakumar, P. V. Balachandran, D. Xue, J. E. Gubernatis, and T. Lookman, “Multi-objective optimization for materials discovery via adaptive design,” *Sci. Rep.*, 2018.
- [6] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, 2006.
- [7] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution strategies as a scalable alternative to reinforcement learning,” *arXiv preprint*, 2017.

- [8] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint*, 2016.
- [9] D. P. Kingma, “Adam: A method for stochastic optimization,” *arXiv preprint*, 2014.
- [10] D. C. Liu and J. Nocedal, “On the limited memory BFGS method for large scale optimization,” *Mathematical Programming*, 1989.
- [11] P. A. Vinkar, “Evolutionary algorithms: A critical review and its future prospects,” in *ICGTSPICC*, 2016.
- [12] J. J. Liang, B. Y. Qu, and P. N. Suganthan, “Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization,” Tech. Rep., 2013.
- [13] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, S. Tiwari *et al.*, “Multiobjective optimization test instances for the CEC 2009 special session and competition,” Tech. Rep., 2008.
- [14] S. Das, S. Maity, B.-Y. Qu, and P. N. Suganthan, “Real-parameter evolutionary multimodal optimization—a survey of the state-of-the-art,” *Swarm Evol. Comput.*, 2011.
- [15] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, “Benchmark functions for the CEC’2010 special session and competition on large-scale global optimization,” Tech. Rep., 2007.
- [16] Q. Xu, N. Wang, L. Wang, W. Li, and Q. Sun, “Multi-task optimization and multi-task evolutionary computation in the past five years: A brief review,” *Mathematics*, 2021.
- [17] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *TEC*, 1997.
- [18] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” *NeurIPS*, 2011.
- [19] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *ACM SIGKDD*, 2019.
- [20] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, “Smac3: A versatile bayesian optimization package for hyperparameter optimization,” *JMLR*, 2022.
- [21] P. Cowling, G. Kendall, and E. Soubeiga, “A hyperheuristic approach to scheduling a sales summit,” in *PATAT*, 2000.
- [22] E. K. Burke, M. R. Hyde, and G. Kendall, “Grammatical evolution of local search heuristics,” *TEC*, 2011.
- [23] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, “A genetic programming hyperheuristic approach for evolving two dimensional strip packing heuristics,” *ASAP*, 2008.
- [24] M. Srinivas and L. M. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” *TSMC*, 1994.
- [25] J. Zhang and A. C. Sanderson, “JADE: Adaptive differential evolution with optional external archive,” *TEC*, 2009.
- [26] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, “Adaptive particle swarm optimization,” *TSMC*, 2009.
- [27] 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,” *TC*, 2015.
- [28] N. Hansen, “The CMA evolution strategy: A tutorial,” *arXiv preprint*, 2016.
- [29] R. Tanabe and A. Fukunaga, “Success-history based parameter adaptation for differential evolution,” in *CEC*, 2013.
- [30] R. Tanabe and A. S. Fukunaga, “Improving the search performance of shade using linear population size reduction,” in *CEC*, 2014.
- [31] V. Stavrov, S. Akhmedova, and E. Semenkin, “Nl-shade-lbc algorithm with linear parameter adaptation bias change for CEC 2022 numerical optimization,” in *CEC*, 2022.
- [32] 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,” *NeurIPS*, 2024.
- [33] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *ICML*, 2017.
- [34] M. Sharma, A. Komninos, M. López-Ibáñez, and D. Kazakov, “Deep reinforcement learning based parameter control in differential evolution,” in *GECCO*, 2019.
- [35] Z. Tan, Y. Tang, K. Li, H. Huang, and S. Luo, “Differential evolution with hybrid parameters and mutation strategies based on reinforcement learning,” *Swarm Evol. Comput.*, 2022.
- [36] Y. Xu and D. Pi, “A reinforcement learning-based communication topology in particle swarm optimization,” *Neural Comput. Appl.*, 2020.
- [37] J. Sun, X. Liu, T. Bäck, and Z. Xu, “Learning adaptive differential evolution algorithm from optimization experiences by policy gradient,” *TEC*, 2021.
- [38] Z. Ma, J. Chen, H. Guo, Y. Ma, and Y.-J. Gong, “Auto-configuring exploration-exploitation tradeoff in evolutionary computation via deep reinforcement learning,” in *GECCO*, 2024.
