Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 21(6): 6289–6335.
DOI: 10.3934/mbe.2024275
Received: 31 December 2023
Revised: 04 March 2024
Accepted: 28 March 2024
Published: 14 June 2024

Survey

Hyperparameter optimization: Classics, acceleration, online, multi-objective, and tools

Jia Mian Tan[1], Haoran Liao[1], Wei Liu[1], Changjun Fan[2,*], Jincai Huang[2], Zhong Liu[2] and Junchi Yan[1,*]

[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2] College of Systems Engineering, National University of Defense Technology, Changsha, China

* **Correspondence:** Email: fanchangjun@nudt.edu.cn, yanjunchi@sjtu.edu.cn.

Abstract: Hyperparameter optimization (HPO) has been well-developed and evolved into a well-established research topic over the decades. With the success and wide application of deep learning, HPO has garnered increased attention, particularly within the realm of machine learning model training and inference. The primary objective is to mitigate the challenges associated with manual hyperparameter tuning, which can be ad-hoc, reliant on human expertise, and consequently hinders reproducibility while inflating deployment costs. Recognizing the growing significance of HPO, this paper surveyed classical HPO methods, approaches for accelerating the optimization process, HPO in an online setting (dynamic algorithm configuration, DAC), and when there is more than one objective to optimize (multi-objective HPO). Acceleration strategies were categorized into multi-fidelity, bandit-based, and early stopping; DAC algorithms encompassed gradient-based, population-based, and reinforcement learning-based methods; multi-objective HPO can be approached via scalarization, metaheuristics, and model-based algorithms tailored for multi-objective situation. A tabulated overview of popular frameworks and tools for HPO was provided, catering to the interests of practitioners.

Keywords: hyperparameter optimization; machine learning; deep neural networks; bayesian optimization; survey

1. Introduction

Machine learning (ML) is a discipline of artificial intelligence (AI) that uses the theory of statistics in building mathematical models to program computers to "learn" or "discover" algorithms, where an algorithm is a sequence of instructions for solving a problem or performing a computation, to optimize the performance criteria for given tasks using example data or past experiences [1]. ML has found applications across various domains, such as image recognition [2], healthcare [3], natural language

processing [4], games and strategy [5], etc. In ML models, parameters fall into two categories: model parameters (e.g., weights and biases of the connections between neurons of a neural network, centroids in k-means clustering) and hyperparameters (e.g., learning rate, number of hidden layers of a neural network, number of clusters in k-means clustering). While model parameters can be learned and adapted during the training process based on the training dataset, hyperparameters serve as external configuration variables that are to be determined before the training commences. The tuning of hyperparameters has played an essential role both from the methodological perspective, e.g., deep neural networks and shallow models, as well as from the application aspect, with models derived from computer vision, natural language models, speech, etc. Though it seems trivial at first glance that hyperparameters can be tuned by humans, in modern systems, manual tuning can be ineffective and is nearly impossible when dealing with large-scale hyperparameters.

Differing from the human tuning process, which can be tedious and heavily dependent on human expertise and experience, hyperparameter optimization (HPO) aims to automatically tune the relevant hyperparameters for the system, where the performance is measured by certain metrics, e.g., classification accuracy for a classifier model could be tuned to a better point. Apart from saving human labor and boosting model performance to its fullest potential, another advantage lies in enhancing the evaluation of different models. This can be achieved by diligently fine-tuning their respective hyperparameters, even when the models differ from one another. In contrast, the traditional trial-and-error procedure by humans can be ad-hoc and less reproducible, posing challenges in the evaluation process.

In recent years, deep learning (DL) techniques have reached remarkable achievements on various AI tasks, including image classification [6], language modeling [4], and speech recognition [7]. However, these models are sensitive to diverse task-specific configurations, costing substantial expert efforts to redesign them through a trial-and-error process. Automated machine learning (AutoML) [8] has been proposed to tackle this problem in areas such as data preparation, feature engineering, model generation, and model evaluation.

Given the search space, optimization methods in model generation can be classified into two categories: HPO and architecture optimization. In this paper, we mainly focus on HPO algorithms. A generic HPO algorithm for an unknown objective function $f\colon \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \in \mathbb{R}^d$ and d is the size of design space of interest, is to find the $\mathbf{x}$, a hyperparameter configuration, e.g., learning rate and batch size that globally minimizes f:

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}). \tag{1.1}$$

Although gradient descent [9] is popular in DL, which is capable of tuning the learning rate, the cases where it is possible to obtain the training gradient of hyperparameters are rare. Therefore, traditional HPO frameworks treat f as a non-convex black-box function and search for optimum without leveraging derivatives. As shown in Figure 1, classical HPO algorithms cover grid search and random search [10], metaheuristic search [11, 12], Bayesian optimization [13], etc.

Unfortunately, despite the success of liberating humans from the loop [14], most conventional HPO algorithms follow a sequential querying approach and therefore suffer delays of the convergence of neural network training, which is always computationally resource-intensive. For instance, the whole training process for the once-for-all (OFA) network [15] took 1200 GPU hours with V100 GPUs. The community has thus proposed many *acceleration* methods to shorten the optimization process and avoid

unnecessary repetition. Another drawback of classical HPOs is that they often yield fixed configurations throughout the entire run of AI models, even though the optimum may change at different phases. To tackle this, some *dynamic algorithm configuration* (DAC) methods [16–19] emerge to optimize *on-the-fly*, i.e., dynamically tune both parameters and hyperparameter schedules during the training procedure. In practical settings, there can be more than one metric to optimize beyond the conventional prediction accuracy (cf. Eqs (1.1) and (5.1)). Additional considerations such as space/time overheads and adherence to specific business constraints can play pivotal roles. The field has thus recently witnessed a growing interest in multi-objective HPO (MOHPO), the HPO counterpart to multi-objective optimization (MOO) [20, 21].

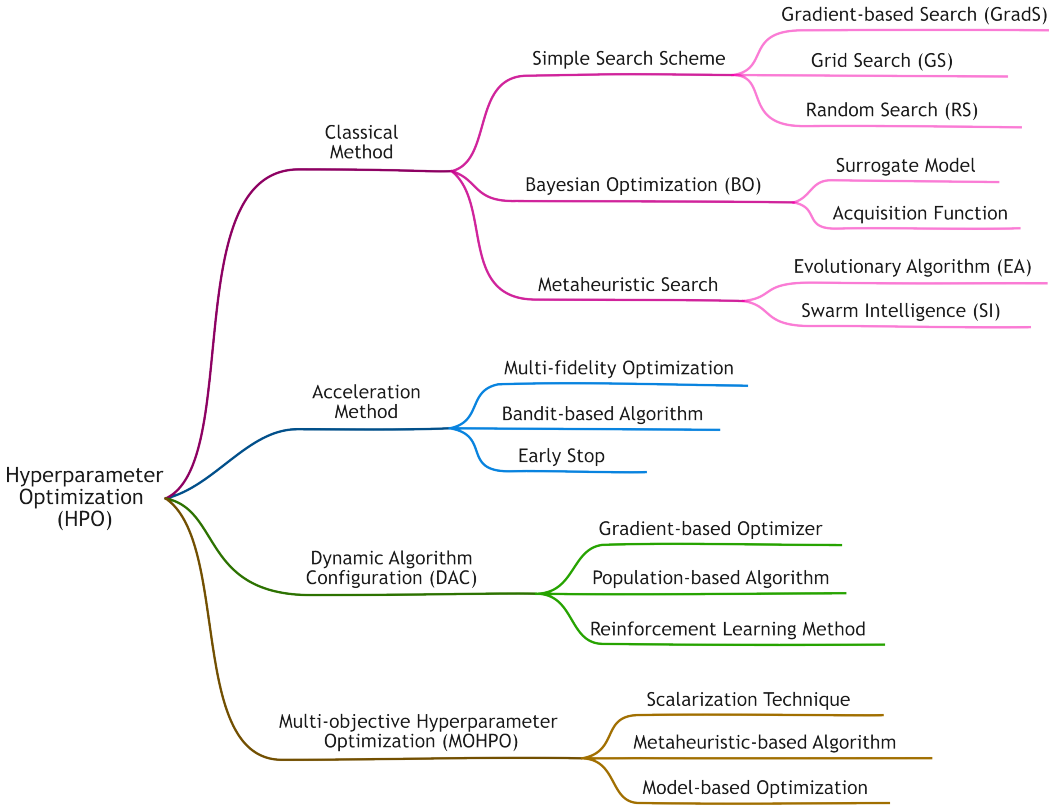

Figure 1. An overview of the lines of research around the topic of HPO surveyed in this paper: classical optimization methods in Section 2, acceleration techniques in Section 3, DAC in Section 4, and MOHPO in Section 5.

As a well-established field, HPO has attracted wide and growing attention over the decades from both academia and industry. There exist numerous surveys covering a range of related topics, including the specific topic of HPO itself [22–27], some closely related topics, e.g., Bayesian optimization [14, 28–30], as well as topics of broader scopes such as AutoML [8, 31, 32].

Existing surveys on general HPO [22–24, 26, 27] introduce HPO algorithms by categorizing them into several major classes: grid search (GS), random search (RS), Bayesian optimization (BO) and variants, multi-fidelity or Hyperband, population- or evolutionary-based, and gradient-based. This work takes a distinctive approach (Figure 1) and considers these algorithms as (i) classical methods for HPO, i.e., simple searches, BOs, and metaheuristics, (ii) techniques for acceleration, i.e., multi-

fidelity, bandit-based, and early stopping, (iii) DAC optimizers that are either gradient-, population-, or reinforcement learning-based, and (iv) strategies for MOHPO. Notably, DAC and MOHPO algorithms in particular have been either minimally explored or entirely overlooked in [22–24, 26, 27]. A list of popular frameworks and tools for HPO is compiled and tabulated, serving as an accessible guide for individuals new to the field, students, researchers, or practitioners in search of diverse options or alternatives for their HPO tasks or workflow integration.

This survey is structured as follows. In Section 2, we provide the basics of the primary classical algorithms in HPO. Section 3 then delves into strategies for accelerating these conventional algorithms from different perspectives. This is followed by Sections 4 and 5, where we explore the increasingly recognized DAC and MOHPO algorithms, respectively. After presenting popular tools and frameworks for HPO in Section 6.1, exploring applications in Section 6.2, and providing further discussions in Section 6.3, the paper is concluded in Section 7.

2. Classical optimization methods

In this section, we introduce three kinds of optimization methods that are commonly applied in HPO. These methods have to handle two key considerations: (i) the exploration vs. exploitation trade-off, which refers to the budget spent on exploring unknown search space or on exploiting known search space, and (ii) the inference vs. search trade-off, referring to the overhead used to analyze existing information to guide the search process versus the budget allocated for the search itself. Table 1 summarizes the classical optimization methods.

Table 1. A concise overview of primary classical optimization methods. The column definitions are as follows: *Variable*: the types of hyperparameters the method can address. *Hierarchy*: whether the method can handle complex hierarchy search spaces. *Parallelizability*: whether the method can propose more than one candidate at the same time. Certain methods, such as sequential model-based algorithm configuration (SMAC), can build the model in parallel but propose only one candidate at a time. *Exploration* and *Exploitation*: how the method explores unknown areas, exploits known areas, and manages the trade-off between them. We assume that there are d hyperparameters with each of them having n distinct values. Note that this table covers only standard algorithms, which can easily be extended. See Section 2 for more details.

Method	Variable	Hier-archy	Parallel-izability	Time complexity	Exploration	Exploitation
GradS	Continuous	×	×	$O(n^d)$	×	Gradient descent
GS	Continuous, Discrete, Categorical	✓	✓	$O(n^d)$	Grid	×
RS	Continuous, Discrete, Categorical	✓	✓	$O(n)$	Random	×
BO-GP	Continuous	×	×	$O(n^3)$	Balanced by acquisition function	
SMAC	Continuous, Discrete, Categorical	✓	×	$O(n \log n)$	Balanced by acquisition function	
BO-TPE	Continuous, Discrete, Categorical	✓	×	$O(n \log n)$	Balanced by acquisition function	
GA	Continuous, Discrete, Categorical	×	✓	$O(n^2)$	Crossover/Mutation	Selection
ES	Continuous	×	✓	$O(n^2)$	Recombination/Mutation	Selection
ω-PSO	Continuous	×	✓	$O(n \log n)$	Balanced by parameter ω	

2.1. Simple search scheme

2.1.1. Gradient-based search (GradS)

Gradient descent-based methods are extensively used for optimization problems including the HPO task, which often involves non-convex optimization, and a local optimum is acceptable. Hypergradients (HGs) refer to the gradients of the model selection criterion such as cross-validation performance and validation error with respect to hyperparameters. With hypergradients, gradient descent can be employed to handle a large number of hyperparameters efficiently.

Reverse-mode differentiation (RMD) [33], known as backpropagation in DL, has been the standard method for computing gradients with respect to parameters in ML models. It was introduced to HPO problems to compute hypergradients [34]. While algorithms for computing hypergradients for optimization methods, including gradient descent, were derived in Domke [35], their impracticality for DL models due to high memory consumption for storing all intermediate variables has been recognized. In Maclaurin et al. [36], reverse-mode differentiation of stochastic gradient descent (SGD) with momentum has been proposed, overcoming the problem by computing intermediate variables during the reverse pass. It is further improved in Pedregosa [37] with the adoption of approximate gradients. In Franceschi et al. [38], reverse-mode algorithms for hypergradients are explained from the perspective of Lagrangian formulation, and a forward-mode algorithm is introduced for situations involving a small number of hyperparameters. Later, Lorraine et al. [39] devised a scalable gradient-based HPO technique capable of handling millions of hyperparameters. Reverse-mode and forward-mode hypergradient algorithms are summarized in Algorithms 1 and 2, respectively, based on the derivations in Franceschi et al. [38]. In these two algorithms, Φ represents parameter optimization methods such as gradient descent, $\mathbf{w}$ denotes parameters or weights of DL models, $\mathbf{x}$ signifies hyperparameters, and E is the validation objective.

Algorithm 1: Reverse-Hypergradient (credit to Franceschi et al. [38])

Input: Initial hyperparameter $\mathbf{x}$, initial parameters $\mathbf{w}_0$

// Train parameters

for $t = 1$ **to** T **do**

  Update $\mathbf{w}_t \leftarrow \Phi_t(\mathbf{w}_{t-1}, \mathbf{x})$

// Compute hypergradients reversely

$\Delta\mathbf{x} \leftarrow 0$

$\alpha_T \leftarrow \nabla E(\mathbf{w}_T)$

for $t = T - 1$ **downto** 1 **do**

  $\Delta\mathbf{x} \leftarrow \Delta\mathbf{x} + \alpha_{t+1}\frac{\partial\Phi_{t+1}}{\partial\mathbf{x}}$

  $\alpha_t \leftarrow \alpha_{t+1}\frac{\partial\Phi_{t+1}}{\partial\mathbf{w}_t}$

Output: Hypergradients $\Delta\mathbf{x}$

Algorithm 2: Forward-Hypergradient (credit to Franceschi et al. [38])

Input: Initial hyperparameters $\mathbf{x}$, initial parameters $\mathbf{w}_0$

$\Delta\mathbf{x} \leftarrow 0$

$\mathbf{Z}_0 \leftarrow 0 \qquad // \ \mathbf{Z}_t = \frac{\partial \mathbf{w}_t}{\partial \mathbf{x}}$

`// Train parameters and compute hypergradients forwardly`

for $t = 1$ **to** T **do**

  $\mathbf{w}_t \leftarrow \Phi_t(\mathbf{w}_{t-1}, \mathbf{x})$

  $\mathbf{Z}_t \leftarrow \frac{\partial \Phi_t}{\partial \mathbf{w}_{t-1}} \mathbf{Z}_{t-1} + \frac{\partial \Phi_t}{\partial \mathbf{x}}$

$\Delta\mathbf{x} \leftarrow \nabla E(\mathbf{w}_T)\mathbf{Z}_T$

Output: Hypergradients $\Delta\mathbf{x}$

2.1.2. Grid search (GS)

GS is the most basic HPO method. It requires the user to select a finite subset for each hyperparameter and then exhaustively evaluate every possible combination point to find the optimum. GS inherently supports parallel implementation but fails in efficiency once the search space is large or objective dimensions get high, as the number of combinations to evaluate grows exponentially. To tackle these challenges, a multi-scale grid approach is used in Hsu et al. [40], where a coarse grid is first applied to identify a good region, and then a finer grid is conducted on that region. Alternatively, direct search [41] queries only the neighbors around current points to update the optimum. When neither improvement nor degradation is observed in any parameter, the search step is reduced until convergence.

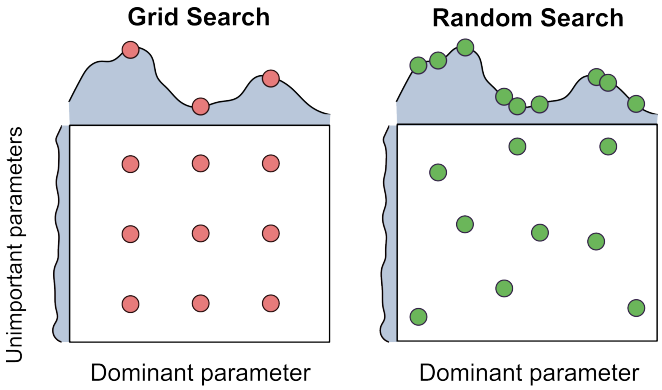

Figure 2. Comparison of GS and RS. As reported in Bergstra and Bengio [10], it is always a few hyperparameters dominating the data. The limitations of GS stem from its inefficient exploration of influential dimensions. Figure adapted from Bergstra and Bengio [10].

