Swarm Intelligence and Evolutionary Computation: Overview and Analysis

Xin-She Yang and Xingshi He

Abstract In many applications, the complexity and nonlinearity of the problems require novel and alternative approaches to problem solving. In recent years, nature-inspired algorithms, especially those based on swarm intelligence, have become popular, due to the simplicity and flexibility of such algorithms. Here, we review briefly some recent algorithms and then outline the self-tuning framework for parameter tuning. We also discuss some convergence properties of the cuckoo search and the bat algorithm. Finally, we present some open problems as further research topics.

Keywords Algorithm · Adaptation · Bat algorithm · Cuckoo search · Diversity · Firefly algorithm · Metaheuristic · Nature-inspired algorithm · Optimization · Parameter tuning · Swarm intelligence

1 Introduction

In many applications, we have to deal with complex optimization problems with complicated constraints. Such problems can be challenging to solve, due to their complexity, nonlinearity and potentially high-dimensionality. These problems can even be NP-hard, and thus require alternative methods because conventional methods usually cannot cope such complex problems. In recent years, nature-inspired metaheuristic algorithms have gained huge popularity because they have demonstrated some promising results in solve tough optimization problems. These metaheuristic algorithms include ant colony optimization, particle swarm optimization, cuckoo

X.-S. Yang (✉)
School of Science and Technology, Middlesex University, London NW4 4BT, UK
e-mail: x.yang@mdx.ac.uk; xy227@cam.ac.uk

X. He
College of Science, Xi'an Polytechnic University, No. 19 Jinhua South Road,
Xi'an, People's Republic of China

search, firefly algorithm, bat algorithm, bee algorithms and others [1–4]. There are many reasons for such popularity. From the algorithm analysis point of view, these algorithms tend to be flexible, efficient and highly adaptable, and yet easy to implement. The high efficiency of these algorithms makes it possible to apply them to a wide range of problems in diverse applications.

Swarm intelligence is quite a general concept that multiple agents interact and exchange information, following simple rules. Rather surprisingly, such simple systems can show complex, self-organized behaviour. Though the characteristics of agent interactions may be drawn from different sources of inspiration in nature [4], algorithmic procedures can be quite simple and flexible, and yet efficient in practice. On the other hand, evolutionary computation is traditionally considered as part of computational intelligence, which concerns optimization with continuous, combinatorial or mixed problems. Algorithms such as genetic algorithms and evolutionary strategy are good examples of evolutionary computation. However, evolutionary computation has broaden its scope and extended to include many areas. Loosely speaking, swarm intelligence is part of the evolutionary computation paradigm, but the interests in swarm intelligence are so overwhelming that swarm intelligence has almost become a field of itself. Here, we will not debate what the right terminology or fields should be. We will discuss both swarm intelligence and evolutionary computation in the most broad sense.

The main purpose of this chapter is to provide an overview and the recent advances concerning swarm intelligence and evolutionary computation. Therefore, the chapter is organized as follows. Section 2 outlines some recent nature-inspired algorithms, followed by the analysis and discussions of adaptation and diversity in these algorithms in Sect. 3. Section 4 discusses the self-tuning framework for parameter tuning and control, while Sect. 5 outlines the convergence analysis of the cuckoo search and the bat algorithm. Finally some discussions and open problems are presented in Sect. 6.

2 Swarm Intelligence, Adaptation and Diversity

Extensive research activities have resulted in significant developments in swarm intelligence (SI) in recent years. It is not possible to cover even a good fraction of the extensive literature in this brief review and thus we have to focus on the most recent algorithms, especially those in the last few years.

2.1 *The Essence of an Algorithm*

Different disciplines may view an algorithm differently, and the point of view all depends on the perspective. In essence, algorithm A is an iterative process, which aims to generate a new and better solution $\boldsymbol{x}^{t+1}$ to a given problem from the current solution $\boldsymbol{x}^t$ at iteration or (pseudo)time t. It can be written as

$$\boldsymbol{x}^{t+1} = A(\boldsymbol{x}^t, p), \tag{1}$$

where p is an algorithm-dependent parameter. For example, the Newton-Raphson method to find the optimal value of $f(\boldsymbol{x})$ is equivalent to finding the critical points or roots of $f'(\boldsymbol{x}) = 0$ in a d-dimensional space. That is,

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \frac{f'(\boldsymbol{x}^t)}{f''(\boldsymbol{x}^t)} = A(\boldsymbol{x}^t). \tag{2}$$

Obviously, the convergence rate may become very slow near the optimal point where $f'(x) \to 0$. Sometimes, the true convergence rate may not be as quick as it should be. A simple way to improve the convergence is to modify the above formula slightly by introducing a parameter p as follows:

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - p\frac{f'(\boldsymbol{x}^t)}{f''(\boldsymbol{x}^t)}, \quad p = \frac{1}{1 - A'(\boldsymbol{x}_*)}. \tag{3}$$

Here, $\boldsymbol{x}_*$ is the optimal solution, or a fixed point of the iterative formula. For simplicity, we can treat p as a step size and this essentially becomes the modified Newton-Raphson method.

The above formula is for a trajectory-based, single agent system. For population-based algorithms with a swarm of n solutions $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$, we can extend the above iterative formula to a more general form

$$\begin{pmatrix} \boldsymbol{x}_1 \\ x_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{t+1} = A\big((\boldsymbol{x}_1^t,\ \boldsymbol{x}_2^t, \ldots, \boldsymbol{x}_n^t); (p_1, p_2, \ldots, p_k); (\epsilon_1, \epsilon_2, \ldots, \epsilon_m)\big) \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{t}, \tag{4}$$

where $p_1, \ldots, p_k$ are k algorithm-dependent parameters and $\epsilon_1, \ldots, \epsilon_m$ are m random variables. An algorithm can be viewed as a dynamical system, Markov chains and iterative maps [4], and it can also be viewed as a self-organized system [5]. Here, the introduction of m random variables captures the essence of all contemporary evolutionary algorithms because they all use some sort of randomization techniques.

Whatever the perspective may be, the aim of such an iterative process is to let the system evolve and converge into some stable optimality. In this case, it has strong similarity to a self-organizing system. Such an iterative, self-organizing system can evolve, according to a set of rules or mathematical equations. As a result, such a complex system can interact and self-organize into certain converged states, showing some emergent characteristics of self-organization. In this sense, the proper design of an efficient optimization algorithm is equivalent to finding efficient ways to mimic the evolution of a self-organizing system [5]. In practice, all nature-inspired algorithms try to mimic some successful characteristics of biological, physical or chemical systems in nature [1, 4, 6].

