Swarm and Evolutionary Computation 93 (2025) 101838

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Survey Paper

Meta-Black-Box optimization for evolutionary algorithms: Review and perspective

Xu Yang [a], Rui Wang [a,b,*], Kaiwen Li [a], Hisao Ishibuchi [c]

[a] College of Systems Engineering, National University of Defense Technology, Deya Road No. 109, Changsha, 410073, Hunan, China
[b] Xiangjiang Laboratory, Changsha, 410205, Hunan, China
[c] Department of Computer Science and Engineering, College of Engineering, Southern University of Science and Technology, Shenzhen, 518055, China

ARTICLE INFO

Keywords:
Black-box optimization
Meta-black-box optimization
Evolutionary algorithms
Automatic algorithm design
Learn to optimize
Comprehensive survey

ABSTRACT

Black-Box Optimization (BBO) is increasingly vital for addressing complex real-world optimization challenges, where traditional methods fall short due to their reliance on expert knowledge and time-consuming processes. Meta-Black-Box Optimization (MetaBBO) emerges as a pivotal solution, leveraging meta-learning to enhance or discover optimization algorithms automatically. Originating from Automatic Algorithm Design (AAD), MetaBBO has branched into areas such as Learn to Optimize (L2O), Automated Design of Meta-heuristic Algorithm (ADMA), and Automatic Evolutionary Computation (AEC), each contributing to the advancement of the field. This comprehensive survey integrates and synthesizes the extant research within MetaBBO for Evolutionary Algorithms (EAs) to develop a consistent community of this research topic. Specifically, a mathematical model for MetaBBO is established, and its boundaries and scope are clarified. The potential optimization objects in MetaBBO for EAs is explored, providing insights into design space. A taxonomy of MetaBBO methodologies is introduced, reflecting the state-of-the-art from a meta-level perspective. Additionally, a comprehensive overview of benchmarks, evaluation metrics, and platforms is presented, streamlining the research process for those engaged in learning and experimentation in MetaBBO for EA. The survey concludes with an outlook on research, underscoring future directions and the pivotal role of MetaBBO in automatic algorithm design and optimization problem-solving.

1. Introduction

Black-box optimization (BBO) has become increasingly crucial in tackling real-world optimization problems where objective functions are complex, unknown, or computationally expensive to evaluate. Applications span diverse domains such as hyperparameter tuning in machine learning (ML) [1–3] including neural architecture search [4], engineering design [5,6], autonomous driving [7] and so on. These problems often involve intricate relationships between design variables and objectives, making traditional gradient-based optimization methods ineffective.

While population-based optimizers like Evolutionary Algorithms (EAs) [1–3,5] and Bayesian Optimization (BO) methods [8,9] have shown promise in solving BBO problems, they often require significant manual design and parameter tuning to achieve optimal performance on specific problem classes. This process is time-consuming, labor-intensive, and demands considerable expert knowledge, hindering the widespread adoption of BBO solvers.

Meta-Black-Box Optimization (MetaBBO) [10] emerges to automatically design BBO algorithms through meta-learning, encompassing the automatic configuration of algorithms without the expert knowledge requirement in manual design mechanisms, as well as the generation of algorithmic strategies. By replacing manual design with automation, MetaBBO can more efficiently find solutions to black box optimization problems. Inspired by the broader field of Automated Algorithm Design(AAD) [11], MetaBBO aims to discover or optimize BBO algorithms by learning on a set of representative optimization problems, enabling them to generalize effectively to previously unseen problems. It has been a powerful tool across diverse practical application domains robot control [12,13], healthcare and diagnosis [14–17], automated design of fan blades [18,19], shop scheduling [20–22], vehicular ad-hoc networks [23], dynamic task allocation of crowdsensing [24] and so on. Given their high dimensionality and substantial computational expense in evaluation, these applications pose significant challenges and are well-suited for meta-optimization strategies. Such strategies are capable

* Corresponding author at: College of Systems Engineering, National University of Defense Technology, Deya Road No. 109, Changsha, 410073, Hunan, China.

E-mail addresses: yangxu616@nudt.edu.cn (X. Yang), rui_wang@nudt.edu.cn (R. Wang), likaiwen@nudt.edu.cn (K. Li), hisao@sustech.edu.cn (H. Ishibuchi).

https://doi.org/10.1016/j.swevo.2024.101838
Received 14 September 2024; Received in revised form 24 December 2024; Accepted 26 December 2024
Available online 8 January 2025

Table 1
An overview of comparing similar surveys.

Reference	Research scope	Application domain	Designed algorithm	Research focus			Methodology type		Optimization object		
				AS	AC	AG	Learning	Non-learning	Numerical	Component	
										Existing	Learnable
[33]	AAD	General optimization	Generalized algorithm	✓			✓	✓			✓
[34]	AAD	General optimization	Generalized algorithm		✓		✓	✓	✓	✓	
[35]	AAD	General optimization	Generalized algorithm		✓	✓	✓		✓	✓	
[25]	L2O	General optimization	General optimizer		✓	✓	✓		✓	✓	✓
[36]	ADMA	Continuous optimization	DE		✓		✓	✓	✓	✓	
[37]	ADMA	General optimization	Meta-heuristic algorithm		✓		✓			✓	
[38]	ADMA	General optimization	Meta-heuristic algorithm		✓		✓	✓	✓	✓	
[14]	AutoML	Healthcare	ML pipeline	✓	✓		✓	✓	✓		
[28]	AutoML	General ML	ML pipeline	✓	✓	✓	✓		✓	✓	
[30]	AutoML	General ML	ML pipeline	✓	✓		✓	✓	✓	✓	✓
[26]	AutoML	General ML	ML system	✓	✓		✓		✓	✓	
[29]	AutoML	General ML	ML system	✓	✓	✓	✓	✓	✓	✓	
[31]	AutoML	RL	RL agent			✓	✓			✓	✓
[39]	AutoML	NAS	NN			✓	✓		✓	✓	
[4]	AutoML	NAS	NN			✓	✓		✓	✓	
[40]	AutoML	RL	RL agent		✓		✓		✓	✓	
[41]	AutoML	ML	NN			✓	✓		✓	✓	
[32]	AutoML	RL	RL agent		✓	✓	✓		✓	✓	
This study	MetaBBO	BBO	EA		✓	✓	✓		✓	✓	✓

of acquiring and transferring optimization insights across analogous problems.

Despite the recent surge in research related to MetaBBO, studies remain scattered across different communities with varying terminologies, including Learn to Optimize (L2O) [25], Automated Machine Learning (AutoML) [26], Automated Design of Meta-heuristic Algorithm (ADMA) [27] and Automatic Evolutionary Computation (AEC). This fragmentation necessitates a comprehensive survey to unify the field and clarify the relationships between these closely related concepts.

There has been extensive related research and review in ML [14,26, 28–32]. However, we have not yet found any comprehensive reviews on the use of MetaBBO for EAs.

To provide a comprehensive context, we analyze key surveys from 2018–2024 across multiple automated algorithm design areas. We dissect these surveys through a multi-dimensional analysis as listed in Table 1. The optimization object refers to the specific object to be optimized when designing the corresponding designed algorithm.

This study, therefore, offers a survey that examines BBO through the problem domains, and explores the design of EAs using learning techniques from a methodological perspective.

Specifically, this study commences with an overview of the foundational knowledge and concepts pertaining to MetaBBO, and then sequentially delves into the base-optimizer and potential optimization objects, meta-optimizer and commonly used optimization methodologies, available benchmarks and platforms. Finally, the perspective and research directions of using MetaBBO for EAs are highlighted. The main contributions of this study can be summarized in the following aspects.

- Clear definitions of MetaBBO and related terms are provided, delineating their scope and highlighting their interconnections.
- The potential optimization objects of MetaBBO from the view of different type of EAs is discussed, encompassing both numerical hyperparameters and components.
- A taxonomy of MetaBBO methodologies at meta-level is proposed according to their domains, with a review of state-of-the-art techniques within each category.
- Existing benchmarks, evaluation metrics, and platforms are summarized to facilitate research and experimentation in MetaBBO for EAs, covering ML and OR.
- Promising research directions and open challenges in the field of MetaBBO for EAs are identified.

The remainder of this study is organized as follows. Section 2 initiates with foundational preliminaries of MetaBBO, encompassing a conceptual exposition, mathematical formalism, and a scoped analysis. Section 3 delves into the optimization objects in MetaBBO for EA, tailored to various EA typologies. Section 4 introduces a taxonomy of MetaBBO methodologies, underpinned by a review of the state-of-the-art techniques categorized within each meta-level approach. Subsequent to this, Section 5 provides a comprehensive synthesis of existing benchmarks, evaluation metrics, and platforms, offering a structured framework to facilitate research and experimentation in the domain of MetaBBO for EAs. Section 6 presents prospective research directions and open challenges, highlighting the pivotal role of MetaBBO in the realms of automatic algorithm design and complex problem-solving. Finally, Section 7 encapsulates the study, offering a synthesis of the contributions and insights garnered throughout the review. The organizational structure of the manuscript is shown as Fig. 1.

2. Preliminary

This section provides the necessary background and definitions for understanding MetaBBO. We begin by formally defining MetaBBO, and the mathematical model of MetaBBO is given. Meta-learning is introduced then due to its pivotal role in MetaBBO by enabling BBOAs to enhance their generalization capabilities and adaptivity through the accumulation of past experiences. Finally, we analyze the scope of MetaBBO, differentiating it from related research areas and clarifying its position within the broader landscape of automated algorithm design.

Some important abbreviations and their full names are listed in Table 2.

2.1. MetaBBO for EAs

MetaBBO refers to an advanced optimization framework that leverages meta-learning to enhance the performance of traditional BBO solvers as defined by Satinder Singh et al. [10]. The prefix “Meta” emphasizes the ability to adapt and refine optimization strategies based on prior experiences, akin to how meta-learning algorithms adapt their learning strategies. Inspired by the idea of meta-learning, BBOAs are trained across a series of representative optimization problems to learn generalizable optimization strategies that can effectively handle novel problems.

MetaBBO research directions can be categorized into two groups from the view of optimization objects:

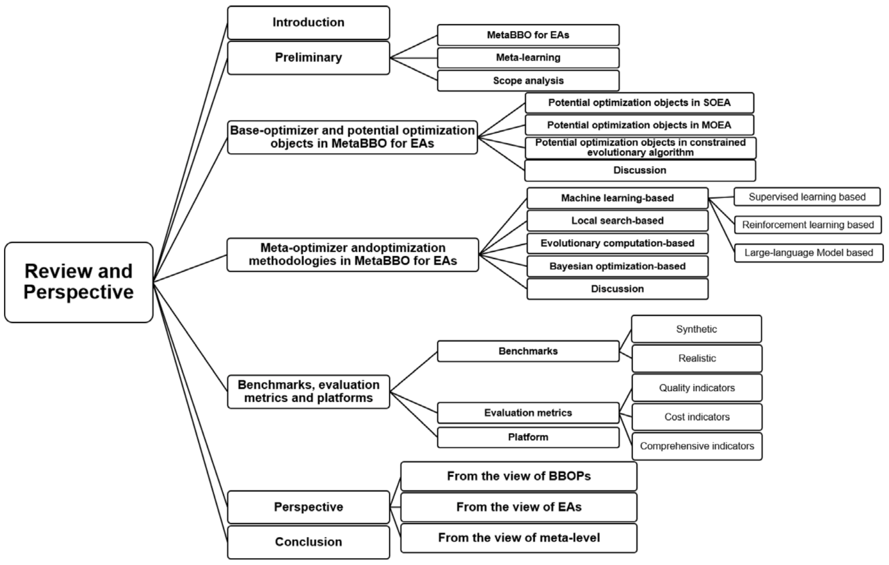

Review and Perspective
Introduction
Preliminary
MetaBBO for EAs
Meta-learning
Scope analysis
Base-optimizer and potential optimization objects in MetaBBO for EAs
Potential optimization objects in SOEA
Potential optimization objects in MOEA
Potential optimization objects in constrained evolutionary algorithm
Discussion
Meta-optimizer andoptimization methodologies in MetaBBO for EAs
Machine learning-based
Supervised learning based
Reinforcement learning based
Large-language Model based
Local search-based
Evolutionary computation-based
Bayesian optimization-based
Discussion
Benchmarks, evaluation metrics and platforms
Benchmarks
Synthetic
Realistic
Evaluation metrics
Quality indicators
Cost indicators
Comprehensive indicators
Platform
Perspective
From the view of BBOPs
From the view of EAs
From the view of meta-level
Conclusion

Fig. 1. Organizational structure of this manuscript.

Table 2
Abbreviations.

Abbr.	Full name	Abbr.	Full name
BBO	Black-Box Optimization	RNN	Recurrent Neural Network
BBOP	Black-Box Optimization Problem	RL	Reinforcement Learning
BBOA	Black-Box Optimization Algorithm	EA	Evolution Algorithm
MetaBBO	Meta-Black-Box Optimization	DE	Differential Evolution
AAD	Automatic Algorithm Design	PSO	Particle Swarm Optimization
L2O	Learn to Optimize	GA	Genetic Algorithm
AutoML	Automated Machine Learning	SI	Swarm Intelligence
NAS	Neural Architecture Search	GP	Genetic Programming
ADMA	Automated Design of Meta-heuristic Algorithm	ES	Evolution Strategy
AEC	Automatic Evolutionary Computation	DPO	Discovered Policy Optimization
EC	Evolutionary Computation	LPO	Learned Policy Optimization
ML	Machine Learning	LES	Learned Evolution Strategy
BO	Bayesian Optimization	LGA	Learned Genetic Algorithm
NN	Neural Network	AS	algorithm selection
AC	algorithm configuration	AG	algorithm generation

- **Parameter Optimization:** This group focuses on optimizing the parameters of existing BBOAs. These parameters include numerical parameters (e.g., crossover rate in EA, learning rate in ML) and component parameters (e.g., initialization strategies in EA, architecture of ML), which belongs to AC. As highlighted in [42], a key objective of parameter optimization is generalizability. The primary objective of MetaBBO in this group is to find parameter configurations that maximize the expected performance across a distribution of problem instances, rather than optimizing for a single specific problem.
- **Generation Strategies:** This direction extends beyond parameter optimization to discover novel optimization strategies for solution generation and algorithm generation. Through learning from diverse problem instances, meta-optimizers can devise innovative strategies that may not be apparent within traditional design paradigms.

In summary, MetaBBO for EAs represents discovering or optimizing EAs to tackle across BBOPs, including parameter optimization and generation of EAs via meta-learning. To this end, it implements the learning process within a bi-level optimization framework as shown in Fig. 2 with different optimization objectives before dealing with unseen tasks.

When learning, the optimization object in inner optimization (base-level) is termed as base-optimizer, where the inner loop aims to find the best solution x^* of the BBOP via base-optimizer. The optimization methodologies used in outer optimization (meta-level) are termed as meta-optimizer, where the outer loop aims to optimize the comprehensive performance of the meta-optimizer across BBOPs, which is evaluated according to the performance of base-optimizers. Both outer and inner optimization will generate a candidate set. For outer optimization, the candidate set represents different base-optimizers, i.e. EAs. For inner optimization, the candidate set represents the solution set of the sampled problem, i.e. population in EAs. The mathematical representations are introduced as follows according to the flowchart of MetaBBO depicted in Fig. 3.

The first step in MetaBBO is to parameterize the base-optimizer. The set of all possible parameter values is made up of the parameterization space, denoted as Θ. The parameterized BBOA with a specific configuration θ (where $\theta \in \Theta$) is represented as $A(\theta)$.

Second, the control parameters and candidate set of outer optimization are initialized. Usually, the control parameters consist of termination condition of outer loop and the control parameters of the meta-optimizer. The candidate set is the configuration set of the base-optimizer.

Then, a training set of BBOPs P is sampled from distribution set $\mathcal{D}$ referring to different problem domains. The inner loop executes solving each $p \in P$ by $A(\theta)$. The performance of $A(\theta)$ is evaluated on the sampled BBOPs P, while the performance of meta-optimizer $C(A(\theta))$ can be formally defined as the expected performance of the base-optimizer $A(\theta)$ across the true distribution of problems $\mathcal{D}$:

$$C(A(\theta)) = \mathbb{E}_{p\in P, A\subseteq P, A\sim d, d\in \mathcal{D}}[\mathcal{J}(A(\theta), p)] \quad (1)$$

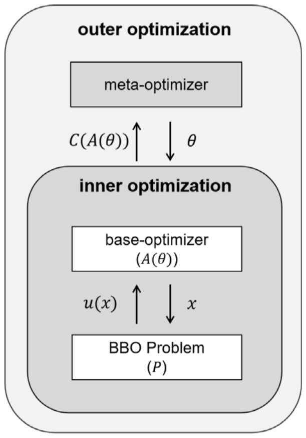


Fig. 2. Bi-level optimization framework of MetaBBO.

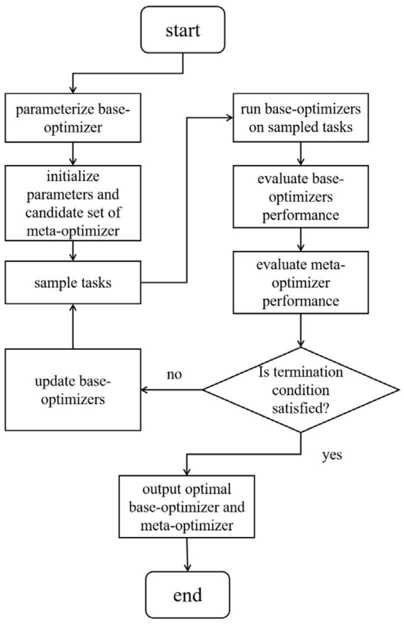

start
parameterize base-optimizer
initialize parameters and candidate set of meta-optimizer
sample tasks
update base-optimizers
run base-optimizers on sampled tasks
evaluate base-optimizers performance
evaluate meta-optimizer performance
Is termination condition satisfied?
no
yes
output optimal base-optimizer and meta-optimizer
end

Fig. 3. The flowchart of MetaBBO.

where $\mathcal{J}(A(\theta), p)$ represents the performance of $A(\theta)$ dealing with p.

Next, the outer loop including updating base-optimizers, sampling new tasks P, running base-optimizers, and evaluating the meta-optimizer is implemented until the termination condition is satisfied.

The optimization objective in MetaBBO can thus be formulated as finding the optimal configuration θ^* that maximizes the expected performance while maintaining robust generalization:

$$\theta^* \in \arg\max_{\theta\in\Theta} \mathbb{E}_{p\in P, A\subseteq P, A\sim d, d\in\mathcal{D}}[\mathcal{J}(A(\theta), p)] \tag{2}$$

To ensure good generalization, several key strategies are employed:

- Diverse training instances to cover the target problem space, like focusing on the sample strategies [43–45], and generating new training instances via generative adversarial construction of parallel portfolios [46] or co-evolution between a configuration population and an instance population [47].
- Prevent overfitting [48] when learning, considering regularization strategies [49–51].
- Estimate true generalization performance via cross-validation [52,53], designing $C(A(\theta))$ with more fitted expectation, and considering uniform distribution of computational budgets across training instances which yields optimal generalization error estimates as proved in [54].
- Robust performance metrics that account for instance variability [55], designing proper $\mathcal{J}(A(\theta), p)$.

More details about theoretical guarantee when learning can be found in [25].

In addition, to avoid confusion between MetaBBO and BBO, the differences have been concluded as given in Table 3. It helps in delineating the scope and objectives of MetaBBO, which focuses on optimizing the optimization algorithms themselves through meta-learning. On the meanwhile, the scope of BBO and MetaBBO are figured in Fig. 4.