2.1.3. Random search (RS)

Contrary to GS, RS randomly and independently samples candidates from the search space until the defined budget is exhausted. While GS is inefficient in high dimensions, RS is less affected, benefiting from lower effective dimensionality where only a few hyperparameters have an influence on the performance [10]. Given the same budget, RS can explore a larger area of the effective dimensions compared to GS, leading to better performance. As shown in Figure 2, RS tends to explore a broader

space than GS with a limited budget, avoiding lingering in less promising areas. Bergstra and Bengio [10] have proved empirically and theoretically that RS is more practical than GS while in some cases sophisticated methods may bring little advantage over RS. Additionally, RS is readily resource-allocated since it can be extended by further samples, and the probability of sampling in different regions can be adjusted manually so exploration of valuable regions can be prioritized. While RS may lead to suboptimal solutions, its performance can be arbitrarily close to the optimum in expectation when provided with enough budgets.

2.2. *Bayesian optimization (BO) with surrogates and acquisitions*

BO [13] is an efficient global black-box optimization framework for expensive functions. Recently, it has gained widespread application in HPO problems and achieved state of the art results across various ML domains like image classification [13] and speech recognition [42].

BO is the most popular sequential model-based optimization (SMBO) algorithm, which has proven its superiority in optimizing expensive black-box functions [14] and exhibits impressive performance on even hard-to-tune hyperparameters [43]. Figure 3 illustrates a general sequential optimization framework that utilizes a model learned from observations to recommend promising candidates, which then gets queried to generate feedback for updating the model.

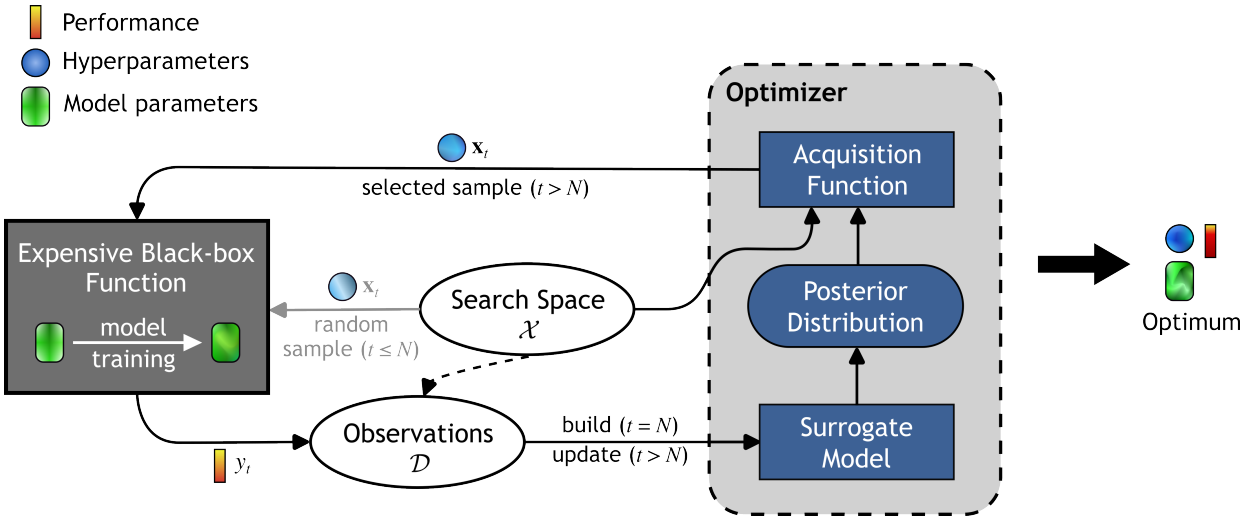

Figure 3. A general pipeline of SMBO. When $t \leq N$, it is the initialization stage where the observation dataset $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}$ is populated by points $\mathbf{x}_t$ sampled randomly from the search space $\mathcal{X}$. When $t > N$, candidate points $\mathbf{x}_t \in \mathcal{X}$ are selected using the acquisition function, which is guided by the posterior distribution derived from the surrogate model.

A classic BO framework comprises a probabilistic surrogate model and an acquisition function. The surrogate model approximates the unknown objective function f based on the observation dataset $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{N}$, where $\mathbf{x}_t \in \mathcal{X}$ and $y_t \in \mathbb{R}$ are the input and the observation value of f, respectively. The prior distribution of the surrogate model captures our knowledge about f and is updated with $\mathcal{D}$ to generate a posterior distribution, which provides predictions and uncertainties over the search space $\mathcal{X}$. The acquisition function α utilizes the posterior distribution to guide the sequential search. Instead of observing the expensive objective function, BO optimizes the cheap acquisition function

globally to generate candidates. The main property of the acquisition function is the trade-off between the exploration of areas with high uncertainty and the exploitation of areas with low predictions (for minimization task).

One common choice for the surrogate model is Gaussian process (GP) [44], and for the acquisition function, it is the expected improvement (EI) [45]. We summarize the BO algorithm with GP and EI in Algorithm 3, where ϕ and Φ denote the standard normal probability density function (p.d.f.) and cumulative distribution function (c.d.f.), respectively, and f^* is the best-known value.

Algorithm 3: Bayesian-Optimization (credit to Shahriari et al. [14])

Input: Initial number N, total number T, noise δ_n

Initialize $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{N}$ randomly

for $t = N + 1$ **to** T **do**

`// Recommend candidates by EI`

Acquire $\alpha_t = \sigma_t \phi(\gamma_t) + (f^* - \mu_t)\Phi(\gamma_t)$

Sample $\mathbf{x}_t = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha_t$

Evaluate $y_t \leftarrow f(\mathbf{x}_t) + \delta_{\mathrm{n}}$ `// Expensive`

Update $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(\mathbf{x}_t, y_t)\}$

Update $f^* = \min_{1\le i\le t} y_i$

`// Estimate posterior by` $\mathcal{GP}(m, K)$

$\mu_{t+1} \leftarrow \mu_0 + \mathbf{k}_*^\top \left[\mathbf{K} + \sigma_y^2 \mathbf{I}\right]^{-1} \mathbf{y}$

$\sigma_{t+1}^2 \leftarrow k_{**} - \mathbf{k}_*^\top \left[\mathbf{K} + \sigma_y^2 \mathbf{I}\right]^{-1} \mathbf{k}_*$

$\gamma_{t+1} \leftarrow (f^* - \mu_{t+1})/\sigma_{t+1}$

Output: Optimal hyperparameters $\mathbf{x}_T$

2.2.1. Surrogate models

The performance of BO significantly hinges on the choice of surrogate model. GP, random forest (RF), tree-structured Parzen estimator (TPE), and Bayesian neural network (BNN) are the commonly employed surrogate models for BO. A concise comparison of surrogates is presented in Table 2. A detailed discussion of these surrogate models follows.

Table 2. Comparison of four common BO surrogate models.

Surrogates	Time complexity	Fit type
GP	$O(n^3)$	Regression
RF	$O(n \log n)$	Regression
TPE	$O(n \log n)$	KDE and Classification
BNN	$O(n)$	Regression

• **GP** is a nonparametric model that is fully specified by a prior mean function $\mu_0 \colon \mathcal{X} \to \mathbb{R}$, which is usually set to a constant, and a covariance function $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Any finite collection of GP induces a multivariate normal distribution. The marginalization properties of GP enable them to compute

marginals and conditionals in closed form simply and flexibly. Given the observation dataset $\mathcal{D}_t$ at step t, the posterior mean and variance functions are:

$$\mu_t(\mathbf{x}) = \mu_0(\mathbf{x}) + \mathbf{k}_*^\top \left[\mathbf{K} + \sigma_y^2 \mathbf{I}\right]^{-1} \mathbf{y} \tag{2.1}$$

$$\sigma_t^2(\mathbf{x}) = k_{**} - \mathbf{k}_*^\top \left[\mathbf{K} + \sigma_y^2 \mathbf{I}\right]^{-1} \mathbf{k}_* \tag{2.2}$$

where $\mathbf{k}_*$ denotes a vector of covariances between $\mathbf{x}$ and all points in $\mathcal{D}_t$, $k_{**} = k(\mathbf{x}, \mathbf{x})$ denotes the variance of $\mathbf{x}$, and $\mathbf{K}$ is the covariance matrix of all points in $\mathcal{D}_t$. The property of GP is determined by the covariance function k, with Mátern $5/2$ kernel being the most commonly used.

While BO with GP performs well on real-valued hyperparameters, it has difficulty with discrete, non-numeric, or conditional hyperparameters. Special kernels are required to address these situations [46]. In addition, GP scales poorly (cubically) with increasing data due to the necessity of the inversion of a dense covariance matrix. To mitigate this, some sparsification techniques have been proposed. Among them are sparse pseudo-input GPs (SPGPs) [47], which select a subset of the original dataset as inducing pseudoinputs to reduce the rank of the covariance matrix and compute the approximate posterior quickly. Another drawback of the standard GP is its poor scalability with dimensions, limiting the number of hyperparameters it can tune. The properties of hyperparameter space have been leveraged to design new kernels, such as cylindrical kernels [48] and additive kernels [49].

• **RF**, which is used in sequential model-based algorithm configuration (SMAC), models the objective function using an ensemble of regression trees [50]. The algorithm works as follows: B regression trees are constructed with n data points randomly sampled with replacement from the dataset of size n. For each split point of a tree, the split criterion is chosen from a subset of size pd that is randomly selected from d hyperparameters, where p is a split ratio that defaults to $5/6$. When the number of data points on a node falls below a threshold n_{min}, the node is set to a leaf and the leaf's prediction is set to the mean or median of the data points on it. Given a new hyperparameter configuration, these trees can produce B predictions for which the mean and variance can be computed.

SMAC can handle continuous, discrete, categorical, and conditional hyperparameters naturally. The time complexities for fitting and predicting are $O(n \log n)$ and $O(\log n)$, respectively. The limitation of data points on leaves and parallel training of trees can further reduce budgets, making RF suitable for larger datasets compared to GP. The subsampling of hyperparameters also helps it work on high-dimensional search spaces. However, despite RF's good predictive performance in the vicinity of training data, it exhibits poor performance far from the data. In areas with missing data, the variance can be highly erratic, ranging from very large to very small.

• **TPE** [51] uses kernel density estimation and classifies observations instead of regression. In contrast to most approaches modeling $P(y \mid \mathbf{x})$ directly, TPE considers Bayes' rule $P(y \mid \mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}$ and models $P(\mathbf{x} \mid y)$ and $P(y)$. Two densities, $l(\mathbf{x})$ and $g(\mathbf{x})$, are built over the search space $\mathcal{X}$ as follows:

$$P(\mathbf{x} \mid y) = \begin{cases} l(\mathbf{x}), & y < y^* \\ g(\mathbf{x}), & y \geq y^* \end{cases} \tag{2.3}$$

where y^* is a predefined percentile determined by a threshold γ such that $P(y < y^*) = \gamma$. The time complexity is $O(n \log n)$. The EI acquisition function is then optimized. By construction and

simplification, we have the result:

$$\alpha_{EI}(\mathbf{x}) \propto \left(\gamma + \frac{g(\mathbf{x})}{l(\mathbf{x})}(1-\gamma)\right)^{-1} \tag{2.4}$$

Based on this expression, when optimizing $EI_{y^*}(\mathbf{x})$, finding the optimum point of $\frac{g(\mathbf{x})}{l(\mathbf{x})}$ suffices and it is not necessary to model $P(y)$. Since the Parzen estimators are organized in a tree structure, TPE can handle conditional hyperparameters naturally and outperform GP in structured HPO tasks [52, 53].

• **BNN** places a distribution over neural network parameters, amalgamating the strengths of neural networks and probabilistic models [54]. Neural networks have strong capabilities of approximating continuous functions, extending the applicability of BO to more complex tasks. Probabilistic models can generate a complete posterior distribution on the predictions, suitable for Bayesian analysis. In Deep Networks for Global Optimization (DNGO) [55], the prior distribution is put on the weights of the output layer, while other parameters are learned via point estimation (typically stochastic training). BOHAMIANN [56] also adopts a BNN to construct the response surface, with weights sampled using a stochastic gradient Hamiltonian Monte-Carlo (SGHMC) method [57] to evaluate the posterior. BNN is more scalable than GP and is faster when the dataset is large [55].

2.2.2. Acquisition functions

Leveraging the predictive posterior, acquisition functions recommend the most promising candidate in the trade-off between the exploitation of known optima and the exploration of uncertainty.

EI is an improvement-based function [14]. It measures both the amount and the probability of improvement:

$$\alpha_{EI}(\mathbf{x}) = \mathbb{E}[\max\{(f^* - f(\mathbf{x})), 0\}] \tag{2.5}$$

When $f(\mathbf{x})$ is normally distributed with mean $\mu(\mathbf{x})$ and variance $\sigma(\mathbf{x})$, the expectation can be computed analytically as:

$$\alpha_{EI}(\mathbf{x}) = (f^* - \mu(\mathbf{x}))\Phi\left(\frac{f^* - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{f^* - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \tag{2.6}$$

where ϕ and Φ denote the standard normal p.d.f. and c.d.f., respectively, and f^* is the best-known value.

Lower confidence bound (LCB), or upper confidence bound (UCB) for maximization problems [58], treats uncertainty as an additive incentive. It uses the optimum point of a fixed probability surface according to the model. In the GP case, LCB is computed as:

$$\alpha_{LCB}(\mathbf{x}) = \mu(\mathbf{x}) - \beta\sigma(\mathbf{x}) \tag{2.7}$$

where β is a control parameter and this function is to be minimized. Some guidelines for choosing β have been proposed to achieve optimal regret [58].

Information-based policies aim to deduce the position of optimum point $\mathbf{x}^*$ by considering the posterior distribution $P(\mathbf{x}^* \mid \mathcal{D})$. Entropy search (ES) [59] selects the point that maximally reduces the entropy of $P(\mathbf{x}^* \mid \mathcal{D})$. It measures the expected information gain about the position of $\mathbf{x}^*$:

$$\alpha_{ES}(\mathbf{x}) = H(\mathbf{x}^* \mid \mathcal{D}) - \mathbb{E}_{y|\mathcal{D},\mathbf{x}}[H(\mathbf{x}^* \mid \mathcal{D} \cup \{(\mathbf{x}, y)\})] \tag{2.8}$$

where $H(\mathbf{x}^* \mid \mathcal{D})$ is the differential entropy of $P(\mathbf{x}^* \mid \mathcal{D})$, and the expectation is over the predictive distribution of $y = f(\mathbf{x}) + \delta_n$, where δ_n is the observation noise. However, this function is intractable and

is usually approximated by expensive methods such as Monte Carlo (MC) sampling, whose computation cost is quartic.

Predictive entropy search (PES) [60] reformulates the function of ES with the symmetric property of mutual information as:

$$\alpha_{PES}(\mathbf{x}) = H(y \mid \mathcal{D}, \mathbf{x}) - \mathbb{E}_{\mathbf{x}^*|\mathcal{D}}[H(y \mid \mathcal{D}, \mathbf{x}, \mathbf{x}^*)] \tag{2.9}$$

where the expectation is over distribution $P(\mathbf{x}^* \mid \mathcal{D})$. This function is approximated by expectation propagation and Thompson sampling. The computation cost is cubic, and the performance is not worse than ES empirically.

Max-value entropy search (MES) [61] uses the information about the optimum value $y^* = f(\mathbf{x}^*)$ instead of $\mathbf{x}^*$. The expected information gain about y^* is expressed as:

$$\alpha_{MES}(\mathbf{x}) = H(y \mid \mathcal{D}, \mathbf{x}) - \mathbb{E}_{\mathbf{y}^*|\mathcal{D}}[H(y \mid \mathcal{D}, \mathbf{x}, \mathbf{y}^*)] \tag{2.10}$$

Here, y^* is sampled via Gumbel distribution or from the posterior distribution, and the expectation is approximated using MC estimation. The computation cost of MES is much lower since the distribution $P(\mathbf{x}^* \mid \mathcal{D})$ is d-dimensional, while $P(y^* \mid \mathcal{D})$ is one-dimensional. Empirically, MES performs at least as well as ES and PES.

2.2.3. Recent BOs

Although optimizers are epoch efficient with little overheads and some parallel versions of algorithms are now available [13], two common drawbacks still exist [62]: First, the intrinsic observation process can be time-consuming; second, SMBO provides only fixed hyperparameters. Many methods emerge to accelerate the vanilla BO as will be discussed in Section 3. Notably, BFO (Bayesian Functional Optimization) [63] has been proposed to optimize hyperparameters on function spaces. 2-OPT (Two-step optimal) [64] enables the acquisition functions to look ahead for two steps to alleviate the shortsightedness of BO. In an attempt to scale BO to high-dimensional domains, while LineBO [65] decomposes iteratively the global problem into a sequence of one-dimensional sub-problems, TuRBO (trust region BO) [66] maintains a collection of local BO models and performs search across trust regions centered around the best solutions. Nguyen and Osborne [67] transform the GP to incorporate more foregone information (e.g., cases where classification accuracy is less than 100%). Meanwhile, MiVaBo (mixed-variable BO) [68] extends BO to optimize variables of mixed types. Several methods, including BOPrO (BO with a Prior for the Optimum) [69], πBO [70], and PriorBand [71], have been proposed to incorporate expert insights on promising configurations into BO, using this prior information to guide the search.

