A generalized evolutionary metaheuristic (GEM) algorithm for engineering optimization

Xin-She Yang

Published online: 01 Jul 2024.

Submit your article to this journal ⤢

View related articles ⤢

View Crossmark data ⤢

COMPUTER SCIENCE | RAPID COMMUNICATION

A generalized evolutionary metaheuristic (GEM) algorithm for engineering optimization

Xin-She Yang Ⓡ

School of Science and Technology, Middlesex University London, London, UK

ABSTRACT

Many optimization problems in engineering and industrial design applications can be formulated as optimization problems with highly nonlinear objectives, subject to multiple complex constraints. Solving such optimization problems requires sophisticated algorithms and optimization techniques. A major trend in recent years is the use of nature-inspired metaheustic algorithms (NIMA). Despite the popularity of nature-inspired metaheuristic algorithms, there are still some challenging issues and open problems to be resolved. Two main issues related to current NIMAs are: there are over 540 algorithms in the literature, and there is no unified framework to understand the search mechanisms of different algorithms. Therefore, this paper attempts to analyse some similarities and differences among different algorithms and then presents a generalized evolutionary metaheuristic (GEM) in an attempt to unify some of the existing algorithms. After a brief discussion of some insights into nature-inspired algorithms and some open problems, we propose a generalized evolutionary metaheuristic algorithm to unify more than 20 different algorithms so as to understand their main steps and search mechanisms. We then test the unified GEM using 15 test benchmarks to validate its performance. Finally, further research topics are briefly discussed.

1. Introduction

Many design problems in engineering and industry can be formulated as optimization problems subject to multiple nonlinear constraints. To solve such optimization problems, sophisticated optimization algorithms and techniques are often used. Traditional algorithms such as Newton-Raphson method are efficient, but they use derivatives and calculations of these derivatives, especially the second derivatives in a high-dimensional space, can be costly. In addition, such derivative-based algorithms are usually local search and the final solutions may depend on the starting point if optimization problems are highly nonlinear and multimodal (Boyd & Vandenberghe, 2004; Yang, 2020a). An alternative approach is to use derivative-free algorithms and many evolutionary algorithms, especially recent nature-inspired

algorithms, do not use derivatives (Kennedy & Eberhart, 1995; Pham et al., 2005; Storn & Price, 1997; Yang, 2020b). These nature-inspired metaheuristic algorithms are flexible and easy to implement, and yet they are usually very effective in solving various optimization problems in practice.

Algorithms have been important through history (Beer, 2016; Chabert, 1999; Schrijver, 2005). There are a vast spectrum of algorithms in the literature, ranging from fundamental algorithms to combinatorial optimization techniques (Chabert, 1999; Schrijver, 2005). In some special classes of optimization problems, effective algorithms exist for linear programming (Karmarkar, 1984) and quadratic programming (Zdenek, 2009) as well as convex optimization (Bertsekas et al., 2003; Boyd & Vandenberghe, 2004). However, for nonlinear optimization problems,

techniques vary and often approximations, heuristic algorithms and metaheuristic algorithms are needed. Even so, optimal solutions cannot always be obtained for nonlinear optimization.

Metaheuristic algorithms are approximation optimization techniques, and they use some form of heuristics with trial and error and some form of memory and solution selections (Glover, 1986; Glover & Laguna, 1997). Most metaheuristic algorithms are evolution-based and/or nature-inspired. Evolution-based algorithms such as genetic algorithm (Holland, 1975; Goldberg, 1989) are often called evolutionary algorithms. Algorithms such as particle swarm optimization (PSO) (Kennedy & Eberhart, 1995), bees algorithm (Pham et al., 2005; Pham & Castellani, 2009) and firefly algorithm (Yang, 2009) are often called swarm intelligence based algorithms (Kennedy et al., 2001).

However, terminologies in this area are not well defined and different researchers may use different terminologies to refer to the same things. In this paper, we use nature-inspired algorithms to mean all the metaheuristic algorithms that are inspired by some forms of evolutionary characteristics in nature, being biological, behaviour, social, physical and chemical characteristics (Yang, 2020a; Yang & He, 2019). In this broad sense, almost all algorithms can be called nature-inspired algorithms, including bees algorithms (Pham & Castellani, 2014, 2015), PSO (Kennedy et al., 2001), ant colony optimization, bat algorithm, flower pollination algorithm, cuckoo search algorithm, genetic algorithm, and many others.

Nature-inspired algorithms have become popular in recent years, and it is estimated that there are several hundred algorithms and variants in the current literature (Yang, 2020a), and the relevant literature is expanding with more algorithms emerging every year. An exhaustive review of metaheuristic algorithms by Rajwar et al. (2023) indicated that there are over 540 metaheuristic algorithms with over 350 of such algorithms that were developed in the last 10 years. Many such new variants have been developed based on different characteristics/species from nature, social interactions and/or artificial systems, or based on the hybridization of different algorithms or algorithmic components, or based on different strategies of selecting candidate solutions and information sharing characteristics (Mohamed et al., 2020; Rajwar et al., 2023; Zelinka, 2015).

