A Survey of Automatic Parameter Tuning Methods for Metaheuristics

Changwu Huang, *Member, IEEE*, Yuanxiang Li, and Xin Yao, *Fellow, IEEE*

***Abstract*—Parameter tuning, that is, to find appropriate parameter settings (or configurations) of algorithms so that their performance is optimized, is an important task in the development and application of metaheuristics. Automating this task, i.e., developing algorithmic procedure to address parameter tuning task, is highly desired and has attracted significant attention from the researchers and practitioners. During last two decades, many automatic parameter tuning approaches have been proposed. This paper presents a comprehensive survey of automatic parameter tuning methods for metaheuristics. A new classification (or taxonomy) of automatic parameter tuning methods is introduced according to the structure of tuning methods. The existing automatic parameter tuning approaches are consequently classified into three categories: 1) simple generate-evaluate methods; 2) iterative generate-evaluate methods; and 3) high-level generate-evaluate methods. Then, these three categories of tuning methods are reviewed in sequence. In addition to the description of each tuning method, its main strengths and weaknesses are discussed, which is helpful for new researchers or practitioners to select appropriate tuning methods to use. Furthermore, some challenges and directions of this field are pointed out for further research.**

***Index Terms*—Automatic parameter tuning, metaheuristics, parameter setting, parameter tuning.**

I. Introduction

OPTIMIZATION methods are extensively required and applied to solve problems from almost all disciplines, whether economics, sciences, or engineering [1]. Generally speaking, optimization approaches can be classified into exact, heuristic, and metaheuristic methods [2]. Exact methods can guarantee the optimality of their solutions. In other words, an exact method can obtain the optimal solution if it is completely executed. However, many computationally challenging problems have emerged in real-world applications. Solving these problems with exact methods would require a huge or unaffordable amount of computing resources since many of these problems are NP-hard [3]. Traveling salesman problem (TSP), vehicle routing problem, capacitated art routing problem, timetabling, and software verification are examples of such hard problems. Unlike exact methods, heuristic and metaheuristic algorithms do not guarantee their solutions' optimality, but they can obtain high-quality solutions in a reasonable time, which is practical for application problems. Thus, heuristics and metaheuristics are the main alternatives to solve hard optimization problems.

Heuristics are problem specific algorithms that implement some reasonable strategies or rules (heuristic mechanisms) to solve problems. Although there is no theoretical guarantee of optimality, heuristics have met some notable success on many difficult problems and thus have been a popular choice for solving NP-hard problems. However, the problem-dependent nature of heuristics restricts the application of a heuristic to one particular class of problems, i.e., heuristics are designed to handle specific problems (or instances). Also, heuristics usually provide only suboptimal solutions because they do not attempt to escape from local optimum. These drawbacks have led to the introduction of metaheuristics.

Metaheuristics are high-level methodologies or general algorithmic templates, which generally do not adapt deeply to specific problem(s) [4]. Hence, they usually can solve a wide range of problems [2]. In fact, the prefix "meta," which means "upper level methodology," indicates that metaheuristic algorithms can be viewed as "higher level" heuristics. Hybrid approaches based on existing metaheuristic(s) are also considered metaheuristics [5]. In last decades, metaheuristics have received widespread attention from researchers and are widely recognized as efficient approaches for hard optimization problems. A number of metaheuristic algorithms have been developed and extensively applied, including simulated annealing (SA) [6], Tabu search (TS) [7], evolutionary algorithms (EAs) [8], ant colony optimization (ACO) algorithm [9], particle swarm optimization (PSO) [10], and so forth. Most of metaheuristics are nature-inspired (inspired from some principles in physics, biology, etc.), contain stochastic components,

Manuscript received September 4, 2018; revised March 29, 2019; accepted June 2, 2019. Date of publication June 7, 2019; date of current version March 31, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFC0804002, in part by the Engineering and Physical Sciences Research Council under Grant EP/J017515/1 and Grant EP/P005578/1, in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X386, in part by the Shenzhen Peacock Plan under Grant KQTD2016112514355531, in part by the Science and Technology Innovation Committee Foundation of Shenzhen under Grant ZDSYS201703031748284 and Grant JCYJ20180504165652917, and in part by the Program for University Key Laboratory of Guangdong Province under Grant 2017KSYS008. *(Corresponding author: Xin Yao.)*

C. Huang is with the Shenzhen Key Laboratory of Computational Intelligence, University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: huangcw3@sustech.edu.cn).

Y. Li is with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: yxli@whu.edu.cn).

X. Yao is with the Shenzhen Key Laboratory of Computational Intelligence, University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China, and also with CERCIA, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: xiny@sustech.edu.cn).

Digital Object Identifier 10.1109/TEVC.2019.2921598

and often have several free parameters that can be set by users according to problem(s) at hand [4].

The setting of parameters (or parameter setting) has strong impact on the performance or efficacy of a metaheuristic, because parameters control the behavior of heuristic mechanisms in the algorithm [11]. Take SA algorithm as an example, it yields good solutions only if its parameters, including initial temperature, cooling factor, number of iterations and so on, are properly chosen [12]. Thus, to obtain high performance, algorithm's parameters should be properly set or fine-tuned. Even though default parameter settings are provided, tuning algorithm's parameters for problems to be solved can result in performance improvement since default parameter settings are often determined with other problems (or application contexts) which are different from the problems at hand.

According to the no free lunch (NFL) theorem of optimization [13], there does not exist an universal algorithm which works well for all optimization problems. This indicates that one need to tailor the adopted algorithm for problems at hand to improve algorithm's performance and to obtain good solutions. Moreover, this also implies that parameter setting is not a one-time task, that is, researchers or end users need to address parameter setting problem again when they face new problems. Consequently, the so-called parameter setting problem (also known as algorithm configuration in literature) [11], which is to properly set algorithm's parameter values for maximizing the empirical performance of the algorithm, is routinely encountered by algorithms designers and users.

Although it has been recognized that the performance of a metaheuristic depends on its parameter values, parameter setting problem has not been formally treated by the academic community until the end of the last century [14]. During the first decades of metaheuristics research, metaheuristics were tuned "by hand," i.e., performing experiments with different parameter settings and selecting the best one, or "by analogy," i.e., adopting parameter setting that is successful on similar problem(s) [15]. The demand of systematic approaches for metaheuristics' parameter setting has been increasingly outlined in the literature since the end of the last century [16], [17]. Subsequently, the parameter setting problem has attracted more attention of developers and end users of metaheuristics and more efforts have been dedicated to developing systematic and sophisticated approaches to address this problem.

Parameter setting problem, i.e., finding appropriate or optimal parameter setting (i.e., configuration) in the parameter space of a metaheuristic, is a broad problem and research topic. According to [18]–[21], it can be mainly divided into two cases.

1) *Parameter tuning* (also termed as off-line tuning), where good parameter values (parameter setting) are identified before applying the algorithm to solve problems at hand. In this case, the results of parameter tuning, i.e., the optimal parameter setting founded by tuning process, is used in solving problems and these parameter values remain unchanged during the run [18].
2) *Parameter control* (also known as on-line tuning), where the values of controlled parameters are changing directly according to some strategies during the execution of algorithm (i.e., during the run). In this situation, initial values and appropriate control strategies for controlled parameters, which change or adapt relevant parameter values during the run, are required. These control strategies could be deterministic, adaptive, or self-adaptive [21].

To avoid confusion, only terms "parameter tuning" (tuning for short) and "parameter control" are used instead of "offline tuning" and "online tuning" hereafter. Parameter tuning processes usually requires a large number of runs of the algorithm to analyze its performance on one instance or a set of problem instances with different parameter settings. This makes tuning process time-consuming, which is the main disadvantage of parameter tuning. The advantage of parameter tuning is its universality, that is, a good tuning method is applicable to handle parameter tuning of many different metaheuristics. While, the obvious drawback of parameter control is its nonuniversality, that is, the proper control strategies for one algorithm usually is not suitable for another algorithm [19]. For example, the self-adaption strategy of mutation strength in $(\mu/\rho, \lambda)$-evolution strategy (ES) [22] is not proper for controlling inertia weight of PSO. Additionally, in order to correctly design parameter control strategies for an algorithm, it is necessary to have a rough idea of how to change parameter(s) during the run to achieve good performance. This usually requires an understanding of the proper parameter values at different phases in the running of the algorithm and involves the utilization of history information generated in iterative process of the algorithm. Hence, in above senses, parameter tuning is easier and more practical than parameter control. In parameter control, self-adaptive parameter control, such as step-size control and covariance matrix adaptation in covariance matrix adaptation ES (CMA-ES) [23], is the state-of-the-art method for numerical parameter control. Adaptive operator selection (AOS) is a popular method to dynamically determine which operator(s) should be applied during the run of an optimization algorithm (i.e., categorical parameter control), based on its performance history of available operators [24]–[26]. More description and achievements in parameter control could be found in [27].

The focus of this paper is on parameter tuning, since many interesting contributions have been published recently within this field. A survey of automatic parameter tuning methods for metaheuristic algorithms is provided. The remainder of this paper is organized as follows. Section II first gives a short overview of the parameter tuning problem including the problem statement and types of parameters, then, a new classification method of automatic parameter tuning approaches is introduced. Sections III–V review the classified three categories of tuning approaches successively. Finally, some future research directions and a concluding summary are given in Section VI.

II. Automatic Parameter Tuning

It is widely recognized that the performance of an optimization algorithm can be improved by parameter tuning.

The task of parameter tuning, however, can be very time-consuming and tedious. In early research, this task was carried out manually in many cases. Thus, automating this tough task, i.e., developing automated approaches to finding good parameter settings (or configurations), is desired and of high practical relevance in several contexts. Hutter *et al.* [28], [29] stated the following main motivation and relevance for developing automatic procedure for parameter tuning.

1) *Development of Complex Algorithms:* Parameter setting is an indispensable but time-consuming step in algorithm development. The use of automatic parameter tuning methods can effectively reduce the time cost of this task, and potentially result in better algorithm's performance than manual methods.
2) *Empirical Studies, Evaluations, and Comparisons of Algorithms:* Comparing performance of different algorithms is a common task in research. An essential question in this task is whether one algorithm outperforms another owing to its fundamental superiority or the optimality of its parameters [17]. Automatic parameter tuning methods can alleviate this problem and thus make comparative studies more fair and meaningful.
3) *Practical Use of Algorithms:* In the application of metaheuristics, to make the algorithm work well (have good performance), users often need to appropriately set algorithm's parameters for the problems they are facing. However, this usually requires users to know how parameters of the algorithm affect its performance. Fortunately, automatic parameter tuning can find proper parameter settings and eliminate the need of prior knowledge for users.