2.3. Metaheuristic search

A metaheuristic is a generic or high-level optimization strategy or algorithmic framework designed to efficiently explore and exploit solution spaces to find approximate solutions to optimization problems [11]. Metaheuristic search often draws inspiration from natural phenomena, such as evolution and annealing. In general, metaheuristics make no assumptions about the objective function, and they do not rely on gradient information, enabling them to tackle non-convex, noncontinuous, and non-smooth optimization problems. According to the number of solutions they hold, metaheuristics can be categorized into single-solution-based methods and population-based methods. Population-based methods run a population of solutions in parallel and evaluate their quality using a fitness function,

demanding significant computing power. Advances in computer technology and parallel architectures have facilitated the realization of many algorithms. The distinctions among population-based methods lie in how they initialize and update populations, with performance being greatly influenced by parameter settings. This section introduces two types of population-based methods: evolutionary algorithms and swarm intelligence.

2.3.1. Evolutionary algorithms (EA)

Evolutionary algorithms (EAs) [73] are inspired by Darwin's evolutionary theory. Generally, EAs update populations through the crossover of two ancestral individuals and mutation. Genetic algorithm (GA) [74] is the most commonly used method. As shown in Figure 4, GA typically represents solutions as chromosomes, often in the form of a fixed-length binary string, and implements crossover and mutation by simple bit manipulation operations. Two critical parameters in GA are the probabilities of crossover and mutation. The procedures of GA are as follows: A population with N individuals is initially generated by randomly initializing chromosomes. Subsequently, a fitness function, whose outcome often reflects the performance of a configuration, is applied to each individual. Based on the results, selection, crossover, and mutation are performed on the population to yield a new generation with N individuals. These two steps are repeated until convergence or some conditions are met. A majority of innovations of GAs are the selection schemes for producing offspring, including roulette-wheel, tournament, and ranking selection [75].

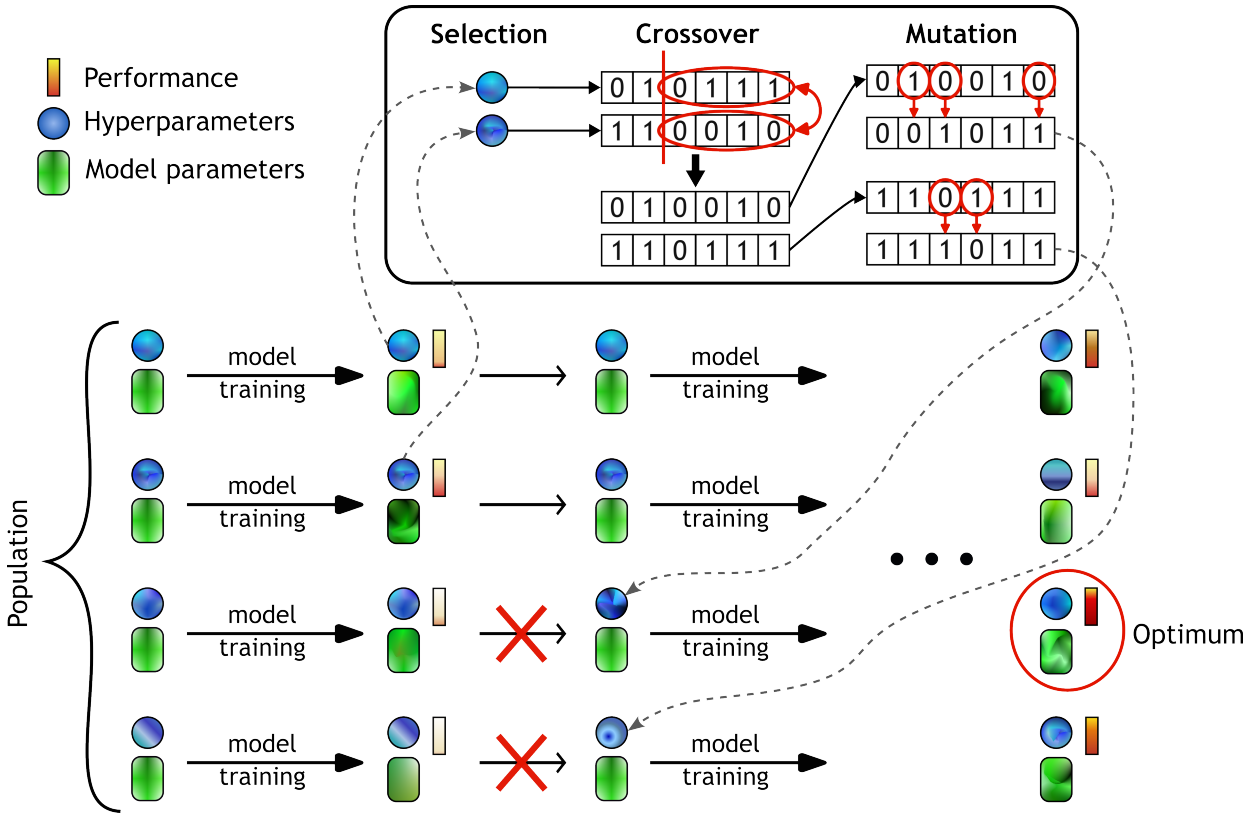

Figure 4. Workflow of GAs for HPO.

In a certain context, evolution strategy (ES) [76] slightly differs from GA by extending the values of GA's chromosomes to the real number domain and introducing the heritable step to guide mutation. Mutation for real numbers is realized by adding noise sampled from a zero-mean normal distribution. The standard deviations for different genes significantly impact the performance of ES. They can be kept constant or adjusted dynamically based on the number of generations and performance [77, 78]. Selection in ES involves eliminating the worst individuals to maintain a constant population size. While ES was originally designed for real numbers, it can be easily extended to other data types, such as integers [79]. ES prefers exploration to exploitation, so it is less prone to getting stuck in local optima.

CMA-ES (Covariance matrix adaptation evolution strategy) [80] is the most representative and effective ES algorithm. It dynamically adapts distribution parameters and the step by modifying the covariance matrix. Differential evolution (DE) [81] inherits from GA but drives the evolution through mutation based on a differential vector rather than relying on crossover. In the recent decade, EAs have enlightened optimization methods for neural architecture search (NAS) [8].

2.3.2. Swarm intelligence (SI)

SI algorithms are inspired by the collective behavior of biological groups, including ants, grey wolves, grasshoppers, etc. [82] Within these groups, each individual has limited capability, but they can jointly accomplish complex tasks through local interactions without any centralized control.

The particle swarm optimization (PSO) algorithm is arguably the most popular SI paradigm [83]. The vanilla PSO imitates the flocking behavior observed in bird communities to address optimization problems [84]. In PSO, each particle represents a potential solution to the problem and is defined by a position and a velocity. The position is initialized randomly, and the velocity starts at zero. A topology is assigned to the swarm to describe the interconnections among particles, where particles connected to a specific particle are considered its neighbors. At each step, a particle's velocity is updated based on the best positions it and its neighbors have found thus far, and the position is updated accordingly. This enables particles to search for the optimum in parallel, sharing the current best position and fitness value with one or more particles to determine their next movements. To prevent the swarm from being trapped in local optima, mutations are introduced by incorporating slight randomness into the update process.

Additionally, inertia PSO (ω-PSO) [85] introduced the inertia weight ω, a positive constant or function, to balance the trade-off between exploration and exploitation. The process of standard ω-PSO is summarized in Algorithm 4 for minimization problems. $\mathbf{x}_i$ and $\mathbf{v}_i$ are d-dimensional vectors representing the position and velocity of particle p_i, respectively. r_1 and r_2 are random variables sampled from the uniform distribution in the range $[0, 1]$. In the algorithm, each particle is the neighbor of all other particles, and thus the topology is a fully connected graph. For alternative topologies, $\mathbf{x}_g^*$ in the update formula of $\mathbf{v}_i$ should be replaced with the best position of p_i's neighbors.

Algorithm 4: Particle-Swarm-Optimization (credit to Houssein et al. [83])

Input: Particle number N, total steps T, learning parameters c_1, c_2, ω, fitness function F

Initialize particles $\mathcal{P} = \{p_i = [\mathbf{x}_i, \mathbf{v}_i]\}_{i=1}^{N}$

`// Local/global best fitness`

Evaluate $\{f_i^* \leftarrow F(\mathbf{x}_i)\}_{i=1}^{N}$, $f_g^* \leftarrow \min_{1 \le i \le N} f_i^*$

`// Local/global best position`

Initialize $\{\mathbf{x}_i^* \leftarrow \mathbf{x}_i\}_{i=1}^{N}, \mathbf{x}_g^*$

for $t = 1$ **to** T **do**

  for $p_1 \in \mathcal{P}$ **to** $p_N \in \mathcal{P}$ **do**

    Calculate fitness value $f_i = F(\mathbf{x}_i)$

    if $f_i \le f_i^*$ **then**

      $f_i^* \leftarrow f_i$

      $\mathbf{x}_i^* \leftarrow \mathbf{x}_i$

  Update $f_g^*, \mathbf{x}_g^*$

  for $p_1 \in \mathcal{P}$ **to** $p_N \in \mathcal{P}$ **do**

    $\mathbf{v}_i \leftarrow \omega \mathbf{v}_i + c_1 r_1 (\mathbf{x}_i^* - \mathbf{x}_i) + c_2 r_2 (\mathbf{x}_g^* - \mathbf{x}_i)$

    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$

  if *error criterion is met* **then**

    break

Output: Optimal position $\mathbf{x}_g^*$

2.4. Summary

Table 1 at the beginning of this section provides a concise overview of classical HPO methods just discussed. GradS methods are suitable only in situations where the objective functions are differentiable and obtaining the gradients of hyperparameters is feasible (e.g., learning rate in neural networks). Since gradient provides only local information, these approaches might converge quickly to a local optimum instead of a global optimum [16, 27]. GS and RS are conceptually simple and are the most basic HPO methods. They can both be readily implemented, and since each hyperparameter configuration evaluation is independent of each other, GS and RS possess the advantage of easier parallelization. In practice, RS is more efficient than GS for large search space and when some hyperparameters have higher importance over others in high-dimensional hyperparameter configuration space [10].

In contrast to the simple search schemes above that are either only exploitative (GradS) or explorative (GS and RS), BO allows the exploration of unknown search space and the exploitation of promising regions. While BO is an efficient strategy for the global optimization of expensive black-box functions, its performance fundamentally hinges on the surrogate model and the acquisition function chosen [13, 86]. Specifically, the quality of BO using GP as the surrogate model, BO-GP, also depends on the choice of the kernel function. Compared to BOs using other surrogates, e.g., RF, TPE, and BNN, BO-GP struggles with large dataset and high dimensions and is only applicable to continuous hyperparameters. Due to the inherently sequential nature of BO algorithms, where the acquisition function guides the selection of subsequent hyperparameter sets based on the current model [14], it is challenging to efficiently parallelize the entire optimization process.

Evolutionary algorithms (EAs, includes GA and ES) and swarm intelligence (SI, includes PSO), owning to their population-based nature and the independence of evaluation for different individuals, support parallelization. Similar to BO, they are capable of both exploration (diversification) and exploitation (intensification). The exploration of GA and ES is primarily driven by crossover (recombination for ES [87]) and mutation, and the exploitation by the selection mechanism; PSO explores by the stochasticity embedded in its update rules and exploits by the convergence of individuals toward the best-known positions. These advantages come with the cost of additional considerations. In the case of GA and ES, this entails managing supplementary hyperparameters like the fitness function and crossover and mutation rates. For PSO, the challenge lies in careful parameter and topology selection, along with proper population initialization to mitigate the risk of premature convergence [83].

3. Acceleration techniques

Approaches addressing HPO can be distilled into two main methodological families: model-free metaheuristic methods and model-based BO methods. Despite the small overheads of meta-guiders, both conventional metaheuristics and BO methods remain computationally resource-intensive. Specifically, metaheuristic methods need to maintain a considerably large population to prevent the collapse of the solution space. On the other hand, BO algorithms suffer from an iterative waiting period due to the time-consuming observation process throughout the sequential search. To overcome these challenges, numerous methods have been proposed to accelerate the search process and optimize the allocation of computational resources.

This section aims to categorize these methods into the following groups: multi-fidelity optimization, bandit-based algorithm, and early stop. These represent prominent lines of research in the ongoing efforts to enhance the efficiency of HPO.

3.1. Multi-fidelity optimization

Multi-fidelity (MF) optimization leverages auxiliary information from various sources to reduce the total evaluation cost. In practice, conventional techniques discard intermediate information and overlook the iterative nature of modern DL algorithms. Strategies that combine information from related tasks to speed up the search efficiency also work. Both approaches revolve around finding a balance between the actual objectives, which are expensive and of high-fidelity, and the auxiliary observations, which are cheap and of low-fidelity.

3.1.1. Multi-fidelity / multi-source

MF-GP-UCB [88] pioneered the formalization of a multi-fidelity bandit algorithm by taking GP to establish the relations between low-fidelity approximations and final performance. Impressively, FABOLAS (fast BO on large datasets) [89] presents to take the subset size of training data as an auxiliary variable, accelerating the search process 10 to 100 times faster than vanilla BOs for large datasets. FABOLAS has also proposed a classical method for modeling the relation via GP using a product kernel. Simultaneously, misoKG (multi-information source optimization with a Knowledge Gradient) [90] describes another generative model that estimates the i-th output by summing up two GPs representing the high-fidelity result and the corresponding discrepancy separately.

Furthermore, methods have emerged to improve multi-fidelity approaches by reinforcing acquisition functions. taKG [91] suggests a trace-aware knowledge-gradient algorithm that is provably convergent. MF-MES [92] incorporates MES [61] to enable a better exploit-explore trade-off for multi-fidelity BO without introducing extra parameters, providing an information-theoretic guarantee.

3.1.2. Multitask / warm start

MTBO (Multitask BO) [93] recommends applying a multitask GP via a Kronecker product kernel for a warm start, avoiding unnecessary re-exploration in familiar search space. MI-SMBO (Meta-learning-based initialization SMBO) [94] proposed a meta-learning-based initialization to warm start BO. Specifically, BO is initialized by well-performing configurations in similar datasets, with a set of meta-features defining the distances between datasets. ABLR (Adaptive Bayesian linear regression) [95] scales MTBO with BNN with a sharing neural network to learn basis features and a Bayesian layer to model the posterior for each output. Recently, WS-CMA-ES (warm starting CMA-ES) [96] has also proposed warm-starting the initialization of CMA-ES to address the conflict between the costly adaptation phase and limited evaluation budget.

3.2. Bandit-based algorithm

Bandit-based algorithms derived from RS have been proven to be compelling in the allocation of limited resources. The successive halving (SHA) algorithm [97] dynamically allocates budgets to top-performing configurations by regularly discarding the least promising half.

A notable extension of SHA is the Hyperband algorithm [98], a multi-armed bandit algorithm designed to terminate poorly performing configurations. Algorithm 5 summarizes the Hyperband process. Compared to SHA, which refrains from allocating resources to underperforming configurations, Hyperband takes a step further by dividing the budget into iterations. This strategy aims to strike a balance between exploration and exploitation, enhancing the algorithm's ability to navigate the search space effectively. Benefiting from the elegant simplicity and flexibility, Hyperband typically outperforms RS and vanilla BO in practice. Another adaptation made to SHA is asynchronous SHA (ASHA) [99], leveraging asynchrony for parallelization. The main idea is to promptly promote configurations to the next rung level, foregoing the necessity to wait for rung completion. While this decision rule may lead to unfavorable configurations being promoted, by the law of large number the impact is expected to diminish as the total number of configurations grows [99].

Algorithm 5: Hyperband (credit to Li et al. [98])

Input: budgets $[b_{\min}, b_{\max}]$, η (default=3)

$s_{\max} \leftarrow \left\lfloor \log_{\eta} \frac{b_{\max}}{b_{\min}} \right\rfloor$

for $s \in \{s_{max}, s_{max} - 1, \cdots, 0\}$ **do**

  $n \leftarrow \left\lceil \frac{s_{\max}+1}{s+1} \cdot \eta^{s} \right\rceil$

  $b_s \leftarrow \eta^{s} \cdot b_{\max}$

  Sample n configurations randomly

  Run SHA with budget b_s

Output: best configuration

However, the convergence of Hyperband is constrained by its randomly drawn strategy, leading to underutilization of known observations. To address this, BOHB (BO and Hyperband) [53] proposed a Hyperband and BO hybrid, replacing the random selection of each Hyperband iteration with a

modified TPE surrogate to guide the search. Notably, BOHB employs a multi-dimensional kernel density estimation (KDE) instead of the hierarchy of one-dimensional KDEs in TPE. BOHB is often regarded as the best previous off-the-shelf optimizer.

Recent developments have seen the emergence of methods building upon Hyperband and BOHB. HyperSTAR [100] adapts the surrogate model to specific tasks and ranks the hyperparameters by estimation in a joint dataset-hyperparameter space. DEHB (DE and Hyperband) [101] combines DE and Hyperband, achieving more robust performance for high-dimensional and combinatorial data than BOHB. The overheads of model-free DE operations remain constant and precede BOHB by up to one order of magnitude. In addition to Hyperband's random sampling, PriorBand [71] includes also prior-based and incumbent-based sampling strategies.

3.3. Early stop

Besides model-free Hyperband, many other stopping criteria are exploited to early terminate poor configurations.

3.3.1. Curve estimation

Modeling the learning curve to allocate resources and stop running individuals dynamically is in vogue. Freeze-thaw BO [102] extends the BO framework with a strong assumption that the training loss curve follows an exponential decay toward some value. A well-designed time kernel gets developed to support this nonstationary prior. However, freeze-thaw BO has been proven ineffective in describing learning curves of deep networks in practice [103].

To enhance representational capacity, learning curve extrapolation (LCE) [103] suggests modeling the curves with a set of parametric model families. Various increasing and saturating functions, each of which may capture certain aspects of curves, are ensembled to describe the entire learning process. To further get rid of manually designed functions, which may introduce undue strong assumptions, Klein et al. [104] recommends using BNN with basis function layers for prediction. Baker et al. [105] incorporates sequential regression models to estimate curves, achieving notable advancements in both NAS and HPO.