2.2. Meta-learning

Meta-learning, commonly characterized as “learning to learn”, represents a sophisticated learning paradigm that systematically optimizes the learning process itself [56]. This paradigm is fundamentally designed to enhance learning efficiency and effectiveness by leveraging experience accumulated from previous learning tasks. Through the systematic exploitation of cross-task learning experiences, meta-learning distills generalizable learning principles that can be effectively applied across different domains and problem spaces. This approach transcends traditional learning paradigms by focusing not only on task-specific knowledge acquisition, but also on the metacognitive aspect of learning optimization. An overview of the difference between traditional learning and meta-learning can be found in Table 4.

To systematically further understand meta-learning, a comprehensive comparison between meta-learning and various established learning paradigms is presented in Table 5. Table 5 delineates the fundamental differences and interconnections across multiple dimensions, including key characteristics, operational mechanisms, and practical applications. This comparative analysis not only highlights meta-learning’s distinctive approach of “learning to learn” but also reveals potential synergies with other paradigms.

Now let us understand MetaBBO again, which can be regarded as “learn to optimize BBOPs”. By training on diverse distributions of BBOPs, meta-learning enables to learn which strategies work well across various BBOPs. Meta-learning can even lead to the discovery of innovative optimization strategies that may not be apparent through human intuition or traditional algorithm design approaches.

In essence, meta-learning in MetaBBO provides a systematic and data-driven approach to the discovery and implementation of optimization strategies, enhancing the capability of algorithms to autonomously improve and adapt to the complexities of various optimization landscapes.

2.3. Scope analysis

While MetaBBO shares conceptual ground with several other fields, it is important to delineate its scope and relationship with these related areas to avoid confusion. This section provides an analysis of similar research domains with MetaBBO including AAD, L2O, AutoML, NAS, ADMA and AEC.

Automated Algorithm Design encompasses the broadest scope and realizes free human creativity for higher level tasks via using computation power to design algorithms automatically, originated in the 1980s [11]. The algorithm in AAD covers almost all algorithms in machine learning (ML) and optimization, such as supervised, unsupervised, and reinforcement learning algorithms, as well as exact, constructive, heuristic algorithms and meta-heuristic algorithms. The studies in AAD can be typically categorized into algorithm selection (AS) [33,57,58], algorithm configuration (AC) [59] and algorithm generation (AG) [60].

Table 3
Differences between BBO and MetaBBO.

Term	Black-Box optimization	Meta-Black-Box optimization
Targeting Object	solving BBOPs	finding optimal BBOAs
Output	x^*, BBOA	BBOA, BBOA optimizer
Decision Space	Variable space of BBOPs	Parameterization space of BBOAs
Objective Space	Objective values of optimization problems	Performance of BBOAs on problems
Process	Trial-and-error on target problem	Training on a set of problem instances

Table 4
Comparison between traditional learning and meta-learning.

Learning type	Traditional learning	Meta-learning
Learning Objective	Learn from data to accomplish specific tasks	Learn from multiple tasks to acquire learning strategies
Task Relationship	Task-specific and independent	Cross-task knowledge transfer
Adaptation Capability	High performance on specific tasks but limited adaptability	Strong generalization with rapid adaptation to novel tasks
Knowledge Acquisition	Direct task-specific knowledge learning	Meta-knowledge extraction from learning experiences
Learning Scope	Limited to predefined task boundaries	Generalizable across different task domains
Optimization Focus	Task-specific performance metrics	Learning process optimization

Learn to Optimize is a more recent development, focusing on the automatic discovery of effective solvers for problem instances with shared characteristics via learning [61,62]. Related studies have been conducted to accelerate the tedious trial-and-error configuration process by learning [25]. From the view of the products of L2O through the training process, the studies can be classified into three groups, performance prediction models (similar to AS in AAD), a single solver and a combination of solvers [25].

Automated Machine Learning pertains to the automated construction of an ML pipeline within a constrained computational budget [29]. AutoML encompasses data preparation, feature engineering, model selection or generation and model estimation. Model generation involves selecting from a search space that includes traditional models (such as support vector machine [63] and K-nearest-neighbor [64]) and deep neural networks (such as conventional neural network [65] and recurrent neural network [66]), as well as optimizing tunable parameters and architectures, i.e. NAS. **Neural Architecture Search**, is an application of AutoML [67], focusing on the automated engineering of neural network architectures [68]. It seeks to identify high-performance architectures by selecting and combining basic operations from a predefined search space. NAS shares significant commonality with hyperparameter optimization [69] and meta-learning [70]. Since it deals with optimizing machine learning models as black-box functions, it can be considered a part of MetaBBO.

Automated Design of Meta-heuristic Algorithms involves the automatic customization of meta-heuristic algorithmic operators, structures, and parameters for specific problems [71]. It includes automated algorithm selection, automated algorithm configuration [38] and algorithm generation (so-called hyper-heuristics) [72] of meta-heuristic algorithms. Within this realm, **Automatic Evolutionary Computation** concentrates on EC as a subset of ADMA.

Based on our investigation and understanding, we have drawn a Venn diagram to represent the hierarchical relationships between these concepts, as shown in Fig. 5. The AAD on the left side of the figure displays its subsets based on the category of the design algorithms, while AutoML displays its subsets based on the entire ML process. The intersection between AAD and AutoML represents the automatic design of algorithm or model used in the ML process. On the right side of the figure, AAD is divided into subsets from another perspective. Firstly, based on the design method, AAD is divided into two complementary subsets, and then L2O and MetaBBO are placed into the right part. It is noted that when it comes to MetaBBO for EA, it belongs to ADMA.

3. Base-optimizer and potential optimization object in MetaBBO for EAs

Within the scope of our study, the base-optimizer consists of various kinds of EAs. According to the number of optimization objectives, EA can be divided into single- and multi-objective evolutionary algorithm (SOEA and MOEA). According to whether there are constraints, EA can be divided into constrained optimization and unconstrained optimization. Due to the importance of constrained optimization, this section extracts the part of constrained optimization from single-objective evolutionary optimization and multi-objective evolutionary optimization, and separately introduces the optimization objects of constrained evolutionary optimization. Hence, the base-optimizer is divided into SOEA, MOEA and constrained evolutionary algorithm.

Whatever kinds of EAs, the potential optimization objects encompass a comprehensive algorithmic elements decomposed from parameterizing base-optimizers, constituting the design space. These objects can be taxonomically categorized into two principal domains: numerical parameters and components.

Numerical parameters comprise both basic control parameters (e.g., population size, number of generations) and advanced regulatory parameters (e.g., selection pressure, elite preservation ratio). Algorithmic components include artificially designed components and learnable components. The former includes existing operators, such as selection, crossover, and mutation operators, each with various implementation variants, and the other adjustable parts such as structural configurations, adaptive mechanisms, termination criteria, constraint handling mechanisms, and multi-objective frameworks. In addition to refining existing operators, MetaBBO delves into the generation of novel, learning-based operators, employing adaptive mechanisms that evolve and refine these operators based on the specific characteristics and requirements of the optimization problems.

These optimization objects exhibit complex interdependencies and present significant challenges in their optimization due to their high-dimensional nature, non-linear interactions, and problem-specific characteristics. To gain a clearer understanding of the potential optimization objects within MetaBBO for EA, all types of optimization objects are introduced, which is organized from the view of EA types in this section.

3.1. Potential optimization objects in SOEA

Single-objective evolutionary optimization aims to find an optimal solution in a given search space by simulating natural evolution processes such as selection, crossover, mutation, etc., in order to achieve the optimal value of a predetermined objective function.

Classic single-objective evolutionary algorithms are population-based optimization algorithms, including GA, DE, and ES. They simulate natural selection and genetic mechanisms, use encoding to represent the solution space of the problem, and evaluate individual strengths and weaknesses through fitness functions. During evolving, operations such as selection, crossover, and mutation are utilized to

Table 5
Comparison of meta-learning with other learning paradigms.

Learning paradigm	Key characteristics	Distinction from meta-learning	Representative applications
Supervised Learning	• Requires labeled data • Direct input–output mapping • Task-specific optimization	• Single-task oriented • Static learning process • Data-intensive	• Image classification • Speech recognition • Machine translation
Unsupervised Learning	• Label-free learning • Structure discovery • Self-organization	• Task-independent • Implicit objectives • Dataset-specific	• Clustering analytics • Dimensionality reduction • Anomaly detection
Reinforcement Learning	• Reward-driven • Trial-and-error • Policy optimization	• Environment-specific • Ab initio learning • Iterative policy refinement	• Game AI • Robotic control • Resource allocation
Transfer Learning	• Cross-domain knowledge transfer • Source–target mapping • Data efficiency	• Direct knowledge transfer • Fixed transfer direction • Knowledge reuse focus	• Pre-training fine-tuning • Cross-lingual transfer • Domain adaptation
Multi-task Learning	• Concurrent task learning • Shared representations • Task correlation	• Parallel task optimization • Fixed task set • Joint learning	• Computer vision • nature language process • Multi-modal learning
Few-shot Learning	• Limited sample learning • Rapid adaptation • Knowledge leverage	• Task-specific adaptation • Fixed learning protocol • Domain-specific	• Image recognition • Personalized recommendation • Drug discovery
Zero-shot Learning	• Unseen class recognition • Semantic mapping • Knowledge generalization	• Auxiliary information dependent • Fixed semantic mapping • Class generalization	• Object recognition • Cross-modal retrieval • Novel class recognition

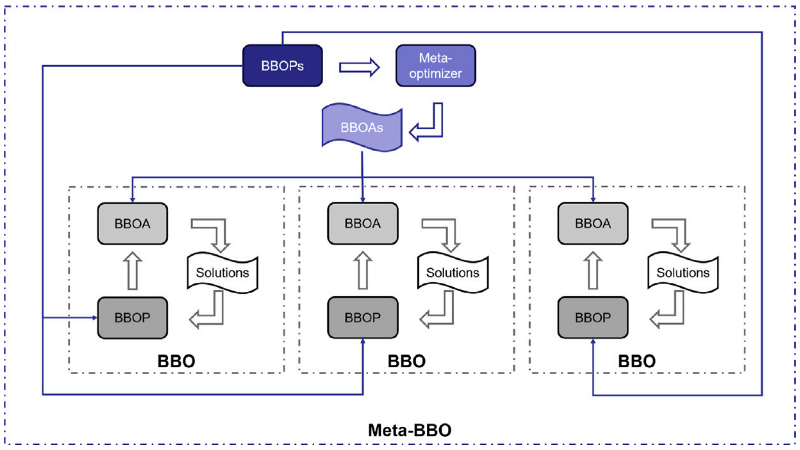


Fig. 4. The scope of BBO and MetaBBO, where the gray dashed box delineates what BBO needs to do, while the blue dashed box represents what MetaBBO needs to do.

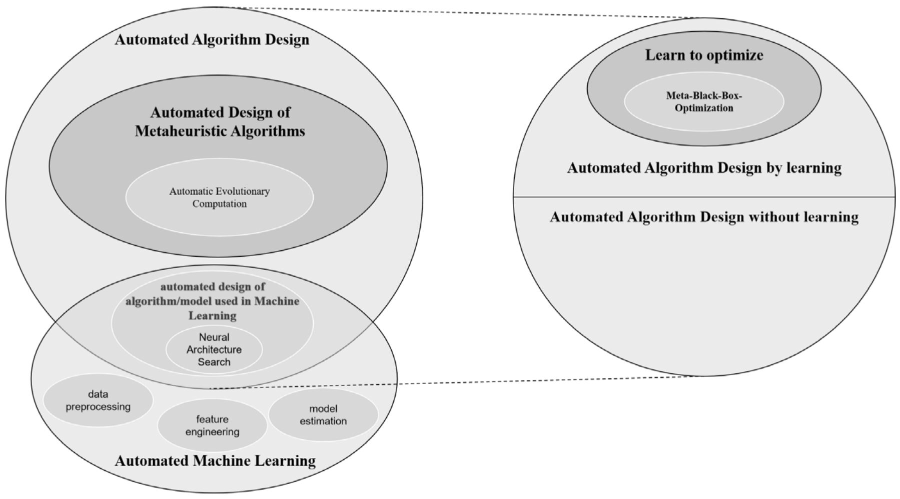

Automated Algorithm Design
Automated Design of Metaheuristic Algorithms
Automatic Evolutionary Computation
automated design of algorithm/model used in Machine Learning
Neural Architecture Search
data preprocessing
feature engineering
model estimation
Automated Machine Learning
Learn to optimize
Meta-Black-Box-Optimization
Automated Algorithm Design by learning
Automated Algorithm Design without learning

Fig. 5. Venn of the related concepts.

Table 6
Optimization objects and their options for SOEA.

Optimization object	Options
InitializationPopulation	Random Initialization Latin Hypercube Sampling Knowledge-Based Initialization Elite Initialization Layered Initialization
Selection	Deterministic Tournament Stochastic Tournament Roulette Wheel Rank-based
GenerateVariants	Mutation: bit-flip, polynomial mutation, Gaussian mutation Crossover: single-point, multi-point, uniform
EvaluateFitness	Objective function evaluation
UpdatePopulation	Generational replacement Steady-state replacement Elitism
TerminationCriterion	Maximum number of evaluations Maximum number of generations Improvement threshold

update the population, gradually searching for the optimal solution. These algorithms each have their own characteristics and demonstrate different advantages in optimization problems. **GA** is the fundamental EA, which has strong global search ability. **DE** generates new mutated individuals through differential vectors and performs cross operations with the target individual to generate experimental individuals, which has the characteristics of fast convergence speed and good robustness, and is more suitable for continuous function optimization. **ES** uses a continuous real number space for search and generates new solutions through probability distributions.

Based on the common characteristics of these algorithms, a unified and flexible framework capturing the common elements of single-objective evolutionary algorithms is created, termed as AutoSOEA template, see in Algorithm 1.

Algorithm 1 AutoSOEA template

```
1: $population \leftarrow$ InitializePopulation()
2: repeat
3:     $candidates \leftarrow$ Selection($population$)
4:     $variants \leftarrow$ GenerateVariants($candidates$)
5:     EvaluateFitness($variants$)
6:     $population \leftarrow$ UpdatePopulation($population$, $variants$)
7: until TerminationCriterion($population$)
8: return BestSolution($population$)
```

In this framework, **InitializePopulation()** is a placeholder for the process of creating the initial population of solutions. **Selection**($population$) represents the process of selecting individuals from the current population to be used in the generation of new solutions. **GenerateVariants**($candidates$) is a generic term that can represent crossover, mutation, or any other variation operation used to create new candidate solutions from the selected individuals. **EvaluateFitness**($variants$) is the process of assessing the fitness of the new candidate solutions. **UpdatePopulation**($population$, $variants$) is the process of integrating the new candidate solutions into the population, which may involve replacing less fit individuals or updating the best-known solutions for each individual. **TerminationCriterion**($population$) is a condition that determines when the algorithm should stop, such as a maximum number of iterations or a satisfactory fitness level. **BestSolution**($population$) returns the best solution found in the population after the termination criterion is met.

According to the AutoSOEA template, the optimization objects of different single-objective evolutionary optimization algorithms can be extracted as shown in Table 6.

In addition, Table 7 lists some numerical parameters, some of which are condition parameters. The "Conditionality" column helps to identify which part of the algorithm the hyperparameter is associated with. If a hyperparameter is marked as "Shared", it means that it is used across multiple components or throughout the entire algorithm. If it is marked as specific to a component, such as "Selection-specific" or "GenerateVariants-specific", it means that the hyperparameter is used only within that particular component of the algorithm. The "-" in the "Symbol" column for some entries indicates that the hyperparameter is not represented by a specific symbol but is rather a general category of options within a component.

3.2. Potential optimization objects in MOEA

Multi-objective optimization problems (MOPs) often have multiple conflicting objectives, indicating optimizing one objective may have adverse effects on the other objectives. It aims to search for multiple optimal solutions by simulating the genetic mechanisms and natural selection principles in biological evolution, providing decision-makers with a comprehensive selection space. These solutions are called Pareto optimal solution sets, while the optimal objectives constitute Pareto Front. Multi-objective evolutionary algorithms (MOEAs) have been a well-known optimizer for MOPs, widely applied in various fields such as engineering, industry, and science. For example, in engineering optimization, it can be used to optimize multiple objectives such as performance and cost of complex systems; In logistics scheduling, it can optimize indicators such as path planning, resource utilization, and transportation costs; In financial investment, it can optimize the balance between risk and return.

According to the way of dealing with multiple objectives, MOEAs can be divided into three categories: decomposition-based MOEA, dominance-based MOEA, and indicator-based MOEA.

Decomposition-based MOEA combines or aggregates all optimized sub-objectives into single objective, thereby transforming MOPs into single-objective optimization problems [73]. The aggregation function can be linear or nonlinear. When the aggregation function is linear, it is difficult to search for non-convex solutions no matter how the weight coefficients are adjusted. But when the aggregation function is non-linear, it can effectively solve the above problems.

Dominance-based MOEA is to use the Pareto-based fitness allocation strategy to identify all non-dominant individuals from the current evolutionary population. There are many MOEA methods based on Pareto, mainly including the following: NSGA-II proposed by Deb et al. [74], SPEA2 proposed by Zitzler Thiele [75], and MOGA proposed by Tadahiko [76].

Indicator-based MOEA uses performance evaluation metrics to guide the search process and solution selection process [77]. Classic

Table 7
Numerical hyperparameters in SOEAs. **Shared** hyperparameters are common to all SOEAs, while **Conditionality** refers to certain hyperparameters that may be related only to specific algorithmic components or used under certain conditions.

Numerical hyperparameters	Symbol	Conditionality	Description
Population Size	N	Shared	Total number of individuals in the population.
Crossover Rate	C_r	Shared	Probability of performing crossover.
Mutation Rate	M_r	Shared	Probability of performing mutation.
Tournament Size	T_s	Selection-specific	Number of individuals in a tournament selection.
Bit-flip Rate	B_r	Mutation-specific	Probability of flipping each bit in a binary representation.
Polynomial Mutation Power	η_m	Mutation-specific	Power parameter for polynomial mutation.
Gaussian Mutation Std. Dev.	σ	Mutation-specific	Standard deviation of the Gaussian distribution for Gaussian mutation.
Gaussian Mutation Mean	μ	Mutation-specific	Mean of the Gaussian distribution for Gaussian mutation.
Max Generations	$G_{\max}$	TerminationCriterion-specific	Maximum number of generations before termination.
Improvement Threshold	ϵ	TerminationCriterion-specific	Threshold for improvement that triggers termination.
Elite Ratio	E_r	UpdatePopulation-specific	Ratio of best individuals to carry over to the next generation.

Table 8
Main optimization objects and their options of MOEAs extracted from AutoMOEA+.

Main optimization objects	Options
Preference	(SetPart, Refinement, Diversity)
Mating	(Mating, Preference$_{Mat}$, Selection, Removal)
Replacement	(Preference$_{Rep}$, Removal$_{Rep}$)
Replacement$_{Ext}$	(Preference$_{Ext}$, Removal$_{Ext}$)
UnderlyingEA	(Mating, Variation)

indicator based MOEA includes weighted indicator-based EA [78] and GDE-MOEA [79].