- [39] Z. Tan and K. Li, “Differential evolution with mixed mutation strategy based on deep reinforcement learning,” *Appl. Soft Comput.*, 2021.
- [40] H. Guo, Y. Ma, Z. Ma, J. Chen, X. Zhang, Z. Cao, J. Zhang, and Y.-J. Gong, “Deep reinforcement learning for dynamic algorithm selection: A proof-of-principle study on differential evolution,” *TSMC*, 2024.
- [41] Z. Ma, H. Guo, J. Chen, G. Peng, Z. Cao, Y. Ma, and Y.-J. Gong, “LLaMoCo: Instruction tuning of large language models for optimization code generation,” *arXiv preprint*, 2024.
- [42] R. Lange, Y. Tian, and Y. Tang, “Evolution transformer: In-context evolutionary optimization,” in *GECCO*, 2024.
- [43] L. Song, C. Gao, K. Xue, C. Wu, D. Li, J. Hao, Z. Zhang, and C. Qian, “Reinforced in-context black-box optimization,” *arXiv preprint*, 2024.
- [44] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. Freitas, “Learning to learn without gradient descent by gradient descent,” in *ICML*, 2017.
- [45] V. TV, P. Malhotra, J. Narwariya, L. Vig, and G. Shroff, “Meta-learning for black-box optimization,” in *ECML PKDD*, 2019.
- [46] X. Li, K. Wu, Y. B. Li, X. Zhang, H. Wang, and J. Liu, “GLHF: General learned evolutionary algorithm via hyper functions,” *arXiv preprint*, 2024.
- [47] H. S. Gomes, B. Léger, and C. Gagné, “Meta learning black-box population-based optimizers,” *arXiv preprint*, 2021.
- [48] R. Lange, T. Schaul, Y. Chen, T. Zahavy, V. Dalibard, C. Lu, S. Singh, and S. Flennerhag, “Discovering evolution strategies via meta-black-box optimization,” in *GECCO*, 2023.
- [49] R. Lange, T. Schaul, Y. Chen, C. Lu, T. Zahavy, V. Dalibard, and S. Flennerhag, “Discovering attention-based genetic algorithms via meta-black-box optimization,” in *GECCO*, 2023.
- [50] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, “Connecting large language models with evolutionary algorithms yields powerful prompt optimizers,” in *ICLR*, 2024.
- [51] R. Lange, Y. Tian, and Y. Tang, “Large language models as evolution strategies,” in *GECCO*, 2024.
- [52] 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 *ICML*, 2024.
- [53] A. AhmadiTeshnizi, W. Gao, and M. Udell, “OptiMUS: Scalable optimization modeling with (MI) LP solvers and large language models,” in *ICML*, 2024.
- [54] F. Tahernezhad-Javazm, D. Rankin, N. D. Bois, A. E. Smith, and D. Coyle, “R2 indicator and deep reinforcement learning enhanced adaptive multi-objective evolutionary algorithm,” *arXiv preprint*, 2024.
- [55] J. Wang, Y. Zheng, Z. Zhang, H. Peng, and H. Wang, “A novel multi-state reinforcement learning-based multi-objective evolutionary algorithm,” *Inf. Sci.*, 2024.
- [56] H. Lian, Z. Ma, H. Guo, T. Huang, and Y.-J. Gong, “Rlemmo: Evolutionary multimodal optimization assisted by deep reinforcement learning,” in *GECCO*, 2024.
- [57] S.-H. Wu, Y. Huang, X. Wu, L. Feng, Z.-H. Zhan, and K. C. Tan, “Learning to transfer for evolutionary multitasking,” *arXiv preprint*, 2024.
- [58] Y. Huang, X. Lv, S. Wu, J. Wu, L. Feng, and K. C. Tan, “Advancing automated knowledge transfer in evolutionary multitasking via large language models,” *arXiv preprint*, 2024.
- [59] Q. Zhao, Q. Duan, B. Yan, S. Cheng, and Y. Shi, “Automated design of metaheuristic algorithms: A survey,” *TMLR*, 2024.
- [60] X. Wu, S.-h. Wu, J. Wu, L. Feng, and K. C. Tan, “Evolutionary computation in the era of large language model: Survey and roadmap,” *arXiv preprint*, 2024.
- [61] T. Stützle and M. López-Ibáñez, “Automated design of metaheuristic algorithms,” *Handbook of metaheuristics*, 2019.
- [62] M. M. Drugan, “Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms,” *Swarm Evol. Comput.*, 2019.
- [63] M. Chernigovskaya, A. Kharitonov, and K. Turowski, “A recent publications survey on reinforcement learning for selecting parameters of meta-heuristic and machine learning algorithms,” in *CLOSER*, 2023.
- [64] Y. Song, Y. Wu, Y. Guo, R. Yan, P. N. Suganthan, Y. Zhang, W. Pedrycz, S. Das, R. Mallipeddi, O. S. Ajani *et al.*, “Reinforcement learning-assisted evolutionary algorithm: A survey and research opportunities,” *Swarm Evol. Comput.*, 2024.
- [65] P. Li, J. Hao, H. Tang, X. Fu, Y. Zhen, and K. Tang, “Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey on hybrid algorithms,” *TEC*, 2024.

