Journal of Computational Science 46 (2020) 101104

Contents lists available at ScienceDirect

Journal of Computational Science

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

Nature-inspired optimization algorithms: Challenges and open problems

Xin-She Yang

School of Science and Technology, Middlesex University, London NW4 4BT, UK

ARTICLE INFO

Article history:
Received 4 January 2020
Accepted 5 March 2020

Keywords:
Algorithm
Bat algorithm
Convergence
Cuckoo search
Differential evolution
Firefly algorithm
Flower pollination algorithm
Metaheuristic
Nature-inspired computation
Optimization
Particle swarm optimization
Stability
Swarm intelligence

ABSTRACT

Many problems in science and engineering can be formulated as optimization problems, subject to complex nonlinear constraints. The solutions of highly nonlinear problems usually require sophisticated optimization algorithms, and traditional algorithms may struggle to deal with such problems. A current trend is to use nature-inspired algorithms due to their flexibility and effectiveness. However, there are some key issues concerning nature-inspired computation and swarm intelligence. This paper provides an in-depth review of some recent nature-inspired algorithms with the emphasis on their search mechanisms and mathematical foundations. Some challenging issues are identified and five open problems are highlighted, concerning the analysis of algorithmic convergence and stability, parameter tuning, mathematical framework, role of benchmarking and scalability. These problems are discussed with the directions for future research.

© 2020 Elsevier B.V. All rights reserved.

1. Introduction

Many real-world applications involve the optimization of certain objectives such as the minimization of costs, energy consumption, environment and the maximization of performance, efficiency and sustainability. In many cases, the optimization problems that can be formulated are highly nonlinear with multimodal objective landscapes, subject to a set of complex, nonlinear constraints. Such problems are challenging to solve. Even with the ever-increasing power of modern computers, it is still impractical and not desirable to use simple brute force approaches. Thus, whenever possible, efficient algorithms are crucially important to such applications. However, efficient algorithms may not exist for most of the optimization problems in applications. Though there are a wide range of optimization algorithms such as gradient-based algorithms, the interior-point method and trust-region method, most of such algorithms are gradient-based and local search algorithms [1,2], which means that the final solutions may depend on the initial starting points. In addition, the computation of derivatives can be computationally expensive, and some problems such as the objective with discontinuities may not have derivatives in certain regions.

A recent trend is to use evolutionary algorithms such as genetic algorithm (GA) [3], and swarm intelligence (SI) based algorithms. In fact, a wide spectrum of SI-based algorithms have emerged in the last decades, including ant colony optimization (ACO) [4], particle swarm optimization (PSO) [5], bat algorithm (BA) [6], firefly algorithm (FA) [7], cuckoo search (CS) [8] and others [2]. These nature-inspired algorithms tend to be global optimizers, using a swarm of multiple, interacting agents to generate the search moves in the search space. Such global optimizers are typically simple, flexible and yet surprisingly efficient, which have been shown in many applications and case studies [9–12,2]. In the last three decades, significant progress has been and various applications have appeared. This paper will briefly summarize some of these important developments.

Despite the extensive studies and developments, there are still some important issues concerning swarm intelligence and nature-inspired algorithms. Firstly, there still lacks a unified mathematical framework to analyze these algorithms. Consequently, it lacks in-depth understanding how such algorithms may converge and how quickly they can converge. Secondly, there are many different algorithms and their comparison studies have mainly based on

E-mail address: x.yang@mdx.ac.uk

https://doi.org/10.1016/j.jocs.2020.101104
1877-7503/© 2020 Elsevier B.V. All rights reserved.

numerical experiments, and it is difficult to justify if such comparison is always fair. Thirdly, most of the applications in the literature concern small-scale problems, and it is not clear if such approaches can be directly applied to large-scale problems. Finally, it is not clear what are the conditions for the emergence of swarming and intelligence behaviour, even though the term 'swarm intelligence' is used widely. All these mean that a systematical review and analysis is needed, and this paper is a preliminary attempt to analyze nature-inspired algorithms in a comprehensive and unified manner.

Therefore, this paper is organized as follows. Section 2 briefly review and summarize some of the recent SI-based algorithms, with the emphasis on their main characteristics. Section 3 focuses on the search mechanisms and their possible mathematical foundations. Section 4 attempts to highlight some of the main issues concerning nature-inspired algorithms from different perspectives, and outline some open problems and future research directions.

2. Nature-inspired optimization algorithms

There are many nature-inspired algorithms in the current literature, it is estimated there are more than 100 different algorithms and variants [13,9,14–18]. It is not our intention to review all of them. Instead, our emphasis will be on the typical characteristics of algorithms and search mechanisms, and consequently we have selected only a few algorithms in our discussions here. Though different algorithms can be described in different ways, it would be convenient for the discussions later if we can subdivide the descriptions of algorithms into two categories: procedure-based and equation-based.

2.1. Procedure-based algorithms

Though the genetic algorithm (GA) can have quite rigorous mathematical analyses [3,19], it is mainly a procedure as an optimization algorithm. Its main steps are carried out in an iterative manner, and its main procedure consists of three parts:

- Solution representations: A solution vector $\boldsymbol{x}$ to a D-dimensional problem is usually represented or encoded as a binary string of a fixed length or a string of real numbers.
- Solution modifications: Solutions can be modified by mutation or crossover. Mutation can be applied to a single solution at a single place or multiple places, while crossover is carried out over two parent solutions by mixing or swap relevant parts to form new solutions.
- Solution selection: The fitness of a solution is evaluated, usually in terms of its objective value. The selection of a solution among a population is carried out according to its fitness (higher values for maximization problems) and the best solutions are usually passed onto the next generation.

This iterative procedure is relative generic for many algorithms. For example, the evolutionary strategy (ES) can also fit into the above procedure, though crossover is not used in ES. In addition, the ant colony optimization (ACO) [4] can also fit into the above steps, though solution modifications are not by mutation or crossover. Chemical pheromone is used to represent the fitness of a solution, and the modification of solutions are by pheromone deposition and evaporation.

2.2. Equation-based algorithms

A vast majority of the recent nature-inspired algorithms for optimization are equation-based where all solution vectors $\boldsymbol{x}_i (i = 1, 2, \ldots, n)$ are represented as a population set of n solutions in a D-dimensional search space. In this sense, all different algorithms use the same type of vector representations of solutions.