Among all evolutionary algorithms, algorithms based on swarm intelligence (SI) dominate the landscape. There are many reasons for this dominance, though three obvious reasons are: (1) swarm intelligence uses multiple agents as an evolving, interacting population, and thus provides good ways to mimic natural systems. (2) Population-based approaches allow parallelization and vectorization implementations in practice, and are thus straightforward to implement. (3) SI-based algorithms are simple and easy to implement, and they are flexible and yet sufficiently efficient. As a result, these algorithms can deal with a relatively wide range of problems in applications.

2.2 *Particle Swarm Optimization*

Among the swarm intelligence (SI) based algorithms, particle swarm optimization (PSO) is among the first. PSO was developed by Kennedy and Eberhart in 1995 [1], based on the swarm behaviour of fish or bird schooling in nature. Each particle updates its position $\boldsymbol{x}_i$ and velocity $\boldsymbol{v}_i$ by

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + \alpha\,\boldsymbol{\epsilon}_1[\boldsymbol{g}^* - \boldsymbol{x}_i^t] + \beta\boldsymbol{\epsilon}_2[\boldsymbol{x}_i^* - \boldsymbol{x}_i^t], \tag{5}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}, \tag{6}$$

where $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ are two random vectors, drawn from a uniform distribution between 0 and 1. Here, α and β are the learning parameters with typical values of $\alpha \approx \beta \approx 2$.

The literature of PSO is vast, with thousands of papers and dozens of books. Therefore, we will not provide detailed literature review here, and readers can find such literature quite easily.

2.3 *Some Recent SI-Based Algorithms*

In this section, we will focus on the SI-based optimization algorithms that have been developed in recent years and these new algorithms have attracted much attention in the last few years.

2.3.1 Firefly Algorithm

The firefly algorithm (FA) is simple, flexible and easy to implement. FA was developed by Yang in 2008 [2], which was based on the flashing patterns and behaviour of tropical fireflies. FA can naturally deal with nonlinear multimodal optimization problems.

The movement of a firefly i is attracted to another more attractive (brighter) firefly j is determined by

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2}(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha\,\boldsymbol{\epsilon}_i^t, \tag{7}$$

where the second term is due to the attraction of fireflies, and β_0 is the attractiveness at $r = 0$. The third term is randomization with α being the randomization parameter, and $\boldsymbol{\epsilon}_i^t$ is a vector of random numbers drawn from a Gaussian distribution at time t. Other studies also use the randomization in terms of $\boldsymbol{\epsilon}_i^t$ that can easily be extended to other distributions such as Lévy flights. A comprehensive review of the firefly algorithm and its variants has been carried out by Fister et al. [7–9].

One novel feature of FA is that distance-related attraction is used, and this is the first of its kind in any SI-based algorithms. Since local attraction is stronger than long-distance attraction, the population in FA can automatically subdivide into multiple subgroups, and each group can potentially swarm around a local mode. Among all the local modes, there is always a global best solution which is the true optimality of the problem. Thus, FA can deal with multimodal problems naturally and efficiently [3].

2.3.2 Cuckoo Search

The cuckoo search (CS) was developed in 2009 by Yang and Deb [10]. CS is based on the brood parasitism of some cuckoo species. In addition, this algorithm is enhanced by the so-called Lévy flights [11], rather than by simple isotropic random walks. Recent studies show that CS is potentially far more efficient than PSO and genetic algorithms [12–14].

In essence, CS uses a balanced combination of a local random walk and the global explorative random walk, controlled by a switching parameter p_a. The local random walk can be written as

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha s \otimes H(p_a - \epsilon) \otimes (\boldsymbol{x}_j^t - \boldsymbol{x}_k^t), \tag{8}$$

where $\boldsymbol{x}_j^t$ and $\boldsymbol{x}_k^t$ are two different solutions selected randomly by random permutation, $H(u)$ is a Heaviside function, ϵ is a random number drawn from a uniform distribution, and s is the step size. On the other hand, the global random walk is carried out by using Lévy flights

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha L(s, \lambda), \tag{9}$$

where

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s \gg s_0 > 0). \tag{10}$$

Here $\alpha > 0$ is the step size scaling factor, which should be related to the scales of the problem of interest.

CS has two distinct advantages over other algorithms such as GA and SA, and these advantages are: efficient random walks and balanced mixing. Since Lévy flights are usually far more efficient than any other random-walk-based randomization techniques, CS can be efficient in global search [3, 11]. In fact, recent studies show that CS can have guaranteed global convergence [4]. In addition, the similarity between eggs can produce better new solutions, which is essentially fitness-proportional generation with a good mixing ability. In other words, CS has varying mutation realized by Lévy flights, and the fitness-proportional generation of new solutions based on similarity provides a subtle form of crossover. In addition, simulations also show that CS can have autozooming ability in the sense that new solutions can automatically zoom into the region where the promising global optimality is located.

In addition, Eq. (9) is essentially simulated annealing in the framework of Markov chains. In Eq. (8), if $p_a = 1$ and $\alpha s \in [0, 1]$, CS can degenerate into a variant of differentia evolution. Furthermore, if we replace $\boldsymbol{x}_j^t$ by the current best solution $\boldsymbol{g}^*$, then (8) can further degenerate into accelerated particle swarm optimization (APSO) [15]. This means that SA, DE and APSO are special cases of CS, and this explains why CS is so efficient [3].

2.3.3 Bat Algorithm

The bat algorithm (BA) was developed by Yang in 2010 [16]. It was inspired by the echolocation behavior of microbats. It is the first algorithm of its kind to use frequency tuning. Each bat is associated with a velocity $\boldsymbol{v}_i^t$ and a location $\boldsymbol{x}_i^t$, at iteration t, in a d-dimensional search or solution space. Among all the bats, there exists a current best solution $\boldsymbol{x}_*$. Therefore, the updating equations for $\boldsymbol{x}_i^t$ and velocities $\boldsymbol{v}_i^t$ can be written as

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{11}$$

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + (\boldsymbol{x}_i^{t-1} - \boldsymbol{x}_*) f_i, \tag{12}$$

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1} + \boldsymbol{v}_i^t, \tag{13}$$

where $\beta \in [0, 1]$ is a random vector drawn from a uniform distribution.

The loudness and pulse emission rates are regulated by the following equations:

$$A_i^{t+1} = \alpha A_i^t, \tag{14}$$

and

$$r_i^{t+1} = r_i^0 [1 - \exp(-\gamma t)], \tag{15}$$

where $0 < \alpha < 1$ and $\gamma > 0$ are constants. In essence, here α is similar to the cooling factor of a cooling schedule in simulated annealing.