Therefore, automatic parameter tuning, also referred as automatic algorithm configuration, is a rapidly growing field because of above motivation and eliminating the limitations and difficulties of manual parameter tuning.

A. Statement of Parameter Tuning Problem

The parameter tuning problem can be briefly described as: given a parameterized algorithm and one instance or a set of instances, find an optimal parameter setting that results in the best possible performance across the given problem instance(s). In other words, the purpose of parameter tuning is to find a configuration that maximizes the performance of an algorithm over the given problem instance(s). A formal statement of parameter tuning problem was concisely given in [11] as follows.

Given:

1) A parameterized algorithm A with free parameters that affect its behavior.
2) A configuration space (or parameter space) C, which defines possible configurations (i.e., parameter settings).
3) A set of problem instances I.
4) A performance metric m that measures the performance of A across I for a given configuration c ($c \in C$).

Find: A configuration $c^* \in C$ that optimizes the performance of A on I according to metric m.

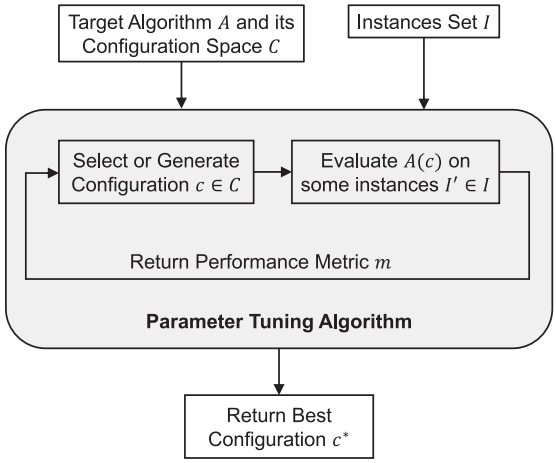


Fig. 1. Illustration of automatic parameter tuning procedure.

In parameter tuning problem, the algorithm A whose performance will be optimized via tuning its parameters is called the target algorithm, the automatic parameter tuning method used for finding the optimal configuration is often referred to as a tuning algorithm (tuner for short) [30]. A parameter setting or configuration c refers to a setting of free parameters (of A) that need to be tuned, and $A(c)$ denotes target algorithm A under specific configuration c. The process of automatic parameter tuning is illustrated in Fig. 1.

Performance evaluation of the target algorithm (with different configurations) is a vital part of the parameter tuning process. Usually, the performance of a run of a metaheuristic on one instance (a run for short) can be evaluated or assessed by the quality of the solution obtained with a fixed computation time, or by the computation time required to find a solution of the desired quality. However, in parameter tuning, target algorithm performance evaluation is not an easy task. Because of the stochasticity of tuning problem, the performance measure of target algorithm is stochastic and computationally expensive. The stochasticity (or randomness) of tuning problem comes from two main sources of randomness: 1) the stochastic nature of target algorithm, which results from the utilization of randomized decisions during the execution of algorithm and 2) the randomness in selecting problem instance(s) to estimate target algorithm's performance [31], [33]. Since the performance metric is a stochastic quantity, usually, the expected performance of the target algorithm is optimized during parameter tuning process [34]. The estimation of expected performance measure cannot be achieved directly through analytical computation, so it is usually estimated by Monte Carlo method [31].

According to the above statements, parameter tuning problem can be viewed as an optimizing problem, often called meta-optimization [35]. Meta-optimization is a research field of searching the right behavioral parameters for some underlying optimizer [36]. This is not a new concept as it was already used by Mercer and Sampson [37] in the late 1970s for optimizing EA. In consequence, a tuning algorithm (tuner) is a meta-optimizer that searches for the optimal (or at least somewhat well performing) set of parameters

for the target algorithm. And this meta-optimization problem is a black-box problem since the relationship between algorithm's parameter values and its performance metric on given instances are unavailable. Considering its stochasticity at the same time, parameter tuning problem can be treated as a stochastic black-box (meta)-optimization problem.

Automatic parameter tuning can also be considered, from a machine learning perspective, as a learning problem of finding a good parameter setting for solving unseen problem instances with high efficacy by learning from a set of training instances [31]. Hence, there are two distinct phases or steps: 1) tuning phase and 2) testing (or production) phase. In tuning phase, a parameter setting that optimizes the performance measure of the algorithm is to be determined, based on the training instances that commonly are representatives of the problem faced in the following reproduction phase. In the later testing (or production) phase, the founded configuration is adopted to solve previously unseen instances [38]. The goal of algorithm configuration is to find a good parameter setting in tuning phase so that it maximizes the performance of target algorithm across instances that will be seen during the test phase [30].

1) Parameter Types: In parameter tuning, virous types of parameters may occurs depending on the given target algorithm. Based on the parameter types classification in existing work [11], [38]–[40], parameters can be mainly classified into categorial and numerical parameters, according to their searchability [41], [42].

1) *Numerical parameters* can be real or integer values. The population size and mutation rate in EAs are typical example of numerical parameters.
2) *Categorical parameters* are related to mechanisms or operators that can be implemented in different ways in the algorithm. They have a finite, unordered set of discrete values. Examples of categorical parameter include the selection operator in an EA, which can be chosen among tournament, roulette, and ranking-based selection [41], and the mutation operator in evolutionary programming, which could be chosen from Gaussian and Cauchy mutation [43].

Numerical parameters define a domain that has distance measures between different parameter values, and thus one can use optimization methods to search the optimal values. But for categorical parameters, this is not possible because the categorical parameters' domain is not exploitable and only sampling methods can be used to search this domain [41].

The number and type of algorithm parameters, as well as the constraints on configurations, determine the nature of configuration space C and have a profound impact on the methods used for finding optimal parameter settings within that space [11]. For instance, if there are only numerical parameters in a tuning problem, it can be solved by a derivative-free optimization algorithm along with an approach of handling the stochasticity.

2) Specialist or Generalist: Besides parameter types, another aspect that has profound influence on solving parameter tuning problem is the goal that one wants to tailor the target algorithm to be a specialist or generalist [44], which are defined as follows.

1) *Generalist:* Denotes a parameter setting that has good performance across a wide range (or a set) of problems (or instances).
2) *Specialist:* Signifies a parameter setting that show excellent performance on only one problem (or instance).

By no-free lunch theorem, the true generalist which performs well on all problems (or instances) does not exist. So, in practice, a generalist is restricted to a set of problem instances not to all the possible instances. The choice of tuning a specialist or a generalist commonly depends on the practitioner and the application context he is facing.

Generally speaking, tuning the target algorithm as a specialist is easier than tuning it as a generalist. The main difference between tuning a specialist and a generalist exists in evaluating the performance of target algorithm. In the case of tuning the algorithm on only one problem instance, the performance metric could be the mean of performance measures from multiple runs on the instance. While, performance metric of the target algorithm on a set of instances means the performance over a whole set of instances. This could be represented by the expected value of the performance over the instance set, roughly speaking, the average performance of the algorithm across the given set of instances [27], [31]. However, for some situations, the performance metrics of the algorithm on different instances may be different tremendously, such as different in orders of magnitude. In this case, the expected value of performance measure over different instance is unreasonable, and normalization of performance measures on each instance or the building block design would be helpful [31]. Apparently, finding the optimal parameter setting (or configuration) usually requires a large number of runs of the target algorithm with different configurations on different instances. Thus, parameter tuning problem is a computational expensive and time-consuming task.

B. Classification of Tuning Methods

Since the end of last century, a number of automatic parameter tuning approaches have been put forward, such as F-Race, REVAC, ParamILS, SPO, SMAC, and so forth. In the overview paper of development in automatic parameter tuning [39], it distinguished the existing automatic parameter tuning algorithms among two types of methods: 1) model-free and 2) model-based methods. In [42], tuning methods were classified as: hand-made tuning, tuning by analogy, experimental design-based tuning, search-based tuning, and hybrid tuning. Eiben and Smit [41] classified tuning methods according to search effort of each method. This paper distinguishes the existing tuning methods from the perspective of tuner's structure and composition.

Essentially, all the existing tuning methods work by the generate and evaluate (or test) principle, that is, by generating different parameter settings (or configurations) and evaluating them by establishing their performance metrics [33], [34], [41]. Based on this principle, this paper classifies the existing tuning methods into three main categories.

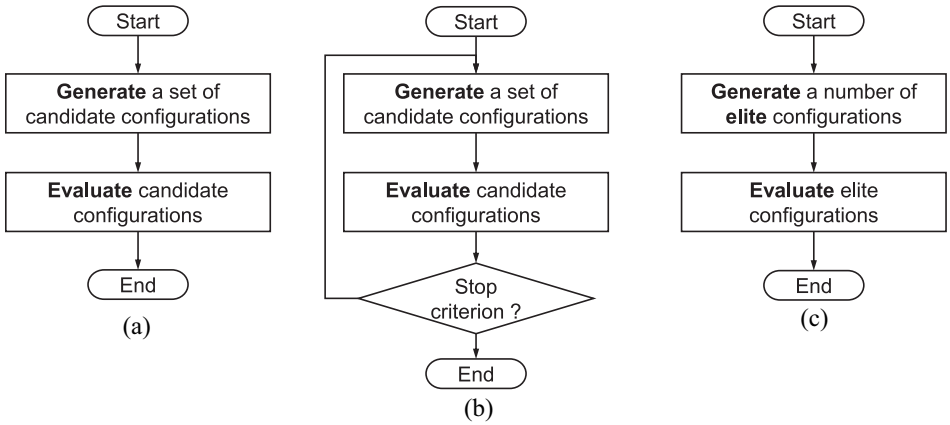


Fig. 2. Illustrations of the general structures of three categories of tuning methods. (a) Simple generate-evaluate methods, (b) iterative generate-evaluate methods, and (c) high-level generate-evaluate methods.

1) *Simple Generate-Evaluate Methods:* Are straightforward approaches that consist of a generate and an evaluate phase. In generate step, a set of candidate configurations are generated. Then, in evaluate phase, each of those configurations is evaluated in order to find the best one from them.
2) *Iterative Generate-Evaluate Methods:* Mainly involve a repeated process of generate and evaluate steps. This kind of methods do not generate all the candidate configurations in one step as like in simple generate-evaluate methods. On the contrary, it starts with a small set of initial configurations and creates new configurations iteratively during execution. After new configurations are generated, they are evaluated to find the incumbent, that is, the best configuration found so far, and at the same time to guide the generating of new configurations for next iteration. This category can be further subcategorized into: experimental design-based tuning, numerical optimization-based tuning, heuristic search-based methods, and model-based optimization approaches. These subcategories will be described in Section IV.
3) *High-Level Generate-Evaluate Methods:* Consist of a high-level generate mechanism and an evaluation step. In generate phase, a number of elite (high-quality) configurations are generated by existing tuners or search methods rather than by random sampling or design of experiments (DOE) methods. In evaluate step, the best of these configurations is selected through careful evaluating.