In addition, the selection of the solutions is mainly based on their fitness values. The fittest solutions (higher objective values for maximization, or lower objective values for minimization) are most likely to be passed onto the next generation in the population. Though there some subtle form of selection, such as fitness-proportional elitism, the essence of solution selection is basically the same.

Consequently, the main differences among different nature-inspired optimization algorithms now are the ways of solution modifications, usually using different mathematical forms or search mechanisms. In general, a solution vector $\boldsymbol{x}_i^t$ at iteration or generation t is a position vector, and the new solution $\boldsymbol{x}_i^{t+1}$ is generated by a modification increment or mutation vector $\Delta\boldsymbol{x}_i^t$. That is

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

which dictates the main differences between different algorithms. Traditionally, this increment is a step size (or a step vector). In case of gradient-based algorithms such as the Newton-Raphson method, this step is linked to the negative gradient

$$\Delta\boldsymbol{x}_i^t = -\eta\nabla f(\boldsymbol{x}), \tag{2}$$

where ∇f is the gradient of the objective function $f(\boldsymbol{x})$, and $\eta > 0$ is the so-called learning parameter [1,2].

In some nature-inspired algorithms, the modification in $\Delta\boldsymbol{x}_i^t$ is often related to the increment of velocity modification $\Delta\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t+1} - \boldsymbol{v}_i^t$ such as

$$\Delta\boldsymbol{x}_i^t = \boldsymbol{v}_i^t \quad \Delta t, \tag{3}$$

where $\boldsymbol{v}_i^t$ and $\boldsymbol{v}_i^{t+1}$ are the velocity for the solution i (particle, agent, etc.) at iterations t and $t+1$, respectively. Here, Δt is the time increment. As all algorithms are iterative or time-discrete dynamical systems, the time step or increment Δt is essentially the difference in the iteration counter t, which means that $\Delta t = 1$ can be used for all these algorithms. Consequently, there is no need to worry about the units of these quantities and thus consider all quantities in the same units.

Now let us discuss the equations for modifying solutions in different algorithms.

- Differential evolution (DE): In differential evolution [20], the main mutation is realized by

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + F(\boldsymbol{x}_j^t - \boldsymbol{x}_k^t), \tag{4}$$

  which can be written as

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \Delta\boldsymbol{x}_i^t, \quad \Delta\boldsymbol{x}_i^t = F(\boldsymbol{x}_j^t - \boldsymbol{x}_k^t), \tag{5}$$

  where $\boldsymbol{x}_i^t, \boldsymbol{x}_j^t$ and $\boldsymbol{x}_k^t$ are three distinct solution vectors from the population. The parameter $F \in (0, 2)$ controls the mutation strength.
- Particle swarm optimization (PSO) [5]: The main inspiration of PSO comes from the swarming behaviour of birds and fish. The position and velocity of particle i at any iteration or pseudo-time t can be updated iteratively using

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

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \Delta\boldsymbol{x}_i^t, \tag{7}$$

  with

$$\Delta\boldsymbol{x}_i^t = \boldsymbol{v}_i^{t+1}\Delta t = \boldsymbol{v}_i^{t+1}, \tag{8}$$

and

$$\Delta \boldsymbol{v}_i^t = \alpha \boldsymbol{\varepsilon}_1 [\boldsymbol{g}^* - \boldsymbol{x}_i^t] + \beta \boldsymbol{\varepsilon}_2 [\boldsymbol{x}_i^* - \boldsymbol{x}_i^t], \tag{9}$$

where $\boldsymbol{\varepsilon}_1$ and $\boldsymbol{\varepsilon}_2$ are two uniformly distributed random numbers in [0,1]. Here, $\boldsymbol{g}^*$ is the best solution of the population at iteration t, while $\boldsymbol{x}_i^*$ is the individual best solution for particle i among its search history up to iteration t.

- Firefly algorithm (FA) [21,7]: The main characteristics of FA are based on the attraction and flashing behaviour of tropical fireflies. The position vector $\boldsymbol{x}_i$ of firefly i at iteration t is updated by

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \Delta \boldsymbol{x}_i^t, \tag{10}$$

and

$$\Delta \boldsymbol{x}_i^t = \beta_0 e^{-\gamma r_{ij}^2} (\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha \boldsymbol{\varepsilon}_i^t, \tag{11}$$

where $\beta_0 > 0$ is the attractiveness at zero distance, that is $r_{ij} = 0$. The scale-depending parameter γ controls the visibility of fireflies, while α essentially controls the strength of randomization in FA.

- Bat algorithm (BA) [6]: The main inspiration of BA is based on the echolocation of microbats and the associated frequency-tuning characteristics in a range from $f_{\min}$ to $f_{\max}$, in combination with varying pulse emission rate and loudness [22,23].

The position of a bat is updated by

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + \Delta \boldsymbol{v}_i^t, \tag{12}$$

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

with

$$\Delta \boldsymbol{v}_i^t = (\boldsymbol{x}_i^{t-1} - \boldsymbol{x}_*)[f_{\min} + \beta(f_{\max} - f_{\min})], \qquad \Delta \boldsymbol{x}_i^t = \boldsymbol{v}_i^t \Delta = \boldsymbol{v}_i^t, \tag{14}$$

where $\boldsymbol{x}_*$ is the best solution among the population of n bats, and β is a random number in [0,1].

- Cuckoo search (CS) [8]: CS was based on the aggressive reproduction strategy of some cuckoo species and their interactions with host species such as warblers [24]. The eggs laid by cuckoos can be discovered and thus abandoned with a probability p_a, realized by a Heaviside step function H with the use of a random number ε in [0,1]. The similarity of two eggs (solutions $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$) can be roughly measured by their difference $(\boldsymbol{x}_j - \boldsymbol{x}_k)$. Thus, the position at iteration t can be updated [25] by

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \Delta \boldsymbol{x}_i^t, \tag{15}$$

where

$$\Delta \boldsymbol{x}_i^t = \alpha s \otimes H(p_a - \varepsilon) \otimes (\boldsymbol{x}_j^t - \boldsymbol{x}_k^t). \tag{16}$$

The step size s, scaled by a parameter α so as to limit its strength, is drawn from a Lévy distribution with an exponent λ [26]. The generation of this step size can be realized by some sophisticated algorithms such as the Mantegna's algorithm [27].