BA has been extended to multiobjective bat algorithm (MOBA) by Yang [17], and Fister et al. have extended to a hybrid bat algorithm [18]. The preliminary results suggested that they are very efficient [19, 20].

2.3.4 Flower Algorithm

Flower pollination algorithm (FPA) was developed by Yang in 2012 [21], inspired by the flower pollination process of flowering plants. It has been extended to multiobjective optimization problems and found to be very efficient [22, 23]. For simplicity, we use the following four rules:

1. Biotic and cross-pollination can be considered as a process of global pollination process, and pollen-carrying pollinators move in a way which obeys Lévy flights (Rule 1).
2. For local pollination, abiotic pollination and self-pollination are used (Rule 2).
3. Pollinators such as insects can develop flower constancy, which is equivalent to a reproduction probability that is proportional to the similarity of two flowers involved (Rule 3).
4. The interaction or switching of local pollination and global pollination can be controlled by a switch probability $p \in [0, 1]$, with a slight bias towards local pollination (Rule 4).

In order to formulate updating formulae, we have to convert the above rules into updating equations. For example, in the global pollination step, flower pollen gametes are carried by pollinators such as insects, and pollen can travel over a long distance because insects can often fly and move in a much longer range. Therefore, Rule 1 and flower constancy can be represented mathematically as

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \gamma L(\lambda)(\boldsymbol{g}_* - \boldsymbol{x}_i^t), \tag{16}$$

where $\boldsymbol{x}_i^t$ is the pollen i or solution vector $\boldsymbol{x}_i$ at iteration t, and $\boldsymbol{g}_*$ is the current best solution found among all solutions at the current generation/iteration. Here γ is a scaling factor to control the step size.

Here $L(\lambda)$ is the parameter that corresponds to the strength of the pollination, which essentially is also a step size. Since insects may move over a long distance with various distance steps, we can use a Lévy flight to mimic this characteristic efficiently. That is, we draw $L > 0$ from a Levy distribution

$$L \sim \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s \gg s_0 > 0). \tag{17}$$

Here $\Gamma(\lambda)$ is the standard gamma function, and this distribution is valid for large steps $s > 0$. This step is essentially a global mutation step, which enables to explore the search space more efficiently.

For the local pollination, both Rule 2 and Rule 3 can be represented as

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \epsilon(\boldsymbol{x}_j^t - \boldsymbol{x}_k^t), \tag{18}$$

where $\boldsymbol{x}_j^t$ and $\boldsymbol{x}_k^t$ are pollen from different flowers of the same plant species. This essentially mimics the flower constancy in a limited neighborhood. Mathematically, if $\boldsymbol{x}_j^t$ and $\boldsymbol{x}_k^t$ comes from the same species or selected from the same population, this equivalently becomes a local random walk if we draw ϵ from a uniform distribution in [0, 1]. In essence, this is a local mutation and mixing step, which can help to converge in a subspace.

In principle, flower pollination activities can occur at all scales, both local and global. But in reality, adjacent flower patches or flowers in the not-so-far-away neighborhood are more likely to be pollinated by local flower pollen than those far away. In order to mimic this feature, we can effectively use a switch probability (Rule 4) or proximity probability p to switch between common global pollination to intensive local pollination. To start with, we can use a naive value of $p = 0.5$ as an initially value. A preliminary parametric showed that $p = 0.8$ may work better for most applications.

Recent studies suggested that flower pollination algorithm is very efficient for multiobjective optimization [23, 24].

2.4 Other Evolutionary Algorithms

There are quite a few other evolutionary algorithms, though they are not swarm intelligence based algorithms. So we also briefly introduce them here.

2.4.1 Differential Evolution

Differential evolution (DE) was developed by Storn and Price in 1996 and 1997 [25, 26]. In fact, modern differential evolution (DE) has strong similarity to the traditional mutation operator in the traditional pattern search. In essence, the mutation in DE can be viewed as the generalized pattern search in any random direction $(\boldsymbol{x}_p - \boldsymbol{x}_q)$ by

$$\boldsymbol{x}_i = \boldsymbol{x}_r + F(\boldsymbol{x}_p - \boldsymbol{x}_q), \tag{19}$$

where F is the differential weight in the range of [0,2]. Here, r, p, q, i are four different integers generated by random permutation.

In addition, DE also has a crossover operator which is controlled by a crossover probability $C_r \in [0,\ 1]$ and the actual crossover can be carried out in two ways: binomial and exponential. Selection is essentially the same as that used in genetic algorithms. It is to select the most fittest, and for the minimization problem, the minimum objective value. Therefore, we have

$$\boldsymbol{x}_i^{t+1} = \begin{cases} \boldsymbol{u}_i^{t+1} & \text{if } f(\boldsymbol{u}_i^{t+1}) \leq f(\boldsymbol{x}_i^t), \\ \boldsymbol{x}_i^t & \text{otherwise.} \end{cases} \tag{20}$$

Most studies have focused on the choice of F, C_r, and the population size n as well as the modification of the mutation scheme. In addition, it can be clearly seen that selection is also used when the condition in the above equation is checked. Almost all variants of DE use crossover, mutation and selection, and the main differences are in the step of mutation and crossover. For example, DE/Rand/1/Bin use the three vectors for mutation, and binomial crossover. There are more than 10 different variants [27].

2.4.2 Harmony Search

Harmony search (HS) is a music-inspired algorithm, developed by Geem et al. in 2001 [28]. It is not swarm-intelligence-based, but it is a metaheuristic algorithm. In the standard HS, solutions are represented in terms of a population of harmonies, using the following three choices/rules (of a musician playing a piece of music): (1) play any famous piece of music from memory, (2) play something similar to a known piece (pitch adjustment), and (3) compose new or random notes. HS uses mainly mutation and selection, while crossover is not explicitly used. The first rule corresponds to selection or elitism, and the second and third rules are mutation.

Mutation can be local and global in HS. For example, the pitch adjustment (the second rule) uses the following equation

$$x_{new} = x_{old} + b_w \varepsilon, \tag{21}$$

where b_w is the bandwidth of the pitch adjustment, while ε is a random number drawn from $[-1, 1]$. This is a local random walk, and the distance of the random walk is controlled by the bandwidth. This part can be considered as a local mutation action with an equivalent mutation rate of 0.1–0.3.