The general structures or frameworks of above three categories are illustrated in Fig. 2. Different generating techniques (also referred to as sampling methods) and evaluation approaches form different tuning algorithms. Generating techniques produce candidate parameter settings (configurations) and evaluation approaches estimate candidate configurations' performance, where the stochasticity of the tuning problem need to be handled. In parameter tuning approaches, the commonly used techniques for creating candidate configurations includes DOE, such as full factorial design (FFD), central composite design, Latin hypercube design (LHD), and random sampling design (RSD). Yuan *et al.* [34] broadly considered black-box optimization algorithms as sampling methods for parameter tuning. Actually, any methods that can effectively generate candidate configurations could be taken as generating techniques.

Evaluation methods in tuning problem distinguish from these methods used in black-box optimization problem due to the stochasticity of tuning problem. Thus, it is worth here to briefly summarize the evaluation methods that are used in the existing tuning algorithms. Evaluation techniques for parameter tuning mainly include the following.

1) *Repeated Evaluation:* For stochastic or random objective function optimization problem, like parameter tuning problem, the most straightforward method for evaluating the objective function is to evaluate the function multiple times and return the average value [34]. In parameter tuning, the repeated evaluation method assesses each candidate configuration by running a number of times of the target algorithm and returning its average performance measure.
2) *F-Racing:* The *F*-Race method proposed by Birattari *et al.* [31], [45] evaluates candidate configurations gradually, i.e., instance by instance, and immediately eliminates inferior configurations as soon as statistical evidence is gathered against them. The early elimination of poor quality configurations can concentrate computing resources on more promising candidates and hence allow them to obtain more reliable performance estimation [42]. Thus, racing method uses computational power more efficiently than repeated evaluation.
3) *Intensification:* Intensification is a method used in comparing a new configuration to the incumbent. In this method, the new candidate is gradually evaluated on the sequence of instances (sequence for short in following of this paragraph) that the incumbent has already evaluated on. During the evaluation, once the new candidate is worse than the incumbent, it is eliminated; otherwise, it is evaluated on next instance from the sequence and compares with the incumbent again. This process

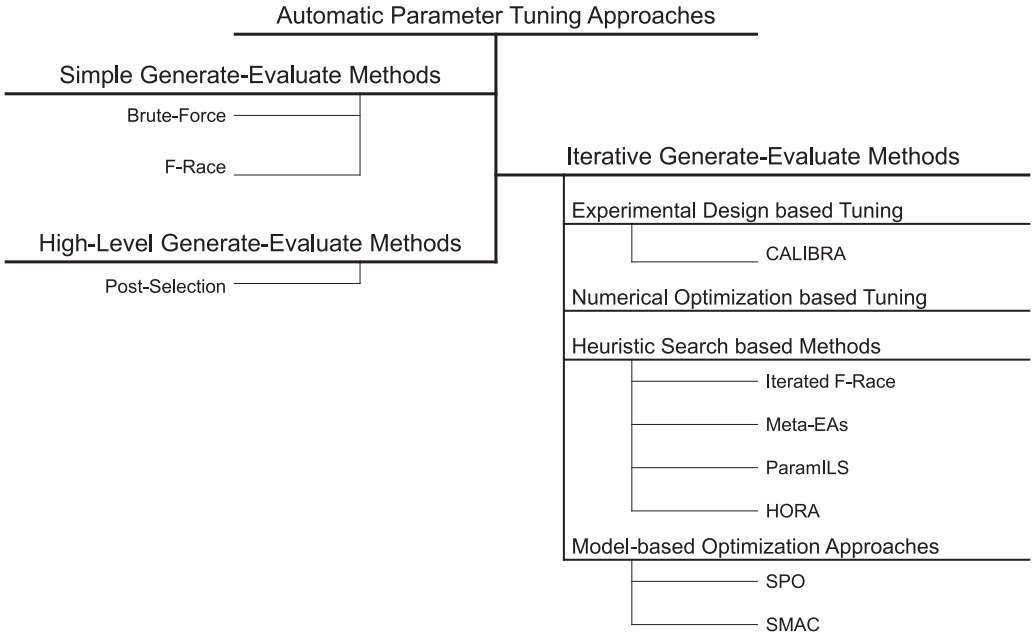


Fig. 3. Classification of automatic parameter tuning approaches.

continues until the candidate has been evaluated on all the instances in the sequence, then a new incumbent is determined. Intensification mechanism has been used in several tuning methods, including ParamILS [28], SPO+ [46], and SMAC [47].

4) *Sharpening:* Sharpening [48] is another intensification technique that makes the promising parameter configurations are tested more thoroughly than the ones that are not as prospective. In the beginning, each configuration is evaluated with a small number of tests, but when certain threshold is met, the number of tests increases, i.e., doubles. In this way, the tuning algorithm can explore the configuration space quickly.
5) *Adaptive Capping:* Adaptive capping proposed in [28] is a method used to cut off the run of unpromising configurations and thus lead to computational savings. The authors present two of its variants, namely, trajectory-preserving capping and aggressive capping.

The parameter tuning algorithms that as far as we are aware from literature are allocated to above three main categories and its subcategories. This classification is presented in Fig. 3. In subsequent sections, each category of tuning algorithms will be reviewed successively.

III. Simple Generate-Evaluate Methods

Since parameter tuning problem is a black-box problem, the most straightforward way to find a good parameter setting is through the generate-evaluate principle. The simple generate-evaluate methods, as illustrated in Fig. 2(a), are noniterative tuners that directly adopt this principle by first creating a number of parameter settings (candidate configurations), then, evaluating each of them to finding the best configuration. The brute-force approach and *F*-Race method fall in this category. The main difference between them is the evaluation method. Specifically, brute-force adopts the repeated evaluation, while *F*-Race employs racing method.

A. Brute-Force Approach

In brute-force approach, at first, a set of parameter configurations are generated usually by FFD or other DOE techniques, then the performance of each candidate configuration is estimated by running the same number of runs on training instances. The configuration with the best estimated performance is considered the optimal parameter setting. In this method, the computing resources are evenly distributed to each candidate configurations [31]. To achieve good results by brute-force method, a sufficiently large number of runs is required for each candidate configurations.

Brute-force approach is the easiest method for parameter tuning, but it presents some serious weakness. First, because of the even allocation of computational resources to each configuration, poor quality configurations are thoroughly tested to the same extent as the good ones are, thus, the computational power is not used efficiently. Additionally, there is no criterion that decides how many runs of each configuration on each instance should be performed to handle the stochasticity of the target algorithm [45].

B. F-Race

F-Race, which is inspired from the algorithm Hoeffding race [49], [50] in machine learning for model selection, was proposed in [45] and comprehensively studied in [31]. The essential idea of racing method is to evaluate candidate configurations incrementally on a stream of instances. As soon as sufficient (statistical) evidence is gathered against some candidates, these configurations are discarded, and the race continues on the surviving candidates. In *F*-Race, after each

evaluation round of the candidate configurations, the nonparametric Friedman test (Friedman two-way analysis of variance by ranks) is used as a family wise test to check whether there is evidence that at least one of the candidate configurations is significantly different from others in terms of performance measures. If the null hypothesis of no differences is rejected, pair-wise comparisons between the best ranked and each other configuration are executed and all candidates that result in significantly worse performance than the incumbent are eliminated and will not appear in next evaluation round [51]. The process is repeated until there are only two candidates remain, and the better one of the two is taken as the result for tuning problem.

The test statistic used in *F*-Race is based on the ranking of candidates' performance metrics. It is worth mentioning that ranking plays a twofold role here. The first role is connected with the nonparametric nature of the statistical test based on the ranking. A second role is to implement a blocking design [52], where only the ranking of different configurations within each instance is considered. The blocking design is an effective way of normalizing or standardizing the performance metrics observed on different instances [38].

A set of candidate configurations should be created before the execution of racing procedure. When *F*-Race was first proposed by Birattari *et al.* [45], this candidates set was generated by an FFD, and this version of *F*-Race using FFD is denoted as FFD/*F*-Race. However, FFD is restricted due to its drawbacks. First, it requires practitioners to determine the levels of each parameters. More importantly, the number of candidates grows exponentially with the number of parameters. Therefore, when the number of parameters is large, this is impractical and computationally prohibitive. Consequently, FFD is usually limited to the case with small number of parameters and reasonable number of levels for each parameter.

Birattari *et al.* [53] showed that *F*-Race with candidates generated by an RSD is significantly better than FFD/*F*-Race in many applications. In RSD, the values of parameters are sampled according to some probability model defined over the parameter space. Usually, priori information is unavailable, and in this case the probability model is set to an uniform distribution. The *F*-Race using RSD is denoted as RSD/*F*-Race. Using RSD in *F*-Race has two main advantages: 1) no priori definition of levels for numerical parameters is required and 2) an arbitrary number of candidates can be generated while also uniformly covering the configuration space [51].

In [31], *F*-Race was compared with other racing algorithms and brute-force approach on two tuning problems. In the first tuning problem, the iterated local search (ILS) [54] is tuned on quadratic assignment problem [55]. And in the second one, the ACO [9] is tuned on TSP [56]. The results showed that *F*-Race is an effective and convenient method to solve parameter tuning problems of metaheuristics. Besides above two applications, the *F*-Race algorithm has been adopted in a number of other studies, as briefly mentioned in [31]. The race package[1] implemented in programming language *R* by Birattari [32] and a simple documentation[2] can be found from the websites.

F-Race uses the computational power more efficiently than repeated evaluation in brute-force approach. It also can stop the search process by itself, i.e., stop when only one configuration left. However, if the target algorithm has a large number of parameters and/or each parameter has a wide range of possible values, a very large number of candidate configurations should be evaluated to obtain high-quality results. In such cases, the adoption of *F*-Race could become impractical or computationally prohibitive. This is a common drawback of simple generate-evaluate methods due to generating candidate configurations in one-stage and without using priori information about the importance and interaction of algorithm parameters in generating step.

C. Remarks on Simple Generate-Evaluate Methods

Simple generate-evaluate methods, as its name implies, are easy to understand and implement. Brute-force and *F*-Race methods are able to tune both numerical and categorical parameters. As the most basic approach, the brute-force approach usually serves as a baseline in experimental evaluation of other parameter tuning algorithms [31], even though it is computationally expensive. *F*-Race method provides a promising approach for evaluating candidate configurations and results in significant computational resource savings. In simple generate-evaluate methods, all candidate configurations are initially generated at once by FFD, RSD, or other sampling strategies, and usually no priori information about the importance and interaction of algorithm parameters is used in candidate configurations generating step, that is, the set of candidates are generated uniformly, i.e., without bias. In order to obtain acceptable optimal configurations, the number of candidates should be large enough. When parameter space is large, this number will increase tremendously. This leads to a huge number of runs in evaluating phase and, thus, is computationally expensive even prohibitive. Therefore, simple generate-evaluate methods are not suitable for parameter tuning problems with large configuration space.