3.3.2. Other criteria

Lately, BO-BOS [106] has been proposed to unify the BO and Bayesian optimal stop (BOS) mechanism to eliminate unnecessary queries. A significant difference between BO-BOS and multi-fidelity methods is that BO-BOS determines whether to stop in the training process. BOIL (BO for iterative learning) [107] inherits the advantages of both multi-fidelity BOs and curve estimation techniques by optimizing the numeric score of curves compression instead of averaged final performance. Makarova et al. [108] suggests an automatic termination criterion based on the discrepancy between the actual HPO objective and the BO-optimized target function.

3.4. Summary

This section has introduced three main strategies for speeding up the hyperparameter optimizing process: Multi-fidelity, bandit-based algorithms, and early stopping. Multi-fidelity optimization methods operate by leveraging auxiliary information and exploiting more cost-effective approximations of the expensive objective function. In navigating the trade-off between optimization performance and runtime,

practical implementations often witness the speed improvements outweighing the errors introduced by the approximations [27]. These methods can either actively or adaptively determine the appropriate fidelity (or fidelities), or transfer knowledge from previous experiments or similar tasks. Some literature, e.g., Yang and Shami [24] and Shawi et al. [31], consider also bandit-based algorithms such as SHA [97] and Hyperband [98] introduced in Section 3.2 as a subset of multi-fidelity approaches. While most methods that enhance the efficiency of HPO have been predominantly proposed within the context of BO frameworks, learning curve-based prediction for early termination proposed by Domhan et al. [103] and Baker et al. [105] work on deep neural networks and are agnostic to the hyperparameter optimizer used.

4. Dynamic algorithm configuration

The above strategies have demonstrated their superiority through various acceleration tricks. However, most HPO algorithms only search for fixed configurations, while dynamic or scheduling hyperparameters can be more welcomed in practice. In this section, we broadly convey recent trends of the emerging DAC algorithms covering the following aspects: gradient-based optimizers, population-based algorithms, and reinforcement learning methods. We also explore a few miscellaneous approaches. A succinct summary of these algorithms is presented in Table 3. They are compared from several aspects, including the types of hyperparameters they can handle and their approach to managing hyperparameters and parameters during the training process.

Table 3. A concise overview of representative methods that optimize hyperparameters *on the fly*, i.e., dynamically tune configurations (schedule) in the training process. Column *Coverage* shows the types of hyperparameters suitable for each algorithm. Some algorithms are designed for specific hyperparameters like learning rate (LR), while others can accommodate most continuous hyperparameters. *Exploration* and *Exploitation* columns show how hyperparameters and parameters are treated separately. In the last column, "Keep" means the parameters remain unaltered, and "Overwrite" means former bad parameters are replaced with parameters copied from other well-performing models.

Method	Coverage	Base	Exploration	Exploitation
HD [109]	LR	Hypergradient	Hypergradient descent	Keep
RTHO [38]	Continuous	Hypergradient	Hypergradient descent	Keep
MARTHE [17]	LR	Hypergradient	Hypergradient descent	Keep
FSLA [111]	Continuous	Hypergradient	Hypergradient descent	Keep
PBT [62]	Continuous	PBT	Mutation / Re-sample	Overwrite
PB2 [18]	Continuous	PBT	Time-varying GP-bandit	Overwrite
HPM [112]	Continuous	PBT	Teacher+Hypergradient	Overwrite
PB2-Mix [113]	Mixed	PBT	Mixed GP-bandit	Overwrite
BG-PBT [114]	Mixed	PBT	TR-GP-BO	Overwrite
HOOF [117]	RL parameters	Reinforcement	Off-policy Estimate	Keep
Biedenkapp et al. [19]	Mixed	Reinforcement	Dynamic movement primitives	Keep
AutoLRS [126]	LR	Curve Estimation	Forecasting+GP-bandit	Keep
Multistage QHM [127]	BS+SGD parameters	Momentum	Quasi-hyperbolic momentum	Keep

4.1. Gradient-based optimizers

In Section 2, we have discussed gradient-based search and introduced both reverse-mode and forward-mode algorithms. While reverse-mode is less computationally demanding with a large number of hyperparameters, forward-mode exhibits greater memory efficiency and is suitable for real-time hyperparameter updates.

Baydin et al. [109] derived the hypergradient with respect to the learning rate and updated the learning rate at each iteration. The hypergradient is based on the partial derivative of one-step parameter training, and parameters from the previous time step are irrelevant to the current learning rate. SGD, SGD with Nesterov, and Adam are generalized to online update forms, termed hypergradient descent (HD) variants. HD is considered shortsighted [110]. RTHO (Real-time hyperparameter optimization) [38] modifies the forward-mode hypergradient algorithm in Algorithm 1 to a real-time update form, which is longsighted but too slow to adapt to abrupt changes in the loss surface. MARTHE (Moving Average Real-Time Hyperparameter Estimation) [17] (Algorithm 6) tries to find the balance between HD and RTHO by introducing a discount factor μ, providing globally useful update directions while formal methods fail to cope with loss surface that varies too fast or too slow. Both RTHO and HD can be interpreted as special cases of MARTHE when $\mu = 1$ and $\mu = 0$, respectively. Li et al. [111] unified hypergradient approximation methods including back propagation through time, Neumann series, and conjugate gradient descent under the same framework, and has proposed a fully single loop algorithm (FSLA) [111] for bi-level optimization based on this framework.

Algorithm 6: MARTHE (credit to Donini et al. [17])

Input: Initial hyperparameters $\mathbf{x}$, initial parameters $\mathbf{w}_0$, hyper-learning rate β, discount factor μ, objective function $E\colon \mathbb{R}^d \to \mathbb{R}^+$, weight update dynamics $\Phi_t\colon \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^d$

$\Delta\mathbf{x} \leftarrow 0$

$\mathbf{Z}_0 \leftarrow 0$ // $\mathbf{Z}_t = \frac{\partial \mathbf{w}_t}{\partial \mathbf{x}}$

for $t = 1$ **to** T **do**

  $\mathbf{x} \leftarrow \max\{\mathbf{x} - \beta\Delta\mathbf{x}, 0\}$

  $\mathbf{w}_t \leftarrow \Phi_t(\mathbf{w}_{t-1}, \mathbf{x})$

  $\mathbf{Z}_t \leftarrow \mu \frac{\partial \Phi_t}{\partial \mathbf{w}_{t-1}} \mathbf{Z}_{t-1} + \frac{\partial \Phi_t}{\partial \mathbf{x}}$

  $\Delta\mathbf{x} \leftarrow \nabla E(\mathbf{w}_t)\mathbf{Z}_t$

4.2. Population-based algorithms

Population-based training (PBT) [62] derives from evolution search but updates both weights and hyperparameters of the population of agents in a single training process instead. Proportional agents with poor performances get eliminated regularly to generate new individuals exploited from the best-performing ones. Typically, neural network weights are copied from one of the tops while hyperparameters are randomly resampled from the whole space or slightly mutated from best values.

However, the reliance on metaheuristics for exploration leads to space collapse when the population is small. PB2 [18] proposed to guide the exploration by a time-varying GP with sublinear regret guarantees, which enable it to search with a small computational budget. A novel trick of batch sampling for the acquisition function helps maintain the diversification of the population. HPM (Hyperparameter mutation) [112] regards the agents as students that update their hyperparameters with hypergradient. Meanwhile, an attention-based

teacher model is leveraged to learn a mutation schedule. While PB2-Mix [113] uses a hierarchical approach to allow PB2 to tackle hyperparameters of continuous and categorical types, BG-PBT (Bayesian generational PBT) [114] extends PBT-style methods to consider also ordinal (in addition to continuous and categorical) variables by employing trust-region (TR), GP-based BO.

4.3. Reinforcement learning methods

Reinforcement learning (RL) [115] has gained widespread application in optimizing algorithm configurations [116]. Typically, a controller is used to sample new candidates to get a return from the environment, where an evaluator scores the return to generate a reward. The controller then gets updated based on the received reward and current network states.

HOOF (Hyperparameter Optimization on the Fly) [117] proposed an off-policy method to adapt sensitive RL hyperparameters to changing environments. In HOOF, candidates are regularly generated and estimated by trajectories sampled from the current policy. The policy then gets updated greedily using the values of candidates. A novel generation strategy with importance sampling and a Kullback-Leibler (KL) constraint reduces the computation cost further.

While HOOF is tailored for RL tasks, Biedenkapp et al. [19] formalized learning DAC as a contextual Markov decision process (MDP) where multiple agents sample sequences of parameters across different stages. A global policy is generated according to the distribution of stages, and self-paced learning (SPL) is adopted to evaluate this policy to maximize its reward.

4.4. Remarks

DAC facilitates the real-time application of HPO while training of the model parameters makes progress. The capability of automatically determining algorithm configuration (or schedule) adaptively in an online fashion is particularly advantageous in scenarios where learning dynamics exhibit high non-stationarity, and a fixed configuration (or schedule) for the entire training duration could potentially be suboptimal [62, 118]. Successful DAC, however, may require more computational resources compared to static methods [16].

Given the iterative nature of many AI algorithms [19], especially in the context of RL recognized as a universal modeling framework for sequential decision problems [119], dynamically learning optimal hyperparameter configurations that may vary over time during the training process becomes a natural proposition. AutoRL (Automated RL) [120, 121], a recent area of research, has arisen to address RL algorithms' sensitivity to design choices [122–124] that often leads to laborious and costly manual tuning. AutoRL aims to automate different components of the RL framework, including (PO)MDP modeling, algorithm selection, HPO, and, when DL is used, the architecture search. Considering the nonstationary nature of RL, which adds to the vulnerability of RL algorithms [125], integrating dynamic HPO in the AutoRL pipeline can be beneficial [120, 124].

As DAC is an emerging field, there are approaches optimizing hyperparameter schedules beyond the major classes discussed earlier. AutoLRS (Automatic learning rate scheduler) [126] trains a time-series forecasting model to estimate performance based on observations from the initial steps at each training stage and utilize a GP to explore the search spaces further. Multistage QHM (quasi-hyperbolic momentum) [127] provides a quasi-hyperbolic momentum modification for almost all momentum variants to dynamically tune hyperparameters, including SGD parameters and batch size.

5. Multi-objective hyperparameter optimization

Methods presented in earlier sections consider HPO problems with a single objective, e.g., prediction accuracy or error-based measure. Many real-world challenges, however, demand the simultaneous optimization of multiple, sometimes conflicting, objectives (or criteria). These objectives or criteria may encompass, for instance, training accuracy/error, generalization capability, model complexity, sensitivity, specificity, energy consumption, and inference time [128].

Karl et al. [20] gave the definition of the general MOHPO problem, representing the generalized HPO problem taking into account $m \in \mathbb{N}$ objectives or criteria,

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) = \arg\min_{\mathbf{x} \in \mathcal{X}} (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})), \tag{5.1}$$

where $f_1 \colon \mathcal{X} \to \mathbb{R}, \ldots, f_m \colon \mathcal{X} \to \mathbb{R}$ and $f \colon \mathcal{X} \to \mathbb{R}^m$. A hyperparameter configuration $\mathbf{x} \in \mathcal{X}$ *(Pareto)* dominates another configuration $\mathbf{x}' \in \mathcal{X}$, if and only if $f(\mathbf{x}) \prec f(\mathbf{x}')$, i.e.,

$$\begin{aligned} &\forall\, i \in \{1, \ldots, m\} \colon f_i(\mathbf{x}) \leq f_i(\mathbf{x}') \wedge \\ &\exists\, j \in \{1, \ldots, m\} \colon f_j(\mathbf{x}) \leq f_j(\mathbf{x}'). \end{aligned} \tag{5.2}$$

A configuration $\mathbf{x}^*$ is *non-dominated*, *Pareto optimal*, or *Pareto efficient* if and only if there exists no other configuration $\mathbf{x} \in \mathcal{X}$ that dominates $\mathbf{x}^*$. From Figure 5 illustrating a simple MOO example with two minimizing objectives, it can be seen that in contrast to single-objective optimization, MOO problems lack a single universally optimal solution that satisfies all objectives simultaneously. The set of Pareto optimal solutions is referred to as the *Pareto set* $\mathcal{P}$ [20],

$$\mathcal{P} := \{\mathbf{x} \in \mathcal{X} \mid \nexists\, \mathbf{x}' \in \mathcal{X} \text{ s.t. } \mathbf{x}' \prec \mathbf{x}\}, \tag{5.3}$$

and $f(\mathcal{P})$ is the *Pareto front*. The Pareto set represents solutions that achieve the best trade-off between objectives, where no objective can be made better without making another worse off. The ideal goals of MOO, and thus MOHPO, algorithms are to [129]: (i) find a set of solutions that lies on the Pareto front, and (ii) ensure the solution set found in (i) is diverse enough to represent the entire range of the Pareto front.

Perhaps the simplest approaches to MOHPO are GS (Section 2.1.2) and RS (Section 2.1.3). To adapt single-objective GS and RS to MOHPO problems, given that each point represents a combination of hyperparameters in the hyperparameter space, one can simply evaluate the performance based on multiple objectives at each chosen point and return all non-dominated solutions as the Pareto set. When dealing with high- or large-dimensional search space, GS and RS can be computationally inefficient and lack the ability to exploit the structure of the search space efficiently. They can, however, serve as simple baselines for specialized MOHPO algorithms, e.g., [130, 131].

Below we introduce the canonical approaches tailored for MO(HP)O problems, distinguishing them into three main categories: scalarization, metaheuristic-based, and model-based. For detailed discussions on MOHPO, interested readers are referred to Karl et al. [20] and Morales-Hernández et al. [21].

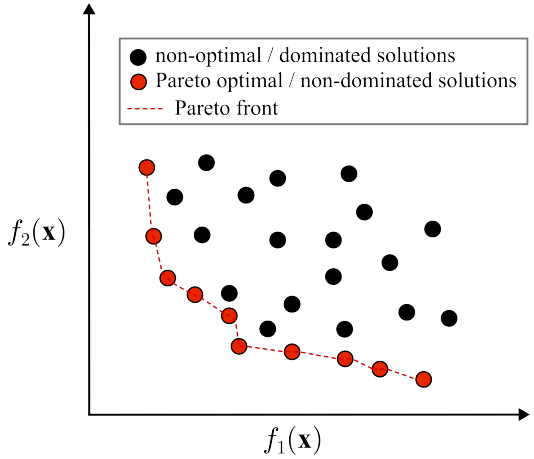

Figure 5. An example of the objective space of a multi-objective optimization problem with two minimizing objectives.

5.1. Scalarization

Scalarization is a straightforward technique to MOHPO. This approach works by transforming multiple objectives into a single objective function, which then single-objective optimizers can be used [132–134]. Two of the most popular scalarization methods are the (i) *weighted sum*, which combines the objectives linearly with each objective $f_i(\mathbf{x})$ multiplied by a user-specified weight λ_i,

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathcal{X}} \sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}), \tag{5.4}$$

and the (ii) ϵ-*constraint* [135], which retains only one objective f_i and turns the rest of the objectives $f_j, \forall j = 1, \ldots, m,\ j \neq i$, into constraints, i.e., restrict them to be within user-specified values ϵ_j,

$$\min_{\mathbf{x}\in\mathcal{X}} f_i(\mathbf{x}) \quad \text{subject to } f_j(\mathbf{x}) \leq \epsilon_j, \forall j = 1, \ldots, m,\ j \neq i. \tag{5.5}$$

Another scalarization function or its variants that has been used in other MOO approaches (such as those introduced in the following sections) is the *(weighted) Chebyshev* or *Tchebycheff function (TCH)* [132],

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathcal{X}} \max_{i=1,\ldots,m} [\lambda_i \mid f_i(\mathbf{x} - z_i^*)], \tag{5.6}$$

where (z_i^*) is usually set to the ideal reference point, i.e. $z_i^* = \min_{\mathbf{x}\in\mathcal{X}} f_i(\mathbf{x})$ for $i = 1, \ldots, m$.

5.2. Multi-objective metaheuristics

NSGA-II (Non-dominated sorting GA II) [136], building on the primitive NSGA algorithm [137], is a widely adopted algorithm for MOO problems. It employs a GA framework and introduces non-dominated sorting to categorize solutions based on their dominance relationships, and crowding distance to sustain a diverse set of solutions on the Pareto front. NSGA-III [138, 139] extends NSGA-II to MOO problems with a larger number of objectives by involving predefined reference points to aid in diversity preservation.

SPEA2 (Strength Pareto EA 2) [140] employs a strength-based approach within its EA framework, using Pareto dominance to guide the selection, favoring individuals that dominate more other individuals. Diversity is achieved through an external, fully-filled archive. SPEA2 and NSGA-II both incorporate elitism in which the best individuals are preserved in each generation.

MOEA/D (Multi-objective EA based on decomposition) [141] takes a decomposition approach, explicitly using scalarization to transform the multi-objective problem into a set of single-objective subproblems. These subproblems are simultaneously optimized, achieving diverse and evenly distributed solutions along the Pareto front through properly selected decomposition methods and weight vectors [141].

To maximize the hypervolume indicator, where the hypervolume (or the $\mathcal{S}$-metric) can be considered as the size covered in the objective space by a set of non-dominated solutions with respect to a user-defined reference point [142], SMS-EMOA ($\mathcal{S}$-metric selection evolutionary MOO algorithm) [143] proposes a steady-state EA with constant population size. SMS-EMOA applies non-dominated sorting from NSGA-II [136] as first ranking criterion and discards individuals with low marginal hypervolume contribution.