- Flower pollination algorithm (FPA) [28]: The FPA was mainly based on the pollination processes and characteristics of flowering plants [28,29], including biotic and abiotic pollination as well as flower constancy [30]. The solution vector $\boldsymbol{x}_i$ of a pollen particle i can be simulated by

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \Delta \boldsymbol{x}_i^t, \tag{17}$$

and

$$\Delta \boldsymbol{x}_i^t = \begin{cases} \gamma L(\lambda)(\boldsymbol{g}_* - \boldsymbol{x}_i^t), & \text{if} r < p, \\ \varepsilon(\boldsymbol{x}_j^t - \boldsymbol{x}_k^t), & \text{otherwise.} \end{cases} \tag{18}$$

Here, r is a uniformly distributed random number in [0, 1], and γ is a scaling parameter. $\boldsymbol{g}_*$ is the best solution found so far at iteration t. In the above equation, $L(\lambda)$ can be considered as a random number vector to be drawn from a Lévy distribution with an exponent of λ [26].

- Other algorithms: There are many other nature-inspired algorithms, such as simulated annealing [31], bacteria foraging optimization [32], biogeography-based optimization [33], gravitational search [34], charged particle system [35], black-hole algorithm [36], krill herd algorithm [37], eagle strategy [38] and others. However, their main differences are in the ways of generating $\Delta \boldsymbol{x}_i^t$ and $\Delta \boldsymbol{v}_i^t$ from the population of the existing solutions.

Most recently, new variants and applications are appearing regularly, including local ant system for allocating robot swarms [39], hybrid ant and firefly algorithms [40], usability feature selection by MBBAT [41], vehicle routing [42,43] and others [17,44,18].

Though the expressions of $\Delta \boldsymbol{x}_i^t$ and $\Delta \boldsymbol{v}_i^t$ may be different and some algorithms do not use velocity at all, the detailed underlying search mechanisms may also be very different, even for seemingly similar expression of $\Delta \boldsymbol{x}_i^t$. For example, the mutation in Eq. (5) of differential evolution seems to be similar to the form in Eq. (7) for PSO. However, the former was done by random permutation, while the latter was done by a difference vector with perturbed directions using a uniform distribution. Similarly, the modification in Eq. (14) in BA uses frequency tuning by varying frequencies, though it has some similarity to the mathematical term in Eq. (7). Therefore, in order to gain better insight, we should analyze the underlying search mechanisms and their mathematical or statistical foundations.

2.3. Applications

Before we discuss various search mechanisms in nature-inspired algorithms, let us briefly outline some of their recent applications. These algorithms have been applied in almost every area of science, engineering and industry, from engineering optimization [45–50] and deep learning [51] to the coordination of swarming robots [44,52] and the travelling salesman problem [53,54,42,43].

For comprehensive reviews, please refer to some recent review articles [55,13,56,16,25,17,18] and books [9,11,12].

3. Search mechanisms and theoretical foundations

Different algorithms usually use different search mechanisms, and these search moves are often based on the underlying probability distributions. Though solution modifications or perturbations are largely part of mutation, we now focus on their statistical foundations and underlying mechanisms. Loosely speaking, we can put the ways of modifying or perturbing existing solutions into five categories: gradient-guided moves (GGM), random permutation (RP), direction-based perturbations (DBP), isotropic random walks (IRW), and long-tailed, scale-free random walks (LTRW).

Gradient-guided moves are mainly used in gradient-based optimization algorithms such as the Newton-Raphson method. The modification is parallel to the gradient direction, and the step length can be controlled by a learning parameter.

Random permutation tends to mix up a set of n solutions, and then $k \geq 1$ solutions are randomly selected to generate new solutions. Random permutations are used in many algorithms, such as DE and FPA.

Direction-based perturbations are used in many algorithms with the term of $(\boldsymbol{g}_* - \boldsymbol{x}_i)$ or $(\boldsymbol{x}_j - \boldsymbol{x}_k)$. The difference between any two vectors such as $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$ determines a direction, but this direction is then perturbed by multiplying by a uniformly distributed random number ε. Thus, the actual directions of the moves are randomly distributed within a cone.

Random walks are a general framework for solution perturbations [57]. If we consider the current solution $\boldsymbol{x}_i^t$ as the current state S_t at time t, the next state (or solution) can be achieved by a local move w_{t+1}

$$S_{t+1} = S_t + w_{t+1}. \tag{19}$$

Here, we use the non-bold form to denote the state in the D-dimensional space, and perturbations can be done in a dimension by dimension manner. Here, w_{t+1} can be an array of random numbers to be drawn from the Gaussian distribution

$$w_{t+1} \sim N(0, 1), \tag{20}$$

which means that the random walk becomes a Brownian motion. Here, the notation '$\sim$' emphasizes that the random steps should be drawn from the probability distribution described by the right-hand side of the equation. As the iteration time is discrete, the pseudotime counter t can be replaced by the number ($N = t$) of steps, which means that the average distance d_N covered by a Brownian random walk is

$$d_N \propto \sqrt{N}. \tag{21}$$

This square-root law is a typical feature for many diffusion phenomena [57]. If the steps are drawn from Gaussian distributions, the random walks are isotropic random walks.

However, some probability density distributions can have a long tail, or a heavy tail. If the steps are drawn from a heavy-tailed or long-tailed distribution, the random walks can become non-isotropic, long-tailed random walks or even scale-free random networks. A good example of heavy-tailed distributions is the Cauchy distribution

$$p(x, \mu, \gamma) = \frac{1}{\pi\gamma}\left[\frac{\gamma^2}{(x-\mu)^2 + \gamma^2}\right], \quad -\infty < x < \infty, \tag{22}$$

with two parameters μ and γ. Both its mean and variance are infinite or undefined [58,59].

Another important long-tailed distribution is the Lévy distribution, which has been used in nature-inspired computation. Lévy flights are a very special random walk whose steps are drawn from the Lévy distribution.

