Nature-Inspired Algorithms: Success and Challenges

Xin-She Yang

Abstract The simplicity and flexibility of nature-inspired algorithms have made them very popular in optimization and computational intelligence. Here, we will discuss the key features of nature-inspired metaheuristic algorithms by analyzing their diversity and adaptation, exploration and exploitation, attractions and diffusion mechanisms. We also highlight the success and challenges concerning swarm intelligence, parameter tuning and parameter control as well as some open problems.

Keywords Algorithm · Adaptation · Bat algorithm · Cuckoo search · Diversity · Firefly algorithm · Metaheuristic · Optimization

1 Introduction

Many applications concern hard optimization problems, which may require sophisticated optimization techniques to deal with. However, traditional algorithms usually cannot cope with such highly nonlinear and multimodal problems. Alternative approaches have to be found. In recent years, nature-inspired metaheuristic algorithms have gained huge popularity, and these algorithms include ant colony optimization, particle swarm optimization, cuckoo search, firefly algorithm, bat algorithm, bee algorithms and others [4, 14, 17, 28]. 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.

The main purpose of this chapter is to highlight some key issues in adaptation and diversity in swarm intelligence. Therefore, the chapter is organized as follows. Section 2 outlines some widely used nature-inspired algorithms, followed by a brief discussion of the main mechanisms of generating new solutions in Sect. 3. Section 4

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

N.D. Lagaros and M. Papadrakakis (eds.), *Engineering and Applied Sciences Optimization*, Computational Methods in Applied Sciences 38,
DOI 10.1007/978-3-319-18320-6_8

analyzes adaptation and diversity in swarm intelligence in detail. Section 5 discusses the parameter tuning and control, and finally some conclusions will be drawn briefly, with some discussions for open problems in Sect. 6.

2 Some Recent Algorithms Based on Swarm Intelligence

Before we proceed to carry out any analysis, let us briefly introduce some popular nature-inspired, swarm-intelligence-based algorithms for global optimization.

From a mathematical point of view, an 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. In general, an algoirthm can be written as

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

where p is an algorithm-dependent parameter. A good example is the so-called quasi-Newton method with a step size parameter.

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

$$\begin{pmatrix} \boldsymbol{x}_1 \\ x_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{t+1} = A\Big((\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \ldots, \boldsymbol{x}_n^t); (p_1, p_2, \ldots, p_k); (\epsilon_1, \epsilon_2, \ldots, \epsilon_m)\Big) \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^t, \tag{2}$$

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 [28], and it can also be viewed as a self-organized system [1].

Most nature-inspired algorithms nowadays are swarm intelligence based. Their updating equations vary significantly. However, most algorithms have linear updating equations. For example, particle swarm optimization has two linear equations in terms of $\boldsymbol{x}$. On the other hand, some algorithms such as the firefly algorithm use nonlinear updating equations, which can lead to rich characteristics and potentially higher efficiency.

Linear systems are easier to analyze, while nonlinear systems can be more challenging to analyze. At the moment, it still lacks in-depth understanding how different systems work. In the rest of this section, we will introduce some nature-inspired algorithms.

2.1 PSO

Particle swarm optimization (PSO) is one of the first algorithms that are based on swarm intelligence. PSO was developed by Kennedy and Eberhart in 1995 [14], based on the swarm behaviour of fish or bird schooling in nature. Each particle updates its position $\boldsymbol{x}_i$ and velocity $\boldsymbol{v}_i$, and their evolution is controlled by two learning parameter α and β with typical values of $\alpha \approx \beta \approx 2$ and two random vectors $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ that are uniformly distributed in [0,1]. Briefly speaking, the main equations of PSO are as follows:

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + \alpha \boldsymbol{\epsilon}_1[\boldsymbol{g}^* - \boldsymbol{x}_i^t] + \beta \boldsymbol{\epsilon}_2[\boldsymbol{x}_i^* - \boldsymbol{x}_i^t], \tag{3}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}. \tag{4}$$

There are more than two dozen variants of PSO. For example, Yang et al. developed the accelerated PSO [20], while Fister Jr. et al. used some reasoning techniques to improve the efficiency of PSO [11].

2.2 Firefly Algorithm

The firefly algorithm (FA) is simple, flexible and easy to implement. FA was developed by Yang in 2008 [17], which was based on the flashing patterns and behaviour of tropical fireflies.

One of the main advantages of the FA is that 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

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2}(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha\, \boldsymbol{\epsilon}_i^t, \tag{5}$$

where the second term is due to the attraction, 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. [6–8]. One novel feature of FA is that 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.