Analogous to EAs, PSO has also been used for MOO. NSPSO (Non-dominated sorting PSO) [144] extends single-objective PSO to MOO through information sharing, motivating the entire swarm population progress toward the Pareto front. Non-dominated sorting and crowding distance from NSGA-II [136] are used in NSPSO. Inspired by multi-objective EAs, Coello et al. [145] incorporated the concept of Pareto dominance and a secondary (external) population in its MOO approach via PSO. An adaptive mutation operator facilitates throughout exploration, while a historical record of previously non-dominated solutions promotes convergence toward the Pareto front.

5.3. Multi-objective model-based optimization

Approaches for multi-objective BO can broadly be categorized into scalarization, leveraging Pareto hypervolume (PHV) indicator [142], and information-theoretic methods.

ParEGO (Pareto efficient global optimization) [146] using GP as the surrogate based on the EGO approach [45], approximates the Pareto front with a single objective scalarized via the augmented Tchebycheff function [146], where the parameterization of the weight vector is sampled uniformly per iteration.

SMS-EGO [147] is another approach based on EGO [45]. Instead of using scalarization, SMS-EGO models each objective with a separate surrogate model, with the optimization and selection based on the hypervolume indicator [142]. SMS-EGO enhances the expected hypervolume improvement (EHI/EHVI) [148] acquisition function, initially proposed for surrogate-assisted evolutionary multi-objective search [149], by introducing adaptive search space reduction for improved scalability. As EHVI demands high computational complexity, Daulton et al. [150] proposed qEHVI, a differentiable hypervolume improvement acquisition function. qEHVI allows gradient-based parallel and sequential greedy optimization via the inclusion-exclusion principle.

PESMO (Predictive entropy search) [151] and MESMO (multi-objective maximum entropy search) [152] are information-theoretic methods rooted in information theory. Both PESMO and MESMO build an individual surrogate model for each objective. PESMO's acquisition function utilizes input space entropy, selecting the next evaluation that maximizes information gain about the optimal Pareto set. In contrast, MESMO proposes an output-space-entropy-based acquisition function for candidate selection, evaluating the input that maximizes information gain about the optimal Pareto front.

5.4. Remarks

Evolutionary-based methods, particularly exemplified by tools like NSGA-II [136] (refer to Table 4), have been predominant in MOO, owing to their derivative information-free nature, simplicity for implementation, and flexibility with widespread applicability [129]. As we have seen in Section 5, MOO aims to find a set of solutions, the Pareto set, making EAs a natural choice as they inherently rely on population for optimization.

Apart from the prominent approaches introduced earlier, recently there has been growing interest in encompassing additional considerations when working with MOO problems. For instance, observing that observations are often subject to noise while optimization formulations assume noiseless observations, Daulton et al. [153] proposed NEHVI (noisy expected hypervolume improvement) for noisy multi-objective BOs. Lin et al. [154] and Misitano et al. [155] advocated incorporating decision-maker preferences. Instead of searching for the Pareto front, Malkomes et al. [156] proposed the constraint active search (CAS) formulation to search for diverse solutions satisfying objectives-turned-constraints.

In MOHPO, efforts have also been put toward adapting advanced single-objective HPO techniques to the multi-objective counterpart. For instance, a multi-fidelity method for MOHPO building upon the Hyperband algorithm [98] and scalarization was proposed by Schmucker et al. [131]. Similarly, Chen et al. [157] adapted BOHB [53] to MOHPO, leveraging the integrated information from a multi-fidelity ensemble model effectively in an online fashion. Dushatskiy et al. [158] introduced MO-PBT, the multi-objective version of PBT [62], demonstrating superior performance over MO-ASHA [159], the multi-objective version of ASHA [99].

6. Tools, applications, and further discussion

6.1. Public frameworks and tools

In the past, hyperparameters were usually tuned manually, which is time-consuming. To reduce the bar of using HPO algorithms and enhance their widespread applicability, various libraries and frameworks have been developed. We tabulate the popular frameworks used for HPO, referencing notable works such as [22–24, 31, 32, 160].

Existing HPO frameworks can be categorized in several ways. Based on the distribution of computing resources, they fall into three main categories: centralized, distributed, and cloud-based [31]. When considering optimization methods, they can be distinguished as RS-based, BO-based, metaheuristic-based, and so on. They can also be classified based on the type of problems they address, including HPO, NAS, and AutoML. HPO tools are primarily designed for tuning given hyperparameters; NAS tools are used to automatically discover the optimal architecture or structure of a neural network for a given task; AutoML considers all or most tasks in building ML models, including feature engineering, model construction, and HPO. Overview of existing HPO frameworks and tools is provided in Table 4.

Table 4. Overview of frameworks/tools for HPO. In *Language* column, cpp: C++, go: Golang, jv: Java, jl: Julia, py: Python, R: R language. MO: multi-objective. MF: multi-fidelity. In *Mode* column, cent.: centralized, dist.: distributed. *UI* denotes user interface. In column *MF*, ✓ is provided for tools that offer either their own multi-fidelity option or the original SHA/Hyperband algorithm. Citations are provided for frameworks/tools with publications. Note that the column *Problem* is relatively loosely specified due to challenges in attributing work to a single problem setting. Frameworks/tools that are not open source are marked with an asterisk (*).

Framework/Tool	Year	Problem	Optimization	Lang-uage	MO	MF	Mode	UI	URL
Optuna [161]	2019	HPO	GS, RS, TPE, CMA-ES, NSGA-II, etc.	py	✓	×	dist.	✓	https://github.com/optuna/optuna
Ray Tune [162]	2018	HPO	GS, RS, BO, EA, Optuna, Nevergrad, etc.	py	✓	✓	cent., dist., cloud	✓	https://docs.ray.io/en/latest/tune/index.html
BoTorch [163]	2019	HPO	GP, qEHVI, etc.	py	✓	✓	cent.	×	https://github.com/pytorch/botorch
Bayesian Optimization	2014	HPO	GP	py	×	×	cent.	×	https://github.com/bayesian-optimization/BayesianOptimization
Hyperopt [164]	2015	HPO	RS, TPE, Adaptive TPE	py	×	×	dist.	×	https://hyperopt.github.io/hyperopt/
SMAC3 [50, 165]	2011	HPO	SMAC, ParEGO, mean aggr.	py	✓	✓	cent.	×	https://github.com/automl/SMAC3
DEAP [166]	2012	HPO	GA, PSO, CMA-ES, NSGA-II, NSGA-III, SPEA2, MO-CMA-ES	py	✓	×	dist.	×	https://github.com/DEAP/deap
Nevergrad	2018	HPO	RS, GA, PSO, BO	py	×	×	cent.	×	https://github.com/facebookresearch/nevergrad
AgileRL	2023	HPO	evolutionary	py	×	×	cent., dist.	×	https://github.com/AgileRL/AgileRL
TuRBO [66]	2019	HPO	GP	py	×	×	cent.	×	https://github.com/uber-research/TuRBO
BayesOpt [167]	2014	HPO	GP	cpp	×	×	cent.	×	https://github.com/rmcantin/bayesopt
HyperMapper [168]	2019	HPO	BO, TCH, etc.	cpp, py	✓	×	cent.	×	https://github.com/luinardi/hypermapper
mlr3 [22, 169, 170]	2019	HPO	GS, RS, BO, SHA, Hyperband, CMA-ES, etc.	R	✓	✓	cent.	×	https://github.com/mlr-org/mlr3
jMetal(Py) [171]	2018	HPO	ES, GA, MOEA/D, NSGA-II, OMOPSO, etc.	jv, py	✓	×	cent.	×	https://github.com/jMetal/jMetalPy
Goptuna	2019	HPO	RS, TPE, CMA-ES, ASHA, etc.	go	×	×	cent.	✓	https://github.com/c-bata/goptuna
EvoTorch [172]	2022	HPO	evolutionary	py	✓	×	cent., dist.	×	https://github.com/nnaisense/evotorch
OpenBox [173]	2021	HPO	BO, Hyperband, EHVI, MESMO, etc.	py	✓	✓	cent., dist., cloud	✓	https://github.com/PKU-DAIR/open-box
Dragonfly [174]	2020	HPO	GP	py	✓	✓	cent.	×	https://github.com/dragonfly/dragonfly
DEHB [101]	2021	HPO	DE, Hyperband	py	×	✓	cent.	×	https://github.com/automl/DEHB
Orion	2022	HPO	RS, GS, Hyperband, PBT, BO, EA, etc.	py	×	✓	dist.	×	https://github.com/Epistimio/orion
KerasTuner	2019	HPO	RS, BO, Hyperband	py	×	✓	cent., dist.	×	https://github.com/keras-team/keras-tuner
Syne Tune [175]	2022	HPO	GS, RS, BO, PBT, DEHB, ASHA, NSGA-II, MO-ASHA, etc.	py	✓	✓	cent., dist., cloud	×	https://github.com/awslabs/syne-tune
Katib [176]	2020	HPO, NAS	BO, CMA-ES, Hyperband, PBT, Optuna, Hyperopt, etc.	py	×	✓	cent., dist., cloud	✓	https://github.com/kubeflow/katib
Propulate [177]	2023	HPO	EA	py	×	×	dist.	×	https://github.com/Helmholtz-AI-Energy/propulate
Determined AI	2020	HPO	GS, RS, ASHA, etc.	py	×	×	cent., dist., cloud	✓	https://github.com/determined-ai/determined
Facebook Ax	2019	HPO	GP, BoTorch	py	✓	✓	cent.	×	https://github.com/facebook/Ax

Continued on next page

Framework/Tool	Year	Problem	Optimization	Lang-uage	MO	MF	Mode	UI	URL
pymoo [178]	2018	HPO	GA, DE, CMA-ES, NSGA-II, NSGA-III, MOEA/D, SMS-EMOA, etc.	py	✓	×	cent.	×	https://github.com/anyoptimization/pymoo
Hypernets	2022	HPO, NAS	MCTS, EA, NSGA-II, NSGA-III, MOEA/D, etc.	py	✓	×	cent.	×	https://github.com/DataCanvasIO/Hypernets
Hyperopt.jl	2018	HPO	BO, Hyperband, BOHB, etc.	jl	×	✓	cent.	×	https://github.com/baggepinnen/Hyperopt.jl
MLJTuning	2020	HPO	GS, RS, TPE, PSO, etc.	jl	×	×	cent., dist.	×	https://github.com/JuliaAI/MLJTuning.jl
DeepHyper	2019	HPO, NAS	BO, scalarization	py	✓	✓	cent., dist.	×	https://github.com/deephyper/deephyper
Weights & Biases	2018	HPO	GS, RS, BO, Hyperband	py	×	✓	cent., dist., cloud	✓	https://github.com/wandb/wandb
Polyaxon (HyperTune)	2018	HPO	GS, RS, BO, Hyperband, Hyperopt	py	×	✓	cent., dist., cloud	✓	https://github.com/polyaxon/polyaxon
Mango [179]	2020	HPO	BO	py	×	×	cent., dist.	×	https://github.com/ARM-software/mango
Gradient-Free-Optimizers (Hyperactive)	2020	HPO	GS, RS, PSO, BO, etc.	py	×	×	cent.	×	https://github.com/SimonBlanke/Gradient-Free-Optimizers
shap-hypetune	2021	HPO	GS, RS, BO	py	×	×	cent.	×	https://github.com/cerlymarco/shap-hypetune
NePS [71]	2023	HPO, NAS	BO, Hyperband, πBO, PriorBand	py	×	✓	cent., dist.	×	https://github.com/automl/neps
Scikit-Optimize	2016	HPO	GS, RS, GP	py	×	×	cent.	×	https://github.com/scikit-optimize/scikit-optimize
Talos	2019	HPO	GS,RS, etc.	py	×	×	cent.	×	https://github.com/autonomio/talos
SHERPA [180]	2018	HPO	GS, RS, BO-GP, Hyperband, ASHA, PBT	py	×	✓	cent.	×	https://github.com/sherpa-ai/sherpa
FAR-HO [38]	2017	HPO	ReverseHG, RTHO, ForwardHG	py	×	×	cent.	×	https://github.com/lucfra/FAR-HO
FEDOT [181, 182]	2021	AutoML	Hyperopt, Optuna, etc.	py	✓	×	cent.	×	https://github.com/aimclub/FEDOT
TPOT [183]	2016	AutoML	GA	py	×	×	dist.	×	https://github.com/EpistasisLab/tpot
AutoGL [184]	2021	AutoML	GS, BO, CMA-ES, MO-CMA-ES, etc.	py	✓	×	cent.	×	https://github.com/THUMNLab/AutoGL
Auto-Sklearn [185, 186]	2015	AutoML	SMAC	py	\	×	dist.	×	https://github.com/automl/auto-sklearn
Auto-PyTorch [187]	2020	AutoML	SMAC, Hyperband	py	×	✓	cent.	×	https://github.com/automl/Auto-PyTorch
AutoKeras [188, 189]	2019	AutoML	GP	py	×	×	cent., cloud	×	https://github.com/keras-team/autokeras
AutoGluon [190]	2020	AutoML	RS, BO	py	×	×	cent.	×	https://github.com/autogluon/autogluon
TransmogrifAI	2018	AutoML	GS, RS, BO	Scala	×	×	cent.	×	https://github.com/salesforce/TransmogrifAI
EvalML	2019	AutoML	GS, RS, BO	py	×	×	cent.	×	https://github.com/alteryx/evalml
MLJAR AutoML	2021	AutoML	RS, TPE (Optuna)	py	×	×	cent.	✓	https://github.com/mljar/mljar-supervised
Microsoft NNI	2021	AutoML	GS, RS, EA, Hyperband, PBT, BO, etc.	py	×	✓	cent., dist., cloud	✓	https://github.com/microsoft/nni
Microsoft FLAML [191]	2021	AutoML	GS, RS, Optuna	py, .NET	✓	×	cent., dist, cloud	×	https://github.com/microsoft/FLAML
Microsoft Archai	2020	NAS	RS, SHA, evolution, etc.	py	✓	×	cent., dist., cloud	×	https://github.com/microsoft/archai
*Microsoft AzureML AutoML [192]	2018	AutoML	GS, RS, BO	py	×	×	cent., cloud	✓	https://azure.microsoft.com/en-us/products/machine-learning/automatedml/
*Oracle AutoML [193]	2020	AutoML	gradient-based	py	×	×	cent., cloud	✓	https://docs.oracle.com/en/database/oracle/machine-learning/
*Google Vizier [194]	2017	HPO	RS, GS, BO	cpp, go, py	×	×	cloud	✓	https://cloud.google.com/vertex-ai
*Amazon SageMaker [195]	2017	AutoML	GS, Hyperband, RS, BO	py, R	✓	✓	cloud	✓	https://aws.amazon.com/machine-learning

6.2. HPO applications

HPO has emerged as a pivotal component of AutoML, finding extensive applications across diverse domains and tasks. For instance, it is applied for software sensor design [196], electroencephalography (EEG) data decoding [197], smart grid [198], P systems optimization [199], and healthcare [200].

Data-driven models have become integral in the realm of software sensor design for vehicles, where the performance is notably influenced by hyperparameters. A specific study [196] focused on designing a roll angle estimator based on an artificial neural network (ANN), employing techniques including GS, Hyperband, BO, and GA. The results emphasized the significant impact of search space size on performance, with knowledge-based methods outperforming their counterparts.

In the domain of EEG data decoding for brain-computer interfaces (BCIs), HPO methods play a crucial role in optimizing DL models for the classification of EEG recordings. In Stober et al. [201], hyperparameters associated with the convolutional neural networks (CNN) used to classify EEG recordings of rhythm perception were optimized with BO. In another work by Drouin-Picaro and Falk [202], GS for HPO was used in the classification of EEG signals corresponding to natural saccade with CNN and multilayer perceptron (MLP). Coonet et al. [197] suggested that more experiments should be conducted to investigate whether the generalization of HPOs across subjects is feasible.

In a smart grid, the consumption of electricity varies from time to time, which burdens the electricity systems. Prediction with the ML models can help achieve the balance between demand and supply, and manage the energy efficiently [198]. ML models such as support vector machine (SVM), ANN, and BNN are commonly used for this problem. To achieve a good predictive performance, tuning the hyperparameters is of vital importance. For example, Zhou et al. [203] proposed a system based on autoencoders and tuned the hyperparameters using GA. Additionally, AutoML frameworks are likely to be applied to this field in the future search.

In recent attempts to create bridges between adaptive P systems and ML paradigms, the optimization of one of the hyperparameters, which is the type of the spiking neuron used in spiking neural (SN) P systems, prior to the training of P systems, can potentially benefit from HPO [199].

Biomedical applications, encompassing healthcare, biomedical research, and big biomedical data, have also witnessed the advantages of integrating HPO methods in the ML pipelines [200]. Hospitals can deploy ML models for health outcome improvement, healthcare cost reduction, and clinical research advancement [204]. In fact, certain frameworks are developed exclusively for healthcare. One example is AutoPrognosis [205], which uses BO to optimize models and hyperparameters for clinical prognosis. HPO also has non-negligible impact when working with medical images using AI techniques [206]. For instance, BO was used to optimize the network architecture of Nishio et al. [207] for lung segmentation using chest X-ray (CXR) images with severe abnormalities. BO with TPE as the surrogate was also used in Abdellatif et al. [208] on the Improved Weighted RF for predicting the presence of cardiovascular disease and patient survival. Experiments by Belete and Huchaiah [209] employing GS-based HPO on eight ML models have shown improvement over statistical way of optimization in predicting HIV/AIDS test results. Similarly, Nematzadeh et al. [210] demonstrated that by employing dedicated HPO algorithms for ML models in the classification and regression of biomedical datasets, training process and model performance improved when compared to using blindly chosen hyperparameters. Despite these advancements, challenges persist. This includes the limited interpretability of ML models contributing to user skepticism regarding predictions, efficiency concerns as current methods struggle to swiftly identify a near-optimal hyperparameter configuration, and the protracted trial-and-error

processes presenting an impracticality given the demand for approaching thousands of predictive modeling problems in personalized medicine. In light of these challenges, DAC algorithms emerge as a promising avenue.