- [66] R. S. Sutton, “Reinforcement learning: An introduction,” *A Bradford Book*, 2018.
- [67] S. Thrun and L. Pratt, “Learning to learn: Introduction and overview,” in *Learning to learn*, 1998.
- [68] J. Schmidhuber, “Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta... hook,” Ph.D. dissertation, Technische Universität München, 1987.
- [69] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei, “On the optimization of a synaptic learning rule,” in *Optimality in Biological and Artificial Networks?*, 2013.
- [70] B. Zoph, “Neural architecture search with reinforcement learning,” *arXiv preprint*, 2016.
- [71] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, “Learning to learn by gradient descent by gradient descent,” *NeurIPS*, 2016.
- [72] W. Kool, H. van Hoof, and M. Welling, “Attention, learn to solve routing problems!” in *ICLR*, 2019.
- [73] R. Storn and K. Price, “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces,” *J. Glob. Optim.*, 1997.
- [74] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart, “Hyper-heuristics: learning to combine simple heuristics in bin-packing problems,” in *GECCO*, 2002.
- [75] H.-L. Fang, P. Ross, and D. Corne, *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*, 1993.
- [76] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, “Exploring hyper-heuristic methodologies with genetic programming,” *Computational intelligence: Collaboration, fusion and emergence*, 2009.
- [77] K. A. Smith-Miles, “Towards insightful algorithm selection for optimisation using meta-learning concepts,” in *IJCNN*, 2008.
- [78] J. Y. Kanda, A. C. de Carvalho, E. R. Hruschka, and C. Soares, “Using meta-learning to recommend meta-heuristics for the traveling salesman problem,” in *ICML*, 2011.
- [79] Y. Tian, S. Peng, T. Rodemann, X. Zhang, and Y. Jin, “Automated selection of evolutionary multi-objective optimization algorithms,” in *SSCI*, 2019.
- [80] A. E. Gutierrez-Rodríguez, S. E. Conant-Pablos, J. C. Ortiz-Bayliss, and H. Terashima-Marín, “Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning,” *Expert Syst. Appl.*, 2019.
- [81] Y. Tian, S. Peng, X. Zhang, T. Rodemann, K. C. Tan, and Y. Jin, “A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks,” *TAI*, 2020.
- [82] Y. Li, J. Liang, K. Yu, K. Chen, Y. Guo, C. Yue, and L. Zhang, “Adaptive local landscape feature vector for problem classification and algorithm selection,” *Appl. Soft Comput.*, 2022.
- [83] X. Wu, Y. Zhong, J. Wu, B. Jiang, and K. C. Tan, “Large language model-enhanced algorithm selection: Towards comprehensive algorithm representation,” in *IJCAI*, 2024.
- [84] N. Zhu, F. Zhao, and J. Cao, “A hyperheuristic and reinforcement learning guided meta-heuristic algorithm recommendation,” in *CSCWD*, 2024.
- [85] G. Cenikj, G. Petelin, and T. Eftimov, “Transoptas: Transformer-based algorithm selection for single-objective optimization,” in *GECCO*, 2024.
- [86] H. Samma, C. P. Lim, and J. M. Saleh, “A new reinforcement learning-based memetic particle swarm optimizer,” *Appl. Soft Comput.*, 2016.
- [87] W. Ning, B. Guo, X. Guo, C. Li, and Y. Yan, “Reinforcement learning aided parameter control in multi-objective evolutionary algorithm based on decomposition,” *Prog. Artif. Intell.*, 2018.
- [88] Y. Liu, H. Lu, S. Cheng, and Y. Shi, “An adaptive online parameter control algorithm for particle swarm optimization based on reinforcement learning,” in *CEC*, 2019.
- [89] Z. Li, L. Shi, C. Yue, Z. Shang, and B. Qu, “Differential evolution based on reinforcement learning with fitness ranking for solving multimodal multiobjective problems,” *Swarm Evol. Comput.*, 2019.
- [90] G. Shala, A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, and F. Hutter, “Learning step-size adaptation in CMA-ES,” in *PPSN*, 2020.
- [91] Y. Huang, W. Li, F. Tian, and X. Meng, “A fitness landscape ruggedness multiobjective differential evolution algorithm with a reinforcement learning strategy,” *Appl. Soft Comput.*, 2020.
- [92] Z. Hu, W. Gong, and S. Li, “Reinforcement learning-based differential evolution for parameters extraction of photovoltaic models,” *Energy Rep.*, 2021.
- [93] S. Yin, Y. Liu, G. Gong, H. Lu, and W. Li, “RLEPSO: Reinforcement learning based ensemble particle swarm optimizer,” in *ACAI*, 2021.