Based on the design of SOEA, the design of MOEA needs to consider the trade-off relationship between multiple objectives [80]. In particular, how to balance the convergence and diversity of solutions is the core challenge of multi-objective evolutionary algorithm design. In general, MOEAs in MetaBBO are commonly proposed as monolithic blocks, assuming that their components are equally effective and need to be jointly used [81]. This monolithic approach is reflected by various MOEA frameworks that offer limited composability of components [82, 83]. Bezerra et al. have proposed a general MOEA template from which a researcher can easily instantiate several existing MOEAs, but also generate a much larger number of novel MOEAs by combining the algorithmic components available from a component-wise algorithmic framework [84]. Based on this work, a strongly extended framework AutoMOEA+ was proposed [81], see in Algorithm 2, which integrates a further level of composability that allows to separate between the multi-objective related aspects of the search from the underlying EA and comprises decomposition-based algorithms in addition to the originally comprised dominance- and indicator-based. Specifically, pop represents the population taking part in the evolutionary process, pop_{ext} represents an optional external archive, while pop_{new} represents the set of new solutions generated after some operations. **Initialization** represents the generation of an initial population. **UnderlyingEA** defines the proper choice of the underlying EA operators which is critical to the effectiveness of a MOEA due to the strong interactions between MO-components and EA operators [85]. GA and DE are considered as the underlying EA in this proposed template, as shown in Algorithm 3 and Algorithm 4 separately. **Evaluation** defines the fitness computation. **Replacement** and **Replacement**$_{Ext}$ define environmental selection and external archive truncation respectively.

The optimization objects in MetaBBO for MOEAs can be defined according to the template AutoMOEA+, where the existing algorithmic

Algorithm 2 AutoMOEA+ template proposed by Bezerra et al.

```
1:  $pop \leftarrow$ Initialization()
2:  if $type(pop_{ext}) \neq none$ then
3:      $pop_{ext} \leftarrow pop$
4:  end if
5:  repeat
6:      $pop_{new} \leftarrow$ UnderlyingEA($pop$)
7:      $pop_{new} \leftarrow$ Evaluation($pop_{new}$)
8:      $pop \leftarrow$ Replacement($pop$, $pop_{new}$)
9:      if $type(pop_{ext}) = bounded$ then
10:         $pop_{ext} \leftarrow$ Replacement$_{Ext}$($pop_{ext}$, $pop_{new}$)
11:     else if $type(pop_{ext}) = unbounded$ then
12:         $pop_{ext} \leftarrow pop_{ext} \cup pop$
13:     end if
14: until termination criteria met
15: if $type(pop_{ext}) = none$ then
16:     return $pop$
17: else
18:     return $pop_{ext}$
19: end if
```

Algorithm 3 UnderlyingEA = (Genetic Algorithm)

```
Require: $pop$
Ensure: $pop_{new}$
1: $pop_{new} \leftarrow \emptyset$
2: $pool \leftarrow$ MatingGA($pop$)
3: $pop_{new} \leftarrow$ VariationGA($pool$)
4: return $pop_{new}$
```

components can be extracted mainly in Table 8. The specific parameter options of each component are listed in Table 9.

In detail, **Preference** models general preference relations [86], where **SetPart** partitions solutions into dominance equivalent, **Refinement** ranks solutions within each partition and **Diversity** is a Pareto-noncompliant metric used to keep the population well-spread across the objective space. **Mating** uses traditional Selection operators to select individuals to undergo variation. In the case of tournaments, solutions are compared based on a preference relation Preference$_{Mat}$. **Preference**$_{Rep}$ and **Preference**$_{Ext}$ are used to compare solutions, while

Algorithm 4 UnderlyingEA = (Differential Evolution)

```
Require: $pop$
Ensure: $pop_{new}$
1: $pop_{new} \leftarrow \emptyset$
2: $i \leftarrow 0$
3: while $i < n$ do
4:     $\{target, donor\} \leftarrow \text{Mating}_{DE}(pop)$
5:     $trial \leftarrow \text{Variation}_{DE}(target, donor)$
6:     $S \leftarrow \text{OnlineReplace}(pop, target, trial)$
7:     $pop_{new} \leftarrow pop_{new} \cup S$
8: end while
9: return $pop_{new}$
```

Table 9
More basic-level options according to main optimization objects.

Option of main optimization objects	Basic-level options
SetPart	none(–) dominance count dominance rank dominance strength dominance depth dominance depth-rank (DR)
Refinement	none(–) binary $I_{\epsilon+}$ binary I_H^- exclusive HV contribution (I_H^1) shared HV contribution (I_H^h) weighted ranking (w-$rank$) [87]
Diversity	none(–) niche sharing (σ_{share}) nearest neighbors crowding distance adaptive grid (AGA) reference lines [88]
Selection	deterministic tournament (DT) stochastic tournament (ST) random
Removal	sequential one-shot
type(pop)	fixed size bounded
type(pop_{ext})	none bounded unbounded
UnderlyingEA	GA DE

the removal policies **Removal**$_{Rep}$ and **Removal**$_{Ext}$ determine the frequency with which the preference relation is computed. **Variation** represents the variation operators that produce new solutions from existing ones.

In addition, there are some condition components depicted in Table 10. When $\Lambda_{dist} = two - layer$, parameters "peripheral", "central" and "balanced" determine the search focus a designer wants to use [88].

Besides the components parameters in the Tables 9 and 10, the design space of MOEAs also includes numerical parameters as depicted in Table 7.

3.3. Potential optimization objects in constrained evolutionary algorithm

The optimization objects in the previous two sections have not involved constraints. However, constraints are an indispensable part of practical optimization. Constraint Handling Technique (CHT) is the core when addressing constrained evolutionary optimization [93], ensuring finding feasible solutions. Penalty function method [94,95], feasibility rule method [96,97], and multi-objective method [98,99] are the most popular CHTs in constrained evolutionary optimization.

Table 10
Algorithmic component options available in the AutoMOEA+ framework.

Condition	Component	Domain
UnderlyingEA = DE	OnlineReplace	none Pareto [89] WeakPareto [90]

Refinement = w-$rank$
Diversity = reference lines
	Λ_{dist}	uniform [91]
		dichotomic [92]
		two-layer [88]

Λ_{dist} = two-layer	Λ_{focus}	peripheral central balanced

Penalty function method is a method of transforming constraints into penalty functions and adding them to the objective function to form a new objective function. It introduces appropriate penalty terms in the objective function to lower the numerical value of the solution that satisfies the constraint conditions in the objective function, thereby encouraging the algorithm to search for the optimal solution that satisfies the constraint conditions as much as possible. The penalty function method is simple, intuitive, and easy to implement. The commonly used penalty function methods include external penalty function method, internal penalty function method, quadratic penalty function method, and linear penalty function method.

Feasibility rule method is a direct processing method based on constraint conditions. It directly determines the feasibility of the solution based on the constraint conditions and conducts selection and evolution operations accordingly. Usually, it prioritizes feasible solutions in the selection process to improve the search efficiency of the algorithm. The feasibility rule method directly deals with constraint conditions, which can better balance objective optimization and constraint satisfaction. However, it may face the problem of insufficient feasible solutions, which can affect the search performance of the algorithm.

Multi-objective method transforms constrained optimization problems into MOPs for processing. It uses constraints as additional objective functions and optimizes them together with the objective function. Through this approach, the algorithm can simultaneously consider both objective optimization and constraint satisfaction, finding solutions that satisfy multiple objectives. Multi-objective methods can handle multiple objective functions and constraints simultaneously, and have strong flexibility and universality. However, it may also face issues such as solution selection and ranking, requiring the design of appropriate fitness evaluation mechanisms.

In addition, there are other constraint processing techniques, such as stochastic ranking method [100], ϵ-constraint handling method [101], etc., whose detail can be found in study [101]. These technologies each have their own characteristics and are suitable for different constrained optimization problems. Hence, it is necessary to consider CHTs as optimization objects when designing EAs. If constraints need to be considered when designing SOEAs or MOEAs, some options of their optimization object may change. Potential optimization objects and options can be found in Table 11.

3.4. Discussion

Designing EAs is a complex and multi-level process that involves multiple considerations. The first step in designing an effective EA is to comprehensively understand and clarify the various optimization objects involved in the algorithm. These optimization objects determine the structure and performance of the algorithm, covering multiple levels from basic control parameters to complex algorithm components. They do not work independently, and many optimization objects have

Table 11
Optimization objects and options when considering constraint handling in designing SOEAs and MOEAs.

SOEA or MOEA	Optimization object	Options
SOEA	EvaluateFitness	Penalty function method
	UpdatePopulation	Feasibility rule method Multi-objective method Constraint Relaxation Fitness Minimization
	TerminationCriteria	Maximum Iterations with Feasibility Check Convergence and Feasibility Threshold Time Limit with Constraint Satisfaction
MOEA	Replacement	Penalty function method Feasibility rule method Multi-objective method Constraint Relaxation

complex interdependent relationships. For example, the efficiency of mutation operators may depend on the size of the population, while selection pressure may affect the design of elite retention ratios. The nonlinear interactions between numerical parameters and algorithm components make the design of evolutionary algorithms highly complex. In addition, the properties of these optimization objects may vary in different optimization problems. For example, in multi-objective optimization problems, the trade-off of the objective function and the satisfaction of the constraint conditions jointly affect the design of the algorithm; In constrained optimization problems, how to efficiently handle constraints and balance the relationship between the objective function and constraints is a key challenge in design. Only after fully understanding these optimization objects can we start making reasonable choices, adjustments, and optimizations to design efficient evolutionary algorithms.

In the MetaBBO framework, these optimization objects together form a highly complex parameterized space, which includes the design of multiple algorithm components and the selection of numerical parameters. Especially, the existence of meta-learning has given rise to learning objects, which can be understood as rules generated during the learning process. In order to effectively design and optimize these objects, MetaBBO automates the algorithm design and optimization process, enabling evolutionary algorithms to adaptively adjust their components to meet different problem requirements.

The next section will delve into the meta-optimizer in MetaBBO, introducing which optimization methodologies play a role as the meta-optimizer in MetaBBO for EAs and how these methodologies automatically select and optimize algorithm objects.

4. Meta-optimizer and optimization methodologies in MetaBBO for EAs

A variety of learning methodologies serve as the meta-optimizer to discover or optimize an EA, including ML techniques and intelligent optimization techniques. Meta-optimizer plays a pivotal role, iteratively generating candidate EAs with different configurations, assessing their performance on BBOPs, and utilizing this feedback to refine the search for improved configurations when learning.

This section delves into the current state-of-the-art methodologies commonly used as the meta-optimizers. They are categorized based on their underlying principles into machine learning-based, local search-based, evolutionary algorithm-based and Bayesian optimization-based.

4.1. Machine learning-based meta-optimizer

The application of machine learning techniques to optimize the process of optimization has a rich history, with neural networks being considered the most general approach due to their ability to modify weights, as proposed by Schmidhuber [102]. Early attempts focused on using simple NN architectures, such as feedforward networks, as surrogate models to approximate the objective function and guide the search for optimal solutions. However, these approaches had limited success compared to traditional optimization methods due to computational constraints and limitations in NN architectures at the time.

Researchers began exploring hybrid approaches that combined NNs with EAs to leverage their respective strengths. The deep learning revolution, fueled by advancements in NN capabilities, led to significant progress in applying deep neural networks, particularly deep reinforcement learning (DRL), to complex BBOPs. More recently, machine learning has also been employed as a meta-optimizer in MetaBBO for EAs.

4.1.1. Supervised learning

Building upon the works of Younger et al. [103] and Hochreiter et al. [104], which allowed for the output of backpropagation from one network to feed an additional learning network and enabled joint training of the networks, Andrychowicz et al. [105] proposed a classic approach that treats the design of optimization algorithms as a learning problem. They utilized a two-layer Long Short-Term Memory (LSTM) network to learn how to design optimization algorithms that achieve the minimum value of the objective function faster. These works laid the foundation for supervised learning-based MetaBBO for EA.

Chen et al. utilized recurrent neural network (RNN) optimizers as a learned component of BBOA in 2017 [106]. First, they summarized the BBOAs as the following loop, covering EAs:

(1) Given the current state of knowledge k_t propose a query point x_t
(2) Observe the response y_t
(3) Update any internal statistics to produce k_{t+1}

RNN here is unrolled to generate new candidates i.e. as the update rule in the loop, while the update of the RNN is optimized by gradient descent. Experimental results demonstrated that the learned optimizers were able to transfer successfully to numerous BBOPs arising in global optimization, control, and hyperparameter tuning.

Later in 2022, they leveraged Transformer models to develop a general hyperparameter optimizer [107]. The proposed method involves training a meta-optimizer based on the Transformer architecture that learns to optimize hyperparameters directly from the problem's context (e.g., dataset, model architecture), without the need for task-specific search strategies. This meta-optimizer is designed to be task-agnostic and can adapt to various machine learning tasks, including classification and regression problems.

Vishnu et al. [108] improved the work of Chen et al. by designing regret-based loss function, incremental normalization and incorporating domain constraints, to obtain a better BBO optimizer termed as RNN-opt. Specifically, unlike previous loss functions based on observed improvements, RNN-Opt uses a loss function based on regret values to train the model, directly optimizing the final optimization objective, which is to minimize the difference between the best value found and the true best value. In response to the problem of unknown numerical range of black-box functions, RNN-opt adopts an incremental normalization method to dynamically normalize function values during training and inference, improving the stability and performance of the model. In order to handle domain constraints in practical problems, RNN-opt introduces penalty functions and domain constraint loss functions to ensure that the generated query points meet the constraint conditions. At meta-level, parameters of RNN-opt are learned through experience on multiple optimization tasks still by gradient optimization.

In contrast to traditional machine learning approaches, which involve training a neural network on a single task using a set of training samples to enable generalization to unseen data from the same distribution, the approach described here involves training the neural network on a distribution of related tasks (in this case, optimization tasks). The goal is for the network to learn a strategy that generalizes to novel, yet related tasks drawn from a similar task distribution. Meta-learning approaches, therefore, aim to train a single model to simultaneously optimize multiple functions, enabling it to effectively generalize and optimize previously unseen functions.

As noted by Fawzi et al. [109], the expansive search space of potential algorithms introduces significant complexity to the process. Moreover, the reliance on extensively annotated datasets for training makes these methodologies inherently susceptible to biases present within the data, potentially yielding suboptimal or inaccurate predictions.

4.1.2. Reinforcement learning

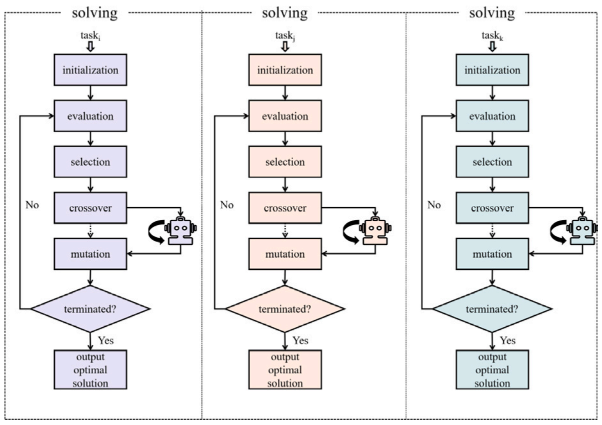
Recent research highlights the efficacy of Reinforcement Learning (RL) in the domain of MetaBBO. Operating at a meta-level, RL exhibits the capacity to alleviate the reliance on extensive fine-tuning by effectively optimizing specific components within the structure of a human-designed algorithm.

It is well known that DE has six similar mutation strategies for DE so far which significantly affect its performance. Li et al. [110] proposed a Differential Evolution based on Reinforcement Learning with Fitness Ranking (DE-RLFR), which uses three typical DE mutation strategies as optional actions, while the distribution characteristics of the population in the objective space, decision space, and fitness-ranking space serve as state data. Specifically, DE-RLFR is based on the Q-learning framework, treating each individual in the population as an agent and encoding hierarchical state variables using adaptive ranking values. The reward function was designed to guide the population to gradually approach PF. Based on the reinforcement learning experience represented by the corresponding Q-table values, each agent can adaptively choose a mutation strategy to generate offspring individuals. The flowchart of DE-RLFR is depicted in Fig. 6(a). Similar works have utilized RL to configure EAs adaptively [111–115], where RL learns with base-optimizer evolving. Hence, the meta-optimizer in these works have poor generalization performance.

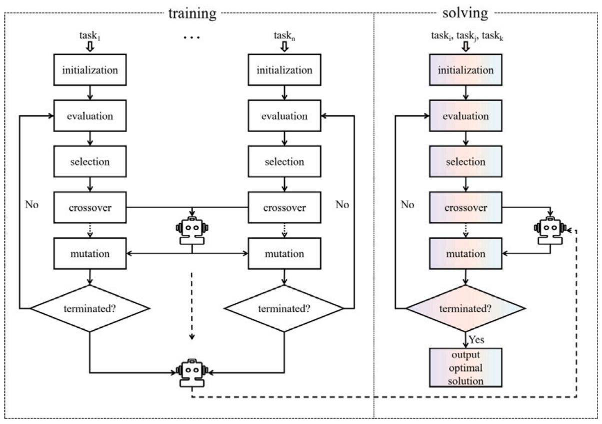
Sharma et al. [116] proposed a more general RL-based DE named DE-DDQN, where a Double Deep Q-network (DDQN) is utilized to select mutation strategies within DE, which is a kind of MetaBBO methodology. DE-DDQN can be divided into two stages. First, by collecting data on the DE state and the benefits (rewards) of applying each mutation strategy in each iteration, a neural network is trained offline, which comes from running DE multiple times to solve the benchmark function; Secondly, when applying DDQN as a parameter controller in DE to another benchmark function test set, DDQN utilizes the trained neural network to predict the mutation strategy that each parent should adopt for each generation based on the DE state. The flowchart of DE-DDQN is shown in Fig. 6.

Similar to DE-DDQN, Tan et al. utilized Deep Q-network to adaptively select the mutation strategies of DE in [117], termed as DEDQN. A historical memory parameter adaption mechanism is adopted to improve DEDQN.

Sun et al. introduced deep reinforcement learning into differential evolution as a parameter controller, and the prediction during the



(a) flowchart of DE-RLFR



(b) flowchart of DE-DDQN

Fig. 6. The flowcharts of DE-RLFR [110] and DE-DDQN [116]. Roberts with different colors represent different agents. DE-RLFR will train one agent while evolving for each optimization problem. DE-DDQN will pre-train only one agent on quantitative optimization tasks and leverage it in unseen tasks.

search process is the control parameters of DE such as the scale factor and crossover rate [118]. Specifically, at each generation, the control parameters are determined by a non-linear function that is emulated through a deep neural network (DNN). DNN's parameters are acquired through the experiences gained from optimizing a set of training functions utilizing the proposed differential evolution. The learning process is rooted in the framing of the evolutionary methodology as a finite-horizon Markov decision process (MDP). To ascertain the optimal parameters for the DNN, policy gradient [119] is employed as the reinforcement learning algorithm.

Shala et al. [120] utilized RL to learn the adaptive strategy of mutation step-size in the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [121]. They formulated the dynamic adjustment of the mutation step-size in CMA-ES as a reinforcement learning problem, aiming to learn policies that can dynamically configure this parameter. The state space includes the current step-size, cumulative path length, changes in objective values, and historical step-size data, considering information from different functions and optimization processes. The action space represents continuous adjustments of the step-size parameter. The reward function is defined based on the function value at each iteration, optimizing performance at any given time. To learn these policies, the guided policy search (GPS) [122] technique is employed, a type of reinforcement learning originating from the robotics community, capable of learning complex policies from limited trials. The study showcases the generalization ability of the learned policies, demonstrating their effectiveness on higher-dimensional functions and unseen test functions. This marks the first application of policy search methods for dynamic configuration in evolutionary algorithms, contrasting with conventional value-based reinforcement learning approaches.