From the application perspective, nature-inspired algorithms have been shown that they can solve a wide range of optimization problems (Abdel-Basset & Shawky, 2019; Bekaşş et al., 2018; Osaba et al., 2016; Pham & Castellani, 2015), from continuous optimization (Pham & Castellani, 2014) and engineering design optimization problems (Bekaşş et al., 2018) to combinatorial optimization problems (Ouaarab et al., 2014; Osaba et al., 2016, 2017), multi-robots systems (Palmieri et al., 2018; Rango et al., 2018) and many other applications (Gavvala et al., 2019; Mohamed et al., 2020; Rajwar et al., 2023; Zelinka, 2015).

Despite the wide applications of nature-inspired algorithms, theoretical analysis in contrast lacks behind. Though there are some rigorous analyses concerning genetic algorithm (Greenhalgh & Marshal, 2000), PSO (Clerc & Kennedy, 2002) and the bat algorithm (Chen et al., 2018), however, many new algorithms have not been analyzed in detail. Ideally, a systematical analysis and review should be carried out in the similar way to convex analysis (Bertsekas et al., 2003) and convex optimization (Boyd & Vandenberghe, 2004). In addition, since there are so many different algorithms, it is difficult to figure out what search mechanisms can be effective in determining the performance of a specific algorithm. Furthermore, some of these 540 algorithms can look very similar, either in terms of their search mechanisms or updating equations, though they may look very different on the surface. This often can cause confusion and frustration to readers and researchers to see what happens in this research community. In fact, there are many open problems and unresolved issues concerning nature-inspired metaheuristic algorithms (Rajwar et al., 2023; Yang et al., 2018; Yang & He, 2019; Yang, 2020b).

Therefore, the purpose of this paper is two-fold: outlining some of the challenging issues and open problems, and then developing a generalized evolutionary metaheuristic (GEM) to unify many existing algorithms. The rest of the paper is organized as follows. Section 2 first provides some insights into nature-inspired computing and then outlines some of the open problems concerning nature-inspired algorithms. Section 3 presents a unified framework of more than 20 different algorithms so as to look all the relevant algorithms in the same set of mathematical equations. Section 4 discusses 15 benchmark functions and case studies, whereas Section 5 carries out some numerical experiments to test and validate the generalized algorithm. Finally, Section 6 concludes with a brief discussion of future research topics.

2. Insights and open problems

Though not much systematical analysis of nature-inspired algorithms exist, various studies from different perspectives have been carried out and different degrees of insights have been gained (Eiben & Smit, 2011; Ser et al., 2019; Yang, 2020b). For any given nature-inspired algorithm, we can analyze its algorithmic components, and their role in exploration and exploitation, and we also study its search mechanism so as to understand ways that local search and global search moves are carried out. Many studies in the literature have provided numerical convergence curves during iterations when solving different function optimization and sometimes real-world case studies, and such convergence curves are often presented with various statistical quantities according to a specific set of performance metrics such as accuracy of the solution and successful rate as well as number of iterations. In addition, stability and robustness are also studies for some algorithms (Clerc & Kennedy, 2002). Such analyses, though very important, are largely qualitative studies of algorithmic features, as summarized in Figure 1.

The analysis of algorithms can be carried out more rigorously from a quantitative perspective, as shown in Figure 2. For a given algorithm, it is possible to analyze the iterative behaviour of the algorithm using fixed-point theory. However, the assumptions required for such theories may not be realistic or relevant to the actual algorithm under consideration. Thus, it is not always possible to carry out such analysis. One good way is to use complexity theory to analyze an algorithm to see its time complexity. Interestingly, most nature-inspired algorithms have the complexity of $O(nt)$ where n is the typically population size and t is the number of iterations. It is still a mystery that how such low complexity algorithms can solve highly complex nonlinear optimization problems that have been shown in various applications.