The rigorous definition of Lévy probability distribution can be tricky, involving an integral [60,26]

$$p(x) = \frac{1}{\pi}\int_0^\infty \cos(kx) e^{-\alpha|k|^\beta} dk \quad (0 < \beta \le 2), \tag{23}$$

where $\alpha > 0$. The case of $\beta = 1$ is equivalent to a Cauchy distribution, while $\beta = 2$ leads to a normal distribution. For the practical purpose, we can use the following approximations for large steps (s)

$$L(s) \to \frac{\alpha\beta\Gamma(\beta)\sin(\pi\beta/2)}{\pi|s|^{1+\beta}}, \quad s \gg 0, \tag{24}$$

where $\Gamma(\beta)$ is the standard gamma function. Comparing the variance of the Brownian random walks, the variance or the distance covered by Lévy flights increases much faster [2]. The mean distance covered by the Lévy flights after N steps is

$$d_N \propto N^{(3-\beta)/2}. \tag{25}$$

This power-law feature is typically for super-diffusion phenomena [26].

By analyzing nature-inspired algorithms in great detail, we can summarize the search mechanisms and their underlying statistical characteristics in Tables 1 and 2 . Most algorithms modify their solution population directly without using velocities as shown in Table 1. However, even for the same types of probability distributions, their role and effects on search characteristics can be subtly different, and we summarize their search characteristics loosely in Table 2.

Table 1
Position and velocity modifications in algorithms.

Algorithm	Position increment $\Delta\boldsymbol{x}$	Velocity increment $\Delta\boldsymbol{v}$
Newton-Raphson	GGM	None
PSO	DBP	DBP
DE	RP, DBP	None
CS	RP, DBP, LTRW	None
SA	IRW	None
FA	DBP, IRW	None
BA	RP, DBP	RP, DBP
FPA	DBP, LTRW	None

Table 2
Nature-inspired algorithms and their search characteristics.

Algorithm	Probability Distribution	Search Characteristics
PSO	Uniform	Guided search towards $\boldsymbol{g}_*$
DE	Uniform	Auto-scaling search
	Permutation	Random mutation
CS	Lévy flights	Self-similar search moves
	Long-tailed	Scale-free search
FA	Gaussian, uniform	Nonlinear attraction
BA	Uniform	Frequency-tuning
		Fitness-dependent switching
FPA	Uniform	Scale-free search
	Lévy flights	Jumps biased towards $\boldsymbol{g}_*$

4. Challenges and open problems

Despite the effectiveness of nature-inspired algorithms and their popularity, there are still many challenging issues concerning such algorithms, especially from theoretical perspectives. Though researchers know the basic mechanisms of how such algorithms can work in practice, it is not quite clear why they work and under exactly what conditions. In addition, all nature-inspired algorithms have algorithm-dependent parameters, and the values of these parameters can affect the performance of the algorithm under consideration. However, it is not clear what the best values or settings are and how to tune these parameters to achieve the best performance. Furthermore, though there are some theoretical analyses of some nature-inspired algorithms [61,62], it still lacks a unified mathematical framework to analyze all algorithms to get in-depth understanding of their stability, convergence, rates of convergence and robustness.

In the rest of this paper, we will highlight five open problems concerning nature-inspired algorithms: mathematical framework for stability and convergence, parameter tuning, role of benchmarking, performance measures for fair comparison, and large-scale scalability.

4.1. Mathematical framework

As almost all algorithms for optimization are iterative, traditional numerical analysis tends to use fixed-point theorems to see if it is possible to show the conditions for such theorems are satisfied. Basically, an iterative algorithm means that a new solution $\boldsymbol{x}_{k+1}$ can be obtained from the current solution $\boldsymbol{x}_k$ by an algorithm A with a parameter α or a set of parameters. That is

$$\boldsymbol{x}_{k+1} = A(\boldsymbol{x}_k, \alpha), \tag{26}$$

If we omit the bold font and use the standard notations in numerical analysis, we can write the above equation simply as

$$x_{k+1} = A(x_k), \tag{27}$$

without stating α explicitly. Using the properties of function composites, we have

$$x_{k+1} = A^{k+1}(x_0) = (A \circ A^k)(x_0), \tag{28}$$

where x_0 is the initial starting point.

From the well-known Banach fixed-point theorem [63,64], we know that a fixed point x_* can exist if $A(x_*) = x_*$ under the condition that certain distance metric $\rho(.,.)$

$$\rho(A(x_i), A(x_j)) \le \theta \quad \rho(x_i, x_j), \quad 0 \le \theta < 1, \tag{29}$$

for all x_i and x_j. This requires that ρ is a shrinking or contracting metric. If a fixed point exists, it is possible to approach this point iteratively via

$$\lim_{k\to\infty} x_{k+1} = \lim_{k\to\infty} A^{k+1}(x_0) = x_*. \tag{30}$$

However, for most nature-inspired algorithms, this condition may not be true at all [57].

Alternatively, we can view the iterative system such as Eq. (26) as a dynamical system, which allows us to analyze its behaviour in the framework of dynamical system theory. For example, the analysis of PSO was first carried out using dynamical system theory [61].

In case when an algorithm is linear in terms of its position or solution vectors. It is possible to write the updating equations as a set of linear equations as a time-discrete linear dynamical system

$$x_{k+1} = Bx_k, \tag{31}$$

where B becomes a linear mathematical operator on x_k [65]. Its solution can be written as

$$x(k) = B^k x_0. \tag{32}$$

The Lyapunov stability requires that all the n eigenvalues λ_i of B must satisfy

$$|\lambda_i| \le 1. \tag{33}$$

If $|\lambda_i| < 1$ (without equality), the algorithm or system becomes globally asymptotically stable.

Using a dynamical system framework, Chen et al. [62] studied an extended bat algorithm system

$$v_{k+1} = -\zeta x_k + \theta v_k + \zeta g, \tag{34}$$

$$x_{k+1} = x_k + \theta v_k + \zeta g - \zeta x_k, \tag{35}$$

where v_k and x_k are the velocity and position of a bat at iteration k. Here, g is the best solution found by the current population. ζ and θ are two parameters.

The above two equations can be rewritten compactly as

$$Y_{k+1} = CY_k + Mg, \tag{36}$$

where

$$Y_k = \begin{bmatrix} x_k \\ v_k \end{bmatrix}, \qquad C = \begin{bmatrix} 1-\zeta & \theta \\ -\zeta & \theta \end{bmatrix}, \qquad M = \begin{bmatrix} \zeta \\ \zeta \end{bmatrix}. \tag{37}$$

Their analysis obtained some stability conditions for the parameters so that

$$\begin{cases} -1 \le \theta \le +1, \\ \zeta \ge 0, \\ 2\theta - \zeta + 2 \ge 0. \end{cases} \tag{38}$$

They also used numerical experiments confirmed such stability [62].