Schuchardt et al. [123] used deep reinforcement learning to learn to dynamically adjust the strategy of evolutionary algorithms to varying circumstances. Yi et al. [124] optimized population-based general algorithms based on deep Q-network methods and proximal strategy optimization methods. The proposed RL-based optimizer can intelligently select and combine appropriate generic algorithmic components at different stages of the optimization process, including selection heuristics and evolution operators. Guo et al. [125] modeled dynamic algorithm selection as a Markov decision process, trains agents in the form of policy gradients, and selects the most suitable algorithm based on the observed features during the optimization process. Ma et al. [126] proposed a deep reinforcement learning-based framework that autonomously configures and adapts the exploration–exploitation trade-off throughout the EC search process. There are also some studies that focus on utilizing Q-learning to design EAs [20–22].

Another novel work by Chen et al. [127] utilized the idea of symbolic regression to generate interpretable update rules, where neural networks are used to represent update formulas and three distinct strategies based on reinforcement learning are developed to meta-learn the Symbolic Equation Generator efficiently.

Besides EAs, Particle Swarm Optimization (PSO) [128], as another important algorithm for BBO, along with genetic algorithms, differential evolution, etc., together constitute the basic framework of evolutionary computing. Despite the commendable optimization performance exhibited by the original PSO algorithm, it remains inherently susceptible to premature convergence, posing a significant challenge. Consequently, numerous researchers have dedicated their efforts to modifying and enhancing the algorithm, yielding a diverse array of PSO variants that offer either incremental or substantial improvements in performance [129,130]. These advancements reflect the ongoing pursuit of refining and optimizing the PSO framework to address its limitations and capitalize on its potential. The study of RL for optimizing PSO has been a hot topic in recent years. Samma et al. [131] proposed a new hybrid particle swarm optimization algorithm based on reinforcement learning (RLMPSO). The algorithm consists of four global search operations and a new local search operation. Under the control of the RL algorithm, each particle can perform five operations: exploration, convergence, high jump, low jump, and local fine-tuning. The historic behavior of the swarm group is used to update the policy network and guide future actions. Specifically, each particle has a Q-table that stores reward values for taking different actions in different states. During the search process, the agent (particle) decides which action to take based on the current state and Q-table, performs the operation, and updates the values in the Q-table based on the results. In this way, the agent can gradually learn which operation to take in a specific state to obtain the maximum cumulative reward. Similarly, there are some state-of-the-art studies employing reinforcement learning to enhance Particle Swarm Optimization via predicting the particle update function [132–136]. It is proven that Reinforcement Learning can take advantage of valid history information to guide the behavior of individuals during evolution and improve the convergence rate of PSO. However, like DE-RLFR, the agents (i.e. particles) in these algorithms initialize, gradually learn and adapt as the search process progresses. They cannot generalize to other problem instances, so they do not belong to MetaBBO strictly speaking.

Song et al. proposed a novel Reinforced In-Context Black-Box Optimization method named RIBBO [137], utilizing end-to-end reinforcement learning black-box optimization algorithms from offline data. Offline data consists of optimization history data generated by multiple behavioral algorithms (such as random search, genetic algorithm, Bayesian optimization, etc.) collected on multiple tasks (such as synthesis function, hyperparameter optimization, robot control, etc.). RIBBO uses highly expressive sequence models to learn various behavioral algorithms and optimization histories generated by tasks, and utilizes the contextual learning ability of large models to extract task information and make decisions based on it.

Reinforcement learning has shown promising results in optimizing evolutionary algorithms. It can guide the behavior of individuals during evolution, improve convergence rates, and dynamically adjust optimization strategies. However, reinforcement learning algorithms are difficult to handle high-dimensional state spaces. A high-dimensional state space can lead to a curse of dimensionality, requiring the design of new algorithms to handle it. Meanwhile, reinforcement learning algorithms are difficult to handle large-scale multitasking learning problems. When learning multiple tasks, the current algorithm is difficult to achieve good results on large-scale tasks.

4.1.3. *Large language model*

The emergence of the Large Language Model (LLM) has attracted widespread attention and discussion. There are also some studies utilizing LLM for MetaBBO.

Zhang et al. explored the novel application of large language models (LLMs) for learning a general hyperparameter optimization optimizer [138]. They treated the process of hyperparameter optimization as a sequence prediction problem, where the objective was to predict the next best hyperparameter configuration given a series of previous configurations and their corresponding performance metrics. By using LLMs, the authors proposed a framework where the model was trained to predict hyperparameter values from the input space, effectively learning patterns and relationships between the hyperparameters and the performance outcomes of machine learning models. Yang et al. leveraged LLM to generate new solutions from the prompt that contains previously generated solutions with their values, and then the new solutions are evaluated by the objective function evaluator and added to the prompt for the next optimization step [139]. Liu et al. proposed an LLM-driven EA (LMEA) [140]. LMEA instructs LLM to select parent solutions from the current population and perform crossover and mutation to generate offspring solutions. Then, LMEA evaluates these new solutions and incorporates them into the next-generation population.

LLM can automate many optimization tasks, simplify research and development processes, and reduce reliance on expert knowledge. However, large language models rely on a large amount of training data, which may lead to dependencies on data quality and diversity, as well as issues of data bias. Meanwhile, compared to traditional optimization algorithms, the decision-making process of LLMs may not be transparent enough, making their interpretability a challenge. LLM for MetaBBO needs further exploration.

4.2. *Local search-based meta-optimizer*

Iterative Local Search (ILS) [141] is a meta-heuristic optimization algorithm that combines the concept of local search and iterative processes to improve the ability to find high-quality solutions in complex optimization problems. ILS enhances the global search capability of the algorithm by applying local search processes multiple times and introducing changes in each iteration to escape the local optimal solution.

ParamILS is the initial MetaBBO framework utilizing ILS to configure algorithm [59]. They provided a method to optimize the performance of the base-optimizer for a given class of problem instances by changing a set of ordered and/or classification parameters. A new technique to accelerate the evaluation of individual configurations by adaptively limiting the time spent evaluating them is proposed as well.

Lin et al. [142] used a multilevel ParamILS to optimize ACO and applied it in transportation planning problems [143]. Blot et al. extended ParamILS framework into multi-objective optimization, termed as MO-ParamILS.

Recently, Turky et al. [144] proposed a hyper-heuristic framework that includes multiple local search algorithms and a set of neighborhood structures. This framework designs a two-stage hyperheuristic structure to control the selection of local search algorithms and their internal operations. In addition, the author proposes an adaptive ranking mechanism to select the most suitable neighborhood structure for the current local search algorithm. The proposed mechanism uses entropy to evaluate the contribution of local search in terms of quality and diversity, and adaptively adjusts the pool of candidate neighborhood structures.

ILS achieves local optima by locally improving only one dimension of the current algorithm configuration in each iteration, and re-initializes the current algorithm configuration with a certain probability through global perturbation to escape local optima. ILS is more suitable for situations where the design space is relatively compact and easy to identify good starting points, as well as problems with limited computational budgets. Additionally, ILS performs poorly in handling high-dimensional design spaces.

4.3. Evolutionary computation-based meta-optimizer

Meta-learning via algorithm-based optimization has been demonstrated effective when learning gradient-based optimization [145] and discovering RL policy optimization [146] recently.

In early work of EA for EA is the meta-GA algorithm that used a genetic algorithm (GA) to tune the parameter settings of another GA [147]. REVAC [148] used multi-parent cross-over and entropy measures to estimate the relevance of parameters. The gender-based GA uses various sub-populations and a specialized cross-over operator to generate new candidate configurations [149]. Recently, Hugo et al. [150] leveraged a modified DeepGA [151] as the meta-optimizer, which maintains a population of the inner population-based algorithm parameters. A recurrent neural network was used to parameterize population-based algorithm, and a meta-loss function based on stochastic algorithm performance was utilized to train efficient data-driven optimizers over several related optimization tasks in this work.

Iterated racing [55] is an ES-based methodology for MetaBBO that consists of three steps: (1) sampling new configurations according to a particular distribution, (2) selecting the best configurations from the newly sampled ones by means of racing, and (3) updating the sampling distribution in order to bias the sampling towards the best configurations. I/F-Race [152,153] is regarded as a special case that uses Friedman's non-parametric two-way analysis of variance by ranks. The study [154] proposed a framework that allows to instantiate the most prominent multi-objective ACO (MOACO) algorithms and used I/F-Race to optimize the MOACO.

Lu et al. [146] utilized evolution strategies in meta-training to discover a new state-of-the-art RL algorithm within the Mirror Learning space [155], which provides an infinite space of provably correct algorithms, all of which share the same template. The outcome of this meta-training is a specific Mirror Learning algorithm named as Learned Policy Optimization (LPO). Building upon LPO's drift discovered, Discovered Policy Optimization (DPO) was proposed as the first theoretically-sound, scalable deep RL algorithm that was discovered via meta-learning. This meta-evolution approach was adopted in MetaBBO [10].

Lange et al. in [10] first proposed the concept of "MetaBBO" to discover effective update rules for evolution strategies via meta-learning. A search strategy parameterized by a self-attention-based architecture is adopted to achieve the learned evolution strategy (LES), where the concept of self-attention was formally proposed by Vaswani et al. [156] and applied to Transformer model [157]. The framework of LES is depicted in Fig. 7. In order to learn the parameters of LES (such as weights of query, key and values, i.e. W_Q, W_K, W_V), at meta-level, a set of classic BBOPs will be sampled to characterize a space of representative optimization problems first. Then a set of candidate LES parameters will be initialized. Next, the LES architecture is introduced as an inner loop search of MetaBBO, where candidate solutions from a Gaussian search distribution are sampled, evaluated and transformed iteratively on a set of inner loop optimization problems. The inner loop will generate a set of recombination weights used to update the diagonal Gaussian search distribution, and an additional MLP module will compute per-dimension learning rates by processing momentum-like statistics and a timestamp. At the end of the MetaBBO loop, the meta-performance of different LES parameter combination is normalized and the LES search distribution is updated. This study made it clear that it is feasible to self-referentially train an ES with the learned update rule used to drive the outer evolution loop. Then, Lange et al. introduced the attention-based architecture in Genetic Algorithms to substitute the generators where selection and mutation rate adaption are cast as dot-product attention modules [158]. At meta-level, meta-evolution using MetaBBO is developed to train the learned genetic algorithm (LGA). The attention-based architecture can help trained LGA derive a neural network-based family of genetic algorithms, which makes LGA can generalize to unseen tasks.

Genetic programming is another important evolutionary search method, which was initially proposed by John Koza as an automated machine learning technique employing natural selection and genetic mechanisms to evolve computer programs [159]. It provides a way to search the space of possible computer programs for a highly fit individual computer program to solve a series of different problems [160]. Genetic programming has been extensively explored and applied to various fields, including machine learning problems such as feature engineering and automatic model discovery, as well as in operation research (OR) such as complex scheduling and routing problems. Recently, Olson et al. have developed a Tree-based Pipeline Optimization Tool (TPOT) for Automated machine learning and scaled it to biomedical big data [161–163], where GP is utilized to optimize a machine learning pipeline that maximizes score on the provided features and target. In the context of MetaBBO, especially in the application of Evolutionary Algorithms, GP can be used to optimize algorithm parameter settings or discover new optimization strategies and operators.

4.4. Bayesian optimization-based meta-optimizer

Bayesian optimization is a method based on sequential models to solve problems, which consists of a prior distribution and an observation model that describes the data generation mechanism. The former captures belief in the behavior of the unknown objective function. The second part is to acquisition function, which is optimized at each step to balance exploration and exploitation. Bayesian optimization is similar to evolution strategies for both including sample and update steps. Differently, Bayesian optimization needs to construct surrogate model to predict the base-optimizer performance.

With the help of Bayesian Optimization, a general-purpose Black-Box optimizer, is learned in [164]. Sergio et al. [164] extended the generic optimizer with RNN as the update rule. First, numerous differentiable functions are generated using Gaussian processes as training data. Then, the RNN optimizer is trained to utilize information from previous query points to determine the next query point. Next, acquisition functions are used to guide the training process and encourage RNN exploration. Through this approach, an RNN optimizer has been trained that can be applied to black-box global optimization, and demonstrated its generalization ability on various functions in experiments.

Besides Gaussian process, random forest is another common surrogate model in BO. Sequential Model Based Optimization for General Algorithm Configuration (SMAC), was proposed by Hutter et al. in 2011 [165]. The proposal of this algorithm solves the problem of parameter types not being discrete in Gaussian regression process. Tang et al. [47] utilized SMAC to optimize generic framework. In outer optimization, SMAC serves as a mutation operation to generate new configurations.

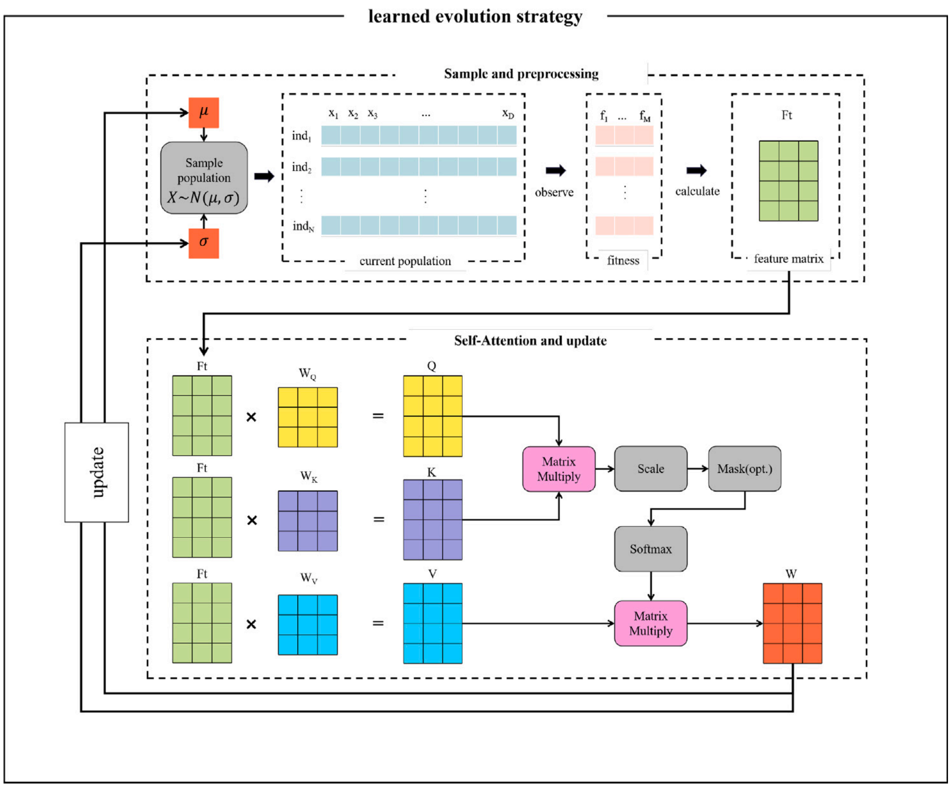


Fig. 7. Framework of LES.

4.5. Discussion

Table 12 summarizes the specific methods or models of optimizers in each MetaBBO related literature mentioned above, for researchers to have a more intuitive overview, while Table 13 presents the advantages and disadvantages of each type of meta-optimizer.

Machine learning, especially reinforcement learning, stands out for its fine-grained search ability and long-term planning of value functions. It can build algorithms one by one operator or one by one primitive, achieving a more efficient design process than building the entire algorithm at once. However, reinforcement learning may be limited by linear arrays in algorithm representation, and further research is needed on advanced encoding decoding mappings to innovate algorithm structures. Reinforcement learning is suitable for scenarios with rich problem instances and the need to gradually optimize solutions in online scenarios.

Local search is an optimization technique that focuses on dense design spaces. It relies on expert knowledge and previous experiments to select the starting point and quickly converge to the local optimal solution. However, local search may lack a global perspective, easily fall into local optima, and be very sensitive to the selection of initial solutions. It is suitable for scenarios where the design space is relatively compact and easy to identify good starting points, as well as when the computational budget for the problem is limited.

Evolutionary algorithms, as adaptive systems, provide higher adaptability than local search and can evolve suitable algorithms for different problem instances. They enhance their adaptability to difficult problems by evolving from simple to complex problem instances. Nevertheless, evolutionary algorithms face the challenge of dealing with endogenous parameters in algorithm operators and require research on strategies for managing dual-layer searches. Evolutionary algorithms are suitable for scenarios where the design space is composed of existing algorithm operators and has predefined algorithm templates.

Bayesian optimization is favored for its good performance in low dimensional problems and effectiveness in expensive computational problems. It predicts the objective function by constructing a probability model and uses the acquisition function to guide the search process. The challenge of Bayesian optimization lies in its scalability in high-dimensional problems and the complexity of algorithm representation. It is suitable for optimization problems with high computational costs and the need to reduce the number of function evaluations.

When developing novel MetaBBOs, it is necessary to select an appropriate methodology according to their advantages and disadvantages, as well as the problem characteristics.

5. Benchmarks, evaluation metrics and platforms

5.1. Benchmarks

Proprietary benchmarks for MetaBBO have not been available yet currently. But since it is fundamentally proposed to solve BBOPs, Some Black-Box Optimization Benchmark (BBOB) test suites are leveraged to evaluate specific aspects of the algorithms, usually sampled as the tasks.

These test suites can be classified into two categories: synthetic and realistic.

5.1.1. Synthetic

There are 24 noise-free real-parameter single-objective synthetic functions proposed by Hansen et al. in 2009 for quantifying and evaluating the performance of BBOAs [166], termed as **BBOB2009**. The dimension D of all the benchmarks is scalable. Most functions have no specific optimal solution x_{opt} which are shifted in x-space randomly, while the optimal function values $f_{opt} = f(x_{opt})$ randomly shifted in f-space are self-selected. Hence, different instances for each function could be generated. Predefined groups are separated into five groups based on their characteristics including multimodality, global structure, separability, variable scaling, homogeneity, basin-sizes and global to local contrast. The specific groups are separable functions, functions with low or moderate conditioning, functions with high conditioning and unimodal, multimodal functions with adequate global structure,

Table 12
Summary of state-of-the-art MetaBBO methodologies and their applications in literature, where “Any” represents any parameterization algorithms.

Reference	Time	Meta-optimizer/methodology	Base-optimizer	Optimization object
[107]	2022	ML/NN	Any	hyperparameters
[105]	2016	ML/RNN	RNN	update rule
[106]	2017	ML/RNN	BBOA	update rule
[108]	2019	ML/RNN	BBOA	update rule
[116]	2019	ML/RL	SOEA/DE	**GenerateVariants**-mutation strategy
[123]	2019	ML/DRL	EA	evolutionary strategy
[120]	2020	ML/RL	SOEA/ES	mutation rate
[117]	2021	ML/RL	SOEA/DE	**GenerateVariants**-mutation strategy
[118]	2021	ML/DRL	SOEA/DE	hyperparameters
[124]	2022	ML/RL	MA-SOEA	components
[137]	2024	ML/RL	BBOA	AS strategy
[138]	2023	ML/LLM	Any	hyperparameters
[139]	2023	ML/LLM	Any	update rule
[140]	2023	ML/LLM	SOEA/EA	**GenerateVariants**
[125]	2024	ML/RL	SOEA/DE	algorithm selection
[126]	2024	ML/DRL	EC-SOEA/PSO	**GenerateVariants**
[20]	2022	ML/Q-learning	MOEA	hyperparameters
[21,22]	2023	ML/Q-learning	SOEA	hyperparameters
[127]	2024	ML/DRL	BBOA	update formulas
[59]	2009	LS/ILS	Any	hyperparameters
[143]	2015	LS/ILS	MOEA	hyperparameters
[142]	2016	LS/ILS	MOEA-ACO	hyperparameters
[144]	2020	LS/ILS	LS	AS+AC
[147]	1986	EC/GA	SOEA/GA	hyperparameters
[148]	2007	EC/GA	SOEA	hyperparameters
[149]	2009	EC/GA	Any	hyperparameters
[150]	2021	EC/GA	BBOA	update rule
[55]	2016	EC/ES	Any	hyperparameters
[153]	2007	EC/ES	Any	hyperparameters
[152]	2010	EC/ES	Any	hyperparameters
[154]	2010	EC/ES	MOEA-ACO	hyperparameters
[146]	2022	EC/ES	RL	update rule
[10]	2022	EC/ES	ES	update rule
[158]	2023	EC/ES	GA	**GenerateVariants**+hyperparameters
[161–163]	–	EC/GP	ML	ML pipeline
[164]	2016	BO	BBOA	update rule
[47]	2021	BO	Any	AC

Table 13
Summary of optimization methodologies in MetaBBO for EAs.