2.3 Cuckoo Search

The cuckoo search (CS) was developed in 2009 by Yang and Deb [23]. CS is based on the brood parasitism of some cuckoo species. In addition, this algorithm is enhanced by the so-called Lévy flights [15], rather than by simple isotropic random walks.

Recent studies show that CS is potentially far more efficient than PSO and genetic algorithms [24, 25]. Mathematically speaking, 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

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha s \otimes H(p_a - \epsilon) \otimes (\boldsymbol{x}_j^t - \boldsymbol{x}_k^t), \tag{6}$$

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:

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha L(s, \lambda), \tag{7}$$

where

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s > 0). \tag{8}$$

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

If we look at the CS equations from a mathematical point of view, we can analyze their key features and characteristics, and thus highlight their advantages. 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. In fact, recent studies show that CS can have guaranteed global convergence [28].

On the other hand, 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 a varying mutation rate realized by Lévy flights, and the fitness-proportional generation of new solutions based on the solution similarity provides a subtle form of crossover. In addition, simulations also show that CS can have an autozooming ability in the sense that new solutions can automatically zoom into the region where the promising global optimality is located.

Using the framework of Markov chains and probability, we can see that equation (7) is essentially simulated annealing in the framework of Markov chains. In Eq. (6), if $p_a = 1$ and $\alpha s \in [0, 1]$, CS can degenerate into a variant of differential evolution. Furthermore, if we replace $\boldsymbol{x}_j^t$ by the current best solution $\boldsymbol{g}^*$, then (6) can further degenerate into accelerated particle swarm optimization (APSO) [20]. This means that SA, DE and APSO are special cases of CS, and that is one of the reasons why

CS is so efficient. A brief literature review has been carried out by Yang and Deb [26] and Fister Jr. et al. [9].

2.4 Bat Algorithm

The bat algorithm (BA) is the first algorithm of its kind to use frequency tuning for the optimization purpose. BA was developed by Yang in 2010 [18], inspired by the echolocation behavior of microbats. 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

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{9}$$

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + (\boldsymbol{x}_i^{t-1} - \boldsymbol{x}_*) f_i, \tag{10}$$

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1} + \boldsymbol{v}_i^t, \tag{11}$$

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

The motion of bats are updated by the above equations, but when to update and which branch is updated first are controlled by the loudness and pulse emission rate of each bat. In the most simplest case, the loudness and pulse emission rates are regulated by the following equations:

$$A_i^{t+1} = \alpha A_i^t, \tag{12}$$

and

$$r_i^{t+1} = r_i^0[1 - \exp(-\gamma t)], \tag{13}$$

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

There have been a lot of interest in the study of BA in recent years, and BA has been extended to multiobjective optimization [19] and various variants. For example, Fister et al. have extended to a hybrid bat algorithm [10, 12]. The preliminary results suggested that they are very efficient [21].

Obviously, there are other nature-inspired algorithms such as the flower pollination algorithm [22]. However, as the main purpose of this chapter is to analyze adaptation and diversity in metaheuristic algorithms, we will now focus on the analysis and discussion of the forms of adaptation and diversity and their roles/representations in the actual algorithms.

3 Mechanisms for Generating New Solutions

There are many ways for generating new solutions. However, from the locality point of view, they can be divided into the following subcategories:

- Modification of selected solutions (from the existing population).
- Local modifications.
- Global modifications.
- Mixed (both local and global as well as selected).

One of the most widely used methods for generating new solutions is to select a subset of existing solutions from the evolving population. For example, if two solutions are randomly selected from the existing population, they can be combined to form two new solutions by crossover or recombination. This is one of fundamental mechanisms in genetic algorithms and many evolutionary algorithms. In the simplest case when one solution is selected, some modification on a part (or a few parts) of the solution can be carried out. This is the main mechanism for mutation in genetic algorithms. In fact, these two ways of generating new solutions have paved the ways for most modern evolutionary algorithms.

The above operations can be converted to mathematical equations. Mathematically speaking, crossover can be written as

$$\begin{pmatrix} \boldsymbol{x}_i^{t+1} \\ \\ \boldsymbol{x}_j^{t+1} \end{pmatrix} = C(\boldsymbol{x}_i^t, \boldsymbol{x}_j^t, p_c), \tag{14}$$

where p_c is the crossover probability, though the exact form of $C(\,)$ depends on the actual crossover manipulations. Mutation can be written schematically as

$$\boldsymbol{x}_i^{t+1} = M(\boldsymbol{x}_i^t, p_m), \tag{15}$$

where p_m is the mutation rate. However, the form $M(\,)$ depends on the coding and the number of mutation sites.