The third rule is essentially mutation on a larger scale, which is essentially equivalent to random walks. The selection is controlled by the probability of choosing a harmony from harmony memory. Similar to genetic algorithms, this choice of harmonies from the population is high with a typical value of 0.9, which enables the system to converge in a subspace. However, this may be at the expense of reduced probability of finding the global optimality in some highly nonlinear problems.

2.4.3 Other Algorithms

Many other algorithms have appeared in the literature, which may require more extensive analysis and comparisons [4]. However, as this is not the main focus of this chapter, we will not go into more details about these algorithms.

One thing we may notice by analyzing these algorithms is that mutation and selection are always used, while crossover are not used in most of these algorithms. This may raise the question and further need to analyze what exactly the role of crossover is. Therefore, there is a strong need to investigate further how two-stage eagle strategy and co-evolutionary methods can work better. In addition, systematic tuning of parameters in algorithms and careful control of these algorithm-dependent parameters may be very useful to understand how these algorithms behave and how to improve them in practice.

3 Adaptation and Diversity

The effectiveness of these algorithms can be attributed to two important characteristics: adaptation and diversity of nature-inspired optimization algorithms.

Adaptation in nature-inspired algorithms can take many forms. For example, the ways to balance exploration and exploitation are the key form of adaptation [29]. As diversity can be intrinsically linked with adaptation, it is better not to discuss these two features separately. If exploitation is strong, the search process will use problem-specific information (or landscape-specific information) obtained during the iterative process to guide the new search moves, this may lead to the focused search and thus reduce the diversity of the population, which may help to speed up the convergence of the search procedure. However, if exploitation is too strong, it can result in the quick loss of diversity in the population and thus may lead to the premature convergence. On the other hand, if new search moves are not guided by local landscape information, it can typically increase the exploration capability and generate new solutions with higher diversity. However, too much diversity and exploration may result in meandered search paths, thus lead to the slow convergence. Therefore, adaptation of search moves so as to balance exploration and exploitation is crucial. Consequently, to maintain a balanced diversity in population is also important.

Adaptation can also be in terms of the representations of solutions of a problem. In genetic algorithms, representations of solutions are usually in binary or real-valued strings [29, 30], while in swarm-intelligence-based algorithms, representations mostly use real number solution vectors. For example, the population size used in an algorithm can be fixed or varying. Adaptation in this case may mean to vary the population size so as to maximize the overall performance.

For a given algorithm, adaptation can also occur to adjust its algorithm-dependent parameters. As the performance of an algorithm can largely depend on its parameters, the choice of these parameter values can be very important. Values can be varied so as to adapt the landscape type of the problem and thus may lead to better search efficiency. Such parameter tuning is in essence parameter adaptation. Once parameters are tuned, they can remain fixed. However, there is no particular reason why parameters should be fixed. In fact, adaptation in parameter can be extended to parameter control. That is to control the parameter values in such a way that their values vary during the iterations so that optimal performance of the algorithm can be achieved.

Diversity in metaheuristic algorithms can also take many forms. The simplest diversity is to allow the variations of solutions in the population by randomization. For example, solution diversity in genetic algorithms is mainly controlled by mutation rates and crossover mechanisms, while in simulated annealing, diversity is achieved by random walks. In most swarm-intelligence-based algorithms, new solutions are generated according to a set of deterministic equations, which also include some random variables. Diversity is represented by the variation, often in terms of population variance. Once the population variance is getting smaller (approaching zero), diversity also decreases, leading to converged solution sets. However, if diversity is reduced too quickly, premature convergence may occur. Therefore, a right amount of randomness and the right form of randomization can be crucial.

In addition, adaptation and diversity can also be related to the selection of solutions among the population and the replacement of the old population. If the selection is based on the fitness, parent solutions with higher fitness will be more likely to pass onto the next generation. In the extreme case, only the best solutions can be selected, which is a kind of elitism. If the replacement of worst solutions by new (hopefully better) solutions, this will ensure that better solutions will remain in the population. The balance of what to replace and what to pass on can be tricky, which requires good adaptation so as to maintain good diversity in the population.

4 Self-Tuning Algorithms

From a mathematical point of view, an algorithm A tends to generate a new and better solution $\boldsymbol{x}^{t+1}$ to a given problem from the current solution $\boldsymbol{x}^t$ at iteration or time t. In modern metaheuristic algorithms, randomization is often used in an algorithm, and in many cases, randomization appears in the form of a set of m random variables $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_m)$ in an algorithm. For example, in simulated annealing, there is one random variable, while in particle swarm optimization, there are two random variables. In addition, there are often a set of k parameters in an algorithm. For example, in particle swarm optimization, there are 4 parameters (two learning parameters, one inertia weight, and the population size). In general, we can have a vector of parameters $\boldsymbol{p} = (p_1, \ldots, p_k)$. Mathematically speaking, we can write an algorithm with k parameters and m random variables as

$$\boldsymbol{x}^{t+1} = \boldsymbol{A}(\boldsymbol{x}^t, \boldsymbol{p}(t), \varepsilon(t)), \tag{22}$$

where $\boldsymbol{A}$ is a nonlinear mapping from a given solution (a d-dimensional vector $\boldsymbol{x}^t$) to a new solution vector $\boldsymbol{x}^{t+1}$.

Representation (22) gives rise to two types of optimality: optimality of a problem and optimality of an algorithm [4, 31]. For an optimization problem such as $\min f(\boldsymbol{x})$, there is a global optimal solution whatever the algorithmic tool we may use to find this optimality. This is the optimality for the optimization problem. On the other hand, for a given problem Φ with an objective function $f(\boldsymbol{x})$, there are many algorithms that can solve it. Some algorithms may require less computational effort than others. There may be the best algorithm with the least computing cost, though this may not be unique. However, this is not our concern here. Once we have chosen an algorithm A to solve a problem Φ, there is an optimal parameter setting for this algorithm so that it can achieve the best performance. This optimality depends on both the algorithm itself and the problem it solves. In the rest of this chapter, we will focus on this type of optimality.

That is, the optimality to be achieved is

$$\text{Maximize the performance of } \xi = A(\Phi, \boldsymbol{p}, \varepsilon), \tag{23}$$

for a given problem Φ and a chosen algorithm $A(., \boldsymbol{p}, \varepsilon)$. We will denote this optimality as $\xi_* = A_*(\Phi, \boldsymbol{p}_*) = \xi(\Phi, \boldsymbol{p}_*)$ where $\boldsymbol{p}_*$ is the optimal parameter setting for this algorithm so that its performance is the best. Here, we have used a fact that ε is a random vector can be drawn from some known probability distributions, thus the randomness vector should not be related to the algorithm optimality.