Methodology	Advantages	Disadvantages	Suitability
ML	Fine-grained search ability, capable of long-term planning	May require large datasets, limited by representation	Complex problems with rich instances and information
LS	Fast convergence to local optima, simple to implement	Lacks global perspective, sensitive to initial solutions	Base-optimizer with compact design spaces
EA	High adaptability, evolves suitable algorithms for different problems	Complex dual-layer search, sensitive to parameter tuning	Complex problems without rich information
BO	Efficient in high-dimensional spaces, balances exploration and exploitation	Computationally expensive, scalability issues	Expensive BBOPs

and multimodal functions with weak global structure in sequence. The detailed characteristics of each function are taken from [167] and the different groups can be distinguished by horizontal lines, as given in Table 14.

All functions can be evaluated over $\mathcal{R}^D$ and the search space is given as $[-5,5]^D$. The location of x_{opt} is drawn uniformly from the search space while f_{opt} is drawn from a Cauchy distributed random variable with zero median and with roughly 50% of the values between −100 and 100. It is noted that some functions cover penalty boundary handling. Detailed function definitions can be found in [166].

Apart from the above noiseless functions, BBOB 2009 workshop provides noisy functions as well, where the noises generated according to Gaussian noise, Uniform noise and Cauchy noise are introduced into the above synthetic noiseless functions [168]. Gaussian noise and Uniform noise are multiplicative noise while the Cauchy noise is additive. Assuming that $f \le 0$, three kinds of noises are applied on f and reveal stochastic dominance between any two solutions and are therefore utility-free.

The Gaussian noise is defined as

$$f_{GN}(f,\beta) = f \times exp(\beta\mathcal{N}(0,1)) \tag{3}$$

where β controls the noise strength. $\beta = 0.01$ corresponds moderate noise, otherwise $\beta = 1$.

The Uniform noise is defined as

$$f_{UN}(f,\alpha,\beta) = f \times \mathcal{U}(0,1)^{\beta} \max(1, (\frac{10^9}{f+\epsilon})^{\alpha\mathcal{U}(0,1)}) \tag{4}$$

where α and β control the noise strength. Moderate noise is corresponding to $\alpha = 0.01(0.49 + D^{-1})$ and $\beta = 0.01$, otherwise $\alpha = 0.49 + D^{-1}$ and $\beta = 1$. ϵ is set to 10^{-99} to avoid division by zero.

The Cauchy noise is defined as:

$$f_{CN}(f,\alpha,p) = f + \alpha \max(0, 1000 + \Pi_{\{\mathcal{U}(0,1)<p\}} \frac{\mathcal{N}(0,1)}{|\mathcal{N}(0,1)|+\epsilon}) \tag{5}$$

where α controls the noise strength and p represents the frequency of the noise disturbance. Moderate noise is corresponding to $\alpha = 0.01$ and

Table 14
Noiseless benchmarks and their characteristics (multimodality, global structure, separability, variable scaling, homogeneity, basin-sizes and global-to-local contrast). 'v' represents the function possesses the corresponding characteristic while 'x' represents the function does not possess the corresponding characteristic. Predefined groups are separated by horizontal lines.

No.	Function	mul.	gl.st.	sep.	var.	hom.	bas.	gl.-loc.	Groups
1	Sphere	x	x	v	x	v	x	x	Separable functions
2	Ellipsoidal separable	x	x	v	v	v	x	x	
3	Rastrigin separable	v	v	x	v	v	v	v	
4	Bueche–Rastrigin	v	v	v	v	v	v	v	
5	Linear Slope	x	x	v	x	v	x	x	
6	Attractive Sector	x	x	v	v	v	x	x	Functions with low or moderate conditioning
7	Step Ellipsoidal	x	x	v	v	v	x	x	
8	Rosenbrock	v	x	x	x	v	v	v	
9	Rosenbrock rotated	v	x	x	x	v	v	v	
10	Ellipsoidal	x	x	x	v	v	x	x	Functions with high conditioning and unimodal
11	Discus	x	x	x	v	v	x	x	
12	Bent Cigar	x	x	x	v	v	x	x	
13	Sharp Ridge	x	x	x	v	v	x	x	
14	Different Powers	x	x	x	v	v	x	x	
15	Rastrigin	v	v	x	v	v	v	v	Multi-modal functions with adequate global structure
16	Weierstrass	v	v	x	v	v	v	v	
17	Schaffer F7	v	v	x	v	v	v	v	
18	Schaffer F7 mod.ill-con	v	v	x	v	v	v	v	
19	Griewank–Rosenbrock	v	v	x	x	v	v	v	
20	Schwefel	v	v	x	x	v	v	v	Multi-modal functions with weak global structure
21	Gallagher 101 Peaks	v	x	x	v	v	v	v	
22	Gallagher 21 Peaks	v	x	x	v	v	v	v	
23	Katsuura	v	x	x	x	v	v	v	
24	Lunacek bi-Rastrigin	v	v	x	v	v	v	v	

$p = 0.05$, otherwise $\alpha = 1$ and $p = 0.2$. The sum of 1000 samples positive and negative 'outliers' and ϵ is set to 10^{-199}.

Above all, the noisy functions can be uniformly defined as

$$f_{..}(f,..) = \begin{cases} f_{..}(f,..) + 1.01 \times 10^{-8}, f \le 10^{-8} \\ f, otherwise \end{cases} \tag{6}$$

There are totally 30 noisy functions within this definition and the summary is given in Table 15.

5.1.2. Realistic

One of the realistic benchmarks is **protein–protein docking benchmark (PPDB)** which provides non-redundant protein–protein complexes for which the complex structure and the constituent unbound structures [169]. The new PPDB version 4.0 provides 176 unbound–unbound cases classified into 121 rigid body cases, 30 medium difficulty cases and 25 difficult cases according to expected difficulty for protein–protein docking algorithms. They are available at http://zlab.umassmed.edu/benchmark/.

When EAs are employed for hyperparameter optimization of ML, ML test suites can be used as well. **UCI** Machine Learning Repository provides more than five hundred datasets from different fields like healthcare, banking, science, and business [170]. Classical datasets include the Iris dataset (about flower types), the Breast Cancer dataset (for medical research), and the Adult Income dataset (for studying salary patterns). Each dataset comes with clear descriptions about what the data means, how it can be used, and which research papers have used it before. This makes it easy for people to understand and use the data correctly. UCI can be found at https://www.uci.org/. Besides, Kaggle stands as the largest data science community platform, offering diverse competition datasets. **MNIST** [171] and **Fashion-MNIST** [172] serve as standard datasets for deep learning beginners, particularly in image classification tasks. **ImageNet** [173] represents a benchmark database in computer vision research.

5.2. Evaluation metrics

Evaluation reflects the performance of the algorithms and guides to find desired algorithms. The specific evaluation metrics may vary depending on different application scenarios and research objectives. In practical applications, it may be necessary to design appropriate evaluation metrics based on specific problems and task requirements to comprehensively evaluate the performance of MetaBBO methodologies. Ideally, the evaluator should be quantitative, ideally with a ratio scale, with wide variation, well-interpretable, relevant and meaningful with respect to the real world, available for most targeted algorithms and problems, readily reproducible independently of specific experimental settings, comparable across different problems and as simple and comprehensible as possible [174,175]. As a counterexample, the third decimal digit of the quality indicator would be a well-defined and achievable goal but neither relevant nor meaningful in practice, unless also the preceding digits are minimal. The norm of the gradient is meaningful and relevant on unimodal differentiable functions, but the gradient is often not available in the BBO setting. Here, general evaluation metrics employed in the literature compliant with MetaBBO are summarized. Through analysis, these indicators can be divided into quality, cost and comprehensive categories.

5.2.1. Quality indicators

Quality indicator is defined to assess the solution quality within a predefined budget. It depends on the sequence of visited solutions and maps it to a real value (i.e. quality indicator value). The specific definition varies from different optimization scenarios. For example, quality indicator is usually the function value of the last solution in the sequence for single-objective unconstrained noiseless optimization, same as taking the best objective value [176] as the evaluator. For multi-objective optimization, quality indicator usually refers to solution convergence and solution diversity [177,178], such as PD [179], IGD [180], HV [181] and so on. For constrained optimization, quality indicator is defined as a mapping of the solution sequence $(x_1, \dots, x_t)$ to the value $|f(x_t) - f_{opt}|^+ + \sigma_i \lambda_i |g_i(x_t)|^+$ where $f(x_t)$ the objective value at time step t with the solution x_t, f_{opt} is the f-value of the feasible optimum, $g_i(x)$ represents the non-positive constraints in the feasible domain, $|.|^+$ clips the argument to positive values, and λ_i are positive weights for the constraints. The detailed definitions in other scenarios can be found in [175].

Table 15
Noisy benchmarks and their groups.

No.	Basic function	Functions	Group
1		Sphere with moderate gaussian noise	
2	Sphere	Sphere with moderate uniform noise	
3		Sphere with moderate seldom Cauchy noise	Functions with moderate noise
4		Rosenbrock with moderate gaussian noise	
5	Rosenbrock	Rosenbrock with moderate uniform noise	
6		Rosenbrock with moderate seldom Cauchy noise	
7		Sphere with gaussian noise	
8	Sphere	Sphere with uniform noise	
9		Sphere with seldom Cauchy noise	
10		Rosenbrock with gaussian noise	
11	Rosenbrock	Rosenbrock with uniform noise	
12		Rosenbrock with seldom Cauchy noise	
13		Step ellipsoid with gaussian noise	
14	Step ellipsoid	Step ellipsoid with uniform noise	Functions with severe noise
15		Step ellipsoid with seldom Cauchy noise	
16		Ellipsoid with gaussian noise	
17	Ellipsoid	Ellipsoid with uniform noise	
18		Ellipsoid with seldom Cauchy noise	
19		Different Powers with gaussian noise	
20	Different Powers	Different Powers with uniform noise	
21		Different Powers with seldom Cauchy noise	
22		Schaffer's F7 with gaussian noise	
23	Schaffer's F7	Schaffer's F7 with uniform noise	
24		Schaffer's F7 with seldom Cauchy noise	
25		Composite Griewank–Rosenbrock with gaussian noise	
26	Composite Griewank–Rosenbrock	Composite Griewank–Rosenbrock with uniform noise	Highly multimodal functions with severe noise
27		Composite Griewank–Rosenbrock with seldom Cauchy noise	
28		Gallagher's Gaussian Peaks 101-me with gaussian noise	
29	Gallagher's Gaussian Peaks, globally rotated	Gallagher's Gaussian Peaks 101-me with uniform noise	
30		Gallagher's Gaussian Peaks 101-me with seldom Cauchy noise	

5.2.2. *Cost indicators*

Cost indicator measures how much resource the algorithm generated via MetaBBO consumes when solving a problem instance to achieve a predefined threshold. The threshold can be the solution quality or a given loss/accuracy, while the resource can be running time or memory. Taking running time for example, formally, running time is represented by evaluations. Taking running time for example, which points to expected running time (ERT) [174], which can be formulated as follows [182]. Given a predefined threshold value ϕ_i for problem instance P_i, the ERT of the algorithm A solving P_i for hitting ϕ_i is

$$ERT(A, P_i, \phi_i) = \frac{\sum_{n=1}^{N} \min\{t_i(A, P_i, \phi_i), t_i^{co}\}}{\sum_{n=1}^{N} \mathbb{1}\{t_n(A, P_i, \phi_i) < \infty\}} \tag{7}$$

where n is the number of independent runs with N repeated runs of A on P_i, $t_i(A, P_i, \phi_i)$ which is in $N \cup \{\infty\}$ denotes the running time (if not hitting the ϕ_i, $t_i(A, P_i, \phi_i) = \infty$), t_i^{co} is the cutoff time(i.e.the maximum number of evaluations manually set up), and $\mathbb{1}$ will be 1 if the content in following curly braces is true, otherwise it will be 0. It is noted that if A can hit ϕ_i in each run, the ERT is equal to the average hitting time (AHT). ERT is expected to be smaller and conducive to firstly-hit the threshold.

To optimize the anytime performance, the area under the empirical cumulative distribution function curve of running times (AUC) is considered further in [182]. Given a series of threshold values $\Phi_i = \{\phi_{i,1}, \dots, \phi_{i,J}\}$ and a series of cutoff time values $T_i^{co} = \{t_i^{co,1}, \dots, t_i^{co,K}\}$ of A on P_i, AUC can be calculated as:

$$AUC(A, P_i, \Phi_i, T_i^{co}) = \frac{\sum_{n=1}^{N} \sum_{j=1}^{J} \sum_{k=1}^{K} \mathbb{1}\{\phi_n(A, P, t_i^{co,k}) \geq \phi_{i,j}\}}{N \times J \times K} \tag{8}$$

where $\phi_n(A, P, t_i^{co,k})$ represents the solution quality, loss or accuracy that A evaluated within its first $t_i^{co,k}$ evaluations in the nth run. The interval AUC is [0,1] and AUC is expected to be larger.

The resource could be substituted with each other above, and another limited resource is utilized as the cutoff. For example, the cost indicator can be also denoted to measure the occupied memory to hit the target threshold. The corresponding cost indicator values can be calculated according to derived Eqs. (7) and (8).

5.2.3. *Comprehensive indicators*

Comprehensive indicators provide an overview of the optimization performance, which consider multiple aforementioned indicators. They are of more meaning in some scenarios.

Zeyuan Ma et al. have proposed three comprehensive indicators to evaluate the methodology performance as follows [183].

Aggregated Evaluation Indicator (AEI) aggregates three traditional BBO performance metrics consisting of the best objective value v_{bobj}, consumed function evaluation times v_{feva}, and the runtime complexity v_{com}. It assumes that the methodology has been tested on I problem instances with N repeated runs. These traditional values have been subjected to a logarithmic transformation and Z-score normalization is applied:

$$Z_*^i = \frac{1}{N} \sum_{n=1}^{N} \frac{log(v_*^{i,n}) - \mu_*}{\sigma_*} \tag{9}$$

where μ_* and σ_* are calculated by using Random Search (RS) as a baseline.

Then AEI could be calculated as follows.

$$AEI = \frac{1}{I} \sum_{i=1}^{I} e^{(Z_{bobj}^i + Z_{feva}^i + Z_{com}^i)} \tag{10}$$

A higher AEI is expected which indicates better performance.

Meta Generalization Decay (MGD) assesses the generalization of a learned methodology across different problems especially unseen tasks. AEI_B represents the values when training a model on a problem set B and is calculated via the corresponding test set. Another model will be trained using the same methodology on problem set A and the AEI on its test set is denoted as AEI_A. The $MGD(A, B)$ could be calculated as follows.

$$MGD(A, B) = 100 \times (1 - \frac{AEI_A}{AEI_B})\% \tag{11}$$

A smaller $MGD(A, B)$ is expected which indicates the methodology generalizes well from A to B.

Meta Transfer Efficiency (MTE) metric quantifies the transfer learning ability of RL-based MetaBBO methodologies when zero-shot generalization is unachievable because of significant task disparity. The checkpoint with the highest cumulative return is located first and its index is denoted as T_{scr} when training on a problem set B. Then a checkpoint of the RL-based MetaBBO approach on another problem set A is pre-trained and loaded back.

Meta-loss function is another comprehensive indicator defined as follows according to [150]:

$$\min_{\theta} \mathbb{E}_{w_k \sim p(w)}[J^k(\pi_\theta)] \tag{12}$$

where $J^k(\pi_\theta)$ is the performance of the policy π_θ, which represents a parameterized learned optimizer, for the task w_k sampled from $p(w)$. The performance of π_θ is calculated as the aggregate of the expected number of function evaluations (FE) to reach a certain function value (success criterion) of w_k by simulating independent restarts of π_θ from various run. The basis is expected runtime of the restart algorithm which can be calculated following the derivations in [184].

Let $p_s \in (0, 1]$ be the probability of success of π_θ to reach a certain function value of w_k and FE_{max} be the maximum number of function evaluations. Then, the performance $J(\pi_\theta, w_k) = FE(\pi_\theta, w_k) = FE_{\theta,w_k}$ is a random variable measuring the number of FE until a success criterion is met by independent restarts of π_θ on the task $w_k \sim p(w)$:

$$FE_{\theta,w_k} = \sum_{i=1}^{N} FE_{max} + FE^{succ}_{\theta,w_k} \tag{13}$$

where N is the random variable that measures the number of unsuccessful runs of π_θ required to reach once the success criterion and FE^{succ}_{θ,w_k} as the number of evaluations for the successful run of π_θ to reach the criterion.

Then the expectation $\mathbb{E}[FE_{\theta,w_k}]$ need to be evaluated. The conditional expectation of Eq. (13) w.r.t. N can be calculated as follows.

$$\mathbb{E}[FE_{\theta,w_k}|N] = NFE_{max} + \mathbb{E}[FE^{succ}_{\theta,w_k}] \tag{14}$$

where $N \sim NB(r = 1, p = 1 - p_s)$ following a negative binomial distribution and its expectation is $\mathbb{E}(N) = \frac{rp}{1-p} = \frac{1(1-p_s)}{1-(1-p_s)} = \frac{1-p_s}{p_s}$. Hence, the expectation $\mathbb{E}[FE_{\theta,w_k}]$ can be calculated continuously as follows:

$$\begin{aligned}\mathbb{E}[FE_{\theta,w_k}] &= \mathbb{E}(N) \times FE_{max} + \mathbb{E}[FE^{succ}_{\theta,w_k}] \\ &= \frac{1-p_s}{p_s} \times FE_{max} + \mathbb{E}[FE^{succ}_{\theta,w_k}]\end{aligned} \tag{15}$$

However, p_s and $\mathbb{E}[FE_{\theta,w_k}]$ are not available and can be estimated as [185].

5.3. Platforms

In the field of OR, there are multiple evaluation platforms available to evaluate algorithm performance, compare different algorithms, and conduct experimental research. These platforms provide great convenience for researcher to develop study.

IRACE is a software package used for automatic algorithm configuration, which is based on Evolutionary Algorithms (EAs) to optimize parameter adjustment tasks [55].

ParamILS [59] **and MO-ParamILS** [142] are iterative local search packages used for single-objective and multi-objective optimization algorithms, respectively.