- [94] T. N. Huynh, D. T. Do, and J. Lee, “Q-learning-based parameter control in differential evolution for structural optimization,” *Appl. Soft Comput.*, 2021.
- [95] D. Wu and G. G. Wang, “Employing reinforcement learning to enhance particle swarm optimization methods,” *Eng. Optim.*, 2022.
- [96] F. Wang, X. Wang, and S. Sun, “A reinforcement learning level-based particle swarm optimization algorithm for large-scale optimization,” *Inf. Sci.*, 2022.
- [97] K. Xue, J. Xu, L. Yuan, M. Li, C. Qian, Z. Zhang, and Y. Yu, “Multi-agent dynamic algorithm configuration,” *NeurIPS*, 2022.
- [98] Z. Hu and W. Gong, “Constrained evolutionary optimization based on reinforcement learning using the objective function and constraints,” *KBS*, 2022.
- [99] Y. Tian, X. Li, H. Ma, X. Zhang, K. C. Tan, and Y. Jin, “Deep reinforcement learning based adaptive operator selection for evolutionary multi-objective optimization,” *TETCI*, 2022.
- [100] I. Fister, D. Fister, and I. Fister Jr, “Reinforcement learning-based differential evolution for global optimization,” in *Differential Evolution: From Theory to Practice*, 2022.
- [101] W. Li, P. Liang, B. Sun, Y. Sun, and Y. Huang, “Reinforcement learning-based particle swarm optimization with neighborhood differential mutation strategy,” *Swarm Evol. Comput.*, 2023.
- [102] H. Zhang, J. Sun, T. Bäck, Q. Zhang, and Z. Xu, “Controlling sequential hybrid evolutionary algorithm by q-learning [research frontier] [research frontier],” *CIM*, 2023.
- [103] X. Liu, J. Sun, Q. Zhang, Z. Wang, and Z. Xu, “Learning to learn evolutionary algorithm: A learnable differential evolution,” *TETCI*, 2023.
- [104] S. Yin, M. Jin, H. Lu, G. Gong, W. Mao, G. Chen, and W. Li, “Reinforcement-learning-based parameter adaptation method for particle swarm optimization,” *Complex Intell. Syst.*, 2023.
- [105] X. Meng, H. Li, and A. Chen, “Multi-strategy self-learning particle swarm optimization algorithm based on reinforcement learning,” *MBE*, 2023.
- [106] Q. Yang, S.-C. Chu, J.-S. Pan, J.-H. Chou, and J. Watada, “Dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning for optimization problems,” *Complex Intell. Syst.*, 2024.
- [107] Y. Han, H. Peng, C. Mei, L. Cao, C. Deng, H. Wang, and Z. Wu, “Multi-strategy multi-objective differential evolutionary algorithm with reinforcement learning,” *KBS*, 2023.
- [108] F. Zhao, Z. Wang, L. Wang, T. Xu, N. Zhu *et al.*, “A multi-agent reinforcement learning driven artificial bee colony algorithm with the central controller,” *Expert Syst. Appl.*, 2023.
- [109] Z. Hu, W. Gong, W. Pedrycz, and Y. Li, “Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization,” *Swarm Evol. Comput.*, 2023.
- [110] T. Li, Y. Meng, and L. Tang, “Scheduling of continuous annealing with a multi-objective differential evolution algorithm based on deep reinforcement learning,” *TASE*, 2023.
- [111] L. Peng, Z. Yuan, G. Dai, M. Wang, and Z. Tang, “Reinforcement learning-based hybrid differential evolution for global optimization of interplanetary trajectory design,” *Swarm Evol. Comput.*, 2023.
- [112] X. Yu, P. Xu, F. Wang, and X. Wang, “Reinforcement learning-based differential evolution algorithm for constrained multi-objective optimization problems,” *EAAI*, 2024.
- [113] J. Hong, B. Shen, and A. Pan, “A reinforcement learning-based neighborhood search operator for multi-modal optimization and its applications,” *Expert Syst. Appl.*, 2024.
- [114] H. Zhang, J. Shi, J. Sun, A. W. Mohamed, and Z. Xu, “A gradient-based method for differential evolution parameter control by smoothing,” in *GECCO*, 2024.
- [115] H. Zhang, J. Sun, T. Bäck, and Z. Xu, “Learning to select the recombination operator for derivative-free optimization,” *Sci. China Math.*, 2024.
- [116] Z. Liao, Q. Pang, and Q. Gu, “Differential evolution based on strategy adaptation and deep reinforcement learning for multimodal optimization problems,” *Swarm Evol. Comput.*, 2024.
- [117] X. Wang, F. Wang, Q. He, and Y. Guo, “A multi-swarm optimizer with a reinforcement learning mechanism for large-scale optimization,” *Swarm Evol. Comput.*, 2024.
- [118] A. Bolufé-Röhler and B. Xu, “Deep reinforcement learning for smart restarts in exploration-only exploitation-only hybrid metaheuristics,” in *MIC*, 2024.