Another common way for analyzing the probabilistic convergence is to use Markov chain theory. There are some extensive studies in this area. For example, the convergence of the genetic algorithm has been analyzed in terms of Markov chains [66–69], and this framework has been applied to analyze the cuckoo search algorithm [70] and the bat algorithm [62]. However, these studies have focused on the probabilistic convergence, but there still lacks information about the rate of convergence.

From the Markov chain theory [58,71], we know that the largest eigenvalue of a proper Markov chain is $\lambda_1 = 1$, it is believed that the second largest eigenvalue $0 < \lambda_2 < 1$ controls the error variations $||E||$ or the rate of convergence

$$||E|| \le C(1-\lambda_2)^k, \tag{39}$$

where $C > 0$ is a positive constant, which depends on the exact forms of the chains. In principle, the chain should converge as $k \to \infty$, but it can be very challenging to figure out this eigenvalue λ_2. There is almost no literature on this topic in the context of nature-inspired algorithms. Thus, an open problem in this area is as follows:

Open problem 1. How to build a unified framework for analyzing all nature-inspired algorithms mathematically, so as to obtain in-depth information about their convergence, rate of convergence, stability, and robustness?

As we have seen from the above, it seems that this framework may require a multidisciplinary approach to combine different mathematical, stochastic and numerical methods so that we can study algorithms from different perspectives.

4.2. Parameter tuning

All nature-inspired algorithms have algorithm-dependent parameters, though the number of parameters can vary greatly. For traditional algorithms such as quasi-Newton methods, the tuning of a single parameter can have rigorous mathematical foundations [1,72–74]. However, for nature-inspired algorithms, the tuning is mainly empirical or by parametric studies [75].

Loosely speaking, an algorithm with m parameters $\boldsymbol{p}_m$=$(p_1, p_2, \ldots, p_m)$ can be written schematically as

$$\boldsymbol{x}_{k+1} = A(\boldsymbol{x}_k | p_1, p_2, \ldots, p_m, \varepsilon_1, \ldots, \varepsilon_s), \tag{40}$$

where $\varepsilon_1, \ldots, \varepsilon_s$ are s different random numbers, which can be drawn from different probability distributions. To a certain degree, all these random numbers are drawn iteratively, thus the tuning of an algorithm will mainly be about the m parameters. Thus, we can compactly write the above as

$$\boldsymbol{x}_{k+1} = A(\boldsymbol{x}_k, \boldsymbol{p}_m). \tag{41}$$

If we use an algorithm B to tune this algorithm, how was algorithm B tuned initially? If we used another algorithm C to tune algorithm B, how did we tune C in the first place? In principle, we should use a well-tuned algorithm (or an algorithm without any parameters) to tune a new algorithm. Thus, a key issue is how to tune an unknown algorithm properly?

Systematical brute-force tuning can be very time-consuming if the number of parameters is large. In addition, there is no guarantee that a well-tuned algorithm works well for one type of problems can work well for a different type of problems. It may be the case that parameter settings of an algorithm can be algorithm-dependent and problem-dependent if we want to maximize the overall performance. In addition, even if an algorithm is tuned, its parameters become fixed after tuning. However, there is no reason that we cannot vary the parameter during iterations. In fact, some studies showed that the variations of a parameter can be advantageous, which leads to self-adaptive variants. For example, self-adaptive differential evolution seems to work better than its original version [76].

One way of tuning algorithms is to consider parameter tuning as a bi-objective process so as to form a self-tuning framework [77],

where the algorithm to be tuned can be used to tune itself. This can still be a very time-consuming approach. Now we have another open problem concerning parameter tuning and parameter control.

Open problem 2. How to best tune the parameters of a given algorithm so that it can achieve its best performance for a given set of problems? How to vary or control these parameters so as to maximize the performance of an algorithm?

4.3. Role of benchmarking and no-free-lunch theorem

For any new algorithm, especially a new nature-inspired algorithm, an important study is to use benchmark functions to test how the new algorithm may perform, in comparison with other algorithms. Such benchmarking allows researchers to gain better understanding of the algorithm in terms of its convergence behaviour, stability and advantages as well as disadvantages. However, the key question is what benchmarks should be used.

In the current literature, the benchmarking practice seems to use a set of test functions with different properties (such as mode shapes, separability and optima locality), there are many such benchmark test functions [78] and some test suites designed by different conferences or research groups. As these functions are typically smooth, defined on some regular domains, they can serve some purpose, but such benchmarking is not actually much use in practice. There are many reasons, but we only highlight two here. One reason is that these functions are often well-designed and sufficiently smooth, while real-world problems are much more diverse and can be very different from these test functions. Another reason is that these test functions are typically unconstrained or with simple constraints on regular domains, while the problems in real-world applications can have many nonlinear complex constraints and the domains can be formed by many isolated regions or islands. Consequently, algorithms work well for test functions cannot work well in applications.

For an algorithm to be validated properly, testing and validation should include test functions with irregular domains, subject to various constraints. For example, if an algorithm has been tested for all the benchmarks in the literature, there is no guarantee that it can still be effective to solve other problems such as the following newly designed, seemingly simple, two-dimensional function $f(x, y)$

$$f(x,y) = \sum_{i=-N}^{N}\sum_{j=-N}^{N}(|i|+|j|)\exp\left[-a(x-i)^2 - a(y-j)^2\right], \tag{42}$$

in the domain of

$$|x-i| + |y-j| \le b = \frac{1}{a}, \quad \forall i, j, \tag{43}$$

where i, j are integers, $N = 100$ and $a = 10$. This function has $4(N+1)^2$ local peaks, but it has four highest peaks at four corners; however, its domain is formed by many isolated regions, or $4(N+1)^2 = 40,401$ regions.

In the current literature, there are many different optimization algorithms. A key question naturally arises: Which is the best one to use? Is there a universal tool that can be used to solve all or at least a vast majority of optimization problems? The simple truth is that there are no such algorithms. This conclusion has been formalized by Wolpert and Macready in 1997 in their influential work on the no-free-lunch (NFL) theorem [79]. The NFL theorem states that if an algorithm A can outperform another algorithm B for finding the optima of some objective functions, then there are some other functions on which B will outperform A. In other words, both A and B can perform equally well over all these functions if their performance is averaged over all possible problems or functions.