6.3. Further discussion

In this paper, we have discussed four kinds of HPO methods that are closely connected. Classical techniques, with their foundational concepts proposed over a decade ago, have evolved alongside advancements in ML models. On one hand, these techniques are employed to tune hyperparameters, enhancing the performance of ML models. On the other hand, ML models contribute to the expansion and evolution of these techniques. ML models such as RF and BNNs can serve as surrogate models of BO. As ML models experience longer training times, there is a growing demand for more efficient HPO techniques. Researchers have responded by designing various frameworks to carefully manage computing resources, while retaining the core ideas. DAC algorithms, for instance, aim to utilize resources efficiently within a single training cycle to achieve near-optimal performance. These algorithms often involve structural modifications to classical techniques, such as gradient-based and population-based methods, allowing for on-the-fly updates to hyperparameters. However, a common challenge faced by these methods is the inherent trade-off between performance and efficiency. Driven by real-world situations where practical considerations involve additional metrics or criteria, framing the HPO problem as an MOHPO problem is a more pragmatic approach. MOHPO, being an extension of the broader MOO, leverages existing MOO methodologies, with recent endeavors adapting single-objective HPO algorithms for multi-objective cases.

HPO algorithms encounter several challenges and issues. First, the diversity of hyperparameter types and complex search spaces can impact the usability of algorithms, often requiring disharmonious extensions that may affect performance and theoretical properties. Second, the high-dimensionality of hyperparameters makes convergence challenging due to the vast number of samples required to explore the search space. Identifying the few influential hyperparameters is a daunting task. Third, handling the intricate relationships among hyperparameters poses difficulty, as the value of one hyperparameter may significantly affect another, while others remain independent. Capturing these relations can reduce the complexity of the search space. Additionally, scalability is a concern, with standard BO-GP having a complexity of $O(n^3)$, which becomes impractical with a large number of samples. Furthermore, the issue of transferring hyperparameters from task to task or leveraging knowledge from previous search results remains a challenge. Most algorithms treat different tasks independently, even when sharing the same datasets or models, leading to inefficiencies. Lastly, the optimal hyperparameters may change as datasets grow in size. Many existing algorithms require a search from scratch, which is not conducive to the dynamic nature of big data.

In this paper, we have only reviewed approaches to general HPO, acceleration techniques, and algorithms for the emerging DAC and MOHPO. It is noteworthy that various efforts explore HPO from distinct perspectives, warranting attention. For instance, work looking at constrained HPO [211–214] or constrained BO [215–217] and investigations on robustness [53, 180, 218–220]. Explorations of HPO with meta-learning and AutoML [221–224] also offer valuable insights into related areas.

7. Conclusion and outlook

Complex computing systems, exemplified by modern ML pipelines, have found diverse applications across various domains. In the pursuit of more effective and efficient deployment of these pipelines, researchers have increasingly focused on the performance and efficiency of HPO algorithms – a focal point of this paper. We begin by providing a broad overview of HPO, delving notably into strategies tailored for DL algorithms. Subsequently, we explore methods to accelerate optimization procedures. This is followed by comprehensive and systematic reviews of the emerging DAC and MOHPO, offering insights for future research. We also present a comparative analysis of existing HPO tools and their applications across different domains, laying the groundwork for potential future research endeavors.

The potential applications of HPO span diverse areas, presenting a promising landscape for exploration and advancement. From the ML application perspective, here we discuss important areas that underscore the impact of HPO. One key domain is NAS, wherein HPO manifests as a task within a discrete search space. First, NAS can be regarded as an HPO task on discrete search space, and recent works have explored various search methods, including RL [225], EA [226], BO [227], and gradient-based methods [228–230]. Noteworthy applications of NAS extend to complex vision tasks such as object detection [231, 232] and segmentation [233, 234]. Recent work by Wang et al. [235] introduces a NAS framework based on the "mergeNAS" technique [229], allowing for searches within any custom search spaces across a spectrum of vision tasks.

In addition to ML, the fusion of combinatorial optimization [236, 237] with HPO within the realm of applied mathematics offers a platform for further exploration. This includes but is not limited to, tackling challenges like the traveling salesman problem [238], vehicle routing problem [239], mix-integer programming [240], boolean satisfiability [241], portfolio optimization [242], graph matching, either through ML approaches [243] or conventional methods [244], graph clustering [245], and graph edit distance computing [246].

Despite HPO's long history spanning over half a century, numerous promising areas warrant continued efforts. We envision that this comprehensive survey will not only equip researchers with diverse backgrounds to understand the current state and evolution of HPO, facilitating its seamless integration into future work, but also provide practical insights for practitioners in ML and HPO-related domains.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was partly supported by National Natural Science Foundation of China (623B1009, 62073333), and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102). The authors are also thankful for the anonymous reviewers for their valuable suggestions and comments to help improve this survey in all aspects.

Conflict of interest

Junchi Yan is an editorial board member for Mathematical Biosciences and Engineering and was not involved in the editorial review or the decision to publish this article. All authors declare that there are no competing interests.

References

1. E. Alpaydin, *Introduction to Machine Learning*, MIT press, Cambridge, 2020.
2. A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Commun. ACM*, **60** (2012), 84–90. https://doi.org/10.1145/3065386
3. A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, et al., Dermatologist-level classification of skin cancer with deep neural networks, *Nature*, **542** (2017), 115–118. https://doi.org/10.1038/nature21056
4. J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, (2019), 4171–4186. https://doi.org/10.18653/v1/n19-1423
5. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, et al., Human-level control through deep reinforcement learning, *Nature*, **518** (2015), 529–533. https://doi.org/10.1038/nature14236
6. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner et al., An image is worth 16x16 words: Transformers for image recognition at scale, in *International Conference on Learning Representations*, 2021.
7. C. C. Chiu, C. Raffel, Monotonic chunkwise attention, in *International Conference on Learning Representations*, 2018.
8. X. He, K. Zhao, X. Chu, AutoML: survey of the state-of-the-art, *Knowl.-Based Syst.*, **212** (2021), 106622. https://doi.org/10.1016/j.knosys.2020.106622
9. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, preprint, arXiv:1412.6980.
10. J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, *J. Mach. Learn. Res.*, **13** (2012), 281–305.
11. K. Hussain, M. N. Mohd Salleh, S. Cheng, Y. Shi, Metaheuristic research: a comprehensive survey, *Artif. Intell. Rev.*, **52** (2018), 2191–2233. https://doi.org/10.1007/s10462-017-9605-z
12. I. Boussaïd, J. Lepagnot, P. Siarry, Survey on optimization metaheuristics, *Inf. Sci.*, **237** (2013), 82–117. https://doi.org/10.1016/j.ins.2013.02.041
13. J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, *Adv. Neural Inform. Process. Syst.*, **25** (2012).

14. B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, N. de Freitas, Taking the human out of the loop: A review of bayesian optimization, *Proc. IEEE*, **104** (2016), 148–175. https://doi.org/10.1109/jproc.2015.2494218

15. H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once-for-all: Train one network and specialize it for efficient deployment, in *International Conference on Learning Representations*, 2020.

16. S. Adriaensen, A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, et al., Automated dynamic algorithm configuration, *J. Artif. Intell. Res.*, **75** (2022), 1633–1699. https://doi.org/10.1613/jair.1.13922

17. M. Donini, L. Franceschi, O. Majumder, M. Pontil, P. Frasconi, Marthe: scheduling the learning rate via online hypergradients, in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, (2021), 2119–2125. https://doi.org/10.24963/ijcai.2020/293

18. J. Parker-Holder, V. Nguyen, S. J. Roberts, Provably efficient online hyperparameter optimization with population-based bandits, *Adv. Neural Inform. Process. Syst.*, **33** (2020), 17200–17211.

19. A. Biedenkapp, H. F. Bozkurt, T. Eimer, F. Hutter, M. T. Lindauer, Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework, in *the 24th European Conference on Artificial Intelligence*, (2020), 427–434. https://doi.org/10.3233/FAIA200122

20. F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, et al., Multi-objective hyperparameter optimization in machine learning–An overview, in *ACM Transactions on Evolutionary Learning and Optimization*, **3** (2023), 1–50. https://doi.org/10.1145/3610536

21. A. Morales-Hernández, I. Van Nieuwenhuyse, S. Rojas Gonzalez, A survey on multi-objective hyperparameter optimization algorithms for machine learning, *Artif. Intell. Rev.*, **56** (2023), 8043–8093. https://doi.org/10.1007/s10462-022-10359-2

22. B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, et al., Hyperparameter optimization: Foundations, algorithms, best practices and open challenges, *WIRES Data Min. Knowl.*, **13** (2023). https://doi.org/10.1002/widm.1484

23. T. Yu, H. Zhu, Hyper-parameter optimization: A review of algorithms and applications, preprint, arXiv:2003.05689.

24. L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, *Neurocomputing*, **415** (2020), 295–316. https://doi.org/10.1016/j.neucom.2020.07.061

25. R. Mohakud, R. Dash, Survey on hyperparameter optimization using nature-inspired algorithm of deep convolution neural network, in *Intelligent and Cloud Computing*, (2020), 737–744. https://doi.org/10.1007/978-981-15-5971-6_77

26. N. Del Buono, F. Esposito, L. Selicato, Methods for hyperparameters optimization in learning approaches: An overview, in *Machine Learning, Optimization, and Data Science* , (2020), 100–112. https://doi.org/10.1007/978-3-030-64583-0_11

27. M. Feurer, F. Hutter, Hyperparameter Optimization, in *Automated Machine Learning*, (2019), 3–33. https://doi.org/10.1007/978-3-030-05318-5_1

28. X. Wang, Y. Jin, S. Schmitt, M. Olhofer, Recent Advances in Bayesian Optimization, *ACM Comput. Surv.*, **55** (2023), 1–36. https://doi.org/10.1145/3582078

29. P. I. Frazier, A tutorial on Bayesian optimization, preprint, arXiv:1807.02811.

30. E. Brochu, V. M. Cora, N. De Freitas, A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, preprint, arXiv:1012.2599.

31. R. E. Shawi, M. Maher, S. Sakr, Automated machine learning: State-of-the-art and open challenges, preprint, arXiv:1906.02287.

32. M. A. Zöller, M. F. Huber, Benchmark and survey of automated machine learning frameworks, *J. Artif. Intell. Res.*, **70** (2021), 409–472. https://doi.org/10.1613/jair.1.11854

33. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, et al., Backpropagation applied to handwritten zip code recognition, *Neural Comput.*, **1** (1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

34. Y. Bengio, Gradient-based optimization of hyperparameters, *Neural Comput.*, **12** (2000), 1889–1900. https://doi.org/10.1162/089976600300015187

35. J. Domke, Generic methods for optimization-based modeling, in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, (2012), 318–326.

36. D. Maclaurin, D. Duvenaud, R. Adams, Gradient-based hyperparameter optimization through reversible learning, in *Proceedings of the 32nd International Conference on Machine Learning*, (2015), 2113–2122.

37. F. Pedregosa, Hyperparameter optimization with approximate gradient, in *Proceedings of The 33rd International Conference on Machine Learning*, (2016), 737–746.

38. L. Franceschi, M. Donini, P. Frasconi, M. Pontil, Forward and reverse gradient-based hyperparameter optimization, in *Proceedings of the 34th International Conference on Machine Learning*, (2017), 1165–1173.

39. J. Lorraine, P. Vicol, D. Duvenaud, Optimizing millions of hyperparameters by implicit differentiation, in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, (2020), 1540–1552.

40. C. W. Hsu, C. C. Chang, C. J. Lin, *A Practical Guide to Support Vector Classification*, 2003. Available from: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

41. C. Audet, J. E. Dennis, Mesh adaptive direct search algorithms for constrained optimization, *SIAM J. Optim.*, **17** (2006), 188–217. https://doi.org/10.1137/040603371

42. G. E. Dahl, T. N. Sainath, G. E. Hinton, Improving deep neural networks for lvcsr using rectified linear units and dropout, in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, (2013), 8609–8613. https://doi.org/10.1109/ICASSP.2013.6639346

43. Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, et al., Bayesian optimization in alphago, preprint, arXiv:1812.06855.

44. C. E. Rasmussen, C. K. I. Williams, *Gaussian Processes for Machine Learning*, The MIT Press, Cambridge, 2005. https://doi.org/10.7551/mitpress/3206.001.0001

45. D. R. Jones, M. Schonlau, W. J. Welch, Efficient global optimization of expensive black-box functions, *J. Glob. Optim.*, **13** (1998), 455–492. https://doi.org/10.1023/A:1008306431147

46. K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, M. A. Osborne, Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces, preprint, arXiv:1409.4011.

47. E. Snelson, Z. Ghahramani, Sparse gaussian processes using pseudo-inputs, *Adv. Neural Inform. Process. Syst.*, **18** (2006), 1259–1266.

48. C. Oh, E. Gavves, M. Welling, Bock: Bayesian optimization with cylindrical kernels, in *Proceedings of the 35th International Conference on Machine Learning*, (2018), 3868–3877.

49. K. Kandasamy, J. Schneider, B. Póczos, High dimensional bayesian optimisation and bandits via additive models, in *Proceedings of the 32nd International Conference on Machine Learning*, (2015), 295–304.

50. F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in *Learning and Intelligent Optimization*, Springer, (2011), 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

51. J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, *Adv. Neural Inform. Process. Syst.*, **24** (2011), 2546–2554.

52. K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, et al., Towards an empirical foundation for assessing bayesian optimization of hyperparameters, in *NIPS workshop on Bayesian Optimization in Theory and Practice*, (2013).

53. S. Falkner, A. Klein, F. Hutter, Bohb: Robust and efficient hyperparameter optimization at scale, in *Bohb: Robust and efficient hyperparameter optimization at scale*, (2018), 1437–1446.

54. E. Goan, C. Fookes, Bayesian neural networks: An introduction and survey, in *Case Studies in Applied Bayesian Data Science*, Springer, (2020), 45–87. https://doi.org/10.1007/978-3-030-42553-1_3

55. J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, et al., Scalable bayesian optimization using deep neural networks, in *Proceedings of the 32nd International Conference on Machine Learning*, **37** (2015), 2171–2180.

56. J. T. Springenberg, A. Klein, S. Falkner, F. Hutter, Bayesian optimization with robust bayesian neural networks, *Adv. Neural Inf. Process. Syst.*, **29** (2016), 4134–4142.

57. T. Chen, E. B. Fox, C. Guestrin, Stochastic gradient hamiltonian monte carlo, in *Proceedings of the 31st International Conference on Machine Learning*, **32** (2014), 1683–1691.

58. N. Srinivas, A. Krause, S. M. Kakade and M. W. Seeger, Gaussian process optimization in the bandit setting: No regret and experimental design, in *Proceedings of the 27th International Conference on Machine Learning. Omnipress*, (2010), 1015–1022.

59. P. Hennig, C. J. Schuler, Entropy search for information-efficient global optimization, *J. Mach. Learn. Res. (JMLR)*, **13** (2012), 1809–1837.

60. J. M. Hernández-Lobato, M. W. Hoffman and Z. Ghahramani, Predictive entropy search for efficient global optimization of black-box functions, *Adv. Neural Inform. Process. Syst.*, **27** (2014), 918–926.

61. Z. Wang, S. Jegelka, Max-value entropy search for efficient bayesian optimization, in *Proceedings of the 34th International Conference on Machine Learning*, (2017), 3627–3635.

62. M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, et al., Population based training of neural networks, preprint, arXiv:1711.09846.

63. N. A. Vien, H. Zimmermann, M. Toussaint, Bayesian functional optimization, in *Proceedings of the AAAI Conference on Artificial Intelligence*, (2018). https://doi.org/10.1609/aaai.v32i1.11830

64. P. F. Jian Wu, Practical two-step lookahead bayesian optimization, *Adv. Neural Inform. Process. Syst.*, **32** (2019), 9813–9823.

65. J. Kirschner, M. Mutný, N. Hiller, R. Ischebeck, A. Krause, Adaptive and safe bayesian optimization in high dimensions via one-dimensional subspaces, in *Proceedings of the 36th International Conference on Machine Learning*, (2019), 3429–3438.

66. D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, M. Poloczek, Scalable global optimization via local Bayesian optimization, *Adv. Neural Inf. Process. Syst.*, **32** (2019), 5497–5508.

67. V. Nguyen, M. A. Osborne, Knowing the what but not the where in bayesian optimization, *Proceedings of the 37th International Conference on Machine Learning*, (2020), 7317–7326.

68. E. A. Daxberger, A. Makarova, M. Turchetta, A. Krause, Mixed-variable bayesian optimization, in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, (2020), 2633–2639. https://doi.org/10.24963/ijcai.2020/365

69. A. Souza, L. Nardi, L. B. Oliveira, K. Olukotun, M. Lindauer, F. Hutter, Bayesian optimization with a prior for the optimum, in *Machine Learning and Knowledge Discovery in Databases. Research Track*, Springer, (2021), 265–296. https://doi.org/10.1007/978-3-030-86523-8_17

70. C. Hvarfner, D. Stoll, A. Souza, L. Nardi, M. Lindauer, F. Hutter, πBO: Augmenting acquisition functions with user beliefs for bayesian optimization, in *International Conference on Learning Representations*, 2022.