IV. Iterative Generate-Evaluate Methods

Iterative generate-evaluate methods are iterative tuners, that is, unlike simple generate-evaluate methods, they repeatedly execute the generate and evaluate steps, as illustrated in Fig. 2(b). More importantly, the information gathered from previous iteration(s) is used to guide the generating (or sampling) of new candidate configurations for next iteration. It is desirable to generate candidates around the promising region of the configuration space in this way. In other words, by taking advantage of history information, iterative generate-evaluate methods could explore parameter space more effectively than simple generate-evaluate methods which usually evenly sampling candidates within the space. Thus, iterative tuners mitigate the main drawback, that is, inefficiency in case of large parameter space, of noniterative tuners presented in the last section.

[1] https://cran.r-project.org/src/contrib/Archive/race/

[2] http://ftp.uni-bayreuth.de/math/statlib/R/CRAN/doc/packages/race.pdf

According to the strategies of generating candidate configurations, i.e., the sampling strategies, iterative generate-evaluate methods can be further divided into the following.

1) Experimental design-based tuning approaches, where experimental design techniques are used to set parameter values, i.e., to generate candidate configurations.
2) Numerical optimization-based methods, where derivative-free numerical optimization algorithms are taken as sampling strategies to create candidate configurations.
3) Heuristic search-based methods, where new candidate configurations are generated by some heuristic rules, such as the crossover and mutation in EAs.
4) Model-based optimization approaches, where a model that describes the relation between parameter configurations and algorithm performance is used to assist the evaluation of candidates and to guide the sampling of new candidates.

Apparently, iterative generate-evaluate methods include the largest class of current parameter tuning algorithms. This section devotes to this category of tuners. Specifically, above four subcategories of tuning approaches are reviewed from Sections IV-A–IV-D, respectively.

A. Experimental Design-Based Tuning

The DOE is a well-established method to plan experiments so that desired data are collected and can be analyzed by statistical methods to draw valid conclusions [57]. In simple generate-evaluate methods, DOE has been used to generate candidate configurations. Nevertheless, DOE can be used beyond the candidates generating. Based on experimental analysis, for instance, one can locate the promising region of search space, analyze the effects of parameter values changing, screen and rank the importance of parameters. This section reviews some tuning approaches that are based on experimental designs and use analysis of experiments to guide the search process.

1) CALIBRA: Adenso-Díaz and Laguna [5] developed an iterative tuning algorithm called CALIBRA which employs DOE coupled with a local search procedure. In CALIBRA, the experimental designs helps the local search to focus on promising region of the parameter space (search space). Before the local search starting, a DOE, that is, a two-level FFD, is used to identify a start point for the search. Apart from keeping the local search starting on promising region, CALIBRA uses experimental analysis to narrows the search space and initiates the next round of experiments. In the local search, optimal configuration is found by incrementally narrowing the range of each parameter through experimental analysis. Specifically, for each iteration of the local search, a Taguchi's fractional experimental design [58] is carried out using the bounds and midpoint of the narrowed range. This is repeated until local optimum criteria are satisfied. If computational budget is available, new local search could be executed to provide more local optimal solutions. The best parameter setting found is the output best solution. For stochastic algorithms, CALIBRA uses repeated evaluation to establish the performance measures of each experiment point (candidate configuration).

CALIBRA is a representative DOE-based tuning method and can focuses on promising region quickly owing to the combining of experimental designs and local search. However, due to the Taguchi's fractional experimental design [the $L_9(3^4)$] is used, CALIBRA can only handle up to five parameters. Additionally, this approach focuses on the main effects of parameters without exploiting the interaction effects between parameters [5], [59]. These two drawbacks limit the application of CALIBRA to situations, where the number of parameters is small (up to five) and the interactions among parameters are negligible.

2) Other DOE-Based Tuning Approaches: Based on experimental designs, Gunawan *et al.* [59] proposed a parameter tuning framework consists of three phases, i.e., screening, exploration, and exploitation. Given a number of parameters of the target algorithm, the screening phase uses a factorial experiment design to rank these parameters so that unimportant parameters, whose values have insignificant impact on the performance of target algorithm, are determined. Values of unimportant parameters can be set as constants and thus reduce the configuration space to be explored. Then, in the exploration phase, a first-order polynomial model is constructed to identify the promising ranges for important parameters. Finally, in the exploitation phase, the promising ranges of important parameters are sent to an automated tuning configurator, such as ParamILS [28] to find the optimal parameter configuration.

Moreover, Gunawan *et al.* [60] improved the efficiency of above tuning framework by adding a parameter space decomposition step at the beginning. The decomposition step aims at reducing the parameter space such that the number of candidates in experimental designs would decreases. In [60], the so-called Resolution IV Design [57] was used to separate the main effects and interactions of parameters.

B. Numerical Optimization-Based Tuning

When all the parameters of target algorithm are numerical, i.e., real or integer valued, the tuning problem could be solved by numerical optimization techniques along with an evaluation method. Integer parameters can be handled by the continuous optimizer through rounding. Yuan *et al.* [34] presented research on derivative-free numerical optimization algorithms for tuning numerical parameters. In that work, bound optimization by quadratic approximation (BOBYQA) [61] was combined with repeated evaluation, and mesh adaptive direct search (MADS) [62] was combined with both repeated evaluation and F-Race to solve parameter tuning problems. We call these methods that combine numerical optimizer and evaluation methods as numerical optimization-based tuning or tuners.

In general, a numerical optimizer can be combined with the repeated evaluation or F-Race to form a tuner. However, for those numerical optimizers that, in each iteration, only one candidate is generated and selection is not required, the F-Race evaluation method is not feasible, only the repeated evaluation is available [34]. The repeated evaluation and F-Race

both can be coupled with numerical optimizers that generate multiple candidates and select the best out of them in each iteration. For computational cost savings, *F*-Race is preferred in this case. From the generate-evaluate principle perspective, numerical optimization-based tuners are apparently iterative generate-evaluate methods, where candidate configurations are generated according to numerical optimizer's search strategies.

Numerical optimization-based tuning methods are easily acquired by combining the existing powerful black-box optimizer and evaluation methods, such as racing or repeated evaluation methods. Practitioners can choose a familiar optimization algorithm to form a tuner and then adopt it to solve their tuning problems. The common limitation is that most of this kind of tuners are only applicable to numerical parameters (cannot deal with categorical parameters) since the majority of derivative-free optimizers only deal with numerical optimization problems.

C. Heuristic Search-Based Methods

Iterative tuning algorithms that use some heuristic rules to generate new candidate configurations are referred to as heuristic search-based methods. The goal of using heuristic rules is to effectively create candidates so that the promising region could be quickly identified and then be focused on. In this way, the tuning algorithm could find good configuration fast. Many of the existing tuning algorithms, including iterated *F*-Race, ParamILS, Meta-EAs and HORA, fall in this category.

1) Iterated F-Race: The main challenge and drawback of *F*-Race is that, when the tuning problem has a large number of parameters and/or each parameter has a wide range of possible values, the number of candidate configurations to be evaluated should be quite large, in order to obtain high-quality solutions. In such cases, *F*-Race method becomes impractical and computationally prohibitive. To alleviate this problem, Balaprakash *et al.* [63] proposed the iterative application of *F*-Race, which is abbreviated to iterated *F*-Race or *I*/*F*-Race and showed its effectiveness through examples on MAX–MIN ant system (MMAS) [64], estimation-based local search [65], and SA algorithm.

The iterated *F*-Race, as its name suggests, use an iterative procedure to find optimal parameter settings. Specifically, in each iteration, first, a set of candidate configurations are generated according to a probabilistic model, then a standard *F*-Race is performed on the candidate set and the survived candidates are used to update the probabilistic model which will be used in next iteration. This cycle is repeated until the stop condition, such as maximum computational budget, is satisfied. It is hopeful to focus candidate configurations around the promising region by using survived candidates to bias the sampling of new candidates [38]. The efficiency of the search procedure is, thus, improved by this way. Details about how to update the probabilistic model in iterated *F*-Race could be found in [51] and [63].

Recently, López-Ibáñez *et al.* [30] provided a software package implemented in programming language *R*, called irace, that includes the iterated *F*-Race algorithm as well as several of its extended and improved variants. This irace package could be found along with user guide document and usage examples from its website.[3] In [66], iterated *F*-Race method was applied to improve the performance of the CMA-ES algorithm with increasing population size (iCMA-ES) [67] on CEC'05 benchmark set [68]. The results showed that the performance of iCMA-ES was significantly improved by automatic parameter tuning procedure. Later, Liao *et al.* [69] employed iterated *F*-Race method to tune seven high-performing continuous optimizers on two different benchmark sets (CEC'05 [68] and SOCO [70]) and also concluded that the performance of tested continuous optimizers were improved by parameter tuning.

Iterated *F*-Race has become one of the competitive automatic parameter tuning approach. It can handle both numerical and categorical parameters. However, it has a few limitations. First, iterated *F*-Race was not primarily designed for reducing computation time [30]. So that, the time-saving techniques, such as sharpening and adaptive capping, is not involved in current iterated *F*-Race method. Moreover, to obtain acceptable results, an adequate number of iterations should be implemented, in other words, a sufficient number of candidate configurations need to be sampled and evaluated. Thus, if the tuning budget is too small, the resulting configuration of iterated *F*-Race might be poor.

2) Meta-EAs: Since parameter tuning problem is a meta-optimization problem, apparently, EAs can be used as meta-EA (meta-EAs) to solve tuning problems. In a meta-EA, its individuals, i.e., numeric vectors, represent parameter configurations and the performance measures established by evaluation method (such as repeated evaluation or *F*-Race) of each configuration are related to the corresponding (meta-) fitness [48]. Meta-EA was first introduced by Mercer and Sampson [37]. Greffenstette [15] conducted experiments with meta genetic algorithm (meta-GA) and showed its effectiveness.

In [48] the CMA-ES [23], which is the state-of-the-art improvement of evolution strategies for numerical optimization, was adopted as a meta-EA and the repeated evaluation was used to establish performance measures (fitness of meta-EA) of each candidate configuration. The CMA-ES was also used as meta-EA by Yuan *et al.* [34] where both repeated evaluation and *F*-Race were taken as evaluation methods.