It is worth pointing out that there is another potential optimality. That is, for a given problem, a chosen algorithm with the best parameter setting $\boldsymbol{p}_*$, we can still use different random numbers drawn from various probability distributions and even chaotic maps, so that even better performance may be achieved. Strictly speaking, if an algorithm $A(., ., \varepsilon)$ has a random vector ε that is drawn from a uniform distribution $\varepsilon_1 \sim U(0,1)$ or from a Gaussian $\varepsilon_2 \sim N(0,1)$, it becomes two algorithms $A_1 = A(.,., \varepsilon_1)$ and $A_2 = A(.,., \varepsilon_2)$. Technically speaking, we should treat them as different algorithms. Since our emphasis here is about parameter tuning so as to find the optimal setting of parameters, we will omit the effect of randomness vectors, and thus focus on

$$\text{Maximize } \xi = A(\Phi, \boldsymbol{p}). \tag{24}$$

In essence, tuning algorithm involves in tuning its algorithm-dependent parameters. Therefore, parameter tuning is equivalent to algorithm tuning in the present context.

4.1 Parameter Tuning

In order to tune $A(\Phi, \boldsymbol{p})$ so as to achieve its best performance, a parameter-tuning tool, i.e., a tuner, is needed. Like tuning a high-precision machinery, sophisticated tools are required. For tuning parameters in an algorithm, what tool can we use? One way is to use a better, existing tool (say, algorithm *B*) to tune an algorithm *A*. Now the question may become: how do you know *B* is better? Is *B* well-tuned? If yes, how do you tune *B* in the first place? Naively, if we say, we use another tool (say, algorithm *C*) to tune *B*. Now again the question becomes how algorithm *C* has been tuned? This can go on and on, until the end of a long chain, say, algorithm *Q*. In the end, we need some tool/algorithm to tune this *Q*, which again comes back to the original question: how to tune an algorithm *A* so that it can perform best?

It is worth pointing out that even if we have good tools to tune an algorithm, the best parameter setting and thus performance all depend on the performance measures used in the tuning. Ideally, these parameters should be robust enough to minor parameter changes, random seeds, and even problem instance. However, in practice, they may not be achievable. According to Eiben [32], parameter tuning can be divided into iterative and non-iterative tuners, single-stage and multi-stage tuners. The meaning of these terminologies is self-explanatory. In terms of the actual tuning methods, existing methods include sampling methods, screening methods, model-based methods, and metaheuristic methods. Their success and effectiveness can vary, and thus there are no well-established methods for universal parameter tuning.

4.2 Framework for Self-Tuning Algorithms

From our earlier observations and discussions, it is clear that parameter tuning is the process of optimizing the optimization algorithm; therefore, it is a hyper-optimization problem. In essence, a tuner is a meta-optimization tool for tuning algorithms [31].

For a standard unconstrained optimization problem, the aim is to find the global minimum f_* of a function $f(\boldsymbol{x})$ in a d-dimensional space. That is,

$$\text{Minimize}\, f(\boldsymbol{x}), \quad \boldsymbol{x} = (x_1, x_2, \ldots, x_d). \tag{25}$$

Once we choose an algorithm A to solve this optimization problem, the algorithm will find a minimum solution $f_{\min}$ which may be close to the true global minimum f_*. For a given tolerance δ, this may requires t_δ iterations to achieve $|f_{\min} - f_*| \leq \delta$. Obviously, the actual t_δ will largely depend on both the problem objective $f(\boldsymbol{x})$ and the parameters $\boldsymbol{p}$ of the algorithm used.

The main aim of algorithm-tuning is to find the best parameter setting $\boldsymbol{p}_*$ so that the computational cost or the number of iterations t_δ is the minimum. Thus, parameter tuning as a hyper-optimization problem can be written as

$$\text{Minimize } t_\delta = A(f(\boldsymbol{x}), \boldsymbol{p}), \tag{26}$$

whose optimality is $\boldsymbol{p}_*$.

Ideally, the parameter vector $\boldsymbol{p}_*$ should be sufficiently robust. For different types of problems, any slight variation in $\boldsymbol{p}_*$ should not affect the performance of A much, which means that $\boldsymbol{p}_*$ should lie in a flat range, rather than at a sharp peak in the parameter landscape.

4.3 A Multiobjective View

If we look the algorithm tuning process from a different perspective, it is possible to construct it as a multi-objective optimization problem with two objectives: one objective $f(\boldsymbol{x})$ for the problem Φ and one objective t_δ for the algorithm. That is

$$\text{Minimize}\, f(\boldsymbol{x}) \text{ and Minimize } t_\delta = A(f(\boldsymbol{x}), \boldsymbol{p}), \tag{27}$$

where t_δ is the (average) number of iterations needed to achieve a given tolerance δ so that the found minimum $f_{\min}$ is close enough to the true global minimum f_*, satisfying $|f_{\min} - f_*| \leq \delta$.

This means that for a given tolerance δ, there will be a set of best parameter settings with a minimum t_δ. As a result, the bi-objectives will form a Pareto front. In principle, this bi-objective optimization problem (27) can be solved by any methods that are suitable for multiobjective optimization. But as δ is usually given, a natural way to solve this problem is to use the so-called ϵ-constraint or δ-constraint methods. The naming may be dependent on the notations; however, we will use δ-constraints.

For a given $\delta \geq 0$, we change one of the objectives (i.e., $f(\boldsymbol{x})$) into a constraint, and thus the above problem (27) becomes a single-objective optimization problem with a constraint. That is

$$\text{Minimize } t_\delta = A(f(\boldsymbol{x}),\boldsymbol{p}), \tag{28}$$

subject to

$$f(\boldsymbol{x}) \leq \delta. \tag{29}$$

In the rest of this chapter, we will set $\delta = 10^{-5}$.

The important thing is that we still need an algorithm to solve this optimization problem. However, the main difference from a common single objective problem is that the present problem contains an algorithm A. Ideally, an algorithm should be independent of the problem, which treats the objective to be solved as a black box. Thus we have $A(.,\boldsymbol{p},\varepsilon)$, however, in reality, an algorithm will be used to solve a particular problem Φ with an objective $f(\boldsymbol{x})$. Therefore, both notations $A(.,\boldsymbol{p})$ and $A(f(\boldsymbol{x}),\boldsymbol{p})$ will be used here.