From the dynamical system point of view, an algorithm is a system of updating equations, which can be formulated as a discrete dynamical system. The eigenvalues of the main matrix of such a system determine the main characteristics of the algorithm. It can be expected that these eigenvalues can depend on the parameter values of the algorithm, and thus parameter settings can be important. In fact, the analyses on PSO (Clerc & Kennedy, 2002) and the bat algorithm (Chen et al., 2018) show that
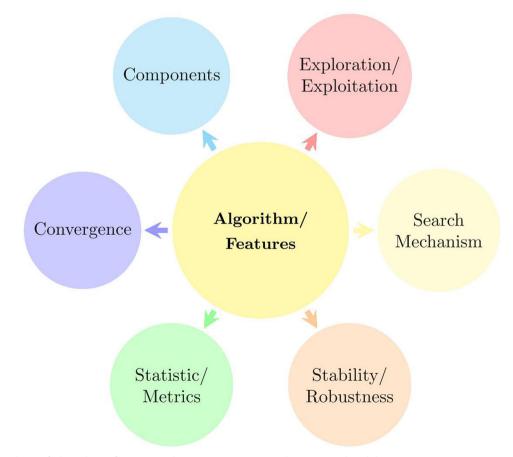

Figure 1. Analysis of algorithmic features such as components, mechanisms and stability.
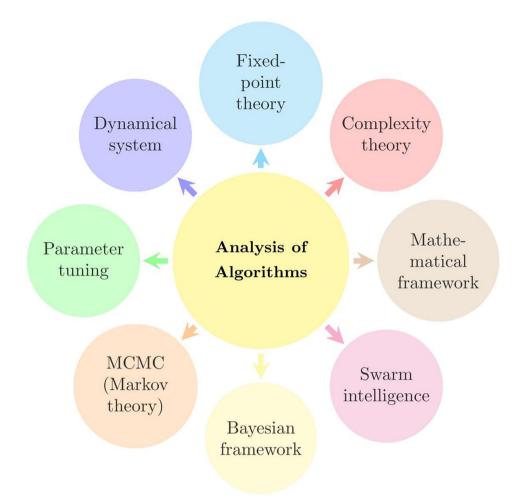
Figure 2. Different perspectives for quantitative analysis of algorithms.

parameter values are important. If the parameter values are in the wrong ranges, the algorithm may become unstable and become less effective. This also indicates the important of parameter tuning in nature-inspired algorithms (Eiben & Smit, 2011; Joy et al., 2023). However, parameter tuning itself is a challenging task because its aim is to find the optimal parameter setting for an optimization algorithm for a given set of optimization problems.

From the probability point of view, an algorithm can be considered as a set of interacting Markov chains, thus it is possible to do some approximation analysis in terms of convergence using Markov chain Monte Carlo (MCMC) theory (Chen et al., 2018). However, the conditions required for MCMC can be stringent, and thus not all algorithms can be analyzed in this way. From the perspective of the analysis of variance, it is possible to see how the variances may change with iteration to gain some useful understanding (Zaharie, 2009).

An alternative approach is to use Bayesian statistical framework to gain some insights into the algorithm under analysis. Loosely speaking, the initialization of the population in an algorithm with a given probability distribution forms the prior of the algorithm. When the algorithm evolves and the solutions also evolve, leading to a posterior distribution of solutions and parameters. Thus, the evolution of algorithm can be understood from this perspective. However, since Bayesian framework often requires a lot of extensive integral evaluations, it is not straightforward to gain rigorous results in general.

A more ambition approach is to build a mathematical framework so as to analyze algorithms in a unified way, though such a framework does not exist in the current literature. Ideally, a theoretical framework should provide enough insights into the rise of swarm intelligence, which is still an open problem (Yang, 2020b; Yang & He, 2019).

As we have seen, algorithms can potentially be analyzed from different perspectives and there are many issues that need further research in the general area of swarm intelligence and nature-inspired computation. We can highlight a few important open problems.

1. *Theoretical framework*. Though there are some good theoretical analyses of a few algorithms such as genetic algorithm (Greenhalgh & Marshal, 2000), PSO (Clerc & Kennedy, 2002) and the bat algorithm (Chen et al., 2018), there is no unified theoretical framework that can be used to analyze all algorithms or at least a major subset of nature-inspired algorithms. There is a strong need to build a mathematical framework so that the convergence and rate of convergence of any algorithm can be analyzed with rigorous and quantitative results.

   In addition, stability of an algorithm and its robustness should also be analyzed using the same mathematical framework, based on theories such as dynamical systems and perturbation as well as probability. The insights gained from such a theoretical framework should provide enough guidance for tuning and setting parameter values for a given algorithm. However, how to construct this theoretical framework is still an open problem. It may be that a multidisciplinary approach is needed to ensure to look at algorithms from different perspectives.

2. *Parameter tuning*. The setting of parameters in an algorithm can influence the performance of an algorithm, though the extent of such influence may largely depend on the algorithm itself and potentially on the problem to be solved (Joy et al., 2023). There are different methods for parameter tuning, but it is not clear which method(s) should be used for a given algorithm. In addition, different tuning methods may produce different results for parameter settings for different problems, which may leads to the question if a truly optimal parameter setting exists. It seems that there are different optimality conditions concerning parameter setting (Joy et al., 2023; Yang et al., 2013), and parameter settings may be both algorithm-dependent and problem-dependent, depending on the performance metric used for tuning. Many of these questions remain unresolved.