The so-called gender-based genetic algorithm (GGA) for automatic parameter tuning was introduced by Ansótegui *et al.* [71]. The GGA uses the concept of competitive and noncompetitive genders [72], [73] in generating candidates configurations. In each generation of GGA, the whole population is divided into two subpopulation with different genders, i.e., competitive and noncompetitive. The individuals (i.e., candidate configurations) in competitive population are evaluated on the set of training instances and compete for the right of mating. The fittest individuals (candidates), that is, candidates that yield better performance, are then mated with the noncompetitive population to generate new candidate configurations [71], [74].

[3] http://iridia.ulb.ac.be/irace/

The parameter relevance estimation and value calibration (REVAC) method was introduced by Nannen and Eiben [75], [76] based on the idea of solving parameter tuning problem by estimating parameter relevance with normalized Shannon entropy [78]. The REVAC is considered a meta estimation of distribution algorithm (meta-EDA) [77] since it use the same general idea as EDA [79], that is, estimating the distribution of promising parameter values [80]. REVAC algorithm is an iterative process that consists of estimating the distributions of promising parameter values for each parameter within the configuration space, and generating parameter configurations by drawing values from these distributions. These parameter distributions are updated after evaluating newly drawn candidates [48]. The tuning process terminates until the maximum number of tested (or evaluated) candidate configurations is reached.

Meta-EAs are automated search methods to identify good parameter settings for metaheuristics. They have the potential to reach the global optimum within parameter space since they are based on EAs, which are well-known global optimizers. It is also important to remark that meta-EAs can stop at any point of the search process and return the current best configuration (incumbent) as a solution. However, the similar as EAs, meta-EAs usually require a large number of evaluations of candidate configurations to obtain desirable solutions. This is not practical when parameter space is large. Additionally, many meta-EAs, such as REVAC and meta-CMA-ES, cannot handle categorical parameters. They are the two main weaknesses of meta-EAs for parameter tuning.

3) ParamILS: ParamILS, which was proposed by Hutter *et al.* [28], [81], is an automatic parameter tuning framework. It combines stochastic local search method with specific mechanisms which exploit some properties of parameter tuning problem [11]. The core of ParamILS is the ILS [54], which is a versatile and well-known stochastic local search method [82]. ILS involves a main loop consist of three components: 1) the perturbation of current best solution to scape from the local optima; 2) the local search procedure to find optima from given start point; and 3) the acceptance criterion to determine whether the currently obtained solution is kept or rejected [54].

ParamILS employs the ILS method to search the optimal parameter setting for target algorithm within its configuration space. It starts the search from the best parameter configuration out of the combination of default configuration and a number of randomly generated configurations. This follows a subsidiary local search procedure, that is, iterative first improvement process that searches a one-exchange neighborhood, where configurations differ in the value given to exactly one parameter, i.e., only one parameter value is changed at a time. Once a local optimal configuration is identified, the main loop of the ILS is entered, which involves three steps: 1) a perturbation step that changes randomly the parameters values; 2) an execution of the iterative improvement process; and 3) an acceptance criterion that decides from which configuration to continue the search process [28]. In local search procedure of the ILS, one needs to test the two candidate configurations and determine which one is preferred, i.e., the evaluation of parameter configurations with consideration of handling the stochasticity of tuning problem.

In ParamILS framework, Hutter *et al.* [28] proposed different ways to evaluate parameter configurations for determining which configuration should be preferred. The most simple and intuitive approach, which is called BasicILS, is to evaluate every candidate configuration by running it on same problem instances (training instances) with the same random number seeds. FocusedILS, which is a variant of BasicILS, uses the intensification mechanism (described in Section II-B) that adaptively changes the number of training instances during evaluating candidates. For situations, where the performance metric is the computation time, i.e., the goal of parameter tuning is to find the optimal parameter setting that minimizes the computation time, ParamILS can use the so-called adaptive capping mechanism, which bounds the execution time of configurations according to the observed performance of the current best configuration and early prune or stop the poor configurations, to reduce the computational cost. The software of ParamILS implemented in programming language Ruby, which is free for academic use, can be found from the website,[4] where a quick start guide could also be downloaded.

Recently, Cáceres and Stützle [83] employed the variable neighborhood search (VNS) [84] mechanism as an alternative for the one-exchange neighborhood in the local search procedure of ParamILS. In this paper, the authors adopted the reduced VNS (RVNS), where various neighborhoods are explored randomly by changing the neighborhood to be explored in a systematic way as it is common in VNS, rather than the one-exchange neighborhood local search in original ParamILS. It has been shown that the search of good algorithm configurations can profit from RVNS' ability of exploring different and also larger neighborhoods but also enabling the intensification of the search when is required.

ParamILS is one of the state-of-the-art automatic parameter tuning methods. It is able to tune both numerical and categorical parameters. ParamILS, like meta-EAs, can be stopped at almost any point in the search process and provide a good quality parameter configuration. Since the neighborhood search is applied, ParamILS requires the discretization for each parameters to define the neighborhood of candidate configurations. This is not so easy-to-handle as meta-EAs, SPO, and SMAC (will be described in the next section) which just require the definition of ranges of values for each parameter.

4) HORA: The so-called heuristic oriented racing algorithm (HORA) introduced by Barbosa and Senne [85], [86] is a relatively new heuristic search method for parameter tuning. HORA is an iterative algorithm which dynamically creates candidate configurations and uses the racing method to evaluate them. Thus, HORA is apparently an iterative generate-evaluate method.

At the start of the HORA tuning process, a number of instances (n instances) are selected arbitrarily from the given set of problem instances. The experimental studies are then performed on each of the selected instances (also called training instances) using the response surface methodology (RSM)

[4]http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/

to identify the best (or promising) parameter settings for each training instance. The experimental studies, thus, result in n different settings, each one being related to a training instance. These identified parameter settings ensure diversity of the parameters and they are used to define the upper and lower bounds of each parameter. After that, HORA enters an iterative procedure consisting of: 1) dynamically creating new candidates in the neighborhood of some best known candidate configurations, i.e., configurations that are preferred in evaluation and 2) evaluating the set of candidate configurations with racing method to discard poor ones according to the statistical evidences [85], [86]. By this repeated procedure, HORA algorithm consistently finds better candidate configurations.

In [85] and [86] a case study about HORA, in which a brute-force and racing methods were also considered for comparison, was performed on GA and SA with instances selected from the OR-Library [87]. The results of the case study showed that HORA achieve similar (or even better) results but with much lower computational cost compared to other approaches. This verified the effectiveness of HORA and indicated it is a fast tuner.

It seems like that HORA and iterated F-Race are similar since they are both iterative tuners that involve iteratively generating new candidates and evaluating candidates with racing method. However, the main difference between them is the way of generating candidate configurations, that is, HORA creating new candidates in the neighborhoods of some good configurations, while iterated F-Race generating candidates according to a probabilistic model. Additionally, HORA uses DOE to establish a set of initial configurations and narrow the original search space of the parameters.

HORA is a relatively new method that combines experimental study, neighborhood search and racing techniques. Current studies in [85] and [86] show that HORA is a promising method for parameter tuning, but comprehensive study on more problems is still required. Since the experimental study based on RSM is used in HORA to initially identify the best parameter settings and to narrow the search space, the search speed could be improved. On the other side, if the landscape of algorithms performance respect to its parameter settings is complex or multimodal, using the simple RSM to identify promising region may lead to the search of HORA focusing on wrong region.

D. Model-Based Optimization Approaches

The use of response surface model, also called surrogate model, is promising and popular in dealing with complex real-world optimization problems, especially expensive optimization problems. This is commonly known as model-based optimization methods [88]. Model-based parameter tuning approaches adopt model-based optimization methods to address tuning problem. They build response surface or surrogate models to describe (or model) the dependence of target algorithm's performance on its parameter settings and then use these models to find good parameter settings for the target algorithm. The existing model-based tuning approaches can be described in a unified framework that involves an iterative procedure consisting of constructing models and using them to determine candidate configurations to be investigated or tested [89]. Apparently, model-based tuners are iterative procedures to tackle parameter tuning problems. The distinctive difference distinguishing model-based tuners from previously presented iterative tuning methods is that, in model-based tuning methods, the candidate configurations to be tested or investigated are determined-based response models which provides desirable information to address the tradeoff between exploration and exploitation.

Model-based parameter tuning approaches could be viewed as extensions of the influential model-based optimization method, that is, the efficient global optimization (EGO) [90] which combines the predictive model (Kriging or Gaussian process model), i.e., design and analysis of computer experiments (DACE) [91], with sequential sampling strategies [commonly based on the expected improvement (EI) criterion], which is used to identify the most promising next design point. The commonly known model-based tuning approaches, including the sequential parameter optimization (SPO) [92] and the more sophisticated sequential model-based algorithm configuration (SMAC) [47].

1) SPO: The SPO procedure was introduced by Bartz-Beielstein *et al.* [92], [93] to optimize algorithm performance. SPO extended EGO to tackle parameter tuning problem by coupling with special techniques to handle stochasticity of tuning problem. SPO starts with a set of initial design points (parameter settings or configurations) which are generated by an LHD [94]. Due to the stochastic nature of the target algorithm, performance for each design point (configuration) is evaluated by means of repeated evaluation. The best configuration from the initial set is chosen as the initial incumbent. Based on the set of parameter settings and their corresponding performance measures, SPO constructs a response surface model called Kriging model [89]. Then, a new set of design points (configurations) are generated and tested using the constructed model. The most promising points (configurations), which have the highest EI, are chosen as new candidate configurations for next iteration. The selected new candidates will be evaluated and compared with the current incumbent to determine the new incumbent through the intensification mechanism [88]. With newly evaluated points, the model can be updated and then used in next iteration. This process is repeated until termination criterion, such as the maximum number of iterations or number of repetitions for the best configuration, has fulfilled.

Sequential Kriging optimization (SKO) [95] is another extension of EGO to noisy function optimization and can be used to parameter tuning problem. Unlike in SPO, SKO uses Gaussian process regression to fit noisy response data, i.e., the algorithm performance metric values, directly. Besides model fitting, SKO and SPO are different in selection of incumbent, generating new parameter settings, and intensification mechanisms. Bartz-Beielstein and Zaefferer [88] have demonstrated the superiority of SPO over SKO, and thus SKO appears rarely in parameter tuning domain. Hutter *et al.* [46], [89] later presented a new version of SPO named SPO+, where log-transformed response data, new EI criterion, and new

intensification procedure were used. This new version performed more robustly and given better results. In addition, in order to avoid the up-front cost of initial design, the time-bounded SPO (TB-SPO) [96] was introduced as another extension of SPO. SPO and its variants are sophisticated model-based parameter tuning approaches for target algorithms with only numerical parameters on single problem instance. The SPO Toolbox (SPOT) package of SPO method implemented in programming language *R* and its reference manual are available in the SPOT website.[5] Further description of this SPOT package and its usage can also be found in [97] and [98].