But the conclusions from most studies in terms of benchmarking seem to indicate that some algorithms are better than others. In practice, we know that some algorithms are indeed better than others, and the quicksort for sorting numbers is indeed better than a method based on simple pair-wise comparison. Now how do we resolve this seemingly contradiction? The key to resolve this issue lies in the keywords 'all' and 'average'. In practical problem-solving, we are always concerned with a particular set of problems, not all problems. We are also concerned with the actual individual performance of solving a particular problem, not the averaged performance over all problems. As result, benchmarking using a finite set of algorithms and a finite set of functions becomes a zero-sum ranking problem [80,57]. On the other hand, recent studies seemed to indicate that free lunches may exist [81,82], especially for continuous optimization [81], multi-objective optimization [82] or co-evolution [83]. Therefore, we now have an open problem concerning benchmarking.

Open problem 3. What types of benchmarking are useful? Do free lunches exist, under what conditions?

4.4. Performance measures

For the benchmarking comparison of different algorithms, the conclusions can be influenced by the performance metrics used. To make a comparison, researchers have to select appropriate performance measures. In the current literature, comparison studies are mainly concerned with the accuracy, computational efforts, stability, and success rates.

For a given set of problems and a few algorithms, the algorithms obtained the most accurate solutions in comparison with some known or analytical solutions are considered better. Obviously, this will depend on the accuracy level and the stopping criteria used. Obviously, if one algorithm runs longer than others, even an ineffective algorithm may be able to obtain sufficiently good results if allowed to run much longer. Thus, to be fair, all algorithms should use the same computational efforts, which is usually realized by fixing the number of function evaluations.

An alternative approach is to use a fixed accuracy and compare the number of function calls or evaluations as a measure of computational costs. Algorithms with the smaller numbers of function evaluations are considered better. Even for the same number N of function calls, there are different ways of using this fixed budget. If one algorithm first runs half of N (or any other values) evaluations and select solutions, and then feed them into the run of the second half of N evaluations, the performance may be different from the execution of the same algorithm with a single run of N evaluations. Such different ways of implementing the same algorithm may lead to mixed conclusions.

Due to the statistical nature of nature-inspired algorithms, results are not exactly repeatable, and thus multiple runs are needed so as to get meaning statistics. Thus, some researchers use the best objective value obtained at the final iteration, together with their means, standard deviations and other statistics. This may give a fuller picture about the algorithms. Though a smaller standard derivation may indicate that the algorithm is more robust, but this may be linked to the problem under consideration. In addition, the ways of initializing the population and the probability distributions used in the algorithm may also influence such results, though it is not clear how initialization may exactly affect the final results.

Another measure used for comparison is the success rate. For multiple runs (N_r), there may be N_s times that an algorithm is able to find the optimal solution, which means that the success rate is the ratio N_s/N_r. However, this depends the way of how the success is defined. For a function $f(x)$ with a known optimal solution x_* and the minimization objective $f_{\min}(x_*)$, the success can be defined by either $|x - x_*| \le \delta$ or $|f(x) - f_{\min}| \le \delta$ for a given small neighborhood such as $\delta = 10^{-5}$. This can be two very different criteria if the landscape is relative flat.

Some studies use one or more performance measures, but it is not clear if the above performance measures are truly fair measures for a fair comparison.

Open problem 4. What are the most suitable performance metrics for fairly comparing all algorithms? Is it possible to design a unified framework to compare all algorithms fairly and rigorously.

4.5. Algorithm scalability

From the application point of view, the most important indicator of the effectiveness of an algorithm is how efficiently it can solve a wide range of problems. Apart from the constraints posed by the no-free-lunch theorem, the efficiency of a given algorithm for a given type of problems can be largely affected by the size of problem instances. A well-known example is the travelling salesman problem (TSP) where a visitor is required to visit each city exactly once so as to minimize the overall distanced travelling through n cities. For a small number of cities (say, $n \leq 5$), it is an easy problem. For a moderate or large n, this problem becomes an NP-hard problem [84–86]. In this case, an algorithm that works well for small-scale problem instances cannot be scaled up to solve large-scale problems in a practically acceptable time scale.

Despite the diverse range of applications concerning nature-inspired algorithms and evolutionary algorithms, the problem sizes tend to be small or moderate, typically under several hundred parameters. It is not clear if these algorithms can be scaled up, by parallel computing, high-performance computing or cloud computing approaches.

Open problem 5. How to best scale up the algorithms that work well for small-scale problems to solve truly large-scale, real-world problems efficiently?

There are other open problems concerning nature-inspired algorithms, including how to achieve the optimal balance of exploitation and exploration, how to deal with nonlinear constraints effectively, and how to use these algorithms for machine learning and deep learning.

Nature-inspired computation is an active area of research. It is hoped that the above five open problems we have just highlighted can inspire more research in this area in the near future.

Conflict of interest

There is no conflict of interest.