4.4 Self-Tuning Framework

This framework has been proposed by Yang et al. in 2013 [31]. In principle, we can solve (28) by any efficient or well-tuned algorithm. Now a natural question is: Can we solve this algorithm-tuning problem by the algorithm A itself? There is no reason why we cannot. In fact, if we solve (28) by using A, we have a self-tuning algorithm. That is, the algorithm automatically tunes itself for a given problem objective to be optimized. This essentially provides a framework for a self-tuning algorithm as shown in Fig. 1.

This framework is generic in the sense that any algorithm can be tuned this way, and any problem can be solved within this framework. This essentially achieves two goals simultaneously: parameter tuning and optimality finding.

4.5 Self-Tuning Firefly Algorithm

Now let us use the framework outlined earlier to tune the firefly algorithm (FA). As we have seen earlier, FA has the following updating equation:

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2}(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha \boldsymbol{\epsilon}_i^t, \tag{30}$$

which contains four parameters: α, β_0, γ and the population size n. For simplicity for parameter tuning, we set $\beta_0 = 1$ and $n = 20$, and therefore the two parameters to be tuned are: $\gamma > 0$ and $\alpha > 0$. It is worth pointing out that γ controls the scaling, while α controls the randomness. For this algorithm to convergence properly, randomness

```
Implement an algorithm $A(.,\boldsymbol{p},\boldsymbol{\varepsilon})$
    with parameters $\boldsymbol{p}=[p_1,...,p_K]$ and random vector $\boldsymbol{\varepsilon}=[\varepsilon_1,...,\varepsilon_m]$;
Define a tolerance (e.g., $\delta = 10^{-5}$);
    Algorithm objective $t_\delta(f(\boldsymbol{x}),\boldsymbol{p},\boldsymbol{\varepsilon})$;
        Problem objective function $f(\boldsymbol{x})$;
        Find the optimality solution $f_{\min}$ within $\delta$;
        Output the number of iterations $t_\delta$ needed to find $f_{\min}$;
    Solve $\min t_\delta(f(\boldsymbol{x}),\boldsymbol{p})$ using $A(.,\boldsymbol{p},\boldsymbol{\varepsilon})$ to get the best parameters;
Output the tuned algorithm with the best parameter setting $\boldsymbol{p}_*$.
```

Fig. 1 A framework for a self-tuning algorithm

should be gradually reduced, and one way to achieve such randomness reduction is to use

$$\alpha = \alpha_0 \theta^t, \quad \theta \in (0,1), \tag{31}$$

where t is the index of iterations/generations. Here α_0 is the initial randomness factor, and we can set $\alpha_0 = 1$ without losing generality. Therefore, the two parameters to be tuned become γ and θ. This framework works wells as shown by Yang et al. [31].

5 Convergence Analysis

There some solid mathematical analysis of nature-inspired algorithms such as differential evolution, genetic algorithms [33], simulated annealing and particle swarm optimization. In recent years, more theoretical results appear, and in the rest of this section, we summarize some of the recent advances.

5.1 Global Convergence of Cuckoo Search

Wang et al. provided a mathematical proof of global convergence for the standard cuckoo search, and their approach is based on the Markov chain theory [34]. Their proof can be outlined as follows:

As there are two branches in the updating formulas, the local search step only contributes mainly to local refinements, while the main mobility or exploration is carried out by the global search step. In order to simplify the analysis and also to emphasize the global search capability, we now use a simplified version of cuckoo search. That is, we use only the global branch with a random number $r \in [0,1]$, compared with a discovery/switching probability p_a. Now we have

$$\begin{cases} \boldsymbol{x}_i^{(t+1)} \leftarrow x_i^{(t)} & \text{if } r < p_a, \\ \boldsymbol{x}_i^{(t+1)} \leftarrow \boldsymbol{x}_i^{(t)} + \alpha \otimes L(\lambda) & \text{if } r > p_a. \end{cases} \tag{32}$$

As our cuckoo search algorithm is a stochastic search algorithm, we can summarize it as the following key steps:

1. Randomly generate an initial population of n nests at the positions, $\boldsymbol{X} = \{\boldsymbol{x}_1^0, \boldsymbol{x}_2^0, \ldots, \boldsymbol{x}_n^0\}$, then evaluate their objective values so as to find the current global best $\boldsymbol{g}_t^0$.
2. Update the new solutions/positions by

$$\boldsymbol{x}_i^{(t+1)} = \boldsymbol{x}_i^{(t)} + \alpha \otimes L(\lambda). \tag{33}$$

3. Draw a random number r from a uniform distribution [0,1]. Update $\boldsymbol{x}_i^{(t+1)}$ if $r > p_a$. Then, evaluate the new solutions so as to find the new, global best $\boldsymbol{g}_t^*$.
4. If the stopping criterion is met, then $\boldsymbol{g}_t^*$ is the best global solution found so far. Otherwise, return to step (2).

The global convergence of an algorithm. If f is measurable and the feasible solution space Ω is a measurable subset on $\Re^n$, algorithm A satisfies the above two conditions with the search sequence $\{x_k\}_{k=0}^{\infty}$, then

$$\lim_{k \to \infty} P(x_k \in R_{\epsilon,M}) = 1. \tag{34}$$

That is, algorithm A can converge globally with a probability of one. Here $P(x_k \in R_{\epsilon,M})$ is the probability measure of the kth solution on $R_{\epsilon,M}$ at the kth iteration.

The state and state space. The positions of a cuckoo/nest and its global best solution g in the search history forms the states of cuckoos: $y = (x, g)$, where $x, g \in \Omega$ and $f(g) \leq f(x)$. The set of all the possible states forms the state space, denoted by

$$Y = \{y = (x, g) | x, g \in \Omega, f(g) \leq f(x)\}. \tag{35}$$

The states and state space of the cuckoo group/population. The states of all n cuckoos/nests form the states of the group, denoted by $q = (y_1, y_2, \ldots, y_n)$. All the states of all the cuckoos form a state space for the group, denoted by

$$Q = \{q = (y_1, y_2, \ldots, y_n), y_i \in Y, 1 \leq i \leq n\}. \tag{36}$$

Obviously, Q contains the historical global best solution g^* for the whole population and all individual best solutions $g_i (1 \leq i \leq n)$ in history. In addition, the global best solution of the whole population is the best among all g_i, so that $f(g^*) = \min(f(g_i)),\ 1 \leq i \leq n$.

The transition probability from state y_1 to y_2 in cuckoo search is

$$P(T_y(y_1) = y_2) = P(x_1 \to x_1')P(g_1 \to g_1')P(x_1' \to x_2)P(g_1' \to g_2), \tag{37}$$

where $P(x_1 \to x_1')$ is the transition probability at Step 2 in cuckoo search, and $P(g_1 \to g_1')$ is the transition probability for the historical global best at this step. $P(x_1' \to x_2)$ is the transition probability at Step 3, while $P(g_1' \to g_2)$ is the transition probability of the historical global best.