2) SMAC: With the aim of removing the two main limitations of SPO and its variants, these are, only support numerical parameters and only optimizes algorithm performance for single instance, the more sophisticated SMAC method [47] was introduced to address general parameter tuning problems.

First, to handle categorical parameters of target algorithm, SMAC uses random forest [99], which is a machine learning method for regression and classification, instead of Kriging in SPO. Random forests are collections of regression trees, which are known to perform well for categorical input data. Thus, random forests share the benefit of regression trees and typically yield more accurate predictions [100]. Furthermore, they also provide quantification of uncertainty in a given prediction.

Another aspect is to extend the model to handle multiple instances so that the tuning approach can optimize the performance of target algorithm on a set of problem instances instead of only on a single instance. In SMAC, information about the instances is explicitly integrated into the response surface model, i.e., the random forests model. Therefore, SMAC learns a joint model that can predicts performance of target algorithm for combinations of parameter settings and instances features. And these predictions are then aggregated across instances to give a statistic performance metric on each parameter configuration.

SMAC also uses the EI criterion as used in [46] to select promising candidate configurations in parameter space. Unlike in SPO, it performs a simple multistart local search to find configurations with maximal EI and considers all resulting configurations as promising candidates for next iteration. With above main modifications and extensions, SMAC can tackle general algorithm parameter tuning problems and yields very good results.

The experimental study that compares the efficacy of SMAC, TB-SPO, GGA, and ParamILS for a range of parameter tuning problems that involve minimizing the runtime of the SAT (propositional satisfiability problem) solver SAOS and SPEAR and MIP (mixed integer programming) solver IBM ILOG CPLEX, was performed in [47]. The empirical results demonstrated that, overall, SMAC yielded statistically significant improvements over the compared approaches. The source code of SMAC in Java[6] and in Python[7] are available for free download. Documentations for both versions are provided and can be found from the websites.

Model-based optimization approaches are efficient methods to solve parameter tuning problems. SPO has shown the potential and efficiency of model-based tuning approach, but SPO and its variants have two main limitations, these are, only support numerical parameters and only optimizes algorithm performance for single instance. The sophisticated SMAC method can tackle both numerical and categorical parameters, and it is especially good at solving tuning problems with many categorical parameters. To the best of our knowledge, SMAC is the currently most powerful automatic parameter tuning method.

E. Remarks on Iterative Generate-Evaluate Methods

Iterative generate-evaluate methods, generally, alleviate the computational burden by iteratively generating a small number of candidate configurations rather than initially generating a large number of candidates in simple generate-evaluate methods. This category is the most fruitful class of approaches for tuning problems. Iterative tuners, like iterated *F*-Race, ParamILS, and SMAC are the state-of-the-art automatic parameter tuning approaches. The numerical optimization methods and heuristic search methods could be modified to solve parameter tuning problem by combining proper evaluation methods. Effective sampling methods, which balance the exploration and exploitation and thus quickly focus on the promising region, are still needed. Prescreening and narrowing the parameter space with small amount of computational resource is advisable for improving the efficiency of tuning process. The use of response model or surrogate model is helpful to reduce the evaluation cost and guide the sampling of new candidate configurations.

V. High-Level Generate-Evaluate Methods

The high-level generate-evaluate methods refer to a recently new category of tuning methods that uses more advanced (high-level) approaches to generate candidate configurations and then carefully evaluates these candidates for selecting the best parameter configuration, as illustrated in Fig. 2(c). In parameter tuning problem, a large number of parameter configuration evaluations, which involve performing a number of runs for each configuration on a set of problem instances, is usually required. This is computationally expensive, in particular when repeated evaluation method is used in tuning algorithm or the size of problem instances is very large. High-level generate-evaluate methods attempt to solve tuning problem in an advanced generate-evaluate framework and to cut down the computational cost.

The essential idea of high-level generate-evaluate methods is to quickly generate a set of elite or high-quality parameter configurations (through coarse evaluations) with small amount of computational resources, and then to carefully select the best one from this set instead of evaluating each candidate configuration thoroughly from the very beginning. By this way, computational resources are saved in exploring the parameter space and identifying promising parameter configurations, and

[5]https://cran.r-project.org/web/packages/SPOT/index.html
[6]http://www.cs.ubc.ca/labs/beta/Projects/SMAC/
[7]https://github.com/automl/SMAC3

thus more resources are available for carefully evaluating of elite configurations. The post-selection mechanism [33], [34] was designed under this direction to solving tuning problems.

A. Post-Selection Mechanism

The post-selection mechanism divides the parameter tuning process into two phases, namely, elite qualification phase and elite selection phase [33]. In elite qualification phase, a number of high-quality or elite candidate configurations are identified by running some tuning algorithms. Then, in elite selection phase, these elite configurations are evaluated thoroughly and the best one is carefully selected. The initial results in [34] showed that, owing to a careful elite selection phase, the post-selection mechanism allows a more rough assessment of candidate configurations during elite qualification phase, i.e., the evaluation of most candidates can be performed with fewer training instances. Therefore, more candidate configurations can be generated (as the total budget of tuning process is limited), and potentially better configurations may be found. Consequently, elite configurations can be collected by running a quick tuner with restart or executing different tuners simultaneously. And in elite selection phase, a racing method or intensification mechanism could be used to select the best configuration. Yuan *et al.* [33] carried out analysis of post-selection mechanism and suggested that this mechanism was helpful to improve automatic parameter tuning methods.

Post-selection method provides a new idea to find good parameter settings by using the existing tuners as generators to create or identify elite candidate configurations and then selecting the best one from them after careful evaluation. This method can easily provide a number of elite configurations rather than only providing the best one. The key problem in post-selection method, however, is to keep diversity of elite candidates in elite qualification phase, so that the global optimum or adequate high-quality parameter setting could be selected finally.

B. Remarks on High-Level Generate-Evaluate Methods

The idea of high-level generate-evaluate methods for parameter tuning is relatively new and there are few researches in this direction. It could be found that the existing tuning approaches can be taken as high-level strategies to systematically generate elite configurations by running them with low computational budget and with restart mechanism. New methods for quickly generate high-quality candidate configurations are of course highly desirable. Racing methods and other intensification mechanisms can be used to carefully evaluate the set of elite configurations. Besides the effectiveness of high-level generate-evaluate methods, another advantage is that they can easily provide a number of high-quality alternative configurations except giving the best one. The positive results already obtained indicate that the idea of this category methods is promising and worth pursuing.

VI. Future Research Prospects and Concluding Summary

A. Future Research Prospects

Based on the development history of automatic parameter tuning methods and the current research status of this field, the following possible research directions are pointed out.

1) Enhance the Efficiency of Solving Parameter Tuning Problems: Since tuning process is usually computational expensive, especially for real-world application problems, it is very important to improve its efficiency, i.e., to reduce computational cost. Generally speaking, there are two possible ways to achieve this goal: 1) to reduce the total number of candidate configurations that are tested and 2) to cut down the average cost of evaluating a configuration. The first way implies to use advanced sampling strategies to generate candidate configurations such that the configuration space could be explored effectively and quickly focuses on promising region of the space. While, the second calls for efficient evaluation methods to test candidate configurations and to identify the elite ones rapidly.

2) Establish Benchmark Test Suite and Easy-to-Use Algorithm Tuning Toolbox: With the development and maturity of algorithm parameter tuning field, a benchmark set of standardized tuning problems and an open algorithm parameter tuning toolbox that allows for simple usage and integration of new tuning methods are highly desirable. Such benchmark set and tuning toolbox would facilitate the empirical studies of tuning algorithms, and reduce barriers faced by new researchers and practitioners in the community. Until now, only the AClib,[8] a library of algorithm parameter tuning (or configuration) benchmarks, has been introduced by Hutter *et al.* [101]. There is still need for more standardized benchmarks that includes more tuning problem from different domain. As mentioned in the previous sections, open source code of *F*-Race, Iterated *F*-Race, ParamILS, SPO, and SMAC are available separately. Those packages do not offer GUI (graphical user interface) visual front-ends for users. They are used through the command lines or scripts in supported programming language (such as Java, Python, or R), which requires users to have adequate programming skills. Therefore, an open and easy-to-use parameter tuning toolbox that allows for simple usage and integration of new tuning methods is in demand. Although developing an algorithm tuning toolbox that includes all the existing tuning approaches is a challenging task, this has very important academic and practical value. Such a toolbox is expected by researchers and end users.

3) Research on Multiobjective Tuning Approaches: Until very recently, almost all the existing tuning approaches are designed to optimize a single performance measure or metric of the target algorithm, such as the solution quality or run time. However, in some case, one may expect to optimize more than one performance metrics simultaneously. This is a new challenging and direction that has not deeply discussed in parameter tuning. Recently, multiobjective racing algorithm has already been newly presented by Zhang *et al.* [102]–[104] for model selection in machine learning, and also has been applied

[8] http://www.aclib.net/

to identify the Pareto optimal parameter settings of ACO algorithm on TSP [103] and artificial bee colony algorithms on numerical optimization problems [104]. Blot *et al.* [105] recently introduced MO-ParamILS, a multiobjective extension of ParamILS, for multiobjective parameter tuning problems. To the best of our knowledge, parameter tuning for multiple performance objectives has been studied only recently and there are few research achievement. Thus, this new subtopic of parameter tuning need to further study and development in the future.

VII. Conclusion

This paper provided a survey of the existing automatic parameter tuning methods for metaheuristics. A common introduction was first given and followed by the general statement of parameter tuning problem. A new classification of tuning approaches was introduced according to tuners' structure or framework. The existing tuning methods were classified into three main categories: 1) simple generate-evaluate methods; 2) iterative generate-evaluate methods; and 3) high-level generate-evaluate methods. Parameter tuning approaches as far as we are aware from literature were, then, reviewed category by category. After the description of each tuning approach, its main strengths and weaknesses were briefly stated, which is helpful for new researcher or practitioners to select appropriate tuner to solve problem at hand. Last but not least, some directions were pointed out for future research. In the conclusion, a comprehensive survey of automatic parameter tuning methods was given for researcher or practitioner of this field.

In the end, it is necessary to highlight the significance of applying automated tuning methods to properly set algorithm parameters. Many works have proved the significant improvement in algorithm's performance with the aid of automatic parameter tuning. In addition to developing new approaches, it is highly desirable to apply frequently the existing tuning methods in industry and research applications, so that the benefits of automatic parameter tuning can be taken adequately, and new challenges may appear and could be figured out.

References

[1] R. K. Arora, *Introduction in Optimization*. Boca Raton, FL, USA: Taylor & Francis, 2015, pp. 1–34.