3. *Benchmarking*. All new algorithms should be tested and validated using a diverse of set of benchmarks and case studies. In the current literature, one of the main issues is that most tests use smooth functions as benchmarks, and it seems that these functions have nothing to do with real-world applications. Thus, it is not clear how such tests can actually validate the algorithm to gain any insight into the potential performance of the algorithm to solve much more complicated real-world applications. There is a strong need to systematically investigate the role of benchmarking and what types of benchmarks and case studies should be used.

4. *Performance metrics*. It can be expected that the performance of an algorithm depends on the performance metrics used to measure the performance. In the current literature, performance measures are mostly accuracy compared to the true objective values, success rate of multiple functions, number of iterations as computational efforts, computational time, and the combination of these measures. Whether these measures are fair or sufficient is still unexplained. In addition, these performance metrics tend to lead to the ranking of algorithms used and the benchmark functions used. Again, this may not be consistent with the no-free-lunch theorems (Wolpert & Macready, 1997). It is not clear if other performance measures should be designed and used, and what theory should be based on to design performance metrics. All these are still open questions.

5. *Search mechanism*. In many nature-inspired metaheuristic algorithms, certain forms of randomization and probability distributions are used to generate solutions with exploration abilities. One of the main tasks is to balance exploration and exploitation or diversification and intensification using different search moves or search mechanisms. However, how to balance these two components is still an open problem. In addition, exploration is usually by randomness, random walks and perturbations, whereas exploitation is usually by using derivative information and memory. It is not clear what search moves can be used to achieve both exploration and exploitation effectively.

6. *Scalability*. Most studies of metaheuristic algorithms in the literature are concerned with problems with a few parameters or a few dozen parameters. These problems, though very complicated and useful, are small-scale problems. It is not clear if the algorithms used and the implementation realized can directly be applied to large-scale problems in practice. Simple scale up by using high-performance computing or cloud computing facilities may not be enough. How to scale up to solve real-world, large-scale problems is still a challenging issue. In fact, more efficient algorithms are always desirable in this context.

7. *Rise of Swarm Intelligence.* Various discussions about swarm intelligence have attracted attention in the current literature. It is not clear what swarm intelligence exactly means and what conditions are necessary to achieve such collective intelligence. There is a strong need to understand swarm intelligence theoretically and practically so as to gain insights into the rise of swarm intelligence. With newly gained insights, we may be able to design better and more effective algorithms.

In addition, from both theoretical perspective and practical point of view, the no-free lunch theorems (Wolpert & Macready, 1997) had some significant impact on the understanding of algorithm behaviour. Studies also indicate that free lunches may exist for co-evolution (Wolpert & Macready, 2005), continuous problems (Auger & Teytaud, 2010) and multi-objective optimization (Corne & Knowles, 2003; Zitzler et al., 2003) under certain conditions. The main question is how to use such possibilities to build more effective algorithms.

3. A generalized evolutionary metaheuristic (GEM) for optimization

From the numerical algorithm analysis point of view (Boyd & Vandenberghe, 2004; Yang, 2020a), an algorithm in essence is a procedure to modify the current solution x^t so as to produce a potentially better solution x^{t+1}. The well-known Newton's method at iteration t can be written as

$$x^{t+1} = x^t - \frac{\nabla f(x^t)}{\nabla^2 f(x^t)}, \qquad (1)$$

where ∇f is the gradient vector of the objective function at x^t and $\nabla^2 f$ is the Hessian matrix. Loosely speaking, all iterative algorithms can schematically be written as

$$x^{t+1} = x^t + S(x^t, x^*, p_1, ..., p_k), \qquad (2)$$

where S is a step size vector, which can in general depend on the current solution x^t, the current best solution x^*, and a set of parameters $(p_1, ..., p_k)$. Different algorithms may differ only in the ways of generating such steps.

In addition to the open problems highlighted above, the other purpose of this paper is to provide a unified algorithm framework, based on more than 20 existing nature-inspired algorithms. This unification may enable us to understand the links and difference among different algorithms. It also allows us

to build a single generic algorithm that can potentially use the advantages of all its component algorithms, leading to an effective algorithm.

As we have mentioned earlier, there are approximately 540 metaheuristic algorithms and variants in the literature (Rajwar et al., 2023), it is impossible to test all algorithms and try to provide a unified approach. This paper is the first attempt to unify multiple algorithms in the same generalized evolutionary perspective. Thus, we have to select over twenty algorithms to see if we can achieve our aim. Obviously, there are two challenging issues: how many algorithms we should use and which algorithms we should choose. For the former question, we think it makes sense that we should choose at least ten different algorithms or more so that we can get a reasonable picture of the unification capabilities. In fact, we have chosen 22 algorithms for this framework. As for the latter question which algorithms to use, it is almost impossible to decide what algorithms to use from 540 algorithms. In the end, the algorithms we have chosen here have different search features or characteristics in terms of their exploration and exploitation capabilities. In addition, in the case of similar exploration and exploitation capabilities, we tend to select the slightly well-established algorithms that appeared earlier in the literature because later/new algorithms may have been based on such mechanisms.