On the other hand, the fitness-dependent reproduction of the offsprings may depend on the relative fitness of the parents in the population. In this case, the function form can be even more complex. For example, $C(\,)$ can depend on all the individuals in the population, which may lead to $C(\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \ldots, \boldsymbol{x}_n^t, p_c)$ where n is the population size.

From the mathematical point of view, local modifications are local random walks, which can take many different forms and are usually around an existing solution. For example, from an existing solution $\boldsymbol{x}_i^t$, new solutions can be generated locally by using

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + s(\boldsymbol{x}_i, \alpha), \tag{16}$$

where $s(\boldsymbol{x}_i^t, \alpha)$ is a step size function that can depend on the current solution and a parameter α. If s is small enough, the distance $d = ||\boldsymbol{x}_i^{t+} - \boldsymbol{x}_i||$ is small, which means the new solutions are limited to a neighborhood of the existing solution $\boldsymbol{x}_i^t$. As random walks are widely used for randomization and local search in metaheuristic algorithms [17, 18], a proper step size is very important. As different algorithms use different forms of randomization techniques, it is not possible to provide a general analysis for assessing randomness. In addition, the above form of equation can in general be written in a more compact form as

$$\boldsymbol{x}_i^{t+1} = N(\boldsymbol{x}_i^t, w, \alpha), \tag{17}$$

where $N(\ )$ depends on the random variable w with a parameter α.

Here, randomness increases the diversity of the solutions and thus enables an algorithm to have the ability to jump out of any local optimum. However, too much randomness may slow down the convergence of the algorithm and thus can waste a lot of computational efforts. Therefore, there is some tradeoff between deterministic and stochastic components, though it is difficult to gauge what is the right amount of randomness in an algorithm? In essence, this question is related to the optimal balance of exploration and exploitation, which still remains an open problem.

Global modifications can also take many forms. For example, the simplest form of global modification or global randomization is

$$\boldsymbol{x}_i = \boldsymbol{L} + (\boldsymbol{U} - \boldsymbol{L})\varepsilon, \tag{18}$$

where ε is a random number drawn in $[0, 1]$. This equation gives new solutions between the lower bound $\boldsymbol{L}$ and the upper bound $\boldsymbol{U}$. On the other hand, random walks can be both local and global simultaneously. For example, the method in the cuckoo search uses Lévy flights in terms of

$$\boldsymbol{x}_i^{t+1}(\text{new solution}) = \boldsymbol{x}_i^t(\text{old solution}) + \alpha L(s, \lambda), \tag{19}$$

which can generate both local and global solutions, controlled by α and the intrinsic nature of Lévy flights that provides occasional long-jumps. However, this is just a simple case where the new solution only depends on one existing solution and the randomization term. In general, the solutions can be generated in parallel by random permutation, and thus we may have a more generic form

$$\begin{pmatrix} \boldsymbol{x}_1^{t+1} \\ \boldsymbol{x}_2^{t+1} \\ \vdots \\ \boldsymbol{x}_n^{t+1} \end{pmatrix} = G(\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \ldots, \boldsymbol{x}_n^t, w, \beta), \tag{20}$$

where $G(\ )$ can be very complex, which also depends on the random variable and parameter β. For example, the mutation operator in differential evolution takes the form

$$\boldsymbol{x}_k^{t+1} = \boldsymbol{x}_k + F(\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{21}$$

where i, j and k are random permutations among $1, 2, \ldots, n$, and F is a parameter or constant.

It is worth pointing out that the difference between global or local modifications are subtle. When the step sizes are large enough, local modifications can be become global. Furthermore, these mechanisms for generating new solutions do not always belong to a single mechanism, and they can be a mixture of two or more components. For example, Lévy flights in the cuckoo search can be considered as a mixture of both local and global modifications, while the bat algorithm uses a combination of simple global randomization in one branch and the local modification in another branch, with the additional control for switching between these two branches depending on the loudness and pulse emission rate.

In fact, all good algorithms uses a combination of the above components, not just a simple component. However, how to combine different modification methods is a challenging problem and what is the most efficient combination is yet to be discovered (if it ever exists). Furthermore, such effective combinations may be problem dependent and should be adaptive as well.