- [119] J. Pei, J. Liu, and Y. Mei, "Learning from offline and online experiences: A hybrid adaptive operator selection framework," in *GECCO*, 2024.
- [120] 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 *AAAI*, 2025.
- [121] M. Chen, C. Feng, and R. Cheng, "Metade: Evolving differential evolution by differential evolution," *TEC*, 2025.
- [122] S. Shao, Y. Tian, Y. Zhang, and X. Zhang, "Knowledge learning-based dimensionality reduction for solving large-scale sparse multiobjective optimization problems," *TC*, 2025.
- [123] H. Guo, S. Ma, Z. Huang, Y. Hu, Z. Ma, X. Zhang, and Y.-J. Gong, "Reinforcement learning-based self-adaptive differential evolution through automated landscape feature learning," in *GECCO*, 2023.
- [124] Z. Ma, Z. Huang, J. Chen, Z. Cao, and Y.-J. Gong, "Surrogate learning in meta-black-box optimization: A preliminary study," in *GECCO*, 2025.
- [125] H. Guo, W. Qiu, Z. Ma, X. Zhang, J. Zhang, and Y.-J. Gong, "Advancing cma-es with learning-based cooperative coevolution for scalable optimization," *arXiv preprint*, 2025.
- [126] S. Chaybouli, L. Dos Santos, C. Malherbe, and A. Virmaux, "Meta-learning of black-box solvers using deep reinforcement learning," in *NeurIPS*, 2022.
- [127] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," 2024.
- [128] S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong, "Large language models as evolutionary optimizers," in *CEC*, 2024.
- [129] F. Liu, X. Lin, Z. Wang, S. Yao, X. Tong, M. Yuan, and Q. Zhang, "Large language model for multi-objective evolutionary optimization," *arXiv preprint*, 2023.
- [130] J. Lehman, J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley, "Evolution through large models," in *Handbook of Evolutionary Machine Learning*, 2023.
- [131] P.-F. Guo, Y.-H. Chen, Y.-D. Tsai, and S.-D. Lin, "Towards optimizing with large language models," *arXiv preprint*, 2023.
- [132] X. Li, K. Wu, X. Zhang, H. Wang, and J. Liu, "B2Opt: Learning to optimize black-box optimization with little budget," *arXiv preprint*, 2023.
- [133] S. Brahmachary, S. M. Joshi, A. Panda, K. Koneripalli, A. K. Sagotra, H. Patel, A. Sharma, A. D. Jagtap, and K. Kalyanaraman, "Large language model-based evolutionary optimizer: Reasoning with elitism," *arXiv preprint*, 2024.
- [134] Z. Wang, S. Liu, J. Chen, and K. C. Tan, "Large language model-aided evolutionary search for constrained multiobjective optimization," in *ICIC*, 2024.
- [135] W. Yi, R. Qu, L. Jiao, and B. Niu, "Automated design of metaheuristics using reinforcement learning within a novel general search framework," *TEC*, 2022.
- [136] F. Liu, X. Tong, M. Yuan, and Q. Zhang, "Algorithm evolution using large language model," *arXiv preprint*, 2023.
- [137] J. Chen, Z. Ma, H. Guo, Y. Ma, J. Zhang, and Y.-J. Gong, "SYMBOL: Generating flexible black-box optimizers through symbolic equation learning," in *ICLR*, 2024.
- [138] N. van Stein and T. Bäck, "LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics," *arXiv preprint*, 2024.
- [139] Y. Huang, S. Wu, W. Zhang, J. Wu, L. Feng, and K. C. Tan, "Autonomous multi-objective optimization using large language model," *arXiv preprint*, 2024.
- [140] R. Zhang, F. Liu, X. Lin, Z. Wang, Z. Lu, and Q. Zhang, "Understanding the importance of evolutionary search in automated heuristic design with large language models," in *PPSN*, 2024.
- [141] Q. Zhao, T. Liu, B. Yan, Q. Duan, J. Yang, and Y. Shi, "Automated metaheuristic algorithm design with autoregressive learning," *arXiv preprint*, 2024.
- [142] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *ECJ*, 2019.
- [143] G. Cenikj, A. Nikolicj, G. Petelin, N. van Stein, C. Doerr, and T. Eftimov, "A survey of meta-features used for automated selection of algorithms for black-box single-objective continuous optimization," *arXiv preprint*, 2024.
- [144] J. R. Rice, "The algorithm selection problem," in *Advances in Computers*, 1976.
- [145] B. Bischi, O. Mersmann, H. Trautmann, and M. Preuß, "Algorithm selection based on exploratory landscape analysis and cost-sensitive learning," in *GECCO*, 2012.
- [146] P. Kerschke and H. Trautmann, "Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning," *ECJ*, 2019.
- [147] A. Liefooghe, "Landscape analysis and heuristic search for multi-objective optimization," Ph.D. dissertation, Université de Lille, 2022.
- [148] A. Liefooghe, S. Verel, B. Lacroix, A.-C. Zăvoianu, and J. McCall, "Landscape features and automated algorithm selection for multi-objective interpolated continuous optimisation problems," in *GECCO*, 2021.
- [149] S. Hochreiter, "Long short-term memory," *Neural Comput.*, 1997.
- [150] Q. Renau and E. Hart, "On the utility of probing trajectories for algorithm-selection," in *EvoAPPS*, 2024.
- [151] A. Kostovska, A. Jankovic, D. Vermetten, J. de Nobel, H. Wang, T. Eftimov, and C. Doerr, "Per-run algorithm selection with warm-starting using trajectory-based features," in *PPSN*, 2022.
- [152] A. Jankovic, D. Vermetten, A. Kostovska, J. de Nobel, T. Eftimov, and C. Doerr, "Trajectory-based algorithm selection with warm-starting," in *CEC*, 2022.
- [153] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, 2017.
- [154] L. Xu, H. Hoos, and K. Leyton-Brown, "Hydra: Automatically configuring algorithms for portfolio-based selection," in *AAAI*, 2010.
- [155] M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub, "Autofolio: An automatically configured algorithm selector," *JAIR*, 2015.
- [156] A. Kostovska, G. Cenikj, D. Vermetten, A. Jankovic, A. Nikolicj, U. Skvoric, P. Korosec, C. Doerr, and T. Eftimov, "Ps-aas: Portfolio selection for automated algorithm selection in black-box optimization," in *AutoML*, 2023.
- [157] K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter, "HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO," in *NeurIPS*, 2021.
- [158] Á. Fialho, "Adaptive operator selection for optimization," Ph.D. dissertation, Université Paris Sud-Paris XI, 2010.
- [159] S. Adriaensen, A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, and F. Hutter, "Automated dynamic algorithm configuration," *JAIR*, 2022.
- [160] S. Biswas, D. Saha, S. De, A. D. Cobb, S. Das, and B. A. Jalaian, "Improving differential evolution through bayesian hyperparameter optimization," in *CEC*, 2021.
- [161] J. Brest, M. S. Maučec, and B. Bošković, "Self-adaptive differential evolution algorithm with population size reduction for single objective bound-constrained optimization: Algorithm j21," in *CEC*, 2021.
- [162] K. M. Sallam, S. M. Elsayed, R. K. Chakrabortty, and M. J. Ryan, "Improved multi-operator differential evolution algorithm for solving unconstrained problems," in *CEC*, 2020.
- [163] ———, "Evolutionary framework with reinforcement learning-based mutation adaptation," *IEEE Access*, 2020.
- [164] S. Wright *et al.*, "The roles of mutation, inbreeding, crossbreeding, and selection in evolution," 1932.
- [165] D. Vermetten, F. Caraffini, A. V. Kononova, and T. Bäck, "Modular differential evolution," in *GECCO*, 2023.
- [166] C. L. Camacho-Villalón, M. Dorigo, and T. Stützle, "Pso-x: A component-based framework for the automatic design of particle swarm optimization algorithms," *TEC*, 2021.
- [167] S. Van Rijn, C. Doerr, and T. Bäck, "Towards an adaptive cma-es configurator," in *PPSN*, 2018.
- [168] M. V. Seiler, J. Rook, J. Heins, O. L. Preuß, J. Bossek, and H. Trautmann, "Using reinforcement learning for per-instance algorithm configuration on the TSP," in *SSCI*, 2023.
- [169] D. Karapetyan and G. Gutin, "Lin-Kernighan heuristic adaptations for the generalized traveling salesman problem," *EJOR*, 2011.
- [170] R. Tanabe and A. S. Fukunaga, "Improving the search performance of shade using linear population size reduction," in *CEC*, 2014.
- [171] A. K. Sadhu, A. Konar, T. Bhattacharjee, and S. Das, "Synergism of firefly algorithm and q-learning for robot arm path planning," *Swarm Evol. Comput.*, 2018.
- [172] J. Snoek, K. Swersky, R. Zemel, and R. Adams, "Input warping for bayesian optimization of non-stationary functions," in *ICML*, 2014.
- [173] R. Lange, Y. Tang, and Y. Tian, "Neuroevobench: Benchmarking evolutionary optimizers for deep learning applications," *NeurIPS*, 2023.
- [174] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, "Rethinking the role of demonstrations: What makes in-context learning work?" *arXiv preprint*, 2022.
- [175] B. Edmonds, "Meta-genetic programming: Co-evolving the operators of variation," *Turk. J. Elec. Eng. & Comp. Sci.*, 2001.