71. N. Mallik, E. Bergman, C. Hvarfner, D. Stoll, M. Janowski, M. Lindauer, et al. Priorband: Practical hyperparameter optimization in the age of deep learning, *Adv. Neural Inform. Process. Syst.*, (2024).

72. S. Katoch, S. S. Chauhan, V. Kumar, A review on genetic algorithm: past, present, and future, *Multimed. Tools Appl.*, **80** (2021), 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

73. C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*, Springer, New York, 2007. https://doi.org/10.1007/978-0-387-36797-2

74. J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, The MIT Press, Cambridge, 1992. https://doi.org/10.7551/mitpress/1090.001.0001

75. T. Blickle, L. Thiele, A comparison of selection schemes used in evolutionary algorithms, *Evol. Comput.*, **4** (1996), 361–394. https://doi.org/10.1162/evco.1996.4.4.361

76. T. Bäck, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*, Oxford University Press, Oxford, 1996. https://doi.org/10.1093/oso/9780195099713.001.0001

77. I. Rechenberg, *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, PhD thesis, Technische Universität, Fakultät für Maschinenwissenschaft, 1970.

78. H. P. Schwefel, G. Rudolph, Contemporary evolution strategies, in *Advances in Artificial Life*, Springer, (1995), 891–907. https://doi.org/10.1007/3-540-59496-5_351

79. R. Li, M. T. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, et al., Mixed integer evolution strategies for parameter optimization, *Evol. Comput.*, **21** (2013), 29–64. https://doi.org/10.1162/evco_a_00059

80. N. Hansen, A. Ostermeier, A. Gawelczyk, On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation, in *Proceedings of the Sixth International Conference on Genetic Algorithms*, (1995), 57–64.

81. R. Storn, K. V. Price, Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces, *J. Glob. Optim.*, **11** (1997), 341–359. https://doi.org/10.1023/A:1008202821328

82. S. Saremi, S. M. Mirjalili, A. Lewis, Grasshopper optimisation algorithm: Theory and application, *Adv. Eng. Softw.*, **105** (2017), 30–47. https://doi.org/10.1016/j.advengsoft.2017.01.004

83. E. H. Houssein, A. G. Gad, K. Hussain, P. N. Suganthan, Major advances in particle swarm optimization: Theory, analysis, and application, *Swarm Evol. Comput.*, **63** (2021), 100868. https://doi.org/10.1016/j.swevo.2021.100868

84. J. Kennedy, R. Eberhart, Particle swarm optimization, in *Proceedings of ICNN'95 - International Conference on Neural Networks*, (1995), 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

85. Y. Shi, R. Eberhart, A modified particle swarm optimizer, in *1998 IEEE International Conference on Evolutionary Computation Proceedings*, (1998), 69–73. https://doi.org/10.1109/ICEC.1998.699146

86. R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, et al., Bayesian optimization is superior to random search for machine learning hyperparameter tuning: analysis of the black-box optimization challenge 2020, in *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, (2021), 3–26.

87. H. G. Beyer, H. P. Schwefel, Evolution strategies–a comprehensive introduction, *Nat. Comput.*, **1** (2002), 3–52. https://doi.org/10.1023/A:1015059928466

88. K. Kandasamy, G. Dasarathy, J. B. Oliva, J. G. Schneider. B. Póczos, Gaussian process bandit optimisation with multi-fidelity evaluations, *Adv. Neural Inform. Process. Syst.*, **29** (2016).

89. A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter, Fast bayesian optimization of machine learning hyperparameters on large datasets, in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, (2017), 528–536.

90. M. Poloczek, J. Wang, P. Frazier, Multi-information source optimization, *Multi-information source optimization*, **30** (2017), 4288–4298.

91. J. Wu, S. Toscano-Palmerin, P. I. Frazier, A. G. Wilson, Practical multi-fidelity bayesian optimization for hyperparameter tuning, in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, (2020), 788–798.

92. S. Takeno, H. Fukuoka, Y. Tsukada, T. Koyama, M. Shiga, I. Takeuchi, et al., Multi-fidelity bayesian optimization with max-value entropy search and its parallelization, in *Proceedings of the 37th International Conference on Machine Learning*, (2020), 9334–9345.

93. K. Swersky, J. Snoek, R. P. Adams, Multi-task bayesian optimization, *Adv. Neural Inform. Process. Syst.*, **26** (2013).

94. M. Feurer, J. T. Springenberg, F. Hutter, Initializing bayesian hyperparameter optimization via meta-learning, *AAAI Conf. Artif. Intell.*, **29** (2015), 1128–1135. https://doi.org/10.1609/aaai.v29i1.9354

95. V. Perrone, R. Jenatton, M. W. Seeger, C. Archambeau, Scalable hyperparameter transfer learning, *Adv. Neural Inform. Process. Syst.*, **31** (2018).

96. M. Nomura, S. Watanabe, Y. Akimoto, Y. Ozaki, M. Onishi, Warm starting CMA-ES for hyperparameter optimization, *AAAI Conf. Artif. Intell.*, **35** (2021), 9188–9196. https://doi.org/10.1609/aaai.v35i10.17109

97. K. G. Jamieson, A. S. Talwalkar, Non-stochastic best arm identification and hyperparameter optimization, in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, (2016), 240–248.

98. L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, A. S. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, *J. Mach. Learn. Res. (JMLR)*, **18** (2018), 1–52.

99. L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur, M. Hardt, et al., A system for massively parallel hyperparameter tuning, *Proc. Mach. Learn. Syst.*, (2020), 230–246.

100. G. Mittal, C. Liu, N. Karianakis, V. Fragoso, M. Chen, Y. R. Fu, Hyperstar: Task-aware hyperparameters for deep networks, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (2020), 8733–8742. https://doi.org/10.1109/cvpr42600.2020.00876

101. N. H. Awad, N. Mallik, F. Hutter, DEHB: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization, in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, (2021), 2147–2153. https://doi.org/10.24963/ijcai.2021/296

102. K. Swersky, J. Snoek, R. P. Adams, Freeze-thaw bayesian optimization, preprint, arXiv:1406.3896.

103. T. Domhan, J. T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in *Twenty-fourth International Joint Conference on Artificial Intelligence*, (2015), 3460–3468.

104. A. Klein, S. Falkner, J. T. Springenberg, F. Hutter, Learning curve prediction with bayesian neural networks, in *International Conference on Learning Representations*, 2017.

105. B. Baker, O. Gupta, R. Raskar, N. Naik, Accelerating neural architecture search using performance prediction, preprint, arXiv:1705.10823.

106. Z. Dai, H. Yu, K. H. Low, P. Jaillet, Bayesian optimization meets bayesian optimal stopping, in *Proceedings of the 36th International Conference on Machine Learning*, (2019), 1496–1506.

107. V. Nguyen, S. Schulze, M. Osborne, Bayesian optimization for iterative learning, *Adv. Neural Inform. Process. Syst.*, **33** (2020), 9361–9371.

108. A. Makarova, H. Shen, V. Perrone, A. Klein, J. B. Faddoul, A. Krause, et al., Automatic termination for hyperparameter optimization, in *Proceedings of the First International Conference on Automated Machine Learning*, 2022.

109. A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, F. Wood, Online learning rate adaptation with hypergradient descent, in *International Conference on Learning Representations*, 2018.

110. Y. Wu, M. Ren, R. Liao, R. Grosse, Understanding short-horizon bias in stochastic meta-optimization, in *International Conference on Learning Representations*, 2018.

111. J. Li, B. Gu, H. Huang, A fully single loop algorithm for bilevel optimization without hessian inverse, in *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, (2022), 7426–7434. https://doi.org/10.1609/aaai.v36i7.20706

112. Z. Tao, Y. Li, B. Ding, C. Zhang, J. Zhou, Y. R. Fu, Learning to mutate with hypergradient guided population, *Adv. Neural Inform. Process. Syst.*, **33** (2020), 17641–17651.

113. J. Parker-Holder, V. Nguyen, S. Desai, S. J. Roberts, Tuning mixed input hyperparameters on the fly for efficient population based autorl, *Adv. Neural Inform. Process. Syst.*, **34** (2021).

114. X. Wan, C. Lu, J. Parker-Holder, P. J. Ball, V. Nguyen, B. Ru, et al., Bayesian generational population-based training, in *Proceedings of the First International Conference on Automated Machine Learning*, (2022), 1–27.

115. R. S. Sutton, A. G. Barto, *Reinforcement learning: An introduction*, 2nd edition, MIT press, Cambridge, 2018.

116. H. S. Jomaa, J. Grabocka, L. Schmidt-Thieme, Hyp-rl: Hyperparameter optimization by reinforcement learning, preprint, arXiv:1906.11527.

117. S. Paul, V. Kurin, S. Whiteson, Fast efficient hyperparameter tuning for policy gradient methods, in *Advances in Neural Information Processing Systems*, **32** (2019), 4616–4626.

118. B. Doerr, C. Doerr, Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices, in *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Springer, Cham, (2020), 271–321. https://doi.org/10.1007/978-3-030-29414-4_6

119. W. B. Powell, *Reinforcement Learning and Stochastic Optimization: A unified framework for sequential decisions*, John Wiley & Sons, Hoboken, 2022. https://doi.org/10.1002/9781119815068

120. J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, et al., Automated reinforcement learning (AutoRL): a survey and open problems, *J. Artif. Intell. Res.*, **74** (2022), 517–568. http://doi.org/10.1613/jair.1.13596

121. R. R. Afshar, Y. Zhang, J. Vanschoren and U. Kaymak, Automated reinforcement learning: An overview, preprint, arXiv:2201.05000.

122. L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, et al., Implementation matters in deep RL: A case study on PPO and TRPO, in *International Conference on Learning Representations*, 2020.

123. M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marninier, et al., What matters for on-policy deep actor-critic methods? a large-scale study, in *International Conference on Learning Representations*, 2021.

124. B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, et al., On the importance of hyperparameter optimization for model-based reinforcement learning, in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, (2021), 4015–4023.

125. M. Igl, G. Farquhar, J. Luketina, W. Boehmer, S. Whiteson, Transient non-stationarity and generalisation in deep reinforcement learning, in *International Conference on Learning Representations*, 2021.

126. Y. Jin, T. Zhou, L. Zhao, Y. Zhu, C. Guo, M. Canini, et al., AutoLRS: Automatic learning-rate schedule by bayesian optimization on the fly, in *International Conference on Learning Representations*, 2020.

127. J. Sun, Y. Yang, G. Xun, A. Zhang, A stagewise hyperparameter scheduler to improve generalization, in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, (2021), 1530–1540. https://doi.org/10.1145/3447548.3467287

128. Y. Jin, *Multi-objective machine learning*, Springer, Berlin, 2006. https://doi.org/10.1007/11399346

129. K. Deb, Multi-objective optimisation using evolutionary algorithms: an introduction, in *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*, Springer, London, (2011), 3–34. https://doi.org/10.1007/978-0-85729-652-8_1

130. M. Parsa, A. Ankit, A. Ziabari, K. Roy, PABO: pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design, in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, (2019), 1–8. https://doi.org/10.1109/iccad45719.2019.8942046

131. R. Schmucker, M. Donini, V. Perrone, C. Archambeau, Multi-objective multi-fidelity hyperparameter optimization with application to fairness, in *NeurIPS 2020 Workshop on Meta-learning*, 2020.

132. K. Miettinen, *Nonlinear multiobjective optimization*, Springer Science & Business Media, New York, 1999.

133. K. Miettinen and M. M. Mäkelä, On scalarizing functions in multiobjective optimization, *OR Spectrum*, **24** (2002), 193–213. https://doi.org/10.1007/s00291-001-0092-9

134. T. Chugh, Scalarizing functions in Bayesian multiobjective Optimization, in *2020 IEEE Congress on Evolutionary Computation (CEC)*, (2020), 1–8. https://doi.org/10.1109/CEC48606.2020.9185706

135. Y. Y. Haimes, L. S. Lasdon, D. A. Wismer, On a bicriterion cormulation of the problems of integrated system identification and system optimization, *IEEE Trans. Syst. Man Cybern.*, (1971), 296–297. https://doi.org/10.1109/tsmc.1971.4308298

136. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE T. Evolut. Comput.*, **6** (2002), 182–197. https://doi.org/10.1109/4235.996017

137. N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms, *Evol. Comput.*, **2** (1994), 221–248. https://doi.org/10.1162/evco.1994.2.3.221

138. 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 T. Evolut. Comput.*, **18** (2014), 577–601. https://doi.org/10.1109/tevc.2013.2281535

139. K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part II: handling constraints and extending to an adaptive approach, *IEEE T. Evolut. Comput.*, **18** (2014), 602–622. https://doi.org/10.1109/tevc.2013.2281534

140. E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, *TIK Report*, **103** (2001), 1–21. https://doi.org/10.3929/ethz-a-004284029

141. Q. Zhang, H. Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition, *IEEE T. Evolut. Comput.*, **11** (2007), 712–731. https://doi.org/10.1109/tevc.2007.892759

142. E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms — A comparative case study, in *Parallel Problem Solving from Nature—PPSN V* , Springer, (1998), 292–301. https://doi.org/10.1007/BFb0056872

143. M. Emmerich, N. Beume, B. Naujoks, An EMO algorithm using the hypervolume measure as selection criterion, in *Evolutionary Multi-Criterion Optimization*, Springer, (2005), 62–76. https://doi.org/10.1007/978-3-540-31880-4_5

144. X. Li, A non-dominated sorting particle swarm optimizer for multiobjective optimization, in *Genetic and Evolutionary Computation—GECCO 2003*, Springer, (2003), 37–48. https://doi.org/10.1007/3-540-45105-6_4

145. C. A. C. Coello, G. T. Pulido, M. S. Lechuga, Handling multiple objectives with particle swarm optimization, *IEEE T. Evolut. Comput.*, **8** (2004), 256–279. https://doi.org/10.1109/tevc.2004.826067

146. J. Knowles, ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems, *IEEE T. Evolut. Comput.*, **10** (2006), 50–66. https://doi.org/10.1109/tevc.2005.851274

147. W. Ponweiset, T. Wagner, D. Biermann, M. Vincze, Multiobjective optimization on a limited dudget of evaluations using model-assisted $\mathcal{S}$-metric selection, in *Parallel Problem Solving from Nature – PPSN X*, Springer, (2008), 784–794. https://doi.org/10.1007/978-3-540-87700-4_78

148. M. T. M. Emmerich, K. C. Giannakoglou, B. Naujoks, Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodels, *IEEE T. Evolut. Comput.*, **10** (2006), 421–439. https://doi.org/10.1109/tevc.2005.859463

149. Y. Jin, Surrogate-assisted evolutionary computation: Recent advances and future challenges, *Swarm Evol. Comput.*, **1** (2011), 61–70. https://doi.org/10.1016/j.swevo.2011.05.001

150. S. Daulton, M. Balandat, E. Bakshy, Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization, *Adv. Neural Inform. Process. Syst.*, **33** (2020), 9851–9864

151. D. Hernández-Lobato, J. Hernández-Lobato, A. Shah, R. Adams, Predictive entropy search for multi-objective bayesian optimization, in *Proceedings of The 33rd International Conference on Machine Learning*, (2016), 1492–1501.

152. S. Belakaria, A. Deshwal, J. R. Doppa, Max-value entropy search for multi-objective bayesian optimization, *Adv. Neural Inform. Process. Syst.*, **32** (2019).

153. S. Daulton, M. Balandat, E. Bakshy, Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement, *Adv. Neural Inform. Process. Syst.*, **34** (2021), 2187–2200.

154. Z. J. Lin, R. Astudillo, P. Frazier, E. Bakshy, Preference exploration for efficient bayesian optimization with multiple outcomes, in *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, (2022), 4235–4258.

155. G. Misitano, B. Afsar, G. Lárrage, K. Miettinen, Towards explainable interactive multiobjective optimization: R-XIMO, *Auton. Agent. Multi-Agent Syst.*, **36** (2022), 43. http://doi.org/10.1007/s10458-022-09577-3

156. G. Malkomes, B. Cheng, E. H. Lee, M. Mccourt, Beyond the pareto efficient frontier: constraint active search for multiobjective experimental design, in *Proceedings of the 38th International Conference on Machine Learning*, (2021), 7423–7434.

157. Z. Chen, Y. Zhou, Z. Huang, X. Xia, Towards efficient multiobjective hyperparameter optimization: a multiobjective multi-fidelity bayesian optimization and hyperband algorithm, in *Parallel Problem Solving from Nature–PPSN XVII* , Springer, (2022), 160–174. http://doi.org/10.1007/978-3-031-14714-2_12

158. A. Dushatskiy, A. Chebykin, T. Alderliesten, P. A. N. Bosman, Multi-objective population based training, in *Proceedings of the 40th International Conference on Machine Learning*, (2023), 8969–8989.

159. R. Schmucker, M. Donini, M. B. Zafar, D. Salinas and C. Archambeau, Multi-objective asynchronous successive halving, preprint, arXiv:2106.12639.

160. F. Hutter, L. Kotthoff, J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*, Springer, Cham, 2019. https://doi.org/10.1007/978-3-030-05318-5

161. T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: a next-generation hyperparameter optimization framework, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, (2019), 2623–2631. https://doi.org/10.1145/3292500.3330701

162. R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, I. Stoica, Tune: a research platform for distributed model Selection and Training, preprint, arXiv:1807.05118.

163. M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, et al., BoTorch: a framework for efficient monte-carlo bayesian optimization, *Adv. Neural Inform. Process. Syst.*, **33** (2020).