For globally optimal solution g_b for an optimization problem $<\Omega, f>$, the optimal state set is defined as $R = \{y = (x, g) | f(g) = f(g_b), y \in Y\}$.

For the globally optimal solution g_b to an optimization problem $<\Omega, f>$, the optimal group state set can be defined as

$$H = \{q = (y_1, y_2, \ldots, y_n) | \exists y_i \in R, 1 \le i \le n\}. \tag{38}$$

All these will ensure that the convergence conditions are met. Further detailed mathematical analysis proves that when the number of iteration approaches sufficiently large [34], the group state sequence will converge to the optimal state/solution set H. Therefore, the cuckoo search has guaranteed global convergence.

5.2 Convergence of the Bat Algorithm

Huang et al. have carried out a detailed convergence analysis for the bat algorithm using the finite Markov process theory [35].

In theory, an algorithm with an order-m reducible stochastic matrix P can be rewritten as

$$P = \begin{pmatrix} S \ldots 0 \\ R \ldots T \end{pmatrix}, \tag{39}$$

where $R \neq 0$, $T \neq 0$, and S is order-q stochastic matrix (with $q < m$). Then, we have

$$\begin{aligned} P^\infty &= \lim_{k \to \infty} P^k \\ &= \lim_{k \to \infty} \begin{pmatrix} S^k & \ldots & 0 \\ \sum_{i=1}^{k-1} T^i R S^{k-i} & \ldots & T^k \end{pmatrix} = \begin{pmatrix} S^\infty \ldots 0 \\ R^\infty \ldots T \end{pmatrix}, \end{aligned} \tag{40}$$

which is a stable stochastic matrix and independent of the initial distribution. In addition, we also have

$$P^{\infty} = [p_{ij}]_{m \times m}, \quad \begin{cases} p_{ij} > 0, & (1 \leq i \leq m, 1 \leq j \leq q), \\ p_{ij} = 0, & (1 \leq i \leq m, q < j \leq m). \end{cases} \tag{41}$$

The search algorithm will converge with almost probability one to the global optimality, starting from any initial random states, if the transition probability p to a better solution/state is $p > 0$. Conversely, if the transition probability p to a worse state is greater, then the algorithm will not converge.

With this main result, it has been proved that PSO will not converge to the global optimality [36], while the bat algorithm will converge to the true global optimality [35].

Huang et al. concluded that for unconstrained function optimization, the bat algorithm satisfies all the conditions for guaranteed global convergence. For non-linear constrained problems, the bat algorithm will converge with additional initialization of orthogonal Latin squares, and has guaranteed global convergence to the true global optimality. They further concluded that

$$S^{\infty} = (1), \quad R^{\infty} = (1, 1, \ldots, 1)^T, \tag{42}$$

and

$$P^{\infty} = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 1 & 0 & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 1 & 0 & \ldots & 0 \end{pmatrix}, \tag{43}$$

which leads to

$$\lim_{t \to \infty} p\{f(\boldsymbol{x}) \to f(\boldsymbol{x}_*)\} = 1. \tag{44}$$

That is, the global convergence is guaranteed.

Huang et al. also proposed a BA variant, called modified bat algorithm (MBA) [35], which can further improve the convergence rate with guaranteed global optimality. They also showed that this variant is suitable for large-scale, global optimization.

For the moment, most convergence studies can provide some results in terms of the long term behaviour of an algorithm during iterations; however, there are not enough results about the convergence rates to indicate how quickly an algorithm can converge and under what conditions. Obviously, more theoretical are highly needed to analyze these algorithms further.

6 Discussions and Open Problems

Despite the huge success of nature-inspired algorithms, there are still some challenging, open problems that need to be addressed. These open problems include the balance of exploration and exploitation, selection mechanisms, right amount of randomization, parameter tuning as well as parameter control, scalability and others.

- **Mathematical Framework**: It still lacks a general mathematical framework for analyzing the convergence and stability of metaheuristic algorithms. There are some good results using Markov chains, dynamic systems and self-organization theory, but a systematic framework is yet to be developed.
- **Exploration and exploitation**: A key problem is how to balance of exploration and exploitation in an algorithm so that it can deal with a vast range of problems efficiently [37]. In reality, the amount of exploration and exploitation may depend on the type of problem, and therefore, some a *priori* knowledge of the problem to be solved can help to determine such a balance. However, it is not known how to incorporate such knowledge effectively. For example, gradient/derivative information obtained from the objective function can be very useful for exploitation, but if such exploitation is too strong, it can cause the system to be trapped in a local optimum, thus sacrificing the possibility of finding the true global optimality. There may not exist such optimal balance for all problems [38].
- **Selection Mechanism**: Selection mechanism is also very important and it is not known what selection is most effective. A proper selection pressure is crucial to maintain a healthy population. For example, when many solutions have similar fitness, numerically speaking, their fitness values may almost be the same, thus how to select certain solutions becomes tricky. Typical approaches include re-scaled fitness values, ranking of solutions, and adaptive elitism. However, it is not clear if they can work for all algorithms and if there is other better ways to handle selection.
- **Right Amount of Randomness**: In order to balance exploration and exploitation, a right amount of randomness is needed. However, no one knows what amount is the right amount. At one extreme, if there is no randomness, an algorithm becomes a deterministic algorithm, and thus loses the ability to explore. At the other extreme, if the search is dominated by high randomness, the algorithm becomes a random search, and thus significantly reduces its ability to exploit the landscape information. In fact, it is not known how to control randomness properly so as to balance exploration and exploitation most effectively.
- **Parameter Tuning and Control**: As the performance of almost any algorithm will depend on its parameter settings, how to tune these parameters to achieve the best performance is a higher level optimization problem. In fact, this is the optimization of an optimization algorithm. It is still an open question. Similarly,

how to control the parameters by varying their values to achieve the best overall performance is also a key challenging issue.

- **Dynamic Landscape**: The problems that have been solved in the current literature usually have fixed landscape. That is, once the problem is defined, its landscape in the search space remain unchanged. However, for dynamic problems and problems with noise, the search landscape can change with time. In such cases, adaptation can be more sophisticated and challenging. It is not clear if most current methods can still work well in such time-dependent, noisy environments.
- **Scalability**: How to solve high-dimensional, large-scale problems effectively? At the moment, most case studies using metaheuristic algorithms are small-scale problems. It is not clear if these algorithms are scalable to deal with large-scale problems effectively.