- [176] J. R. Woodward and J. Swan, "The automatic generation of mutation operators for genetic algorithms," in *GECCO*, 2012.
- [177] ——, "Automatically designing selection heuristics," in *GECCO*, 2011.
- [178] R. Rivers and D. R. Tauritz, "Evolving black-box search algorithms employing genetic programming," in *GECCO*, 2013.
- [179] R. Bellman, "A markovian decision process," *JMM*, 1957.
- [180] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, 1992.
- [181] H. Xia, C. Li, S. Zeng, Q. Tan, J. Wang, and S. Yang, "A reinforcement-learning-based evolutionary algorithm using solution space clustering for multimodal optimization problems," in *CEC*, 2021.
- [182] V. Mnih, "Playing atari with deep reinforcement learning," *arXiv preprint*, 2013.
- [183] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI*, 2016.
- [184] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *ICRA*, 2019.
- [185] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, 1992.
- [186] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *NeurIPS*, 1999.
- [187] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint*, 2017.
- [188] M. Gao, X. Feng, H. Yu, and X. Li, "An efficient evolutionary algorithm based on deep reinforcement learning for large-scale sparse multiobjective optimization," *Appl. Intell.*, 2023.
- [189] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, 1990.
- [190] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff, "COCO: A platform for comparing continuous optimizers in a black-box setting," *Optim. Methods Softw.*, 2021.
- [191] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," *NeurIPS*, 2007.
- [192] A. Maraval, M. Zimmer, A. Grosnit, and H. Bou Ammar, "End-to-end meta-bayesian optimisation with transformer neural processes," *NeurIPS*, 2024.
- [193] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi *et al.*, "Mathematical discoveries from program search with large language models," *Nature*, 2024.
- [194] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint*, 2023.
- [195] A. W. Mohamed, A. A. Hadi, A. K. Mohamed, P. Agrawal, A. Kumar, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2021 special session and competition on single objective bound constrained numerical optimization," *Tech. Rep.*, 2021.
- [196] U. Škvorec, T. Eftimov, and P. Korošec, "GECCO black-box optimization competitions: progress from 2009 to 2018," in *GECCO*, 2019.
- [197] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *TEC*, 2006.
- [198] X. Li, A. Engelbrecht, and M. G. Epitropakis, "Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization," *Tech. Rep.*, 2013.
- [199] C. Li, S. Yang, T.-T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H. Beyer, and P. N. Suganthan, "Benchmark generator for CEC 2009 competition on dynamic optimization," *Tech. Rep.*, 2008.
- [200] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China, "Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization," *Tech. Rep.*, 2013.
- [201] M. A. Muñoz and K. Smith-Miles, "Generating new space-filling test instances for continuous black-box optimization," *ECJ*, 2020.
- [202] D. Vermetten, F. Ye, T. Bäck, and C. Doerr, "MA-BBOB: A problem generator for black-box optimization using affine combinations and shifts," *TELO*, 2024.
- [203] F. Hutter, M. López-Ibáñez, C. Fawcett, M. Lindauer, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "AClib: A benchmark library for algorithm configuration," in *LION*, 2014.
- [204] A. Kumar, G. Wu, M. Z. Ali, R. Mallipeddi, P. N. Suganthan, and S. Das, "A test-suite of non-convex constrained optimization problems from the real-world and some baseline results," *Swarm Evol. Comput.*, 2020.
- [205] C. Doerr, H. Wang, F. Ye, S. Van Rijn, and T. Bäck, "IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics," *arXiv preprint*, 2018.
- [206] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *ICNN*, 1995.
- [207] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies—a comprehensive introduction," *Natural Computing*, 2002.
- [208] A. Kostovska, A. Janković, D. Vermetten, S. Džeroski, T. Eftimov, and C. Doerr, "Comparing algorithm selection approaches on black-box optimization problems," in *GECCO*, 2023.
- [209] A. Kostovska, A. Janković, D. Vermetten, S. Džeroski, T. Eftimov, and C. Doerr, "Comparing algorithm selection approaches on black-box optimization problems," 2023.
- [210] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory landscape analysis," in *GECCO*, 2011.
- [211] M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue, "A study of fitness distance correlation as a difficulty measure in genetic programming," *TEC*, 2005.
- [212] K. M. Malan and A. P. Engelbrecht, "Quantifying ruggedness of continuous landscapes using entropy," in *CEC*, 2009.
- [213] G. Merkuryeva and V. Bolshakov, "Benchmark fitness landscape analysis," *IJSST*, 2011.
- [214] M. Lunacek and D. Whitley, "The dispersion metric and the CMA evolution strategy," in *GECCO*, 2006.
- [215] L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, and S. Vérel, "Fitness clouds and problem hardness in genetic programming," in *GECCO*, 2004.
- [216] L. Vanneschi, P. Collard, S. Verel, M. Tomassini, Y. Pirola, and G. Mauri, "A comprehensive view of fitness landscapes with neutrality and fitness clouds," in *EuroGP*, 2007.
- [217] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking 2010: Experimental setup," Ph.D. dissertation, INRIA, 2010.
- [218] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," *Tech. Rep.*, 2005.
- [219] U. Škvorec, T. Eftimov, and P. Korošec, "The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis," in *CEC*, 2021.
- [220] R. P. Prager and H. Trautmann, "Nullifying the inherent bias of non-invariant exploratory landscape analysis features," in *EvoAPPS*, 2023.
- [221] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big Data*, 2016.
- [222] Y. Zhang and Q. Yang, "A survey on multi-task learning," *TKDE*, 2021.
- [223] M. V. Seiler, P. Kerschke, and H. Trautmann, "Deep-ELA: Deep exploratory landscape analysis with self-supervised pretrained transformers for single-and multi-objective continuous optimization problems," *arXiv preprint*, 2024.
- [224] Z. Ma, J. Chen, H. Guo, and Y.-J. Gong, "Neural exploratory landscape analysis," *arXiv preprint*, 2024.
- [225] H.-G. Huang and Y.-J. Gong, "Contrastive learning: An alternative surrogate for offline data-driven evolutionary computation," *TEC*, 2022.
- [226] Y.-J. Gong, Y.-T. Zhong, and H.-G. Huang, "Offline data-driven optimization at scale: A cooperative coevolutionary approach," *TEC*, 2023.
- [227] Y. Zhong, X. Wang, Y. Sun, and Y.-J. Gong, "Sddobench: A benchmark for streaming data-driven optimization with concept drift," in *GECCO*, 2024.
- [228] X. Song, Y. Tian, R. T. Lange, C. Lee, Y. Tang, and Y. Chen, "Position: Leverage foundational models for black-box optimization," in *ICML*, 2024.
- [229] S. I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar, "GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models," in *ICLR*, 2025.
- [230] Y. Zhang, "Training and evaluating language models with template-based data generation," *arXiv preprint*, 2024.