4 Adaptation and Diversity in Swarm Intelligence

Adaptation and diversity in metaheuristic algorithms can take many forms, including the balance of exploration and exploitation, generations or moves of new solutions, the right amount of randomness, parameter adjustment and parameter control, and other subtle forms. We will discuss the role of adaptation and diversity in such cases.

4.1 Diversity and Adaptation

The effectiveness of swarm intelligence based 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 [2]. 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. However, 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 the balanced diversity in a population is also important.

On the other hand, 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 [2, 13], 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.

Parameter 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 a parameter is tuned, it 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.

Similarly, 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 the mutation rate 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 variations, often in terms of the 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.

From a different perspective, we can also say that 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 a higher level of 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.

4.2 Exploration and Exploitation

Adaptation and diversity are just one side of the coin. In the context of nature-inspired metaheuristics, the characteristics of an algorithm can also be analyzed in terms of basic components: exploitation and exploration, which are also referred to as intensification and diversification [3, 17].

Roughly speaking, exploitation uses any information obtained from the problem of interest so as to help to generate new solutions that are better than existing solutions. However, this process is typically local, and information (such as gradients) is also local. Therefore, it is for local search. For example, hill-climbing is a method that uses derivative information to guide the search procedure. In fact, new steps always try to climb up the local gradient. The advantage of exploitation is that it usually leads to very high convergence rates, but its disadvantage is that it can get stuck in a local optimum because the final solution point largely depends on the starting point. On the other hand, exploration makes it possible to explore the search space more efficiently, and it can generate solutions with enough diversity and far from the current solutions. Therefore, the search is typically on a global scale. The advantage of exploration is that it is less likely to get stuck in a local mode, and the global optimality can be more accessible. However, its disadvantages are slow convergence and waste of lot computational efforts because many new solutions can be far from global optimality.

Therefore, a fine balance is required so that an algorithm can achieve the best performance. Too much exploitation and too little exploration means the system may converge more quickly, but the probability of finding the true global optimality may be low. On the other hand, too little exploitation and too much exploration can cause the search path meander with very slow convergence. The optimal balance should mean the right amount of exploration and exploitation, which may lead to the optimal performance of an algorithm. Therefore, a proper balance is crucially important.

In essence, the optimal balance is itself a higher-level optimization problem. However, how to achieve such a balance is still an open problem. In fact, no algorithm can claim to have achieve such an optimal balance in the current literature. In essence, the balance itself is a hyper-optimization problem, because it is the optimization of an optimization algorithm. In addition, such a balance may depend on many factors such as the working mechanism of an algorithm, its setting of parameters, tuning and control of these parameters and even the problem to be considered. Furthermore, such a balance may not universally exist [16], and it may vary from problem to problem, thus requiring an adaptive strategy.

4.3 Attraction and Diffusion

The novel idea of attraction via light intensity as an exploitation mechanism was first used by Yang in the firefly algorithm (FA) in 2007 and 2008. In FA, the attractiveness

(and light intensity) is intrinsically linked with the inverse-square law of light intensity variations and the absorption coefficient. As a result, there is a novel but nonlinear term of $\beta_0 \exp[-\gamma r^2]$ where β_0 is the attractiveness at the distance $r = 0$, and $\gamma > 0$ is the absorption coefficient for light [17]. The main function of such attraction is to enable an algorithm to converge quickly because these multi-agent systems evolve, interact and attract, leading to some self-organized behaviour and attractors. As the swarming agents evolve, it is possible that their attractor states will move towards to the true global optimality.

The novel attraction mechanism in FA is the first of its kind in the literature of nature-inspired computation and computational intelligence. This also motivated and inspired others to design similar or other kinds of attraction mechanisms. Other algorithms that were developed later also used inverse-square laws, derived from nature. For example, the charged system search (CSS) used Coulomb's law, while the gravitational search algorithm (GSA) used Newton's law of gravitation.

Whatever the attraction mechanism may be, from the metaheuristic point of view, the fundamental principles are the same: that is, they allow the swarming agents to interact with one another and provide a forcing term to guide the convergence of the population. Attraction mainly provides the mechanisms for exploitation, but, with proper randomization, it is also possible to carry out some degree of exploration. However, the exploration is better analyzed in the framework of random walks and diffusive randomization. From the Markov chain point of view, random walks and diffusion are both Markov chains. In fact, Brownian diffusion such as the dispersion of an ink drop in water is a random walk. Lévy flights can be more effective than standard random walks. Therefore, different randomization techniques may lead to different efficiency in terms of diffusive moves. In fact, it is not clear what amount of randomness is needed for a given algorithm.