References

[1] S.P. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, Cambridge, UK, 2004.
[2] X.-S. Yang, Nature-Inspired Optimization Algorithms, Elsevier Insight, London, 2014.
[3] J. Holland, Adaptation in Nature and Artificial Systems, University of Michigan Press, Ann Arbor, MI, USA, 1975.
[4] M. Dorigo, Optimization, learning, and natural algorithms, Ph.D. Thesis, Politecnico di Milano, Milan, Italy, 1992.
[5] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, IEEE, Piscataway, NJ, USA, 1995, pp. 1942–1948.
[6] X.-S. Yang, A new metaheuristic bat-inspired algorithm, in: C. Cruz, J.R. González, D.A. Pelta, G. Terrazas (Eds.), Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), volume 284 of Studies in Computational Intelligence, Springer, Berlin, Germany, 2010, pp. 65–74.
[7] X.-S. Yang, Firefly algorithms for multimodal optimization, in: O. Watanabe, T. Zeugmann (Eds.), Proceedings of Fifth Symposium on Stochastic Algorithms, Foundations and Applications, volume 5792, Lecture Notes in Computer Science, Springer, 2009, pp. 169–178.
[8] X.-S. Yang, S. Deb, Cuckoo search via Lévy flights, in: Proceedings of World Congress on Nature and Biologically Inspired Computing (NaBIC 2009), IEEE Publications, USA, 2009, pp. 210–214.
[9] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence, Academic Press, London, UK, 2001.
[10] A.P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, Hoboken, NJ, USA, 2005.
[11] X.-S. Yang, Z.H. Cui, X.R. B, A.H. Gandom, M. Karamanoglu, Swarm Intelligence and Bio-Inspired Computation: Theory and Applications, Elsevier, London, UK, 2013.
[12] X.-S. Yang, Cuckoo Search and Firefly Algorithm: Theory and Applications, volume 516 of Studies in Computational Intelligence, Springer, Heidelberg, Germany, 2013.
[13] M. Reyes-Sierra, A.C. Coello Coello, Multi-objective particle swarm optimizers: a survey of the state-of-the-art, Int. J. Comput. Intell. Res. 2 (2006) 287–308.
[14] K. Price, R. Storn, J. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, Berlin, Germany, 2005.
[15] I. Fister, I. Fister Jr., J. Brest, X.-S. Yang, A comprehensive review of firefly algorithms, Swarm Evol. Comput. 13 (2013) 34–46.
[16] X.-S. Yang, X.-S. He, Bat algorithm: literature review and applications, Int. J. Bio-Inspired Comput. 5 (2013) 141–149.
[17] Z.A.A. Alyasseri, A.T. Khader, M.A. Al-Betar, M.A. Awadallah, X.-S. Yang, Variants of the flower pollination algorithm: a review, in: X.-S. Yang (Ed.), Nature-Inspired Algorithms and Applied Optimization, Springer, Cham, 2018, pp. 91–118.
[18] M. Abdel-Basset, L.A. Shawky, Flower pollination algorithm: a comprehensive review, Artif. Intell. Rev. 52 (2019) 2533–2557.
[19] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, USA, 1989.
[20] R. Storn, K. Price, Differential evolution: a simple and efficient heuristic for global optimization, J. Global Optim. 11 (1997) 341–359.
[21] X.-S. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press, Bristol, UK, 2008.
[22] J.D. Altringham, Bats: Biology and Behaviour, Oxford University Press, Oxford, UK, 1996.
[23] T. Colin, The Variety of Life, Oxford University Press, Oxford, UK, 2000.
[24] N.B. Davies, Cuckoo adaptations: trickery and tuning, J. Zool. 284 (2011) 1–14.
[25] X.-S. Yang, S. Deb, Cuckoo search: recent advances and applications, Neural Comput. Appl. 24 (2014) 169–174.
[26] I. Pavlyukevich, Lévy flights, non-local search and simulated annealing, J. Comput. Phys. 226 (2007) 1830–1844.
[27] R.N. Mantegna, Fast, accurate algorithm for numerical simulation of Lévy stable stochastic process, Phys. Rev. E 49 (1994) 4677–4683.
[28] X.-S. Yang, Flower pollination algorithm for global optimization, in: J. Durand-Lose, N. Jonoska (Eds.), Unconventional Computation and Natural Computation (UCNC 2012), volume 7445, Springer, Berlin Heidelberg, Germany, 2012, pp. 240–249.
[29] X.-S. Yang, M. Karamanoglu, X.S. He, Multi-objective flower algorithm for optimization, Procedia Comput. Sci. 18 (2013) 861–868.
[30] N.M. Waser, Flower constancy: definition, cause and measurement, Am. Nat. 127 (1986) 596–603.
[31] S. Kirkpatrik, C.D. GEllat, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[32] K. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Syst. 22 (2002) 52–67.
[33] D. Simon, Biogeography-based optimization, IEEE Trans. Evol. Comput. 12 (2008) 702–713.
[34] E. Rashedi, H.H. Nezamabadi-Pour, S. Saryazdi, GSA: a gravitational search algorithm, Inf. Sci. 179 (2009) 2232–2248.
[35] A. Kaveh, S. Talatahari, A novel heuristic optimization method: charged system search, Acta Mech. 213 (2010) 267–289.
[36] A. Hatamlou, Black hole: a new heuristic optimization approach for data clustering, Inf. Sci. 222 (2012) 175–184.
[37] A. Gandomi, A. Alavi, Krill herd: a new bio-inspired optimization algorithm, Comput. Nonlinear Sci. Numer. Simul. 17 (2012) 4831–4845.
[38] X.-S. Yang, S. Deb, Two-stage eagle strategy with differential evolution, Int. J. Bio-Inspired Comput. 4 (2012) 1–5.
[39] Y. Khaluf, S. Vanhee, P. Simoens, Local ant system for allocating robot swarms to time-constrained tasks, J. Comput. Sci. 31 (2019) 33–44.
[40] R. Goel, R. Maini, A hybrid of ant colony and firefly algorithms (HAFA) for solving vehicle routing problems, J. Comput. Sci. 25 (2018) 28–37.
[41] D. Gupta, A. Ahlawat, Usability feature selection via MBBAT: a novel approach, J. Comput. Sci. 23 (2017) 195–203.
[42] E. Osaba, X.-S. Yang, F. Diaz, E. Onieva, A. Masegosa, A. Perallos, A discrete firefly algorithm to solve a rich vehicle routing problem modelling a newspaper distribution system with recycling policy, Soft Comput. 21 (2017) 5295–5308.
[43] E. Osaba, X.-S.I.F. Yang Jr., P. Lopez-Garcia, A. Vazquez-Paravila, A discrete and improved bat algorithm for solving a medical goods distribution problem with pharmacological waste collection, Swarm Evol. Comput. 44 (2019) 273–286.
[44] F.D. Rango, N. Palmieri, X.-S. Yang, S. Marano, Swarm robotics in wireless distributed protocol design for coordinating robots involved in cooperative tasks, Soft Comput. 22 (2018) 4251–4266.
[45] L.C. Cagnina, S.C. Esquivel, A.C. Coello Coello, Solving engineering optimization problems with the simple constrained particle swarm optimizer, Informatica 32 (2008) 319–326.
[46] K. Deb, A. Pratap, S. Agarwal, T. Mayarivan, A fast and elitist multiobjective algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2002) 182–197.
[47] X.-S. Yang, A.H. Gandomi, Bat algorithm: a novel approach for global engineering optimization, Eng. Comput. 29 (2012) 464–483.
[48] X.-S. Yang, S. Deb, Multiobjective cuckoo search for design optimization, Comput. Oper. Res. 40 (2013) 1616–1624.