APPENDIX

A. Baseline Selection Criteria

We conduct the proof-of-principle evaluation of several representative BBO and MetaBBO methods to provide community some insights of MetaBBO method and metric designs. Therefore we select baselines in the empirical evaluation showcase following two motivations:

- Since a large number of MetaBBO methods focus on bound-constrained single-objective numerical optimization and existing MetaBBO methods are not capable across different problem types, we evaluate traditional BBO and MetaBBO methods on single-objective numerical optimization testsuites as a showcase.
- To conduct a comprehensive evaluation, we attempt to cover all categories summarized in the paper, including different meta tasks (AS, AC, SM and AG), different learning paradigm (RL, SL and NE) and different low-level BBO algorithm classes (i.e., DE, PSO and ES). We did not include MetaBBO-ICL methods with LLMs because of their high time cost and API fee. But we notice the concerns raised by reviewers about the in-depth analysis on LLM-aided MetaBBO, therefore we add a classic MetaBBO-ICL method, OPRO [127], in the comparison in Section V, Part B, which shows poor performance due to the weak optimization embedded in LLM and the long LLM conversation time cost.

Following the two motivations above, we adopted a recent single-objective numerical optimization benchmark MetaBox [32] to evaluate the performance traditional EC algorithms and MetaBBO methods. For traditional EC algorithms, we select algorithms from three mainstream EC classes for BBO: DE [73], PSO [206] and ES [207]. For each of the classes, we choose one representative, highly cited and advanced algorithm as the representative, therefore for DE we select JADE [25], for PSO we choose GLPSO [27] and for ES we integrate CMA-ES [28]. Further, we include recent and advanced MetaBBO methods covering all four meta-tasks: for algorithm selection we adopt a recent RL-based method RL-DAS [40]. For algorithm configuration, we include the RL-based robust operator selection method DE-DDQN [34] in MetaBox. Three hyper-parameter optimization MetaBBO methods, RL-based LDE [37], RLEPSO [93] and NE-based LES [48], are included covering the three mainstream EC classes DE, PSO and ES respectively. Besides, a recent advanced hyper-parameter optimization framework GLEET [38] which could compatibly control both DE and PSO is also selected. For algorithm generation, a recent MetaBBO-RL method SYMBOL [137] is included. For solution manipulation, we choose two MetaBBO-SL methods: the earliest work RNN-OI [44] and a recent method GLHF [46]. For all baselines above we use the implementations integrated in MetaBox. Besides, OPRO [127] as a MetaBBO-ICL solution manipulation method is also included in the revised paper.

B. Experimental Setup

In the paper, all baselines are tested on three single-objective numerical testsuites: 10D Synthetic Testsuite with 24 problem instances, 10D Noisy Synthetic Testsuite with 24 problem instances and 12D Protein-Docking Testsuite with 280 problem instances. The “easy” postfix indicates that 75% problem instances are used for the training of MetaBBO methods and the rest 25% instances are taken as testing set. The “difficult” postfix on the contrary uses 25% for training and 75% for testing. The maximal function evaluations of all baselines are 20,000 for Synthetic and Noisy Synthetic Testsuites, 1,000 for Protein-Docking Testsuite. All baselines are trained for 1.5×10^6 learning steps and tested over 51 independent runs to ensure fairness. Other baseline hyper-parameters follow the settings in MetaBox. The experiments are conducted on Intel(R) Xeon(R) Gold 6254 CPU with 64GB RAM and NVIDIA GeForce RTX 4090 GPU. More details of train-test instance split and baseline settings can be found in the MetaBox repository. We hope this response could clear the reviewers’ concerns.

C. AEI Results

For evaluating more algorithms, we could have evaluated the AEI scores of additional traditional algorithms covering DE, PSO, GA and ES on Synthetic and Protein-docking testsuites in Table II below, as well as the AEI of MetaBBO methods in Table III.

D. Comparison on the Learning Capabilities

As a learning based paradigm, MetaBBO should also be evaluated using metrics that reflect its learning effectiveness. Next, we evaluate the MetaBBO methods using the Meta Generalization Decay (MGD) and Meta Transfer Efficiency (MTE) from MetaBox. By presenting the scores of MGD and MTE of these MetaBBO methods, the learning true learning abilities of these MetaBBO methods could be in-depth discussed. We note that this learning ability testing procedure could be generalized to other MetaBBO scenarios beyond single-objective optimization. We here use single-objective testsuites in MetaBox as a showcase for our readers.

TABLE II

THE AEI SCORES OF TRADITIONAL ALGORITHMS ON SYNTHETIC AND PROTEIN-DOCKING TESTSUITES. THE UNDERLINED ALGORITHMS ARE THE ALGORITHMS PRESENTED IN THE PAPER.

	DE	<u>JADE</u>	SHADE	MadDE	AMCDE	PSO	FIPSO	sDMSPSO	<u>GLPSO</u>	EPSO	GA	AGA	ES	<u>CMA-ES</u>	Sep-CMA-ES	IPOP-CMA-ES
Synthetic easy	16.10	13.23	14.11	12.97	12.88	14.13	12.93	9.25	13.40	13.17	12.86	13.61	14.26	22.61	23.31	22.22
	± 1.68	± 1.46	± 1.54	± 1.35	± 1.39	± 1.45	± 1.52	± 1.25	± 1.08	± 1.31	± 1.38	± 1.44	± 1.43	± 4.62	± 4.98	± 4.59
Synthetic difficult	9.45	9.90	11.24	9.70	9.83	7.21	9.69	6.77	10.22	10.01	9.22	11.65	15.23	15.98	16.50	16.24
	± 1.12	± 1.19	± 1.42	± 1.25	± 1.12	± 0.95	± 1.24	± 0.89	± 1.15	± 1.14	± 1.06	± 1.20	± 1.36	± 2.88	± 2.29	± 2.58
Protein easy	0.82	0.95	0.96	1.00	0.94	0.84	0.87	1.13	1.19	1.09	1.08	1.13	0.90	0.93	0.94	0.94
	± 0.03	± 0.03	± 0.03	± 0.04	± 0.02	± 0.03	± 0.02	± 0.04	± 0.04	± 0.03	± 0.02	± 0.03	± 0.03	± 0.04	± 0.04	± 0.04
Protein difficult	0.76	0.93	0.89	0.86	0.86	0.84	0.86	0.85	1.05	1.02	1.02	1.09	0.88	0.92	0.94	0.93
	± 0.02	± 0.03	± 0.03	± 0.04	± 0.04	± 0.02	± 0.02	± 0.03	± 0.03	± 0.03	± 0.02	± 0.04	± 0.04	± 0.04	± 0.04	± 0.04

TABLE III
THE AEI SCORES OF METABBO ALGORITHMS ON SYNTHETIC AND PROTEIN-DOCKING TESTSUITES.