This unified algorithm is called Generalized Evolutionary Metaheuristic (GEM) with multiple parameters and components to unify more than 20 algorithms.

A solution vector x in the D-dimensional space is denoted by

$$x = (x_1, x_2, ..., x_D). \qquad (3)$$

For a set of n solution vectors x_i where $i = 1, 2, ..., n$, these vectors evolve with the iteration t (the pseudo-time) and they are denoted by x_i^t. Among these n solutions, there is one solution g_* that gives the best objective value (i.e. highest for maximization or lowest for minimization). For minimization,

$$g_* = \text{argmin}\{f(x_i^t), f(x_i^*), f(\bar{x}_g)\},$$
$$(\text{for minimization}), \qquad (4)$$

where the argmin is to find the corresponding solution vector with the best objective value. Here, $\bar{x}_g$ is the average of the top m best solutions among n solutions ($m \leq n$). That is

$$\bar{\boldsymbol{x}}_g = \frac{1}{m} \sum_{j=1}^{m} x_j. \tag{5}$$

In case of $m = n$, this becomes the centre of the gravity of the whole population. Thus, we can refer to this step as the *centrality move*.

The main updating equations for our proposed unified algorithm are guided randomization and position update. In the *guided randomization search* step, it consists of

$$\boldsymbol{v}_i^{t+1} = \underbrace{p\boldsymbol{v}_i^t}_{\text{inertia term}} + \underbrace{q\epsilon_1(\boldsymbol{g}_* - \boldsymbol{x}_i^t)}_{\text{motion towards current best}} + \underbrace{r\epsilon_2(\boldsymbol{x}_i^* - \boldsymbol{x}_i^t)}_{\text{motion to individual best}}. \tag{6}$$

In the *position update* step, it means

$$\boldsymbol{x}_{i,new}^{t+1} = a\boldsymbol{x}_i^t + \underbrace{(1-a)\bar{\boldsymbol{x}}_g}_{\text{Centrality}} + \underbrace{b(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t)}_{\text{similarity convergence}} + \underbrace{c\boldsymbol{v}_i^{t+1}}_{\text{kinetic motion}} + \underbrace{\Theta h(\boldsymbol{x}_i^t)\zeta_i}_{\text{perturbation}}, \tag{7}$$

where $a, b, c, \Theta, p, q,$ and r are parameters, and $\boldsymbol{x}_i^*$ is the individual best solution for agent or particle i. Here, $h(\boldsymbol{x}_i^t)$ is a function of current solution, and in most case we can set $h(\boldsymbol{x}_i^t) = 1$, which can be a constant. In addition, ζ_i is a vector of random numbers, typically drawn from a standard normal distribution. That is

$$\zeta_i \sim \mathcal{N}(0, 1). \tag{8}$$

It is worth pointing out the effect of each term can be different, and thus we can name each term, based on their effect and meaning. The inertia term is $p\boldsymbol{v}_i^t$, whereas the $q\epsilon_1(\boldsymbol{g}_* - \boldsymbol{x}_i^t)$ simulates the motion or move towards the current best solution. The term $r\epsilon_2(\boldsymbol{x}_i^* - \boldsymbol{x}_i^t)$ means to move towards the individual best solution. The centrality term has a weighting coefficient $(1 - a)$, which tries to balance between the importance of the current solution $\boldsymbol{x}_i^t$ and the importance of the centre of gravity of the swarm $\bar{\boldsymbol{x}}_g$. The main term $b(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t)$ shows similar solutions should converge with subtle or minor changes. $c\boldsymbol{v}_i^t$ is the kinetic motion of each solution vector. The perturbation term $\Theta h(\boldsymbol{x}_i^t)\zeta_i$ is controlled by the strength parameter Θ where $h(\boldsymbol{x}_i)$ can be used to simulate certain specialized moves for each solution agent. If there is no specialization needed, $h(\boldsymbol{x}_i) = 1$ can be used.

The *selection and acceptance criterion* for minimization is

$$\boldsymbol{x}_i^{t+1} = \begin{cases} \boldsymbol{x}_{i,new}^t & \text{if } f(\boldsymbol{x}_{i,new}^{t+1}) \leq f(\boldsymbol{x}_i^t), \\ \boldsymbol{x}_i^t & \text{otherwise}. \end{cases} \tag{9}$$

These four main steps/equations are summarized in the pseudocode, shown in Algorithm 1 where the initialization of the population follows the standard Monte Carlo randomization

$$\boldsymbol{x}_i = Lb + rand(1, D)(Ub - Lb), \tag{10}$$

where Lb and Ub are, respectively, the lower bound and upper bound of the feasible decision variables. In addition, rand(1,D) is a vector of D random numbers drawn from a uniform distribution.