164. J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D. D. Cox, Hyperopt: a python library for model selection and hyperparameter optimization, *Comput. Sci. Discov.*, **8** (2015), 014008. https://doi.org/10.1088/1749-4699/8/1/014008

165. M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, et al., SMAC3: A versatile bayesian optimization package for hyperparameter optimization, *J. Mach. Learn. Res.*, **23** (2022), 1–9.

166. F. A. Fortin, F. M. De Rainville, M. A. G. Gardner, M. Parizeau, C. Gagné, DEAP: Evolutionary algorithms made easy, *J. Mach. Learn. Res.*, **13** (2012), 2171–2175.

167. R. Martinez-Cantin, Bayesopt: a bayesian optimization library for nonlinear optimization, experimental design and bandits, *J. Mach. Learn. Res.*, **15** (2014), 3735–3739.

168. L. Nardi, A. Souza, D. Koeplinger, K. Olukotun, HyperMapper: a practical design space exploration framework, in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, (2019), 425–426. https://doi.org/10.1109/MASCOTS.2019.00053

169. M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, et al., mlr3: A modern object-oriented machine learning framework in R, *J. Open Source Softw.*, **4** (2019), 1903. https://doi.org/10.21105/joss.01903

170. B. Bischl, R. Sonabend, L. Kotthoff, M. Lang, *Applied Machine Learning Using mlr3 in R* , Chapman and Hall/CRC, New York, 2023. https://doi.org/10.1201/9781003402848

171. A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi and J. Del Ser, jMetalPy: A Python framework for multi-objective optimization with metaheuristics, *Swarm Evol. Comput.*, **51** (2019), 100598. https://doi.org/10.1016/j.swevo.2019.100598

172. N. E. Toklu, T. Atkinson, V. Micka, P. Liskowski and R. K. Srivastava, EvoTorch: Scalable Evolutionary Computation in Python, preprint, arXiv:2302.12600.

173. Y. Li, Y. Shen, W. Zhang, Y. Chen, H. Jiang, M. Liu, et al., OpenBox: a generalized black-box optimization service, in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, (2021), 3209–3219. https://doi.org/10.1145/3447548.3467061

174. K. Kandasamy, K. R. Vysyaraju, W. Neiswanger, B. Paria, C. R. Collins, J. Schneider, et al., Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly, *J. Mach. Learn. Res.*, **21** (2020), 1–27.

175. D. Salinas, M. Seeger, A. Klein, V. Perrone, M. Wistuba, C. Archambeau, Syne Tune: a library for large scale hyperparameter tuning and reproducible research, in *Proceedings of the First International Conference on Automated Machine Learning*, (2022), 1–23.

176. J. George, C. Gao, R. Liu, H. G. Liu, Y. Tang, R. Pydipaty, et al., A scalable and cloud-native hyperparameter tuning system, preprint, arXiv:2006.02085.

177. O. Taubert, M. Weiel, D. Coquelin, A. Farshian, C. Debus, A. Schug, et al., Massively parallel genetic optimization through asynchronous propagation of populations, in *High Performance Computing*, Springer, (2023), 106–124. https://doi.org/10.1007/978-3-031-32041-5_6

178. J. Blank, K. Deb, Pymoo: multi-objective optimization in Python, *IEEE Access*, **8** (2020), 89497–89509. http://doi.org/10.1109/access.2020.2990567

179. S. S. Sandha, M. Aggarwal, I. Fedorov, M. Srivastava, Mango: A Python Library for Parallel Hyperparameter Tuning, in *ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (2020), 3987–3991. https://doi.org/10.1109/icassp40776.2020.9054609

180. L. Hertel, J. Collado, P. Sadowski, J. Ott, P. Baldi, Sherpa: Robust hyperparameter optimization for machine learning, *SoftwareX*, **12** (2020), 100591. https://doi.org/10.1016/j.softx.2020.100591

181. N. O. Nikitin, P. Vychuzhanin, M. Sarafanov, I. S. Polonskaia, I. Revin, I. V. Barabanova, et al., Automated evolutionary approach for the design of composite machine learning pipelines, *Future Gener. Comput. Syst.*, **127** (2022), 109–125. https://doi.org/10.1016/j.future.2021.08.022

182. I. S. Polonskaia, N. O. Nikitin, I. Revin, P. Vychuzhanin, A. V. Kalyuzhnaya, Multi-objective evolutionary design of composite data-driven model, in *2021 IEEE Congress on Evolutionary Computation (CEC)*, (2021), 926–933. https://doi.org/10.1109/CEC45853.2021.9504773

183. R. S. Olson, J. H. Moore, Tpot: A tree-based pipeline optimization tool for automating machine learning, in *Proceedings of the Workshop on Automatic Machine Learning*, 2016, 66–74.

184. C. Guan, Z. Zhang, H. Li, H. Chang, Z. Zhang, Y. Qin, et al., AutoGL: A library for automated graph learning, in *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021.

185. M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, *Adv. Neural Inform. Process. Syst.*, **28** (2015).

186. M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, F. Hutter, Auto-sklearn 2.0: Hands-free automl via meta-learning, *J. Mach. Learn. Res.*, **23** (2022), 1–61.

187. L. Zimmer, M. Lindauer, F. Hutter, Auto-Pytorch: Multi-fidelity metaLearning for efficient and robust autoDL, *IEEE T. Pattern Anal.*, **43** (2021), 3079–3090. https://doi.org/10.1109/tpami.2021.3067763

188. H. Jin, Q. Song, X. Hu, Auto-Keras: an efficient neural architecture search system, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, (2019), 1946–1956. https://doi.org/10.1145/3292500.3330648

189. H. Jin, F. Chollet, Q. Song, X. Hu, AutoKeras: An autoML library for deep learning, *J. Mach. Learn. Res.*, **24** (2023), 1–6.

190. N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, et al., AutoGluon-Tabular: robust and accurate autoML for structured data, preprint, arXiv:2003.06505.

191. C. Wang, Q. Wu, M. Weimer, E. Zhu, FLAML: a fast and lightweight autoML library, in *Proceedings of Machine Learning and Systems 3 (MLSys 2021)*, 2021.

192. N. Fusi, R. Sheth, M. Elibol, Probabilistic matrix factorization for automated machine learning, *Adv. Neural Inform. Process. Syst.*, **31** (2018), 3166–3180.

193. A. Yakovlev, H. F. Moghadam, A. Moharrer, J. Cai, N. Chavoshi, V. Varadarajan, et al., Oracle AutoML: a fast and predictive AutoML pipeline, in *Proc. VLDB Endow.*, **13** (2020), 3166–3180. https://doi.org/10.14778/3415478.3415542

194. D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, D. Sculley, Google vizier: A service for black-box optimization, in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (2017), 1487–1495. https://doi.org/10.1145/3097983.3098043

195. E. Libery, Z. Karning, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati et al., Elastic machine learning algorithms in Amazon SageMaker, in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, (2020), 731–737. https://doi.org/10.1145/3318464.3386126

196. S. Blume, T. Benedens, D. Schramm, Hyperparameter optimization techniques for designing software sensors based on artificial neural networks, *Sensors*, **21** (2021), 8435. https://doi.org/10.3390/s21248435

197. C. Cooney, A. Korik, R. Folli, D. Coyle, Evaluation of hyperparameter optimization in machine and deep learning methods for decoding imagined speech EEG, *Sensors*, **20** (2020), 4629. https://doi.org/10.3390/s20164629

198. R. Khalida, N. Javaida, Survey on hyperparameters optimization algorithms of forecasting models in smart grid, *Sustain. Cities Soc.*, **61** (2020), 102275. https://doi.org/10.1016/j.scs.2020.102275

199. R. Andonie, Hyperparameter optimization in learning systems, *J. Membr. Comput.*, **1** (2019), 279–291. https://doi.org/10.1007/s41965-019-00023-0

200. G. Luo, A review of automatic selection methods for machine learning algorithms and hyper-parameter values, *Netw. Model. Anal. Health Inform. Bioinform.*, **5** (2016), 1–16. https://doi.org/10.1007/s13721-016-0125-6

201. S. Stober, D. J. Cameron, J. A. Grahn, Using convolutional neural networks to recognize rhythm stimuli from electroencephalography recordings, *Adv. Neural Inform. Process. Syst.*, **27** (2014), 1449–1457.

202. A. Drouin-Picaro, T. H. Falk, Using deep neural networks for natural saccade classification from electroencephalograms, in *2016 IEEE EMBS International Student Conference (ISC)*, 2016, 1–4. https://doi.org/10.1109/embsisc.2016.7508606

203. Z. Zhou, F. Xiong, B. Huang, C. Xu, R. Jiao, B. Liao, et al., Game-theoretical energy management for energy internet with big data-based renewable power forecasting, *IEEE Access*, **5** (2017), 5731–5746. https://doi.org/10.1109/access.2017.2658952

204. 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. https://doi.org/10.1016/j.artmed.2020.101822

205. A. Alaa, M. Schaar, Autoprognosis: Automated clinical prognostic modeling via bayesian optimization with structured kernel learning, in *Proceedings of the 35th International Conference on Machine Learning*, (2018), 139–148.

206. I. Castiglioni, L. Rundo, M. Codari, G. Di Leo, C. Salvatore, M. Interlenghi, et al., AI applications to medical images: From machine learning to deep learning, *Phys. Med.*, **83** (2021), 9–24. https://doi.org/10.1016/j.ejmp.2021.02.006

207. M. Nishio, K. Fujimoto, K. Togashi, Lung segmentation on chest X-ray images in patients with severe abnormal findings using deep learning, *Int. J. Imag. Syst. Tech.*, **31** (2021), 1002–1008. https://doi.org/10.1002/ima.22528

208. A. Abdellatif, H. Abdellatef, J. Kanesan, C. O. Chow, J. H. Chuah, H. M. Gheni, An Effective Heart Disease Detection and Severity Level Classification Model Using Machine Learning and Hyperparameter Optimization Methods, *IEEE Access*, **10** (2022), 79974–79985. https://doi.org/10.1109/ACCESS.2022.3191669

209. D. M. Belete, M. D. Huchaiah, Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results, *Int. J. Comput. Appl.*, **44** (2022), 875–886. https://doi.org/10.1080/1206212X.2021.1974663

210. S. Nematzadeh, F. Kiana, M. Torkamanian-Afshar, N. Aydin, Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases, *Comput. Biol. Chem.*, **97** (2022), 107619. https://doi.org/10.1016/j.compbiolchem.2021.107619

211. G. I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, H. Samulowitz, An effective algorithm for hyperparameter optimization of neural networks, *IBM J. Res. Dev.*, **61** (2017), 9:1–9:11. https://doi.org/10.1147/JRD.2017.2709578

212. D. Stamoulis, E. Cai, D. C. Juan, D. Marculescu, HyperPower: Power- and memory-constrained hyper-parameter optimization for neural networks, in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (2018), 19–24. https://doi.org/10.23919/DATE.2018.8341973

213. Z. Lu, L. Chen, C. K. Chiang, F. Sha Hyper-parameter Tuning under a Budget Constraint, in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, (2019), 5744–5750. https://doi.org/10.24963/ijcai.2019/796

214. C. Wang, H. Wang, C. Zhou, H. Chen, ExperienceThinking: Constrained hyperparameter optimization based on knowledge and pruning, *Knowl.-Based Syst.*, **223** (2018), 106602. https://doi.org/10.1016/j.knosys.2020.106602

215. B. Letham, B. Karrer, G. Ottoni, E. Bakshy, Constrained Bayesian Optimization with Noisy Experiments, *Bayesian Anal.*, **14** (2019), 495–519. https://doi.org/10.1214/18-BA1110

216. T. P. Papalexopoulos, C. Tjandraatmadja, R. Anderson, J. P. Vielma, D. Belanger, Constrained discrete black-box optimization using mixed-integer programming, in *Proceedings of the 39th International Conference on Machine Learning*, 2022, 17295–17322.

217. F. Berkenkamp, A. Krause, A. P. Schoellig, Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics, *Mach. Learn.*, **112** (2023), 3713–3747. https://doi.org/10.1007/s10994-021-06019-1

218. F. Wenzel, J. Snoek, D. Tran, R. Jenatton, Hyperparameter ensembles for robustness and uncertainty quantification, *Adv. Neural Inform. Process. Syst.*, **33** (2020), 6514–6527.

219. F. Seifi, M. J. Azizi, S. T. A. Niaki, A data-driven robust optimization algorithm for black-box cases: An application to hyper-parameter optimization of machine learning algorithms, *Comput. Ind. Eng.*, **160** (2021), 107581. https://doi.org/10.1016/j.cie.2021.107581

220. G. Sunder, T. A. Albrecht, C. J. Nachtsheim, Robust sequential experimental strategy for black-box optimization with application to hyperparameter tuning, *Qual. Reliab. Eng. Int.*, **38** (2022), 3992–4014. https://doi.org/110.1002/qre.3181

221. L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, M. Pontil, Bilevel Programming for Hyperparameter Optimization and Meta-Learning, in *Proceedings of the 35th International Conference on Machine Learning*, (2018), 1568–1577.

222. O. Bohdal, Y. Yang, T. Hospedales, EvoGrad: Efficient Gradient-Based Meta-Learning and Hyperparameter Optimization, *Adv. Neural Inform. Process. Syst.*, **34** (2021).

223. X. He, K. Zhao, X. Chu, AutoML: A survey of the state-of-the-art, *Knowl.-Based Syst.*, **212** (2021), 106622. https://doi.org/10.1016/j.knosys.2020.106622

224. A. M. Vincent, P. Jidesh, An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms, *Sci. Rep.*, **13** (2023), 4737. https://doi.org/10.1038/s41598-023-32027-3

225. B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, 8697–8710. https://doi.org/10.1109/cvpr.2018.00907

226. E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized evolution for image classifier architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, (2019), 4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780

227. C. White, W. Neiswanger, Y. Savani, BANANAS: Bayesian optimization with neural architectures for neural architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, (2021), 10293–10301. https://doi.org/10.1609/aaai.v35i12.17233

228. H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in *International Conference on Learning Representations* , (2019).

229. X. Wang, C. Xue, J. Yan, X. Yang, Y. Hu, K. Sun, Mergenas: Merge operations into one for differentiable architecture search, in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, (2020), 3065–3072. https://doi.org/10.24963/ijcai.2020/424

230. X. Wang, W. Guo, J. Su, X. Yang, J. Yan, Zarts: On zero-order optimization for neural architecture search, *Adv. Neural Inform. Process. Syst.*, **35** (2022).

231. Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, J. Sun, Detnas: Backbone search for object detection, *Adv. Neural Inform. Process. Syst.*, **32** (2019), 6642–6652.

232. X. Wang, J. Lin, J. Zhao, X. Yang, J. Yan, Eautodet: Efficient architecture search for object detection, in *Computer Vision–ECCV 2022*, Springer, (2022), 668–684. https://doi.org/10.1007/978-3-031-20044-1_38

233. L. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, et al., Searching for efficient multi-scale architectures for dense image prediction, *Adv. Neural Inform. Process. Syst.*, **31** (2018).

234. C. Liu, L. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, et al., Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (2019), 82–92. https://doi.org/10.1109/cvpr.2019.00017

235. X. Wang, Z. Lian, J. Lin, C. Xue, J. Yan, DIY your easynas for vision: Convolution operation merging, map channel reducing, and search space to supernet conversion tooling, *IEEE T. Pattern Anal.*, **45** (2023), 13974–13990. https://doi.org/10.1109/tpami.2023.3298296

236. Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, *Eur. J. Oper. Res.*, **290** (2021), 405–421. https://doi.org/10.1016/j.ejor.2020.07.063

237. J. Yan, S. Yang, E. Hancock, Learning for graph matching and related combinatorial optimization problems, in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, (2020), 4988–4996. https://doi.org/10.24963/ijcai.2020/694

238. E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, *Adv. Neural Inform. Process. Syst.*, **30** (2017), 6351–6361.

239. M. Nazari, A. Oroojlooy, L. Snyder, M. Takác, Reinforcement learning for solving the vehicle routing problem, *Adv. Neural Inform. Process. Syst.*, **31** (2018), 9839–9849.

240. C. Liu, Z. Dong, H. Ma, W. Luo, X. Li, B. Pang, et al., L2P-MIP: Learning to presolve for mixed integer programming, in *The Twelfth International Conference on Learning Representations*, (2024).

241. Y. Li, X. Chen, W. Guo, X. Li, W. Luo, J. Huang, et al., Hardsatgen: Understanding the difficulty of hard sat formula generation and a strong structure-hardness-aware baseline, in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, (2023), 4414–4425. https://doi.org/10.1145/3580305.3599837

242. R. Wang, L. Shen, Y. Chen, X. Yang, D. Tao, J. Yan, Towards one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-constrained case, in *The Eleventh International Conference on Learning Representations*, (2022).

243. Q. Ren, Q. Bao, R. Wang, J. Yan, Appearance and structure aware robust deep visual graph matching: Attack, defense and beyond, in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (2022), 15242–15251. https://doi.org/10.1109/cvpr52688.2022.01483

244. J. Yan, M. Cho, H. Zha, X. Yang, S. M. Chu, Multi-graph matching via affinity optimization with graduated consistency regularization, *IEEE T. Pattern Anal.*, **38** (2016), 1228–1242. https://doi.org/10.1109/tpami.2015.2477832

245. T. Wang, Z. Jiang, J. Yan, Multiple graph matching and clustering via decayed pairwise matching composition, in *Proceedings of the AAAI Conference on Artificial Intelligence*, (2020), 1660–1667. https://doi.org/10.1609/aaai.v34i02.5528

246. R. Wang, T. Zhang, T. Yu, J. Yan, X. Yang, Combinatorial learning of graph edit distance via dynamic embedding, in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (2021), 5237–5246. https://doi.org/10.1109/CVPR46437.2021.00520

AIMS Press