IOHexperimenter is a framework designed specifically for evaluating heuristic optimization algorithms for experimental research. IOHexperimenter allows researchers to compare the performance of different algorithms under the same conditions, ensuring the reproducibility and fairness of experimental results. IOHexperimenter does not rely on specific algorithm implementations and can be used in conjunction with any optimization algorithm that follows a certain interface. The framework has multiple performance indicators built-in, such as convergence speed, solution quality, stability, etc., to comprehensively evaluate algorithm performance. Importantly, IOHexperimenter supports automated experimental processes, including algorithm configuration, experimental operation, and result analysis.

Such platforms are general for optimization algorithms while the following platforms focus black-box optimization algorithm much more.

Moreover, **COCO** (Comparing Continuous Optimizers in a Black Box Setting) is a platform that provides standardized comparisons for continuous optimization algorithms, particularly suitable for black-box optimization [176]. The COCO platform provides a rich set of test cases, including various benchmark functions with different characteristics, such as multimodality, noise, conditionality, and separability, aimed at comprehensively testing the performance of optimization algorithms. Through a unified experimental protocol and evaluation metrics, COCO enables researchers to compare different optimization algorithms, including evolutionary strategies, gradient descent, Bayesian optimization, etc., on a fair and consistent basis. In addition, the automated experimental process, data visualization tools, and scalability design of COCO have further promoted the research and development of optimization algorithms. As an open source project, COCO has active community support, providing a powerful and flexible tool for comparing and analyzing optimization algorithms.

Nevergrad [186], is an open-source BBO platform that provides a range of tools and algorithms. It includes various heuristic algorithms such as Bayesian optimization, differential evolution, particle swarm optimization, etc., and supports custom complex optimization tasks. Nevergrad's design emphasizes flexibility and ease of use, allowing researchers and practitioners to quickly implement and test new optimization ideas. In addition, Nevergrad also provides rich benchmark testing and evaluation tools to help users compare the performance of different algorithms and make detailed adjustments and improvements to the algorithms. However, it provides a platform for comparing built-in optimizers but not customized algorithms and which does not provide logging functionality beyond fixed-budget simple regret.

The most noteworthy thing is that **MetaBox** developed by Zeyuan Ma et al. [183], tailors for designing and evaluating RL based MetaBBO methodology especially. MetaBox provides a flexible algorithmic template for designers and over 300 benchmark problem instances with various landscape characteristics such as multimodality, condition, global structure and so on. Moreover, three standardized metrics, including AEI, MGD and MTE, as stated in Section 5.2, are introduced to comprehensively evaluate the optimization performance and learning results of MetaBBO methodologies.

In addition, there are also many tools where MetaBBO is used for non-EA. In ML, **OpenML** [187] is an open platform for sharing datasets, algorithms, and experiments — to learn how to learn better, containing thousands of machine learning tasks and over 250 algorithm implementations, providing standardized evaluation frameworks and analysis tools. **AutoGluon** [188], supports multimodal tasks including tabular data, image, and text processing. It provides automatic feature engineering, hyperparameter optimization, and neural architecture search capabilities, while maintaining a user-friendly API interface that enables end-to-end automation of the machine learning pipeline. Notable for its advanced model stacking and ensemble learning techniques, AutoGluon can automatically construct and optimize complex multi-layer models, though it may require substantial computational resources for optimal performance. **MLBench** [189] focuses on distributed machine learning algorithm comparison with mainstream deep learning model implementations and unified performance metrics. TPOT [190] employs genetic programming to optimize complete machine learning pipelines including preprocessing and feature engineering, supporting the full range of scikit-learn algorithms, though at the cost of higher computational overhead. **H2O AutoML** [191] provides enterprise-grade deployment capabilities with automated model stacking and ensemble learning, incorporating GBM, RF, GLM, DNN, and XGBoost models, along with built-in feature engineering. **FLAML** [192] emphasizes computational efficiency with a cost-based optimization strategy, supporting mainstream ML models like XGBoost and LightGBM, making it particularly suitable for resource-constrained environments. **Auto-Sklearn** [193] integrates 15 classifiers and 14 regressors with 14 preprocessing methods, utilizing meta-learning for warm-starting and Bayesian optimization (SMAC) for hyperparameter tuning.

In RL, **Stable Baselines3** [194] provides a reliable collection of mainstream RL algorithms with unified interfaces and extensive documentation, accompanied by RL Baselines3 Zoo for systematic benchmarking across diverse environments. **RLlib** [195] stands out as a scalable RL library supporting both model-free and model-based algorithms, offering distributed training capabilities and integration with multiple deep learning frameworks. **CleanRL** [196] emphasizes single-file implementations of popular RL algorithms with comprehensive benchmarking tools, facilitating reproducible research. For card game domains, **RLCard** [197] implements multiple RL algorithms with standardized environments and evaluation protocols.

These platforms not only serve as standalone tools in ML domain but also provide foundational frameworks that researchers and developers can build upon.

6. Perspective

The field of MetaBBO for EAs continues to evolve rapidly, presenting several promising research directions. Key areas that deserve further investigation and development are identified in this section, potentially leading to significant advancements in the MetaBBO for EA.

6.1. From the view of BBOPs

6.1.1. Problem characteristics identification

A core challenge within MetaBBO is conducting a systematic analysis of problem characteristics and effectively translating these into actionable features that guide algorithm design. While current approaches primarily focus on candidate solution characteristics, a more comprehensive framework is needed that considers:

(1) **Explicit Features:** Directly observable characteristics such as objective dimensionality, constraint types, and function properties.
(2) **Implicit Features:** Hidden characteristics including landscape such as multimodality, separability, and basin of attraction structures.
(3) **Dynamic Features:** Temporal changes in problem characteristics during the optimization process.

Related studies have attempted to address this through exploratory landscape analysis [167], but significant gaps remain in developing reliable, automated feature extraction methods that can inform algorithm selection and adaptation in real-time. There is a lack of a reliable tool to analyze the problem characteristics directly before MetaBBO and transfer them into an adaptive input of the following learning process. Future research can focus on developing more intelligent frameworks that can:

- Automatically identify and quantify relevant problem characteristics
- Establish clear relationships between problem characteristics and algorithm performance
- Adapt feature extraction methods to evolving problem landscapes

6.1.2. Auto-formulation of BBOPs

Another active area from the view of BBOPs falls into auto-formulation of optimization models. It plays a crucial role in purely data-driven optimization, with many new methods and techniques being developed all the time. It refers to the ability of an algorithm or system to automatically generate the mathematical representation of an optimization problem based on a high-level problem description. This process involves understanding the problem context, identifying the decision variables, objective function, and constraints, and formulating them into a structured optimization problem. The auto-formulation process can be broken down into several steps:

(1) **Problem Understanding:** The first step is to understand the problem context and the desired outcome. This involves interpreting the high-level description such as video and text provided by the user and determining the core elements of the optimization problem, such as the objective to be minimized or maximized, and any constraints that must be satisfied. It involves the ability to infer the underlying structure of the problem from the input–output data as well, without explicit knowledge of the mathematical model.
(2) **Decision Variable Identification:** Once the problem context is understood, the next step is to identify the decision variables that will be used to solve the problem. These are the variables that the optimizer will adjust to find the optimal solution.
(3) **Objective Function Formulation:** The objective function is a mathematical representation of what you want to optimize. However, surrogate model is usually leveraged to express the mapping relationship between decision variable and objective. The auto-formulation process should focus on which surrogate model should be used based on the problem description.
(4) **Constraints Definition:** Constraints are the limitations or conditions that the solution must adhere to. These could be inequalities or equalities that the decision variables must satisfy. The auto-formulation need to provide an automated process to define the constraints.
(5) **Model Encoding:** The identified decision variables, objective function, and constraints need to be encoded into a unified format that can be processed by BBOAs. This typically involves converting the high-level problem description into a structured decoding that can be solved using established optimization techniques.

Rindranirina et al. described an augmented intelligence system for simplifying and enhancing the modeling experience for operations research [198]. User can receive a suggested formulation of an optimization problem based on its description with this system.

Advancements in AI are enabling the development of more sophisticated algorithms that can automatically learn patterns and structure from data, allowing for even more complex auto-formulation processes. Especially LLM provides various conveniences and possibilities for automated modeling of optimization problems through its powerful text processing, learning, and generative modeling capabilities. With the continuous development of technology, the application of LLM in the field of automated modeling of optimization problems will become increasingly widespread.

6.1.3. More complex BBOPs

BBOPs tackled by MetaBBO in current studies currently exhibit a notable limitation. For example, most existing studies concentrate on problems with dimensions below 100, dense parameter spaces, simple bound constraints, and relatively straightforward objective landscapes.

This restricted focus creates a substantial gap between theoretical research and practical applications. Real-world optimization problems frequently present multiple challenging characteristics simultaneously: high dimensionality often exceeding 1000 parameters, inherent sparsity in the solution space, complex nonlinear constraints, and highly multi-modal objective landscapes.

To bridge this gap, MetaBBO needs to expand its scope in several critical directions. High-dimensional problems require scalable surrogate models and efficient sampling strategies that can maintain effectiveness despite the curse of dimensionality. Sparse optimization scenarios demand mechanisms for identifying relevant variables and leveraging structure in the parameter space. Constrained optimization problems necessitate sophisticated handling techniques, while multi-modal landscapes require robust strategies for exploring multiple promising regions simultaneously.

Future research should prioritize developing unified approaches that can simultaneously handle multiple challenging problem characteristics, rather than addressing each aspect in isolation. This integration would better reflect the nature of real-world optimization problems and make MetaBBO more applicable to practical challenges in various domains.

6.2. From the view of EAs

Many MetaBBO studies for EAs struggle with a lack of flexibility because they usually follow set patterns based on human knowledge, which restricts how they can adapt. [81]. On the meanwhile, the existing learnable objects are limited to the selection of operators and the adjustment of parameters with few new algorithms or algorithm components being created. In the learning process, feedback is mainly given based on the characteristics of the population, ignoring the characteristics of the problem itself. This exposes significant shortcomings when dealing with complex optimization problems. Hence, we can develop further studies in the following aspects.

6.2.1. Flexible design paradigm

A flexible design paradigm in MetaBBO for EA is essential to overcome the current limitations in existing research, where the optimization process typically adheres to rigid templates and predefined structures. In this context, two key aspects can be developed to enhance the adaptability and efficiency of EAs for complex optimization problems.

One the one hand, current EAs in MetaBBO often follow a fixed updating mechanism, such as the well-known $\mu, \mu + 1, and \mu + \lambda$, where the number of parents and offspring are predefined, as well as the order and execution time of each evolution strategies. While these templates have proven effective for many problems, they lack flexibility when utilizing MetaBBO for EAs. By considering more flexible templates as one of the optimization objects, EAs designed by MetaBBO can accommodate a wider range of problem characteristics and improve overall algorithm performance.

On the other hand, instead of relying on a single population that evolves over time, multiple populations provide more design paradigms, which explore different regions of the search space in parallel, enabling a more thorough exploration and exploitation of the problem domain. Each population could specialize in different subproblems or aspects of the solution. The interactions between these populations (e.g., through migration or information exchange) could be guided adaptively based on the problem characteristics, improving the overall performance and robustness of the optimization process. Furthermore, by integrating multi-stage strategy into the design paradigm, EAs can become more versatile and capable of tackling complex optimization problems that require diverse strategies across different stages of the search.

6.2.2. Learnable objects

Learnable objects are pivotal for the future of MetaBBO, aligning with its aspiration to transcend the limitations of current methodologies and to innovate in the sphere of automatic algorithm design and optimization problem-solving.

One of the most important learnable objects is learnable CHT due to the fundamental aspect of real-world BBOPs [199]. Current studies to constraint handling in MetaBBO typically rely on simple penalty methods or feasibility rules, treating constraints as fixed procedures rather than learnable components. This limits the algorithm's ability to adapt its constraint handling strategy based on problem characteristics and optimization progress. Moreover, the transfer of constraint handling knowledge across similar problems remains largely unexplored, despite its potential to significantly improve optimization efficiency.

A promising direction for future research lies in developing learnable constraint handling mechanisms within the MetaBBO framework. This could include learning adaptive penalty coefficients, intelligent feasibility restoration strategies, and constraint approximation models. The learning process could capture patterns in constraint violations, understand the relationship between constraints and objective functions, and develop strategies for balancing feasibility and optimality.

This enhanced focus on learnable constraint handling would significantly improve MetaBBO's practical applicability. By treating constraint handling as a learnable component rather than a fixed procedure, it could develop more efficient strategies for navigating constrained optimization landscapes.

Besides, the current learnable objects in MetaBBO for EAs concentrate on rule selection and solution generation. They are typically output from NN or RL agents including LLMs, which are inexplicable. Another kind of learnable object is more attractive, that is rule generation, combining symbolic regression with GP or NN for interpretable rule generation. The opacity of inexplicable rules hinders their interpretability, despite their potential for generating effective solutions, while the rule space of symbolic equation generators is inherently limited.

The design of advanced learnable objects for MetaBBO presents an enticing avenue for research, which could significantly bolster the optimization process. This involves novel rule representation such as graph or tree, leveraging machine learning techniques, encompassing supervised learning, unsupervised learning, and reinforcement learning, to discern patterns and relationships and be effective and explainable.

6.3. From the view of meta-level

6.3.1. Advanced learning mechanisms

From the view of meta-level methodologies in MetaBBO, significant advancements can be achieved by introducing multiple, sophisticated learning paradigms as listed in Table 4. These hybrid approaches not only enhance the algorithm's ability to generalize across different problem domains but also enable dynamic, context-aware optimization strategies.

For example, combining transfer learning and meta-learning. By combining these two paradigms, we can create a meta-level learning process based on a pre-configured EA, which refers to already optimized configurations learned from previous optimization tasks. These pretrained configurations can be reused as the starting point for new learning, potentially reducing the time to convergence and enhancing algorithm efficiency.

Otherwise, combine multi-task learning and meta-learning. In a multi-task meta-learning framework, an EA could be trained on a set of optimization tasks. While meta-learning helps the algorithm learn to adapt to new tasks quickly, multi-task learning helps the EA understand which features or strategies are shared across tasks. This shared knowledge can then be applied to optimize EAs more effectively across different domains. For example, an EA optimized for solving various types of vehicle routing problems can apply similar strategies when tackling a delivery scheduling problem or a resource allocation problem, even if these tasks have different specifics.

Another promising direction is accelerating the learning process through distributed computing and multi-GPU/TPU acceleration, enabling it to handle very large datasets and deep learning tasks. Given the computationally intensive nature of many learning algorithms (especially when involving large datasets or deep learning models), parallelization can dramatically reduce training times. The ability to offload computational tasks to multiple GPUs or TPUs enables the simultaneous exploration of different hyperparameter settings, strategies, and solutions, resulting in a more efficient learning process.

6.3.2. Novel methodologies

The design space of EAs is extensive, covering many optimization objects from the selection of basic operators, the adjustment of hyperparameters, to higher-level strategy optimization, etc. In particular, the design space of EAs exhibits an exponential growth characteristic. Each operator, parameter, and mechanism may have multiple implementations and combinations, which makes the design space very large. Moreover, the design of EAs in MetaBBO is not only to perform well on a specific problem, but also to consider the generalization ability of the algorithm. Further research is needed to address these challenges and develop novel techniques to design EAs.

A possible direction falls into directly utilizing gradient-based methodologies in the MetaBBO for EAs. Gradient optimization algorithms, such as Gradient Descent (GD) [200] and stochastic variants Stochastic Gradient Descent (SGD) [201], Adam [202], and RMSProp [203], occupy a central position in the search for solutions that minimize or maximize an objective function. The key advantages of gradient-based optimization include their computational efficiency, theoretical guarantees of convergence, and their ability to effectively handle large-scale optimization problems. These algorithms leverage the gradient of the objective function, which represents the direction of steepest ascent or descent at a given point in the parameter space. The gradient informs the algorithm about the most promising direction to move in the search space, providing crucial information to steer the optimization process towards optimal solutions.

Gradient optimization used at meta-level, termed as meta-gradient, has been utilized to optimize reinforcement learning's value functions in 2018 [204], i.e. hyperparameter optimization of RL. Mathematically, the policy parameters θ of ith RL agent whose learning rate is α_i and loss function is L_i, is updated at tth iteration of the inner loop using gradient descent:

$$\theta_{t+1} = \theta_t - \alpha_i \nabla_\theta L_i(\theta_t|\alpha_i) \tag{16}$$

While at meta-level, α_i and L_i are collectively called meta-parameters η, which is optimized as well using gradient descent:

$$\eta_{new} = \eta - \beta \nabla_\eta \mathcal{L}(\eta) \tag{17}$$

where β is the learning rate of the meta-optimizer.

Oh et al. introduced a meta-learning framework to discover a comprehensive update rule for reinforcement learning algorithms via meta-gradient optimization [205]. The discovered RL agent exhibits generalization capabilities, transitioning from simplified toy environments to complex Atari games. Xu et al. considered the parameters of reward function as learnable meta-parameters, such as discount factor and bootstrapping parameter [206]. By learning the optimal return function parameters via meta-gradient, the RL agent can more accurately estimate the state value function and strategy, thereby improving decision quality. They indicated as well that meta-learning the objectives is more effective than fully updating the rules. Bonnet et al. presented a multi-step meta-gradient reinforcement learning algorithm [207]. Flennerhag et al. proposed an algorithm to solve the difficult meta optimization problem in meta-learning via bootstrapped meta-gradients [208], and demonstrated that it can learn to explore in an ϵ-greedy Q-learning agent. In NAS, Miao et al. employed Differentiable Architecture Search (DARTS) [209] to optimize RL architecture via meta-gradient [210]. By integrating a DARTS hypernetwork into the image encoder, they demonstrated the hypernetwork's ability to learn superior cell structures, leading to competitive reinforcement learning strategies.

Evidently, gradient-based optimization has been predominantly employed in the domain of AutoML. In the context of MetaBBO for EAs, gradient-based optimization offers promising potential to address challenges related to hyperparameter tuning and adaptive strategy design. Although evolutionary algorithms do not rely on gradient information, they can estimate the gradients of hyperparameters and combine them with approximate gradients (such as numerical gradients, simulated gradients, etc.) or reinforcement learning to compensate for the challenges brought by the non-differentiability of the meta-objective function and find parameter combinations that lead to faster and more stable convergence of evolutionary algorithms.

Moreover, it is possible to design hybrid optimization techniques, thereby improving convergence speed without sacrificing the global search capability. Hybrid optimization techniques could lead to more efficient, adaptive, and robust EAs, particularly in complex or high-dimensional optimization tasks, where both global exploration and local refinement are critical.

7. Conclusion

This comprehensive review has delved into the MetaBBO for EAs, showcasing its significance in tackling complex optimization challenges. We have traversed through the theoretical underpinnings, methodological advancements, and empirical evaluations that constitute the core of MetaBBO. By establishing a mathematical model, the scope of MetaBBO and its distinction from traditional BBO have been demarcated. The exploration of the parameterization space for EAs within MetaBBO has unveiled potential configurations that enhance the generalization capabilities of EAs.

The taxonomy of MetaBBO methodologies, presented from a meta-level perspective, reflects the state-of-the-art techniques and their applicability to diverse optimization landscapes. The overview of benchmarks, evaluation metrics, and platforms has streamlined the research and experimentation process in MetaBBO for EAs, providing a solid foundation for future investigations.

As we look towards the horizon, the role of MetaBBO in automatic algorithm design and optimization problem-solving becomes increasingly pivotal. The prospective research directions, including the auto-formulation of optimization models, intelligent exploration of parameterized spaces, and the application of MetaBBO in constrained multi-objective optimization, underscore the field's potential for growth and innovation.