Special cases can correspond to more than 20 different algorithms. It is worth pointing out that in many case there are more than one way to represent the same algorithm by setting different values of parameters. The following representations are one of the possible ways for representing these algorithms:

1. Differential evolution (DE) (Price et al., 2005; Storn & Price, 1997): $a = 1$, $b = F$, $c = 0$ and $\Theta = 0$.
2. Particle swarm optimization (PSO) (Kennedy & Eberhart, 1995; Kennedy et al., 2001): $a = 1$, $b = 0$, $p = 1$, $c = 1$, and $\Theta = 0$. In addition, $q = \alpha$ and $r = \beta$.
3. Firefly algorithm (FA) (Yang, 2009, 2013): $a = 1$, $b = \beta \exp(-\gamma r_{ij}^2)$, $c = 0$ and $\Theta = 0.97^t$.
4. Simulated annealing (SA) (Kirkpatrick et al., 1983): $a = 1$, $b = c = 0$ and $\Theta = 1$.
5. Artificial bee algorithm (ABC) (Karaboga & Basturk, 2008): $a = 1$, $b = \Phi \in [-1, 1]$, $c = 0$ and $\Theta = 0$.
6. Artificial cooperative search (ACS) (Civicioglu, 2013): $a = 1$, $b = R$. $c = 0$, $\Theta = 0$ and its two predator keys are drawn from its α and β.
7. Charged system search (CSS) (Kaveh & Talatahari, 2010): $a = 1$, $b = A(R)$ where R is the normalized distance with the maximum at $R = 1$. In addition, $c = 0$ and $\Theta = 0$.
8. Cuckoo search (CS) (Yang & Deb, 2010): $a = 1$, $c = 0$, and $b = \alpha s H(p_a - \epsilon)$ where α is a scaling parameter, H is a Heaviside function with a switch probability p_a and a uniformly distributed random number ϵ. The step size s is drawn from a Lévy distribution. The other branch with the switch probability p_a is $a = 1$, $b = c = 0$, $\Theta = 1$, but ζ is drawn from the Lévy distribution

$$L(s, \beta) \sim \frac{\beta \Gamma(\beta) \sin(\pi\beta/2)}{\pi} \frac{1}{s^{1+\beta}}, \quad \beta = 1.5,$$

   where Γ is the standard Gamma function.
9. Gravitational search algorithm (GSA) (Rashedi et al., 2009): $a = 1$, $p = rand \in [0, 1]$, $q = r = 0$, $c = 1$ and $\Theta = 0$. In addition, $b = rand * G(t)$ where $G(t) = G_0 \exp(-\alpha t/T)$.

10. Gradient evolution algorithm (GEA) (Kuo & Zulvia, 2015): $a = 1$, $c = 0$ and $\Theta = 0$, but

$$b = r_g \Delta x_{ij}^t / 2(x_{ij}^w - x_{ij}^t + x_{ij}^B).$$

In addition, the parameter is set to $r_a = 0$ in GEA.

11. Harris hawk optimizer (HHO) (Heidari et al., 2019): $a = 1$, $b = 0$, $c = 1$, $\Theta = 0$, $p = 0$, $r = 0$, and $q = -E$ where E varies with iteration t.

12. Henry gas solubility optimization (HGSO) (Hashim et al., 2019): $a = 1$, $b = r$, $c = 1$ and $\Theta = 0$. In addition, $p = 1$ and $r = 0$, but q is related to the solubility affected by the Henry constant and parochial pressure. Here, the Henry's constant changes with iteration t by $H_j^{t+1} = H_j^t \exp[-C_j(1/T(t) - 1/T_0)]$ where $T_0 = 298.15$ K, C_j is a given constant and $T(t) = e^{-t/t_{max}}$.

13. Harmony search (HS) (Geem et al., 2001): $a = 0$, $b = c = 0$, $\Theta = 1$, $h(\boldsymbol{x}) = \boldsymbol{x}_i^t$ for pitch adjustment, but for harmony selection $b = 1$ and $\Theta = 0$ can be used.

14. Ant lion optimizer (ALO) (Mirjalili, 2015): $a = 1$, $b = 0$, $c = 1$, $\Theta = d_i - c_i$ where d_i is the scaled random walk length and c_i is its corresponding minimum. In addition, $p = 0$, $q = 0$, $r = 1$ with some variation of $\boldsymbol{g}_*$ is just the average of two selected elite ant-lion solutions.

15. Whale optimization algorithm (WOA) (Mirjalili & Lewis, 2016): $a = 1$, $c = 0$, $p = q = r = 0$, $b = -A$ where $A = 2dr - a$ with r be drawn randomly from $[0,1]$ and d be linearly decreasing from 2 to 0. In their spiral branch, $\Theta = 1$ with an equivalent $b = e^R \cos(2\pi R)$ where R is uniformly distributed in $[[-1,1]$.