	RL-DAS	DE-DDQN	LDE	RLEPSO	LES	GLEET	SYMBOL	RNN-OI	GLHF	OPRO
Synthetic easy	14.29	8.97	12.06	16.09	2.92	5.43	12.67	1.40	7.53	1.72
	± 1.45	± 0.98	± 1.21	± 2.43	± 0.98	± 1.08	± 1.08	± 0.13	± 2.81	± 0.73
Synthetic difficult	10.86	7.85	9.84	9.36	2.16	5.02	12.02	0.02	7.12	1.61
	± 1.19	± 0.82	± 1.09	± 1.39	± 0.82	± 1.15	± 1.15	± 0.01	± 1.38	± 1.33
Protein easy	0.99	0.71	0.98	1.03	1.11	1.21	0.99	1.00	1.13	0.22
	± 0.03	± 0.02	± 0.03	± 0.03	± 0.04	± 0.03	± 0.04	± 0.03	± 0.03	± 0.01
Protein difficult	0.91	0.70	0.90	1.02	0.95	1.01	0.91	0.98	1.01	0.22
	± 0.02	± 0.02	± 0.03	± 0.03	± 0.03	± 0.02	± 0.04	± 0.05	± 0.02	± 0.01

1) *Meta Generalization Decay*: MGD measures the generalization performance of a MetaBBO method for unseen tasks. Concretely, MetaBox trains two models for the MetaBBO method on two source suites (A and B) and test them on the target suit B . We record the AEI scores of these two models on the target suit as AEI_A and AEI_B respectively. The $MGD(A, B)$ is computed as

$$MGD(A, B) = 100 \times (1 - \frac{AEI_A}{AEI_B})\%, \quad (13)$$

A smaller MGD score indicates that the method generalizes well from A to B .

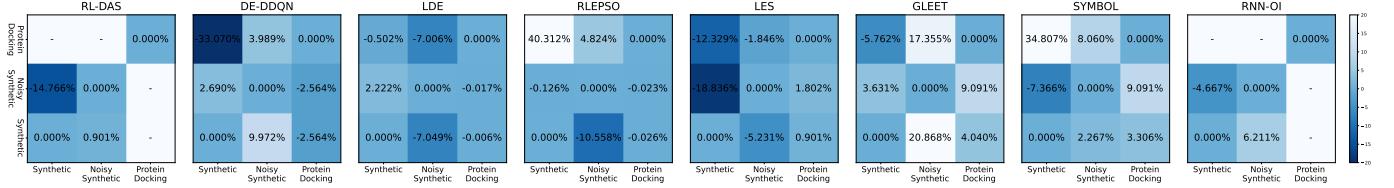


Fig. 10. MGD scores of baselines. The value at i -th row and j -th column is the $MGD(i, j)$, with smaller value indicating better performance.

Fig. 10 shows the MGD plot of MetaBBO baselines, with GLHF omitted since it does not provide training codes. The ‘-’ indicates that the model fails to generalize to target testsuit. We can observe that: 1) RL-DAS and RNN-OI cannot be generalized from Synthetic-10D to Protein Docking-12D due to their optimization state features are dimension-dependent, highlighting the importance of optimization state design. 2) MetaBBO-NE methods (LES) achieves more robust generalization than MetaBBO-RL (RL-DAS, DE-DDQN, LDE, RLEPSO, GLEET, SYMBOL) and MetaBBO-SL (RNN-OI) baselines, possibly revealing the learning effectiveness advantage of neuroevolution paradigm due to its global learning ability. However, neuroevolution is exponentially resource-consuming as the neuron counts scale, implying a tradeoff between effectiveness and efficiency. 3) Larger models (e.g., GLEET with 3 Transformer layers) underperform smaller ones (e.g., DE-DDQN with a single MLP) in the generalization evaluation, even though they outperform within the training distribution. Since the generalization performance is closely tied to the model capacity and the data scale, further investigation into the scaling laws in MetaBBO is highly anticipated.

2) *Meta Transfer Efficiency*: MTE score measures the transfer learning capability. For a MetaBBO method, its MTE from a problem set A to B is computed as:

$$MTE(A, B) = 100 \times (1 - \frac{T_{\text{finetune}}}{T_{\text{scratch}}})\%, \quad (14)$$

TABLE IV
MTE SCORES OF THE METABBO METHODS IN THE TRANSFER FROM SOURCE TESTSUITES TO TARGET TESTSUITES.

Source Testsuites	Synthetic		Noisy-Synthetic		Protein-Docking	
	Noisy Synthetic	Protein Docking	Synthetic	Protein Docking	Synthetic	Noisy Synthetic
Target Testsuites						
RL-DAS	fail	-	1	-	-	-
DE-DDQN	-9.5	1	-3.2	fail	-1.41	-9.5
LDE	-0.05	-0.17	0.19	-0.17	1	-0.05
RLEPSO	fail	-4.25	-0.39	-4.25	1	fail
LES	-1.65	-0.05	fail	-0.05	fail	1
GLEET	1	-0.05	fail	-0.05	fail	-0.39
SYMBOL	-0.16	fail	0.97	-2.17	-0.05	fail
RNN-OI	1	-	0.01	-	-	-

where T_{scratch} is the learning steps used to attain best performance when training on B . T_{finetune} is the learning steps used to fine-tune a model trained on A to attain the same performance level. A larger MTE score indicates that the knowledge learned in A can be easily transferred to solve B . Table IV presents the MTE scores of all baselines under each pair of source-target problem collections, where ‘fail’ indicates the baseline can not be fine-tuned to achieve similar performance level on the target problems. Results show that: 1) While many baselines highlight their transfer learning ability under some cases (e.g., GLEET: Synthetic to Noisy-Synthetic), they show transfer limitations in other cases, suggesting room for improvement. 2) The overall transferring performances across all baselines and problem collections are relatively noisy, making it difficult to determine whether some transfer failures stem from the algorithm designs or the diversity of the problem collections. This opens up a research opportunity to explore the relationship between problem diversity and generalization, as well as how to construct “good” training set for MetaBBO.