[2] E.-G. Talbi, "Common concepts for metaheuristics," in *Metaheuristics: From Design to Implementation*. Hoboken, NJ, USA: Wiley, 2009, pp. 1–86.

[3] D. P. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*. New York, NY, USA: Prentice-Hall, 1994.

[4] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci.*, vol. 237, pp. 82–117, Jul. 2013.

[5] B. Adenso-Díaz and M. Laguna, "Fine-tuning of algorithms using fractional experimental designs and local search," *Oper. Res.*, vol. 54, no. 1, pp. 99–114, Feb. 2006.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[7] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, Jan. 1986.

[8] L. D. Davis, K. D. Jong, M. D. Vose, and L. D. Whitley, *Evolutionary Algorithms* (The IMA Volumes in Mathematics and Its Applications), vol. 111. New York, NY, USA: Springer, 1999.

[9] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[10] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. ICNN*, Perth, WA, Australia, 1995, pp. 1942–1948.

[11] H. H. Hoos, "Automated algorithm configuration and parameter tuning," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds. Heidelberg, Germany: Springer, 2012, pp. 37–71.

[12] D. Weyland, "Simulated annealing, its parameter settings and the longest common subsequence problem," in *Proc. GECCO*, Atlanta, GA, USA, 2008, pp. 803–810.

[13] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[14] L. Calvet, A. A. Juan, C. Serrat, and J. Ries, "A statistical learning based approach for parameter fine-tuning of metaheuristics," *SORT Stat. Oper. Res. Trans.*, vol. 1, no. 1, pp. 201–224, 2016.

[15] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 16, no. 1, pp. 122–128, Jan. 1986.

[16] J. N. Hooker, "Testing heuristics: We have it all wrong," *J. Heuristics*, vol. 1, no. 1, pp. 33–42, Sep. 1995.

[17] D. Johnson, "A theoretician's guide to the experimental analysis of algorithms," in *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*. Providence, RI, USA: Amer. Math. Soc., Dec. 2002, pp. 215–250.

[18] A. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.

[19] E. S. Skakov and V. N. Malysh, "Parameter meta-optimization of metaheuristics of solving specific NP-hard facility location problem," *J. Phys. Conf. Series*, vol. 973, Mar. 2018, Art no. 012063.

[20] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Heidelberg, Germany: Springer, 2015.

[21] G. Karafotias, M. Hoogendoorn, and A. E. Eiben, "Parameter control in evolutionary algorithms: Trends and challenges," *IEEE Trans. Evol. Comput.*, vol. 19, no. 2, pp. 167–187, Apr. 2015.

[22] S. Meyer-Nieberg and H.-G. Beyer, "Self-adaptation in evolutionary algorithms," in *Studies in Computational Intelligence, Parameter Setting in Evolutionary Algorithms*, F. G. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Heidelberg, Germany: Springer, 2007, pp. 47–75.

[23] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms* (Studies in Fuzziness and Soft Computing), vol. 192, J. A. Lozano, Ed. Heidelberg, Germany: Springer, 2006, pp. 75–102.

[24] P. A. Consoli, Y. Mei, L. L. Minku, and X. Yao, "Dynamic selection of evolutionary operators based on online learning and fitness landscape analysis," *Soft Comput.*, vol. 20, no. 10, pp. 3889–3914, 2016.

[25] P. Consoli and X. Yao, "Diversity-driven selection of multiple crossover operators for the capacitated arc routing problem," in *Evolutionary Computation in Combinatorial Optimisation*, C. Blum and G. Ochoa, Eds. Heidelberg, Germany: Springer, 2014, pp. 97–108.

[26] X. Wu, P. Consoli, L. Minku, G. Ochoa, and X. Yao, "An evolutionary hyper-heuristic for the software project scheduling problem," in *Parallel Problem Solving From Nature—PPSN XIV* (Lecture Notes in Computer Science), J. Handl *et al.*, Eds. Cham, Switzerland: Springer Int., 2016, pp. 37–47.

[27] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith, "Parameter control in evolutionary algorithms" in *Studies in Computational Intelligence, Parameter Setting in Evolutionary Algorithms*, F. G. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Heidelberg, Germany: Springer, 2007, pp. 19–46.

[28] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *J. Artif. Intell. Res.*, vol. 36, pp. 267–306, Oct. 2009.

[29] F. Hutter, "Automated configuration of algorithms for solving hard computational problems," Ph.D. dissertation, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, 2009.

[30] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Oper. Res. Perspectives*, vol. 3, pp. 43–58, Sep. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214716015300270

[31] M. Birattari, *Tuning Metaheuristics: A Machine Learning Perspective*. Heidelberg, Germany: Springer, 2009.

[32] M. Birattari, "The race package for R. Racing methods for the selection of the best," IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, Rep. TR/IRIDIA/2003–37, 2003.

[33] Z. Yuan, T. Stützle, M. A. M. de Oca, H. C. Lau, and M. Birattari, "An analysis of post-selection in automatic configuration," in *Proc. GECCO*, Amsterdam, The Netherlands, 2013, pp. 1557–1564.

[34] Z. Yuan, M. A. M. de Oca, M. Birattari, and T. Stützle, "Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms," *Swarm Intell.*, vol. 6, no. 1, pp. 49–75, Mar. 2012.

[35] M. E. H. Pedersen, "Tuning & simplifying heuristical optimization," Ph.D. dissertation, School Eng. Sci., Univ. Southampton, Southampton, U.K., 2010.

[36] B. M. Rudolf, "Parameter tuning for numerical optimization algorithms," M.S. thesis, Dept. Comput. Sci. Eng., Czech Tech. Univ. Prague, Prague, Czech Republic, 2017.

[37] R. E. Mercer and J. R. Sampson, "Adaptive search using a reproductive meta-plan," *Kybernetes*, vol. 7, no. 3, pp. 215–228, Mar. 1978.

[38] M. Birattari, "F-Race for Tuning Metaheuristics" in *Studies in Computational Intelligence, Tuning Metaheuristics*, J. Kacprzyk and M. Birattari, Eds. Heidelberg, Germany: Springer, 2009, pp. 85–115.

[39] F. Dobslaw, "Recent development in automatic parameter tuning for metaheuristics," in *Proc. 19th Annu. Conf. Doctoral Students WDS*, Prague, Czech Republic, 2010, pp. 134–137.

[40] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 19–31, 2011.

[41] A. E. Eiben and S. K. Smit, "Evolutionary algorithm parameters and methods to tune them," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds. Heidelberg, Germany: Springer, 2012, pp. 15–36.

[42] E. Montero, M.-C. Riff, and B. Neveu, "A beginner's guide to tuning methods," *Appl. Soft. Comput.*, vol. 17, pp. 39–51, Apr. 2014.

[43] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.

[44] S. K. Smit and A. E. Eiben, "Parameter tuning of evolutionary algorithms: Generalist vs. specialist," in *Applications of Evolutionary Computation* (Lecture Notes in Computer Science), C. Di Chio *et al.*, Eds. Heidelberg, Germany: Springer, 2010, pp. 542–551.

[45] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *Proc. GECCO*, San Francisco, CA, USA, 2002, pp. 11–18.

[46] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy, "An experimental investigation of model-based parameter optimisation," in *Proc. GECCO*, Montreal, QC, Canada, 2009, pp. 271–278.

[47] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, C. A. Coello Coello, Ed. Heidelberg, Germany: Springer, 2011, pp. 507–523.

[48] S. K. Smit and A. E. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," in *Proc. CEC*, Trondheim, Norway, 2009, pp. 399–406.

[49] O. Maron and A. W. Moore, "Hoeffding races: Accelerating model selection search for classification and function approximation," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. San Mateo, CA, USA: Morgan-Kaufmann, 1994, pp. 59–66.

[50] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artif. Intell. Rev.*, vol. 11, nos. 1–5, pp. 193–225, Feb. 1997.

[51] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-Race and iterated F-Race: An overview," in *Experimental Methods for the Analysis of Optimization Algorithms*, T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, Eds. Heidelberg, Germany: Springer, 2010, pp. 311–336.

[52] A. Dean, D. Voss, and D. Draguljić, "Principles and techniques," in *Springer Texts in Statistics, Design and Analysis of Experiments*, A. Dean, D. Voss, and D. Draguljić, Eds. Cham, Switzerland: Springer Int., 2017, pp. 1–5.

[53] M. Birattari, P. Balaprakash, and M. Dorigo, "The ACO/F-race algorithm for combinatorial optimization under uncertainty," in *Metaheuristics: Progress in Complex Systems Optimization* (Operations Research/Computer Science Interfaces Series), K. F. Doerner, Ed. New York, NY, USA: Springer, 2007, pp. 189–203.

[54] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of Metaheuristics* (International Series in Operations Research & Management Science), vol. 57, F. Glover and G. A. Kochenberger, Eds. Boston, MA, USA: Kluwer Acad., 2003, pp. 320–353.

[55] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis, "The quadratic assignment problem," in *Handbook of Combinatorial Optimization*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA, USA: Springer-Verlag, 2014, pp. 1713–1809.

[56] S. Näher, "The traveling salesman problem," in *Algorithms Unplugged*, B. Vöcking, Ed. New York, NY, USA: Springer, 2011, pp. 383–391.

[57] D. C. Montgomery, *Design and Analysis of Experiments*. Hoboken, NJ, USA: Wiley, 2017.

[58] R. K. Roy, *A Primer on the Taguchi Method*, 2nd ed. Dearborn, MI, USA: Soc. Manuf. Eng., 2010.

[59] A. Gunawan, H. C. Lau, and Lindawati, "Fine-tuning algorithm parameters using the design of experiments approach," in *Learning and Intelligent Optimization*, C. A. Coello Coello, Ed. Heidelberg, Germany: Springer, 2011, pp. 278–292.

[60] A. Gunawan, H. C. Lau, and E. Wong, "Real-world parameter tuning using factorial design with parameter decomposition," in *Advances in Metaheuristics*, L. Di Gaspero, A. Schaerf, and T. Stützle, Eds. New York, NY, USA: Springer, 2013, pp. 37–59.

[61] M. J. D. Powell, "The BOBYQA algorithm for bound constrained optimization without derivatives," Dept. Appl. Math. Theor. Phys., Univ. Cambridge, Cambridge, U.K., Rep. NA2009/06, 2009, pp. 26–46.

[62] C. Audet and J. E. Dennis, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM J. Optim.*, vol. 17, no. 1, pp. 188–217, 2006.

[63] P. Balaprakash, M. Birattari, and T. Stützle, "Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement," in *Hybrid Metaheuristics*, T. Bartz-Beielstein *et al.*, Eds. Heidelberg, Germany: Springer, 2007, pp. 108–122.