16. Lion optimization algorithm (LOA) (Yazdani & Jolai, 2016): $a = 1$, $b = 0$, $c = 1$, $\Theta = 0$, $p = 0$, $q = -PI$, and $r = 0$. In the other moving branch in their LOA, Θ is proportional to DR where D is their scaled distance and R can be drawn either from [0, 1] or [−1, 1].

17. Mayfly optimization algorithm (MOA) (Zervoudakis & Tsafarakis, 2020): $a = 1$, $b = 0$, $c = 1$, $\Theta = 0$, $p = g$, $q = a_1 \exp(-\beta r_p^2)$, and $r = a_2 \exp(-\beta r_g^2)$.

18. Big bang-big crunch (BBBC) (Erol & Eksin, 2006): the modification of solutions is mainly around the centre of gravity $\boldsymbol{x}_g$ of the population with $a = 0$. $b = c = 0$, and $\Theta = Ub/t_{max}$.

19. Social spider algorithm (SSA) (James & Li, 2015): $a = 1$, $b = we^{d_{ij}^2}$ where w is a weight coefficient and d_{ij} is the distance between two spiders. In addition, $c = 0$, $\Theta = 1$, $\zeta = rand - 1/2$.

20. Moth search algorithm (MSA) (Wang, 2018): $a = \lambda$, $b = 0$, $c = 1$, $p = 0$, $q = \phi$, $r = 0$, and $\Theta = 0$ in one equation. The other equation corresponds to $a = 1$, $b = c = 0$, $\Theta = S_{max}/t^2$ and $\zeta = L(s)$ drawn from a Lévy distribution.

21. Multi-verse optimizer (MVO) (Mirjalili et al., 2016): $a = 1$, $b = c = 0$, but $\Theta = 1 - (t/t_{max})^{1/p}$ where t_{max} is the maximum number of iterations. Its randomized perturbation is given by $\zeta = \pm[Lb + rand(Ub - Lb)]$.

22. Water cycle algorithm (WCA) (Eskandar et al., 2012): $a = 1$, $c = \Theta = 0$, $b = C$ $rand$ where $C \in [1, 2]$ and $rand$ is a uniformly distributed random number in [0, 1] for both water drops in river and stream in the WCA. For the additional search for the new stream step, $a = 1$, $b = c = 0$, but $\Theta = \sqrt{\mu}$ as its standard deviation and ζ is drawn from a Gaussian distribution with a unity mean.

Algorithm 1: GEM: A generalized evolutionary meta-heuristic for optimization.

Data: Define optimization objective ($f(\boldsymbol{x})$) with proper constraint handling
Result: Best or optimal solution (f_{min}) with archived solution history

1 Initialize parameter values (n, a, b, c, Θ, p, q, r);
2 Generate an initial population of n solutions using Eq. (10);
3 Find the initial best solution $\boldsymbol{g}_*$ and f_{min} among the initial population;
4 **while** ($t < MaxGeneration$) **do**
5   Generate random numbers ($\epsilon_1, \epsilon_2, \zeta_i$);
6   **for** $i = 1 : n$ (all solutions) **do**
7   Modify solutions by Eq. (6);
8   Update solution vectors by Eq. (7);
9   Accept a solution by checking criterion Eq. (9);
10   **end**
11   Rank all the solutions to find the current best f_{min};
12   Update the best solution so far $\boldsymbol{g}_*$ and $\bar{\boldsymbol{x}}_g$ by Eq. (4);
13   Save and archive the current population for next generation;
14   Update the iteration counter $t \leftarrow t + 1$;
15 **end**
16 Post-process the solutions for constraint verification and visualization;

As pointed out earlier, there are more than one ways of representing an algorithm under consideration

using the unified framework, and the minor details and un-important components of some algorithms may be ignored. In essence, the unified framework intends to extract the key components of multiple algorithms so that we can figure out what main search mechanisms or moves are important in metaheuristic algorithms. Therefore, it can be expected that many other algorithms may be considered as special cases of the GEM framework if the right combinations of parameter values are used.

4. Benchmarks and parameter settings

To test the unified GEM algorithm, we have selected 15 benchmark functions and case studies. There are many different test function benchmarks, including multivariate functions (Jamil & Yang, 2013), CEC2005 test suite (Suganthan et al., 2005), unconstrained benchmark function repository (Al-Roomi, 2015) and engineering optimization problems (Cagnina et al., 2008; Coello, 2000). The intention is to select a subset of optimization problems with a diverse range of properties such as modality, convexity, nonlinear constraints, separability, and landscape variations. The case studies also include a mixed-integer programming pressure vessel design problem, and a parameter estimation based on data for a vibrations system governed by an ordinary differential equation (ODE).