Therefore, in-depth understanding and theoretical results are needed. Possible research routes may require a combination of mathematical analysis, numerical simulations, empirical observations as well as other tools such as dynamical system theories, Markov theory, self-organization theory and probability. It may even require a paradigm shift in analyzing metaheuristic algorithms. There is no doubt that any theoretical results will provide tremendous insight into understanding metaheursitic algorithms.

All these challenges can present golden opportunities for further research in analyzing adaptation and diversity in metaheuristic algorithms. It can be expected that more efficient tools may be developed to solve more complex, real-world problems with a diverse range of applications.

References

1. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 1942–1948. Piscataway, NJ (1995)
2. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, UK (2008)
3. Yang, X.S.: Cuckoo Search and Firefly Algorithm: Theory and Applications, Studies in Computational Intelligence, vol. 516, Springer, Berlin (2014)
4. Yang, X.S.: Nature-Inspired Optimization Algorithms. Elsevier, Amsterdam (2014)
5. Ashby, W.R.: Princinples of the self-organizing sysem. In: Von Foerster, H., Zopf, G.W., Jr. (eds.) Pricinples of Self-Organization: Transactions of the University of Illinois Symposium, pp. 255–278. Pergamon Press, London (1962)
6. Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithms for discrite optimization. Artif. Life **5**(2), 137–172 (1999)
7. Fister, I., Fister Jr, I., Yang, X.S., Brest, J.: A comprehensive review of firefly algorithms. Swarm Evol. Comput. **13**(1), 34–46 (2013)
8. Fister, I., Yang, X.S., Brest, J., Fister Jr, I.: Modified firefly algorithm using quaternion representation. Expert Syst. Appl. **40**(18), 7220–7230 (2013)
9. Fister, I., Mernik, M., Filipic, B.: Graph 3-coloring with a hybrid self-adaptive evolutionary algorithm. Comput. Optim. Appl. **54**(3), 741–770 (2013)

10. Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: Proceedings of World Congress on Nature and Biologically Inspired Computing (NaBIC 2009), pp. 210–214. IEEE Publications, USA (2009)
11. Pavlyukevich, I.: Lévy flights, non-local search and simulated annealing J. Comput. Phys. **226** (2), 1830–1844 (2007)
12. Yang, X.S., Deb, S.: Engineering optimization by cuckoo search. Int. J. Math. Model. Num. Optim. **1**(4), 330–343 (2010)
13. Yang, X.S., Deb, S.: Multiobjective cuckoo search for design optimization. Comput. Oper. Res. **40**(6), 1616–1624 (2013)
14. Yang, X.S., Deb, S.: Cuckoo search: recent advances and applications. Neural Comput. Appl. **24**(1), 169–174 (2014)
15. Yang, X.S., Deb, S., Fong, S.: Accelerated particle swarm optimization and support vector machine for business optimization and applications. In: Networked Digital Technologies, Communications in Computer and Information Science, vol. 136, pp. 53–66 (2011)
16. Yang, X.S.: A new metaheuristic bat-inspired algorithm. In: Nature Inspired Cooperative Strategies for Optimisation (NICSO 2010), Studies in Computational Intelligence, vol. 284, pp. 65–74. Springer, New York (2010)
17. Yang, X.S.: Bat algorithm for multi-objective optimisation. Int. J. Bio-Inspired Comput. **3**(5), 267–274 (2011)
18. Fister Jr, I., Fister, D., Yang, X.S.: A hybrid bat algorithm. Elektrotehniski Vestn. **80**(1–2), 1–7 (2013)
19. Yang, X.S., Gandomi, A.H.: Bat algorithm: a novel approach for global engineering optimization. Eng. Comput. **29**(5), 1–18 (2012)
20. Yang, X.S., He, X.S.: Bat algorithm: literature review and applications. Int. J. Bio-inspired Comput. **5**(3), 141–149 (2013)
21. Yang, X.S.: Flower pollination algorithm for global optimization. In: Unconventional Computation and Natural Computation, pp. 240–249. Springer, New York (2012)
22. Yang, X.S.: Flower pollination algorithm for global optimization. In: Unconventional Computation and Natural Computation, Lecture Notes in Computer Science, vol. 7445, pp. 240–249. Springer, New York (2012)
23. Yang, X.S., Karamanoglu, M., He, X.S.: Multi-objective flower algorithm for optimization. Procedia Comput. Sci. **18**(1), 861–868 (2013)
24. Yang, X.S., Karamanoglu, M., He, X.S.: Flower pollination algorithm: a novel approach for multiobjective optimization. Eng. Optim. **46**(9), 1222–1237 (2014)
25. Storn, R.: On the usage of differential evolution for function optimization. Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), pp. 519–523. Berkeley, CA (1996)
26. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)
27. Price, K., Storn, R., Lampinen, J.: Differential Evolution: A Practical Approach to Global Optimization. Springer, Berlin (2005)
28. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization: Harmony search. Simulation **76**(2), 60–68 (2001)
29. Booker, L., Forrest, S., Mitchell, M., Riolo, R.: Perspectives on Adaptation in Natural and Artificial Systems. Oxford University Press, Oxford (2005)
30. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Anbor (1975)
31. Yang, X.S., Deb, S., Loomes, M., Karamanoglu, M.: A framework for self-tuning optimization algorithm. Neural Comput. Appl. **23**(7–8), 2051–2057 (2013)
32. Eiben, A.E., Smit, S.K.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol. Comput. **1**(1), 19–31 (2011)
33. Belavkin, R.V.: Optimal measures and Markov transition kernels. J. Global Optim. **55**(2), 387–416 (2013)

34. Wang, F., He, X.S., Wang, Y., Yang, S.M.: Markov model and convergence analysis based on cuckoo search algorithm. Comput. Eng. **38**(11), 180–185 (2012). (in Chinese)
35. Huang, G.Q., Zhao, W.J., Lu, Q.Q.: Bat algorithm with global convergence for solving large-scale optimization problem. Appl. Res. Comput. **30**(5), 1323–1328 (2013). (in Chinese)
36. Ren, Z.H., Wang, J., Gao, Y.L.: The global convergence of particle swarm optimization based on Markov chain. Control Theory Appl. **2011**, 462–466 (2011). (in Chinese)
37. Blum, C., Roli, A.: Metaheuristics in combinatorial optimisation: overview and conceptural comparision. ACM Comput. Surv. **35**(2), 268–308 (2003)
38. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)