[64] T. Stützle and H. H. Hoos, "MAX–MIN ant system," *Future Gener. Comput. Syst.*, vol. 16, no. 8, pp. 889–914, 2000.

[65] M. Birattari, P. Balaprakash, T. Stützle, and M. Dorigo, "Estimation-based local search for stochastic combinatorial optimization using delta evaluations: A case study on the probabilistic traveling salesman problem," *INFORMS J. Comput.*, vol. 20, no. 4, pp. 644–658, Nov. 2008.

[66] T. Liao, M. A. M. de Oca, and T. Stützle, "Computational results for an automatically tuned CMA-ES with increasing population size on the CEC'05 benchmark set," *Soft Comput.*, vol. 17, no. 6, pp. 1031–1046, 2013.

[67] A. Auger and N. Hansen, "A restart CMA evolution strategy with increasing population size," in *Proc. CEC*, Edinburgh, U.K., 2005, pp. 1769–1776.

[68] P. N. Suganthan *et al.*, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," School Electr. Electron. Eng., Nanyang Technol. Univ., Singapore, KanGAL Rep. 2005005, 2005.

[69] T. Liao, D. Molina, and T. Stützle, "Performance evaluation of automatically tuned continuous optimizers on different benchmark sets," *Appl. Soft. Comput.*, vol. 27, pp. 490–503, Feb. 2015.

[70] M. Lozano, D. Molina, and F. Herrera, "Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems," *Soft Comput.*, vol. 15, no. 11, pp. 2085–2087, Nov. 2011.

[71] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *Principles and Practice of Constraint Programming—CP 2009*, I. P. Gent, Ed. Heidelberg, Germany: Springer, 2009, pp. 142–157.

[72] J. Rejeb and M. AbuElhaij, "New gender genetic algorithm for solving graph partitioning problems," in *Proc. 43rd IEEE Midwest Symp. Circuits Syst.*, vol. 1. Lansing, MI, USA, Aug. 2000, pp. 444–446.

[73] J. Lis and A. E. Eiben, "A multi-sexual genetic algorithm for multiobjective optimization," in *Proc. IEEE Int. Conf. Evol. Comput. (ICEC)*, Indianapolis, IN, USA, 1997, pp. 59–64.

[74] Y. Malitsky, *Instance-Specific Algorithm Configuration*. Cham, Switzerland: Springer Int., 2014.

[75] V. Nannen and A. E. Eiben, "A method for parameter calibration and relevance estimation in evolutionary algorithms," in *Proc. GECCO*, Seattle, WA, USA, 2006, pp. 183–190.

[76] V. Nannen and A. E. Eiben, "Relevance estimation and value calibration of evolutionary algorithm parameters," in *Proc. 20th Int. Joint Conf. Artif. Intell.*, Hyderabad, India, 2007, pp. 975–980.

[77] V. Nannen and A. E. Eiben, "Efficient relevance estimation and value calibration of evolutionary algorithm parameters," in *Proc. CEC*, Singapore, Sep. 2007, pp. 103–110.

[78] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2003.

[79] P. Larrañaga and J. A. Lozano, Eds., *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Boston, MA, USA: Springer, 2002.
[80] S. K. Smit and A. E. Eiben, "Using entropy for parameter analysis of evolutionary algorithms," in *Experimental Methods for the Analysis of Optimization Algorithms*, T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, Eds. Heidelberg Germany: Springer, 2010, pp. 287–310.
[81] F. Hutter, H. H. Hoos, and T. Stützle, "Automatic algorithm configuration based on local search," in *Proc. AAAI*, Vancouver, BC, Canada, 2007, pp. 1152–1157.
[82] H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. San Francisco, CA, USA: Elsevier, 2005.
[83] L. P. Cáceres and T. Stützle, "Exploring variable neighborhood search for automatic algorithm configuration," *Electron. Notes Discr. Math.*, vol. 58, pp. 167–174, Apr. 2017.
[84] P. Hansen and N. Mladenović, "Variable neighborhood search: Principles and applications," *Eur. J. Oper. Res.*, vol. 130, no. 3, pp. 449–467, 2001.
[85] E. B. M. Barbosa and E. L. F. Senne, "A heuristic for optimization of metaheuristics by means of statistical methods," in *Proc. 6th Int. Conf. Oper. Res. Enterprise Syst.*, Porto, Portugal, 2017, pp. 203–210.
[86] E. B. M. Barbosa and E. L. F. Senne, "Improving the fine-tuning of metaheuristics: An approach combining design of experiments and racing algorithms," *J. Optim.*, vol. 2017, pp. 1–7, Jun. 2017. [Online]. Available: https://www.hindawi.com/journals/jopti/2017/8042436/cta/
[87] J. E. Beasley, "OR-Library: Distributing test problems by electronic mail," *J. Oper. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, Nov. 1990.
[88] T. Bartz-Beielstein and M. Zaefferer, "Model-based methods for continuous and discrete global optimization," *Appl. Soft. Comput.*, vol. 55, pp. 154–167, Jun. 2017.
[89] F. Hutter, T. Bartz-Beielstein, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy, "Sequential model-based parameter optimization: An experimental investigation of automated and interactive approaches," in *Experimental Methods for the Analysis of Optimization Algorithms*, T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, Eds. Heidelberg, Germany: Springer, 2010, pp. 363–414.
[90] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Glob. Optim.*, vol. 13, no. 4, pp. 455–492, Dec. 1998.
[91] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn, "Design and analysis of computer experiments," *Stat. Sci.*, vol. 4, no. 4, pp. 409–423, 1989.
[92] T. Bartz-Beielstein, C. W. G. Lasarczyk, and M. Preuss, "Sequential parameter optimization," in *Proc. CEC*, Edinburgh, U.K., 2005, pp. 773–780.
[93] T. Bartz-Beielstein, K. E. Parsopoulos, and M. N. Vrahatis, "Design and analysis of optimization algorithms using computational statistics," *Appl. Numer. Anal. Comput. Math.*, vol. 1, no. 2, pp. 413–433, Dec. 2004.
[94] T. J. Santner, B. J. Williams, and W. I. Notz, "Space-filling designs for computer experiments," in *The Design and Analysis of Computer Experiments* (Springer Series in Statistics), T. J. Santner, B. J. Williams, and W. I. Notz, Eds. New York, NY, USA: Springer, 2003, pp. 121–161.
[95] D. Huang, T. T. Allen, W. I. Notz, and N. Zeng, "Global optimization of stochastic black-box systems via sequential kriging meta-models," *J. Glob. Optim.*, vol. 34, no. 3, pp. 441–466, Mar. 2006.
[96] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. Murphy, "Time-bounded sequential parameter optimization," in *Proc. 4th Int. Conf. Learn. Intell. Optim.*, 2010, pp. 281–298.
[97] T. Bartz-Beielstein, "SPOT: An R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization," *CoRR*, vol. abs/1006.4645, pp. 1–42, 2010. [Online]. Available: http://arxiv.org/abs/1006.4645
[98] T. Bartz-Beielstein, L. Gentile, and M. Zaefferer, "In a nutshell: Sequential parameter optimization," *CoRR*, vol. abs/1712.04076, pp. 1–40, 2017. [Online]. Available: http://arxiv.org/abs/1712.04076
[99] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY, USA: Springer, 2009.
[100] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.
[101] F. Hutter *et al.*, "AClib: A benchmark library for algorithm configuration," in *Learning and Intelligent Optimization*, P. M. Pardalos, M. G. Resende, C. Vogiatzis, and J. L. Walteros, Eds. Cham, Switzerland: Springer Int., 2014, pp. 36–40.
[102] T. Zhang, M. Georgiopoulos, and G. C. Anagnostopoulos, "S-Race: A multi-objective racing algorithm," in *Proc. GECCO*, Amsterdam, The Netherlands, 2013, pp. 1565–1572.
[103] T. Zhang, M. Georgiopoulos, and G. C. Anagnostopoulos, "SPRINT multi-objective model racing," in *Proc. GECCO*, Madrid, Spain, 2015, pp. 1383–1390.
[104] T. Zhang, M. Georgiopoulos, and G. C. Anagnostopoulos, "Multi-objective model selection via racing," *IEEE Trans. Cybern.*, vol. 46, no. 8, pp. 1863–1876, Aug. 2016.
[105] A. Blot, H. H. Hoos, L. Jourdan, M.-É. Kessaci-Marmion, and H. Trautmann, "MO-ParamILS: A multi-objective automatic algorithm configuration framework," in *Learning and Intelligent Optimization*, P. Festa, M. Sellmann, and J. Vanschoren, Eds. Cham, Switzerland: Springer Int., 2016, pp. 32–47.

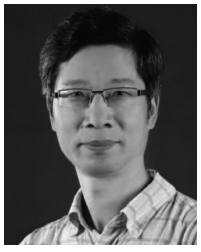
Changwu Huang (M'19) received the Ph.D. degree from the Institut National des Sciences Appliquées de Rouen Normandie (INSA Rouen Normandie), Saint-Étienne-du-Rouvray, France, in 2018.

He is currently a Post-Doctoral Researcher with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. His current research interests include evolutionary computation and automated algorithm configuration, and their applications to routing problems.

Yuanxiang Li received the Ph.D. degree from the Department of Computer Science, Wuhan University (WHU), Wuhan, China, in 1993.

He is a Professor of computer science with WHU. His current research interests include parallel computing, evolutionary computation, and cellular automata modeling for complex systems.

Dr. Li was a recipient of the National Natural Science Prize of China for the Project "Asynchronous Parallel Algorithms and Domain Decompositions" and the Advanced Prize of Science and Technology of the National Educational Ministry for the Project "Parallel Computational Models and Algorithms for Simulating Complex Systems."



Xin Yao (M'91–SM'96–F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technologies, Langfang, China, in 1985, and the Ph.D. degree from USTC in 1990.

He is a Chair Professor of computer science with the Southern University of Science and Technology, Shenzhen, China, and a part-time Professor of computer science with the University of Birmingham, Birmingham, U.K. His current research interests include evolutionary computation and ensemble learning, and their applications to software engineering.

Dr. Yao was a recipient of the 2001 IEEE Donald G. Fink Prize Paper Award for his paper on evolving artificial neural networks, the 2010, 2016, and 2017 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Awards, the 2011 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, the prestigious Royal Society Wolfson Research Merit Award in 2012, the IEEE CIS Evolutionary Computation Pioneer Award in 2013, and many other best paper awards. He was the President of IEEE Computational Intelligence Society (CIS) from 2014 to 2015 and the Editor-in-Chief from 2003 to 2008 of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION. He was a Distinguished Lecturer of IEEE CIS.