In conclusion, MetaBBO stands as a testament to the power of meta-learning in advancing the frontiers of optimization. Through continuous technological innovation and theoretical research, it is poised to revolutionize the way we approach and solve complex real-world problems, thereby cementing its place as a cornerstone in the edifice of automatic algorithm design and optimization.

CRediT authorship contribution statement

Xu Yang: Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Rui Wang:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Kaiwen Li:** Writing – review & editing, Methodology. **Hisao Ishibuchi:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors gratefully acknowledge the financial support provided by the Major Open Project of Xiangjiang Laboratory (No. 22XJ02003), the National Natural Science Foundation of China (No. 62122093), the Science and Technology Project for Young and Middle-aged Talents of Hunan (2023TJ-Z03), the Science and Technology Innovation Program of Hunan Province (No. 2023RC1002), the Scientific Project of NUDT (ZK21-07, 23-ZZCX-JDZ-28), the National Natural science Foundation of China (No. 72071205), and the National Natural Science Funds of China(No. 62303476).

Data availability

No data was used for the research described in the article.

References

[1] Y. Wang, H. Zhang, G. Zhang, cPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks, Swarm Evol. Comput. 49 (4) (2019) 114–123.
[2] J. Zhou, Y. Qiu, S. Zhu, D.J. Armaghani, C. Li, H. Nguyen, S. Yagiz, Optimization of support vector machine through the use of metaheuristic algorithms in forecasting TBM advance rate, Eng. Appl. Artif. Intell. 97 (2021) 104015.
[3] Z. Mustaffa, M.H. Sulaiman, M.N.M. Kahar, LS-SVM hyper-parameters optimization based on GWO algorithm for time series forecasting, in: 2015 4th International Conference on Software Engineering and Computer Systems, ICSECS, IEEE, 2015, pp. 183–188.
[4] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, X. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, ACM Comput. Surv. 54 (4) (2021) 1–34.
[5] G. Schneider, J. Schuchhardt, P. Wrede, Artificial neural networks and simulated molecular evolution are potential tools for sequence-oriented protein design, Bioinformatics 10 (6) (1994) 635–645.
[6] S. Pierret, R. Van den Braembussche, Turbomachinery blade design using a Navier–Stokes solver and artificial neural network, J. Turbomach. 121 (3) (1999) 326–332.
[7] E. Yurtsever, J. Lambert, A. Carballo, K. Takeda, A survey of autonomous driving: Common practices and emerging technologies, IEEE Access 8 (2020) 58443–58469.
[8] V. Nguyen, Bayesian optimization for accelerating hyper-parameter tuning, in: 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering, AIKE, 2019.
[9] Y. Xia, C. Liu, Y. Li, N. Liu, A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring, Expert Syst. Appl. 78 (2017) 225–241.
[10] R. Lange, T. Schaul, Y. Chen, T. Zahavy, V. Dalibard, C. Lu, S. Singh, S. Flennerhag, Discovering evolution strategies via meta-black-box optimization, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, 2023, pp. 29–30.
[11] E. Kant, Understanding and automating algorithm design, IEEE Trans. Softw. Eng. (11) (1985) 1361–1374.
[12] P. Narayan, P. Wu, D. Campbell, R. Walker, An intelligent control architecture for unmanned aerial systems (UAS) in the national airspace system (NAS), in: Proceedings of AIAC12: 2nd Australasian Unmanned Air Vehicles Conference, Waldron Smith Management, 2007, pp. 1–12.
[13] L. Roveda, M. Forgione, D. Piga, Robot control parameters auto-tuning in trajectory tracking applications, Control Eng. Pract. 101 (2020) 104488.
[14] J. Waring, C. Lindvall, R. Umeton, Automated machine learning: Review of the state-of-the-art and opportunities for healthcare, Artif. Intell. Med. 104 (2020) 101822.
[15] A. Mustafa, M. Rahimi Azghadi, Automated machine learning for healthcare and clinical notes analysis, Comput (2021).
[16] J. Adamczyk, F. Malawski, Comparison of manual and automated feature engineering for daily activity classification in mental disorder diagnosis, Comput. Inform. 40 (4) (2021) 850–879.
[17] J. An, I.S. Kim, K.-J. Kim, J.H. Park, H. Kang, H.J. Kim, Y.S. Kim, J.H. Ahn, Efficacy of automated machine learning models and feature engineering for diagnosis of equivocal appendicitis using clinical and computed tomography findings, Sci. Rep. 14 (1) (2024) 22658.
[18] J.M. Sanz, Automated design of controlled-diffusion blades, J. Turbomach. 110 (4) (1988) 540–544, http://dx.doi.org/10.1115/1.3262228, arXiv:https://asmedigitalcollection.asme.org/turbomachinery/article-pdf/110/4/540/5839607/540_1.pdf.
[19] U. Idahosa, V.V. Golubev, V.O. Balabanov, An automated optimal design of a fan blade using an integrated CFD/MDO computer environment, Eng. Appl. Comput. Fluid Mech. 2 (2) (2008) 141–154.
[20] L. Cheng, Q. Tang, L. Zhang, Z. Zhang, Multi-objective Q-learning-based hyper-heuristic with Bi-criteria selection for energy-aware mixed shop scheduling, Swarm Evol. Comput. 69 (2022) 100985.
[21] Z.-Q. Zhang, F.-C. Wu, B. Qian, R. Hu, L. Wang, H.-P. Jin, A Q-learning-based hyper-heuristic evolutionary algorithm for the distributed flexible job-shop scheduling problem with crane transportation, Expert Syst. Appl. 234 (2023) 121050.
[22] Z.-Q. Zhang, B. Qian, R. Hu, J.-B. Yang, Q-learning-based hyper-heuristic evolutionary algorithm for the distributed assembly blocking flowshop scheduling problem, Appl. Soft Comput. 146 (2023) 110695.
[23] A. Nahar, D. Das, MetaLearn: Optimizing routing heuristics with a hybrid meta-learning approach in vehicular ad-hoc networks, Ad Hoc Netw. 138 (2023) 102996.
[24] J.-J. Ji, Y.-N. Guo, X.-Z. Gao, D.-W. Gong, Y.-P. Wang, Q-learning-based hyper-heuristic evolutionary algorithm for dynamic task allocation of crowdsensing, IEEE Trans. Cybern. 53 (4) (2021) 2211–2224.
[25] K. Tang, X. Yao, Learn to optimize-a brief overview, Natl. Sci. Rev. (2024) nwae132.
[26] H.J. Escalante, Automated machine learning – a brief review at the end of the early years, 2020, pp. 11–28, arXiv:2008.08516v2.
[27] T. Stützle, M. López-Ibáñez, Automated design of metaheuristic algorithms, in: Handbook Metaheuristics, Springer, 2019, pp. 541–579.
[28] T. Nagarajah, G. Poravi, A review on automated machine learning (AutoML) systems, in: 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), IEEE, 2019, pp. 1–6.
[29] X. He, K. Zhao, X. Chu, AutoML: A survey of the state-of-the-art, Knowl.-Based Syst. 212 (2021) 106622.
[30] M. Baratchi, C. Wang, S. Limmer, J.N. van Rijn, H. Hoos, T. Bäck, M. Olhofer, Automated machine learning: past, present and future, Artif. Intell. Rev. 57 (5) (2024) 1–88.
[31] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, et al., Automated reinforcement learning (autorl): A survey and open problems, J. Artificial Intelligence Res. 74 (2022) 517–568.
[32] R.R. Afshar, Y. Zhang, J. Vanschoren, U. Kaymak, Automated reinforcement learning: An overview, 2022, arXiv Preprint arXiv:2201.05000.
[33] P. Kerschke, H.H. Hoos, F. Neumann, H. Trautmann, Automated algorithm selection: Survey and perspectives, Evol. Comput. 27 (1) (2019) 3–45.
[34] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, K. Tierney, A survey of methods for automated algorithm configuration, J. Artificial Intelligence Res. 75 (2022) 425–487.
[35] A. Gupta, A. Sivakumar, Deep learning for automated algorithm design: A review, Neural Comput. Appl. (2021).
[36] R.D. Al-Dabbagh, F. Neri, N. Idris, M.S. Baba, Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy, Swarm Evol. Comput. 43 (2018) 284–311.
[37] Y. Yang, Y. Gao, Z. Ding, J. Wu, S. Zhang, F. Han, X. Qiu, S. Gao, Y.-G. Wang, Advancements in Q-learning meta-heuristic optimization algorithms: A survey, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. (2024) e1548.
[38] C. Huang, Y. Li, X. Yao, A survey of automatic parameter tuning methods for metaheuristics, IEEE Trans. Evol. Comput. 24 (2) (2019) 201–216.
[39] Y. Jaafra, J.L. Laurent, A. Deruyver, M.S. Naceur, Reinforcement learning for neural architecture search: A review, Image Vis. Comput. 89 (2019) 57–66.
[40] F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E.C. Garrido-Merchan, J. Branke, B. Bischl, Multi-objective hyperparameter optimization – an overview, 2022, arXiv (Cornell University).
[41] M. Chakraborty, W. Pal, S. Bandyopadhyay, U. Maulik, Survey on multi-objective-based parameter optimization for deep learning, CoRR 24 (3) (2023).
[42] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, K. Tierney, A survey of methods for automated algorithm configuration, 2202, 2022, http://dx.doi.org/10.48550/ARXIV.
[43] F. Hutter, Y. Hamadi, H.H. Hoos, K. Leyton-Brown, Performance prediction and automated tuning of randomized and parametric algorithms, in: Principles and Practice of Constraint Programming-CP 2006: 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006. Proceedings 12, Springer, 2006, pp. 213–228.
[44] W. Zhang, M. Gen, J. Jo, Hybrid sampling strategy-based multiobjective evolutionary algorithm for process planning and scheduling problem, J. Intell. Manuf. 25 (2014) 881–897.
[45] M.-A. Carbonneau, V. Cheplygina, E. Granger, G. Gagnon, Multiple instance learning: A survey of problem characteristics and applications, Pattern Recognit. 77 (2018) 329–353.
[46] S. Liu, K. Tang, X. Yao, Generative adversarial construction of parallel portfolios, IEEE Trans. Cybern. 52 (2) (2020) 784–795.

[47] K. Tang, S. Liu, P. Yang, X. Yao, Few-shots parallel algorithm portfolio construction via co-evolution, IEEE Trans. Evol. Comput. 25 (3) (2021) 595–607.

[48] X. Ying, An overview of overfitting and its solutions, J. Phys.: Conf. Series 1168 (2019) 022022.

[49] C.F.G.D. Santos, J.P. Papa, Avoiding overfitting: A survey on regularization methods for convolutional neural networks, ACM Comput. Surv. 54 (10s) (2022) 1–25.

[50] R. Moradi, R. Berangi, B. Minaei, A survey of regularization strategies for deep models, Artif. Intell. Rev. 53 (6) (2020) 3947–3986.

[51] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, M. Pontil, Bilevel programming for hyperparameter optimization and meta-learning, in: International Conference on Machine Learning, PMLR, 2018, pp. 1568–1577.

[52] S. Bates, T. Hastie, R. Tibshirani, Cross-validation: what does it estimate and how well does it do it? J. Amer. Statist. Assoc. 119 (546) (2024) 1434–1445.

[53] C. Thornton, F. Hutter, H.H. Hoos, K. Leyton-Brown, Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013, pp. 847–855.

[54] S. Liu, K. Tang, Y. Lei, X. Yao, On performance estimation in automatic algorithm configuration, in: Proceedings of the AAAI Conference on Artificial Intelligence, 34, (03) 2020, pp. 2384–2391.

[55] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Oper. Res. Perspect. 3 (2016) 43–58.

[56] R. Vilalta, Y. Drissi, A perspective view and survey of meta-learning, Artif. Intell. Rev. 18 (2002) 77–95.

[57] J.R. Rice, The Algorithm Selection Problem**This work was partially supported by the National Science Foundation through Grant GP-32940X. This chapter was presented as the George E. Forsythe Memorial Lecture at the Computer Science Conference, February 19, 1975, Washington, D. C., in: M. Rubinoff, M.C. Yovits (Eds.), in: Advances in Computers, vol. 15, Elsevier, 1976, pp. 65–118, http://dx.doi.org/10.1016/S0065-2458(08)60520-3, URL https://www.sciencedirect.com/science/article/pii/S0065245808605203.

[58] Y. Tian, S. Peng, X. Zhang, T. Rodemann, K.C. Tan, Y. Jin, A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks, IEEE Trans. Artif. Intell. 1 (1) (2020) 5–18.

[59] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: an automatic algorithm configuration framework, J. Artificial Intelligence Res. 36 (2009) 267–306.

[60] A.R. KhudaBukhsh, L. Xu, H.H. Hoos, K. Leyton-Brown, SATenstein: Automatically building local search SAT solvers from components, Artificial Intelligence 232 (2016) 20–42.

[61] C. Daniel, J. Taylor, S. Nowozin, Learning step size controllers for robust neural network training, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2016.

[62] Z. Xu, A.M. Dai, J. Kemp, L. Metz, Learning an adaptive learning rate schedule, 2019, arXiv Preprint arXiv:1909.09712.

[63] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, B. Scholkopf, Support vector machines, IEEE Intell. Syst. Appl. 13 (4) (1998) 18–28.

[64] L.E. Peterson, K-nearest neighbor, Scholarpedia 4 (2) (2009) 1883.

[65] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern Recognit. 77 (2018) 354–377.

[66] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, S. Valaee, Recent advances in recurrent neural networks, 2017, arXiv Preprint arXiv:1801.01078.

[67] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning: Methods, Systems, Challenges, Springer Nature, Germany, 2019, http://dx.doi.org/10.1007/s10462-024-10726-1.

[68] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, J. Mach. Learn. Res. 20 (55) (2019) 1–21.

[69] M. Feurer, F. Hutter, Hyperparameter optimization, Autom. Mach. Learn.: Methods Syst., Challenges (2019) 3–33.

[70] M. Feurer, F. Hutter, Meta-learning, Autom. Mach. Learn.: Methods Syst., Challenges (2019) 39–68.

[71] Q. Zhao, Q. Duan, B. Yan, S. Cheng, Y. Shi, A survey on automated design of metaheuristic algorithms, 2023, CoRR arXiv:2303.06532.

[72] R. Qu, G. Kendall, N. Pillay, The general combinatorial optimization problem: Towards automated algorithm design, IEEE Comput. Intell. Mag. 15 (2) (2020) 14–23.

[73] Q. Zhang, H. Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731.

[74] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[75] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, TIK Report, 103, ETH Zurich, 2001.

[76] T. Murata, H. Ishibuchi, et al., MOGA: multi-objective genetic algorithms, in: IEEE International Conference on Evolutionary Computation, Vol. 1, IEEE Piscataway, 1995, pp. 289–294.

[77] J.G. Falcón-Cardona, C.A.C. Coello, Indicator-based multi-objective evolutionary algorithms: A comprehensive survey, ACM Comput. Surv. 53 (2) (2020) 1–35.

[78] W. Li, T. Zhang, R. Wang, H. Ishibuchi, Weighted indicator-based evolutionary algorithm for multimodal multiobjective optimization, IEEE Trans. Evol. Comput. 25 (6) (2021) 1064–1078.

[79] A. Menchaca-Mendez, C.A.C. Coello, GDE-MOEA: A new MOEA based on the generational distance indicator and ϵ-dominance, in: 2015 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2015, pp. 947–955.

[80] C.M. Fonseca, P.J. Fleming, Multiobjective optimization, Handbook of evolutionary computation 1 (1997) C4.

[81] L.C. Bezerra, L. Manuel, et al., Automatically designing state-of-the-art multi- and many-objective evolutionary algorithms, Evol. Comput. 28 (2) (2020) 195–226.

[82] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, PISA—a platform and programming language independent interface for search algorithms, in: Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings 2, Springer, 2003, pp. 494–508.

[83] F. Biscani, D. Izzo, C.H. Yam, A global optimisation toolbox for massively parallel engineering optimisation, 2010, arXiv Preprint arXiv:1004.3824.

[84] L.C. Bezerra, M. López-Ibánez, T. Stützle, Automatic component-wise design of multiobjective evolutionary algorithms, IEEE Trans. Evol. Comput. 20 (3) (2015) 403–417.

[85] L.C. Bezerra, M. López-Ibáñez, T. Stützle, A large-scale experimental evaluation of high-performing multi-and many-objective evolutionary algorithms, Evol. Comput. 26 (4) (2018) 621–656.

[86] E. Zitzler, L. Thiele, J. Bader, On set-based multiobjective optimization, IEEE Trans. Evol. Comput. 14 (1) (2009) 58–79.

[87] E.J. Hughes, Multiple single objective Pareto sampling, in: The 2003 Congress on Evolutionary Computation, 2003. CEC'03, Vol. 4, IEEE, 2003, pp. 2678–2684.

[88] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2013) 577–601.

[89] T. Robič, B. Filipič, Differential evolution for multiobjective optimization, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2005, pp. 520–533.

[90] S. Kukkonen, J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 1, IEEE, 2005, pp. 443–450.

[91] I. Das, J.E. Dennis, A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems, Struct. Optim. 14 (1997) 63–69.

[92] Y.P. Aneja, K.P. Nair, Bicriteria transportation problem, Manage. Sci. 25 (1) (1979) 73–78.

[93] C.A. Coello Coello, Constraint-handling techniques used with evolutionary algorithms, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, 2016, pp. 563–587.

[94] B.W. Kort, D.P. Bertsekas, A new penalty function method for constrained minimization, in: Proceedings of the 1972 Ieee Conference on Decision and Control and 11th Symposium on Adaptive Processes, IEEE, 1972, pp. 162–166.

[95] Ö. Yeniay, Penalty function methods for constrained optimization with genetic algorithms, Math. Comput. Appl. 10 (1) (2005) 45–56.

[96] Q. He, L. Wang, A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization, Appl. Math. Comput. 186 (2) (2007) 1407–1422.

[97] Y. Wang, B.-C. Wang, H.-X. Li, G.G. Yen, Incorporating objective function information into the feasibility rule for constrained evolutionary optimization, IEEE Trans. Cybern. 46 (12) (2015) 2938–2952.

[98] W. Ning, B. Guo, Y. Yan, X. Wu, J. Wu, D. Zhao, Constrained multi-objective optimization using constrained non-dominated sorting combined with an improved hybrid multi-objective evolutionary algorithm, Eng. Optim. 49 (10) (2017) 1645–1664.

[99] Y. Wang, Z. Cai, G. Guo, Y. Zhou, Multiobjective optimization and hybrid evolutionary algorithm to solve constrained optimization problems, IEEE Trans. Syst. Man Cybern. B 37 (3) (2007) 560–575.

[100] T.P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Trans. Evol. Comput. 4 (3) (2000) 284–294.

[101] S. TakahamaT, Constrained optimization by the ϵ constrained differential evolution with gradient-based mutation and feasible elites, in: Proceeding of IEEE Congress on Evolutionary Computation, IEEE, 2006, p. r8.

[102] J. Schmidhuber, Evolutionary principles in self-referential learning, On Learn how to learn: meta-meta-... hook. Diploma thesis, 1, (2) Institut f. Informatik, Tech. Univ. Munich, 1987, p. 48.

[103] A.S. Younger, S. Hochreiter, P.R. Conwell, Meta-learning with backpropagation, in: IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222), Vol. 3, IEEE, 2001.

[104] S. Hochreiter, A.S. Younger, P.R. Conwell, Learning to learn using gradient descent, in: Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11, Springer, 2001, pp. 87–94.

[105] M. Andrychowicz, M. Denil, S. Gomez, M.W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. De Freitas, Learning to learn by gradient descent by gradient descent, Adv. Neural Inf. Process Syst. 29 (2016).