[49] A.H. Gandom, X.-S. Yang, Chaotic bat algorithm, J. Comput. Sci. 5 (2014) 224–232.
[50] A. Chakri, R. Khelif, M. Benouaret, X.-S. Yang, New directional bat algorithm for continuous optimization problems, Expert Syst. Appl. 69 (2017) 159–175.
[51] J.P. Papa, G.H. Rosa, D.R. Pereira, X.-S. Yang, Quaternion-based deep belief networks fine-tuning, Appl. Soft Comput. 60 (2017) 328–335.
[52] N. Palmieri, X.-S. Yang, F.D. Rango, A.F. Santamaria, Self-adaptive decision-making mechanisms to balance the execution of multiple tasks for a multi-robots team, Neurocomputing 306 (2018) 17–36.
[53] A. Ouaarab, B. Ahiod, X.-S. Yang, Discrete cuckoo search algorithm for the travelling salesman problem, Neural Comput. Appl. 24 (2014) 1659–1669.
[54] E. Osaba, X.-S. Yang, F. Diaz, P. Lopez-Garcia, R. Carballedo, An improved discrete bat algorithm for symmetric and assymmetric travelling salesman problems, Eng. Appl. Artif. Intell. 48 (2016) 59–71.
[55] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Comput. Surv. 25 (2003) 268–308.
[56] J. Senthilnath, S.N. Omkar, V. Mani, Clustering using firefly algorithm: performance study, Swarm Evol. Comput. 1 (2011) 164–171.
[57] X.-S. Yang, X.-S. He, Mathematical Foundations of Nature-Inspired Algorithms, Springer Briefs in Optimization, Springer, Cham, Switzerland, 2019.
[58] C.M. Grindstead, J.L. Snell, Introduction to Probability, 2nd ed., American Mathematical Society, Providence, Rhode Island, 1997.
[59] U.N. Bhat, G.K. Miller, Elements of Applied Stochastic Processes, 3rd ed., John Wiley & Sons, New York, 2002.
[60] M. Gutowski, Lévy flights as an underlying mechanism for global optimization algorithms, ArXiv Math. Phys. E-Prints (2001) (accessed 1.9.2019).
[61] M. Clerc, J. Kennedy, The particle swarm: explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 6 (2002) 58–73.
[62] S. Chen, G.-H. Peng, Xing-Shi, X.-S. Yang, Global convergence analysis of the bat algorithm using a Markovian framework and dynamic system theory, Expert Syst. Appl. 114 (2018) 173–182.
[63] A. Granas, J. Dugundji, Fixed Point Theory, Springer-Verlag, New York, 2003.
[64] M.A. Khamsi, W.A. Kirk, An Introduction to Metric Space and Fixed Point Theory, John Wiley & Sons, New York, 2001.
[65] H. Khalil, Nonlinear Systems, 3rd ed., Prentice Hall, New Jersey, 1996.
[66] J. Suzuki, A Markov chain analysis on simple genetic algorithms, IEEE Trans. Syst. Man Cybern. 25 (1995) 655–659.
[67] H. Aytug, S. Bhattacharrya, G.J. Koehler, A Markov chain analysis of genetic algorithms with power of 2 cardinality alphabets, Eur. J. Oper. Res. 96 (1996) 195–201.
[68] D. Greenhalgh, S. Marshal, Convergence criteria for genetic algorithm, SIAM J. Comput. 30 (2000) 269–282.
[69] W.J. Gutjahr, Convergence analysis of metaheuristic, Anal. Inf. Syst. 10 (2010) 159–187.
[70] X.S. He, F. Wang, Y. Wang, X.S. Yang, Global convergence analysis of cuckoo search using Markov theory, in: X.-S. Yang (Ed.), Nature-Inspired Algorithms and Applied Optimization, 744, Springer Nature, Cham, Switzerland, 2018, pp. 53–67.
[71] A. Ghate, R. Smith, Adaptive search with stochastic acceptance probability for global optimization, Oper. Res. Lett. 36 (2008) 285–290.
[72] D.P. Bertsekas, A. Nedic, A. Ozdaglar, Convex Analysis and Optimization, 2nd ed., Athena Scientific, Belmont, MA, 2003.
[73] J.L. Chabert, A History of Algorithms: From the Pebble to the Microchips, Springer-Verlag, Heidelberg, 1999.
[74] D. Zdenek, Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities, Springer, Heidelberg, 2009.
[75] A.E. Eiben, S.K. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, Swarm Evol. Comput. 1 (2011) 19–31.
[76] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark functions, IEEE Trans. Evol. Comput. 10 (2006) 646–657.
[77] X.-S. Yang, S. Deb, M. Loomes, M. Karamanoglu, A framework for self-tuning optimization algorithm, Neural Comput. Appl. 23 (2013) 2051–2057.
[78] M. Jamil, X.-S. Yang, A literature survey of benchmark functions for global optimisation problems, Int. J. Math. Modell. Numer. Optim. 4 (2013) 150–194.
[79] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1997) 67–82.
[80] T. Joyce, J.M. Herrmann, A review of no free lunch theorems, and their implications for metaheuristic optimisation, in: X.-S. Yang (Ed.), Nature-Inspired Algorithms and Applied Optimization, Springer, Cham, Switzerland, 2018, pp. 27–52.
[81] A. Auger, O. Teytaud, Continuous lunches are free plus the design of optimal optimization algorithms, Algorithmica 57 (2010) 121–146.
[82] D. Corne, J. Knowles, Some multiobjective optimizers are better than others, Evol. Comput. 4 (2003) 2506–2512.
[83] D.H. Wolpert, W.G. Macready, Coevolutionary free lunches, IEEE Trans. Evol. Comput. 9 (2005) 721–735.
[84] S. Cook, An overview of computational complexity, Commun. ACM 26 (1983) 400–408.
[85] S. Arara, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, Cambridge, UK, 2009.
[86] O. Goldreich, Computational Complexity: A Conceptual Perspective, Cambridge University Press, Cambridge, UK, 2008.