All these unresolved issues and problems discussed so far may motivate more research in this area and thus the relevant literature can be expected to expand in the near future.

5 Parameter Tuning and Parameter Control

Adaptation and diversity can also take the form of parameter tuning and parameter control. In fact, one of the most challenging issues when designing metaheuristic algorithms is probably to control exploration and exploitation properly in terms of controlling algorithm-dependent parameters, which is still an open question. It is possible to control attraction and diffusion in algorithms that use such features so that the performance of an algorithm can be influenced in the right way.

Ideally we should have some mathematical relationships that can explicitly show how parameters can affect the performance of an algorithm, but this is an un-resolved problem. In fact, unless for very simple cases under very strict, (often) unrealistic assumptions, no theoretical results exist at all. Obviously, one of the key questions

is how to tune parameters to gain the best parameter values so that an algorithm can perform in the most effective way.

5.1 Parameter Tuning

As an algorithm is a set of interacting Markov chains, we can in general write an algorithm as

$$\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{t+1} = A[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, p_1(t), \ldots, p_k(t), \epsilon_1, \ldots, \epsilon_m] \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}^{t}, \tag{22}$$

which generates a set of new solutions $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^{t+1}$ from the current population of n solutions. In principle, the behaviour of an algorithm is largely determined by the eigenvalues of the matrix A that are in turn controlled by the parameters $p_k(t)$ and the randomness vector $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_m)$.

From the Markovian theory, we know that the first largest eigenvalue is typically 1, and therefore the convergence rate of an algorithm is mainly controlled by the second largest eigenvalue $0 \leq \lambda_2 < 1$ of A. However, it is extremely difficult to find this eigenvalue in general. Therefore, the tuning of parameters becomes a very challenging task. In fact, parameter tuning, or tuning of parameters, is an important topic under active research [5, 27]. The aim of parameter tuning is to find the best parameter setting so that an algorithm can perform most efficiently for a wider range of problems. At the moment, parameter tuning is mainly carried out by detailed, extensive parametric studies, and there is no efficient method in general.

In essence, parameter tuning itself is an optimization problem which requires higher-level optimization methods to tackle. However, a recent study shows that a framework for self-tuning algorithms can be established with promising results [27]. For example, Yang et al. used the firefly algorithm to tune itself so that the firefly algorithm can achieve optimal performance for a given set of problems. This framework can be expected to be applicable to other algorithms and a range of applications.

5.2 Parameter Control

Related to parameter tuning, there is another issue of parameter control. Parameter values after parameter tuning are often fixed during iterations, while parameters should vary for parameter control.

The main idea of parameter control is to vary the parameters so that the algorithm of interest can provide the best convergence rate and thus may achieve the best performance. Again, parameter control is another tough optimization problem to be yet resolved. In the bat algorithm, some basic form of parameter control has been attempted and found to be very efficient [18]. By controlling the loudness and pulse emission rate, BA can automatically switch from explorative moves to local exploitation that focuses on the promising regions when the global optimality may be nearby. Similarly, the cooling schedule in simulated annealing can be considered as a form of basic parameter control.

Both parameter tuning and parameter control are crucial to the performance of all algorithms, and thus deserve more research attention.

6 Discussions and Open Problems

As we have seen from the above detailed analysis, proper adaptation and diversity are crucial to ensure the good performance of an algorithm. Adaptation can be carried out in different components (of an algorithm), such as the generation of the population, selection of solutions, elitism, replacement of solutions, adjustment of parameters and overall balance of exploration and exploitation. Diversity can also appear in many places such as the ways to generate new solutions, selection and replacement of existing solutions, explorative moves, randomization, and most importantly to maintain a good balance in exploration and exploitation.

Despite the 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, and parameter tuning as well as parameter control.

As mentioned in the main text, one of the most challenging problems is how to balance exploration and exploitation in an algorithm so that it can deal with a vast range of problems efficiently. 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. 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 a high level of 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.

Another important issue is the selection mechanism 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 [2]. However, it is not clear if they can work for all algorithms and if there is other better ways to handle selection.

On the other hand, 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.

From the landscape point of view, 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.

It is worth pointing out that whatever the algorithms may be, the role of adaptation and diversity may be subtle in affecting the performance of an algorithm. 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.