[106] Y. Chen, M.W. Hoffman, S.G. Colmenarejo, M. Denil, T.P. Lillicrap, M. Botvinick, N. Freitas, Learning to learn without gradient descent by gradient descent, in: International Conference on Machine Learning, PMLR, 2017, pp. 748–756.

[107] Y. Chen, X. Song, C. Lee, Z. Wang, R. Zhang, D. Dohan, K. Kawakami, G. Kochanski, A. Doucet, M. Ranzato, et al., Towards learning universal hyperparameter optimizers with transformers, Adv. Neural Inf. Process. Syst. 35 (2022) 32053–32068.

[108] V. TV, P. Malhotra, J. Narwariya, L. Vig, G. Shroff, Meta-learning for black-box optimization, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2019, pp. 366–381.

[109] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F.J. R Ruiz, J. Schrittwieser, G. Swirszcz, et al., Discovering faster matrix multiplication algorithms with reinforcement learning, Nature 610 (7930) (2022) 47–53.

[110] Z. Li, L. Shi, C. Yue, Z. Shang, B. Qu, Differential evolution based on reinforcement learning with fitness ranking for solving multimodal multiobjective problems, Swarm Evol. Comput. 49 (2019) 234–244.

[111] J.E. Pettinger, R.M. Everson, Controlling genetic algorithms with reinforcement learning, in: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, 2002, pp. 692–692.

[112] Á. Fialho, M. Schoenauer, M. Sebag, Toward comparison-based adaptive operator selection, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, 2010, pp. 767–774.

[113] M. Sharma, M. López-Ibáñez, D. Kazakov, Performance assessment of recursive probability matching for adaptive operator selection in differential evolution, in: Parallel Problem Solving from Nature–PPSN XV: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part II 15, Springer, 2018, pp. 321–333.

[114] W. Gong, Á. Fialho, Z. Cai, Adaptive strategy selection in differential evolution, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, 2010, pp. 409–416.

[115] Y. Sakurai, K. Takada, T. Kawabe, S. Tsuruta, A method to control parameters of evolutionary algorithms by using reinforcement learning, in: 2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems, IEEE, 2010, pp. 74–79.

[116] M. Sharma, A. Komninos, M. López-Ibáñez, D. Kazakov, Deep reinforcement learning based parameter control in differential evolution, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 709–717.

[117] Z. Tan, K. Li, Differential evolution with mixed mutation strategy based on deep reinforcement learning, Appl. Soft Comput. 111 (2021) 107678.

[118] J. Sun, X. Liu, T. Bäck, Z. Xu, Learning adaptive differential evolution algorithm from optimization experiences by policy gradient, IEEE Trans. Evol. Comput. 25 (4) (2021) 666–680, http://dx.doi.org/10.1109/TEVC.2021.3060811.

[119] R.S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, NIPS '99, MIT Press, Cambridge, MA, USA, 1999, pp. 1057–1063.

[120] G. Shala, A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, F. Hutter, Learning step-size adaptation in CMA-ES, in: Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, the Netherlands, September 5-9, 2020, Proceedings, Part I 16, Springer, 2020, pp. 691–706.

[121] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.

[122] S. Levine, P. Abbeel, Learning neural network policies with guided policy search under unknown dynamics, Adv. Neural Inf. Process Syst. 27 (2014).

[123] J. Schuchardt, V. Golkov, D. Cremers, Learning to evolve, 2019, arXiv Preprint arXiv:1905.03389.

[124] W. Yi, R. Qu, L. Jiao, B. Niu, Automated design of metaheuristics using reinforcement learning within a novel general search framework, IEEE Trans. Evol. Comput. (2022).

[125] H. Guo, Y. Ma, Z. Ma, J. Chen, X. Zhang, Z. Cao, J. Zhang, Y.-J. Gong, Deep reinforcement learning for dynamic algorithm selection: A proof-of-principle study on differential evolution, IEEE Trans. Syst., Man, Cybern.: Syst. (2024).

[126] Z. Ma, J. Chen, H. Guo, Y. Ma, Y.-J. Gong, Auto-configuring exploration-exploitation tradeoff in evolutionary computation via deep reinforcement learning, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2024, pp. 1497–1505.

[127] J. Chen, Z. Ma, H. Guo, Y. Ma, J. Zhang, Y. jiao Gong, Symbol: Generating flexible black-box optimizers through symbolic equation learning, 2024, http://dx.doi.org/10.48550/arXiv.2402.02355, arXiv arXiv:2402.02355, URL https://www.semanticscholar.org/paper/6fa259299737da470c24d0ed44c3093a06407edd.

[128] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-International Conference on Neural Networks, Vol. 4, ieee, 1995, pp. 1942–1948.

[129] D. Wang, D. Tan, L. Liu, Particle swarm optimization algorithm: an overview, Soft Comput. 22 (2018) 387–408.

[130] T.M. Shami, A.A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M.A. Summakieh, S. Mirjalili, Particle swarm optimization: A comprehensive survey, IEEE Access 10 (2022) 10031–10061, http://dx.doi.org/10.1109/ACCESS.2022.3142859.

[131] H. Samma, C.P. Lim, J.M. Saleh, A new reinforcement learning-based memetic particle swarm optimizer, Appl. Soft Comput. 43 (2016) 276–297.

[132] Y. Xu, D. Pi, A reinforcement learning-based communication topology in particle swarm optimization, Neural Comput. Appl. 32 (2020) 10007–10032.

[133] L. Lu, H. Zheng, J. Jie, M. Zhang, R. Dai, Reinforcement learning-based particle swarm optimization for sewage treatment control, Complex Intell. Syst. 7 (5) (2021) 2199–2210.

[134] F. Wang, X. Wang, S. Sun, A reinforcement learning level-based particle swarm optimization algorithm for large-scale optimization, Inform. Sci. 602 (2022) 298–312.

[135] D. Wu, G.G. Wang, Employing reinforcement learning to enhance particle swarm optimization methods, Eng. Optim. 54 (2) (2022) 329–348.

[136] W. Li, P. Liang, B. Sun, Y. Sun, Y. Huang, Reinforcement learning-based particle swarm optimization with neighborhood differential mutation strategy, Swarm Evol. Comput. 78 (2023) 101274.

[137] L. Song, C. Gao, K. Xue, C. Wu, D. Li, J. Hao, Z. Zhang, C. Qian, Reinforced in-context black-box optimization, 2024, arXiv Preprint arXiv:2402.17423.

[138] M.R. Zhang, N. Desai, J. Bae, J. Lorraine, J. Ba, Using large language models for hyperparameter optimization, in: NeurIPS 2023 Foundation Models for Decision Making Workshop, 2023.

[139] C. Yang, X. Wang, Y. Lu, H. Liu, Q.V. Le, D. Zhou, X. Chen, Large language models as optimizers, 2023, arXiv Preprint arXiv:2309.03409.

[140] S. Liu, C. Chen, X. Qu, K. Tang, Y.-S. Ong, Large language models as evolutionary optimizers, in: 2024 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2024, pp. 1–8.

[141] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated local search, in: Handbook of Metaheuristics, Springer, Germany, 2003, pp. 320–353.

[142] A. Blot, H.H. Hoos, L. Jourdan, M.-É. Kessaci-Marmion, H. Trautmann, MO-paramils: A multi-objective automatic algorithm configuration framework, in: Learning and Intelligent Optimization: 10th International Conference, LION 10, Ischia, Italy, May 29–June 1, 2016, Revised Selected Papers 10, Springer, 2016, pp. 32–47.

[143] P. Lin, J. Zhang, M.A. Contreras, Automatically configuring ACO using multilevel ParamILS to solve transportation planning problems with underlying weighted networks, Swarm Evol. Comput. 20 (2015) 48–57.

[144] A. Turky, N.R. Sabar, S. Dunstall, A. Song, Hyper-heuristic local search for combinatorial optimisation problems, Knowl.-Based Syst. 205 (2020) 106264.

[145] P. Vicol, L. Metz, J. Sohl-Dickstein, Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies, in: International Conference on Machine Learning, PMLR, 2021, pp. 10553–10563.

[146] C. Lu, J. Kuba, A. Letcher, L. Metz, C. Schroeder de Witt, J. Foerster, Discovered policy optimisation, Adv. Neural Inf. Process. Syst. 35 (2022) 16455–16468.

[147] J.J. Grefenstette, Optimization of control parameters for genetic algorithms, IEEE Trans. Syst., Man, Cybern. 16 (1) (1986) 122–128.

[148] V. Nannen, A.E. Eiben, Efficient relevance estimation and value calibration of evolutionary algorithm parameters, in: 2007 IEEE Congress on Evolutionary Computation, IEEE, 2007, pp. 103–110.

[149] C. Ansótegui, M. Sellmann, K. Tierney, A gender-based genetic algorithm for the automatic configuration of algorithms, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2009, pp. 142–157.

[150] H.S. Gomes, B. Léger, C. Gagné, Meta learning black-box population-based optimizers, 2021, arXiv Preprint arXiv:2103.03526.

[151] F.P. Such, V. Madhavan, E. Conti, J. Lehman, K.O. Stanley, J. Clune, Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, 2017, arXiv Preprint arXiv:1712.06567.

[152] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: An overview, Exp. Methods Anal. Optim. Algorithms (2010) 311–336.

[153] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the F-race algorithm: Sampling design and iterative refinement, in: Hybrid Metaheuristics: 4th International Workshop, HM 2007, Dortmund, Germany, October 8-9, 2007. Proceedings 4, Springer, 2007, pp. 108–122.

[154] M. López-Ibáñez, T. Stützle, Automatic configuration of multi-objective ACO algorithms, in: International Conference on Swarm Intelligence, Springer, 2010, pp. 95–106.

[155] J. Grudzien, C.A.S. De Witt, J. Foerster, Mirror learning: A unifying framework of policy optimisation, in: International Conference on Machine Learning, PMLR, 2022, pp. 7825–7844.

[156] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process Syst. 30 (2017).

[157] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, Y.W. Teh, Set transformer: A framework for attention-based permutation-invariant neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 3744–3753.

[158] R. Lange, T. Schaul, Y. Chen, C. Lu, T. Zahavy, V. Dalibard, S. Flennerhag, Discovering attention-based genetic algorithms via meta-black-box optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2023, pp. 929–937.
[159] J.R. Koza, Genetic Programming: a Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, Vol. 34, Stanford University, Department of Computer Science Stanford, CA, 1990.
[160] J.R. Koza, Genetic programming as a means for programming computers by natural selection, Stat. Comput. 4 (1994) 87–112.
[161] T.T. Le, W. Fu, J.H. Moore, Scaling tree-based automated machine learning to biomedical big data with a feature set selector, Bioinformatics 36 (1) (2020) 250–256.
[162] R.S. Olson, R.J. Urbanowicz, P.C. Andrews, N.A. Lavender, L.C. Kidd, J.H. Moore, Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I, Springer International Publishing, Germany, 2016, pp. 123–137, http://dx.doi.org/10.1007/978-3-319-31204-0_9.
[163] R.S. Olson, N. Bartley, R.J. Urbanowicz, J.H. Moore, Evaluation of a tree-based pipeline optimization tool for automating data science, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, ACM, New York, NY, USA, 2016, pp. 485–492, http://dx.doi.org/10.1145/2908812.2908918, URL http://doi.acm.org/10.1145/2908812.2908918.
[164] Y. Sergio, G. Colmenarejo, Learning to learn for global optimization of black box functions, Stat 1050 (2016) 18.
[165] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5, Springer, 2011, pp. 507–523.
[166] N. Hansen, S. Finck, R. Ros, A. Auger, Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions (Ph.D. thesis), INRIA, 2009.
[167] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, G. Rudolph, Exploratory landscape analysis, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, 2011, pp. 829–836.
[168] N. Hansen, S. Finck, R. Ros, A. Auger, Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions (Ph.D. thesis), INRIA, 2009.
[169] H. Hwang, T. Vreven, J. Janin, Z. Weng, Protein–protein docking benchmark version 4.0, Proteins: Struct., Funct., Bioinform. 78 (15) (2010) 3111–3114.
[170] S.D. Bay, D. Kibler, M.J. Pazzani, P. Smyth, The UCI kdd archive of large data sets for data mining research and experimentation, ACM SIGKDD Explor. Newslett. 2 (2) (2000) 81–85.
[171] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[172] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017, arXiv Preprint arXiv:1708.07747.
[173] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.
[174] N. Hansen, A. Auger, S. Finck, R. Ros, Real-parameter black-box optimization benchmarking 2009: Experimental setup, Hal Inria (2009).
[175] N. Hansen, A. Auger, D. Brockhoff, T. Tušar, Anytime performance assessment in blackbox optimization benchmarking, IEEE Trans. Evol. Comput. 26 (6) (2022) 1293–1305.
[176] H. Nikolaus, A. Anne, R. Ros, O. Mersmann, T. Tušar, D. Brockhoff, COCO: a platform for comparing continuous optimizers in a black-box setting, Optim. Methods Softw. 36 (1) (2021) 114–144, http://dx.doi.org/10.1080/10556788.2020.1808977.
[177] A. Blot, A. Pernet, L. Jourdan, M.-É. Kessaci-Marmion, H.H. Hoos, Automatically configuring multi-objective local search using multi-objective optimisation, in: Evolutionary Multi-Criterion Optimization: 9th International Conference, EMO 2017, Münster, Germany, March 19-22, 2017, Proceedings 9, Springer, 2017, pp. 61–76.
[178] S. Tari, N. Szczepanski, L. Mousin, J. Jacques, M.-E. Kessaci, L. Jourdan, Multi-objective automatic algorithm configuration for the classification problem of imbalanced data, in: 2020 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2020, pp. 1–8.
[179] H. Wang, Y. Jin, X. Yao, Diversity assessment in many-objective optimization, IEEE Trans. Cybern. 47 (6) (2016) 1510–1522.
[180] C.A.C. Coello, N.C. Cortés, Solving multiobjective optimization problems using an artificial immune system, Genetic Program. Evol. Mach. 6 (2005) 163–190.
[181] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.
[182] F. Ye, C. Doerr, H. Wang, T. Bäck, Automated configuration of genetic algorithms by tuning for anytime performance, IEEE Trans. Evol. Comput. 26 (6) (2022) 1526–1538.
[183] Z. Ma, H. Guo, J. Chen, Z. Li, G. Peng, Y. jiao Gong, Y. Ma, Z. Cao, MetaBox: A benchmark platform for meta-black-box optimization with reinforcement learning, 2023, http://dx.doi.org/10.48550/arXiv.2310.08252, arXiv arXiv:2310.08252, URL https://www.semanticscholar.org/paper/c512d1a6ebf4173d94ffb07bd1b69b7c1ec8adad.
[184] A. Auger, N. Hansen, Performance evaluation of an advanced local search evolutionary algorithm, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 2, IEEE, 2005, pp. 1777–1784.
[185] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, KanGAL Report 2005005 (2005) (2005) 2005.
[186] P. Bennet, C. Doerr, A. Moreau, J. Rapin, F. Teytaud, O. Teytaud, Nevergrad: black-box optimization platform, ACM SIGEVOlution 14 (1) (2021) 8–15.
[187] J. Vanschoren, J.N. Van Rijn, B. Bischl, L. Torgo, OpenML: networked science in machine learning, ACM SIGKDD Explor. Newsl. 15 (2) (2014) 49–60.
[188] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, A. Smola, AutoGluon-tabular: Robust and accurate AutoML for structured data, 2020, arXiv Preprint arXiv:2003.06505.
[189] L. Li, Y. Kochura, G. Wang, J. Liu, M. Kumar, N. Mahajan, E.P. Xing, C. Zhang, Mlbench: Benchmarking machine learning services against human performance, Proc. VLDB Endow. 13 (12) (2020) 3005–3019.
[190] R.S. Olson, R.J. Urbanowicz, P.C. Andrews, N.A. Lavender, J.H. Moore, Automating biomedical data science through tree-based pipeline optimization, Appl. Evol. Comput. (2016) 123–137.
[191] E. LeDell, S. Poirier, H2O AutoML: Scalable Automatic Machine Learning, H2O.ai, 2020.
[192] C. Wang, Q. Wu, M. Weimer, E. Zhu, Flaml: A fast and lightweight automl library, Proc. Mach. Learn. Syst. 3 (2021) 434–447.
[193] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, Adv. Neural Inf. Process Syst. 28 (2015).
[194] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann, Stable Baselines3: Reliable reinforcement learning implementations, J. Mach. Learn. Res. 22 (268) (2021) 1–8.
[195] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, I. Stoica, RLlib: Abstractions for distributed reinforcement learning, in: International Conference on Machine Learning, PMLR, 2018, pp. 3053–3062.
[196] S. Huang, R.F.J. Dossa, A. Raffin, A. Kanervisto, W. Wang, CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms, J. Mach. Learn. Res. 23 (274) (2022) 1–18.
[197] D. Zha, K.-H. Lai, S. Huang, Y. Cao, K. Reddy, J. Vargas, A. Nguyen, R. Wei, J. Guo, X. Hu, RLCard: a platform for reinforcement learning in card games, in: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, 2021, pp. 5264–5266.
[198] R. Ramamonjison, H. Li, T.T. Yu, S. He, V. Rengan, A. Banitalebi-Dehkordi, Z. Zhou, Y. Zhang, Augmenting operations research with auto-formulation of optimization models from problem descriptions, in: Conference on Empirical Methods in Natural Language Processing, EMNLP, 2022, pp. 29–62.
[199] K. Deb, A. Pratap, Evolutionary algorithms for constrained optimization problems, IEEE Trans. Evol. Comput. 4 (3) (2000) 265–278.
[200] D.P. Mandic, A generalized normalized gradient descent algorithm, IEEE Signal Process Lett. 11 (2) (2004) 115–118.
[201] M. Zinkevich, M. Weimer, L. Li, A. Smola, Parallelized stochastic gradient descent, Adv. Neural Inf. Process Syst. 23 (2010).
[202] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv Preprint arXiv:1412.6980.
[203] G. Hinton, N. Srivastava, K. Swersky, Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, Coursera (2012) 29.
[204] Z. Xu, H.P. van Hasselt, D. Silver, Meta-gradient reinforcement learning, Adv. Neural Inf. Process Syst. 31 (2018).
[205] J. Oh, M. Hessel, W.M. Czarnecki, Z. Xu, H.P. van Hasselt, S. Singh, D. Silver, Discovering reinforcement learning algorithms, Adv. Neural Inf. Process. Syst. 33 (2020) 1060–1070.
[206] Z. Xu, H.P. van Hasselt, M. Hessel, J. Oh, S. Singh, D. Silver, Meta-gradient reinforcement learning with an objective discovered online, Adv. Neural Inf. Process. Syst. 33 (2020) 15254–15264.
[207] C. Bonnet, P. Caron, T.D. Barrett, I. Davies, A. Laterre, One step at a time: Pros and cons of multi-step meta-gradient reinforcement learning, 2021, arXiv arXiv:2111.00206.
[208] S. Flennerhag, Y. Schroecker, T. Zahavy, H. van Hasselt, D. Silver, S. Singh, Bootstrapped meta-learning, 2021, arXiv Preprint arXiv:2109.04504.
[209] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, 2018, arXiv Preprint arXiv:1806.09055.
[210] Y. Miao, X. Song, J.D. Co-Reyes, D. Peng, S. Yue, E. Brevdo, A. Faust, Differentiable architecture search for reinforcement learning, in: International Conference on Automated Machine Learning, PMLR, 2022, pp. 20–21.