Obviously, there are other issues and open problems as well. The above discussion has just focused a few key issues. All these challenges can present golden opportunities for further research in analyzing adaptation and diversity in metaheuristic algorithms. It can be expected more theoretical results will appear in the future, and any theoretical results will provide tremendous insight into understanding metaheursitic algorithms. It is hoped that efficient tools can be developed to solve a wide range of large-scale problems in real-world applications. Future research directions should focus on such key issues and challenges.

References

1. Ashby WR (1962) Princinples of the self-organizing sysem, in: Pricinples of self-organization: transactions of the University of illinois symposium Von Foerster H, Zopf Jr. GW (eds) Pergamon Press, London, pp 255–278
2. Booker L, Forrest S, Mitchell M, Riolo R (2005) Perspectives on adaptation in natural and artificial systems. Oxford University Press, Oxford
3. Blum C, Roli A (2003) Metaheuristics in combinatorial optimisation: overview and conceptural comparision. ACM Comput Surv 35:268–308

4. Dorigo M, Di Caro G, Gambardella LM (1999) Ant algorithms for discrite optimization. Artif Life 5(2):137–172
5. Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evolutionary Comput 1(1):19–31
6. Fister I, Fister I Jr, Yang XS, Brest J (2013) A comprehensive review of firefly algorithms. Swarm Evol Comput 13(1):34–46
7. Fister I, Yang X-S, Brest J, Fister I Jr (2013) Modified firefly algorithm using quaternion representation. Expert Syst Appl 40(18):7220–7230
8. Fister I, Yang XS, Fister D, Fister Jr. I (2014) Firefly algorithm: a brief review of the expanding literature. In: Cuckoo Search Firefly Algorithm: Theor Appl Stud Comput Intell 516:347–360 (Springer, Heidelberg)
9. Fister Jr I, Yang XS, Fister D, Fister I (2014) Cuckoo search: a brief literature review. In: Cuckoo Search Firefly Algorithm: Theor Appl Stud Comput Intell 516:49–62 (Springer, Heidelberg)
10. Fister I Jr, Fister D, Yang XS (2013) A hybrid bat algorithm. Elektrotehniski Vestn 80(1–2):1–7
11. Fister Jr I, Yang XS, Ljubič K, Fister D, Brest J, Fister I (2014) Towards the novel reasoning among particles in PSO by the use of RDF and SPARQL. Sci World J 2014, article ID 121782. doi:10.1155/2014/121782
12. Fister Jr I, Fong S, Brest J, Fister I (2014) A novel hybrid self-adaptive bat algorithm, Sci World J, 2014, article ID 709738. doi:10.1155/2014/709738
13. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Anbor
14. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, Piscataway, NJ, pp 1942–1948
15. Pavlyukevich I (2007) Lévy flights, non-local search and simulated annealing. J. Comput Phys 226(12):1830–1844
16. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
17. Yang XS (2008) Nature-Inspired metaheuristic algorithms. Luniver Press, Bristol
18. Yang XS (2010) A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimisation (NICSO 2010), vol. 284. Springer, Berlin, Studies in Computational Intelligence, pp 65–74
19. Yang XS (2011) Bat algorithm for multi-objective optimisation. Int J Bio-Inspired Computat 3(5):267–274
20. Yang XS, Deb S, Fong S (2011) Accelerated particle swarm optimization and support vector machine for business optimization and applications. Netw Digital Technol 2011, Commun Comput Inf Sci 136:53–66
21. Yang XS, Gandomi AH (2012) Bat algorithm: a novel approach for global engineering optimization. Eng Comput 29(5):1–18
22. Yang XS (2012) Flower pollination algorithm for global optimization. In: Unconventional computation and natural computation, Springer, Berlin, pp. 240–249
23. Yang XS, Deb S (2009) Cuckoo search via Lévy flights. In: Proceeings of world congress on nature & biologically inspired computing (NaBIC 2009). IEEE Publications, USA
24. Yang XS, Deb S (2010) Engineering optimization by cuckoo search. Int J Math Model Numer Optisation 1(4):330–343
25. Yang XS, Deb S (2013) Multiobjective cuckoo search for design optimization. Comput Oper Res 40(6):1616–1624
26. Yang XS, Deb S (2014) Cuckoo search: recent advances and applications. Neural Comput Appl 24(1):169–174
27. Yang XS, Deb S, Loomes M, Karamanoglu M (2013) A framework for self-tuning optimization algorithm. Neural Comput Appl 23(7–8):2051–2057
28. Yang XS (2014) Nature-Inspired optimization algorithms. Elsevier, London