4.1. Test functions

The ten test functions are outlined as follows.

1. Sphere function

$$f_1(\mathbf{x}) = \sum_{i=1}^{D} x_i^2, \quad -10 \le x_i \le +10, \qquad (11)$$

its global optimality $f_{\min} = 0$ occurs at $\mathbf{x}_* = (0, 0, ..., 0)$.

2. Rosenbrock function (Rosenbrock, 1960)

$$f_2(\mathbf{x}) = (x_1 - 1)^2 + 100 \sum_{i=1}^{D-1} (x_{i+1} - x_i^2)^2,$$

$$x_i \in [-10, 10],$$

$$(12)$$

whose global minimum $f_{\min} = 0$ is located at $\mathbf{x}_* = (1, 1, ..., 1)$.

3. Ackley function

$$f_3(\mathbf{x}) = -20 \exp\left[-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right]$$

$$- \exp\left[\frac{1}{D}\sum_{i=1}^{D} \cos(2\pi x_i)\right] + 20 + e,$$

$$(13)$$

with

$$x_i \in [-32.768, 32.768], \qquad (14)$$

whose global minimum $f_{\min} = 0$ occurs at $\mathbf{x}_* = (0, 0, ..., 0)$.

4. Dixon-Price function

$$f_4(\mathbf{x}) = (x_1 - 1)^2 + \sum_{i=2}^{D} i(2x_i^2 - x_{i-1})^2, \qquad (15)$$

$$x_i \in [-10, 10],$$

whose global minimum $f_{\min} = 0$ at $x_i = 2^{-(2^i-2)/2^i}$ for $i = 1, 2, ..., D$.

5. Schwefel function (Schwefel, 1995)

$$f_5(\mathbf{x}) = -x_1 x_2 (72 - 2x_1 - 2x_2), \quad x_i \in [0, 500], \qquad (16)$$

its global minimum $f_{\min} = -3456$ occurs at $\mathbf{x}_* = (12, 12)$.

6. Booth function

$$f_6(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2,$$

$$x_i \in [-10, 10],$$

$$(17)$$

whose global minimum $f_{\min} = 0$ occurs at $\mathbf{x}_* = (1, 3)$.

7. Holder table function

$$f_7(\mathbf{x}) = -\left|\sin(x_1)\cos(x_2)e^{\left|1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right|}\right|, \qquad (18)$$

$$x_i \in [-10, 10],$$

whose four global minima $f_{\min} = -19.2085$ occur at $\mathbf{x}_* = (\pm 8.05502, \pm 9.66459)$.

8. Beale function

$$f_8(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2$$

$$+ (2.625 - x_1 + x_1 x_2^3)^2,$$

$$(19)$$

where

$$x_i \in [-4.5, +4.5]. \qquad (20)$$

The global minimum $f_{\min} = 0$ occurs at $\mathbf{x}_* = (3, 0.5)$.

9. Trid function

$$f_9(\boldsymbol{x}) = \sum_{i=1}^{D} (x_i - 1)^2 - \sum_{i=2}^{D} x_i x_{i-1}, \tag{21}$$

$$x_i \in \left[-d^2, d^2\right],$$

whose global minimum $f_{min} = -D(D+4)(D-1)/6$ occurs at $x_i = i(D+1-i)$ for $i = 1, 2, ..., D$.

10. Rastrigin function

$$f_{10}(\boldsymbol{x}) = 10D + \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i)\right], \tag{22}$$

$$x_i \in [-5.12, 5.12],$$

whose global minimum $f_{min} = 0$ occurs at $\boldsymbol{x}_* = (0, 0, ..., 0)$.

4.2. Case studies

The five design case studies or benchmarks are described as follows.

11. Spring design. The three design variables are: the wire diameter w (or x_1), mean coil diameter D (or x_2) and the number N (or x_3) of active coils.

$$\min f(\boldsymbol{x}) = (2 + x_3) x_1^2 x_2, \tag{23}$$

subject to

$$g_1(\boldsymbol{x}) = 1 - \frac{x_2^3 x_3}{71785 x_1^4} \le 0, \tag{24}$$

$$g_2(\boldsymbol{x}) = \frac{4x_2^2 - x_1 x_2}{12566(x_2 x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} - 1 \le 0, \tag{25}$$

$$g_3(\boldsymbol{x}) = 1 - \frac{140.45 x_1}{x_2^2 x_3} \le 0, \tag{26}$$

$$g_4(\boldsymbol{x}) = \frac{x_1 + x_2}{1.5} - 1 \le 0. \tag{27}$$

The simple lower and upper bounds are

$$\begin{aligned}
Lb &= [0.05, \quad 0.25, \quad 2], \\
Ub &= [1.00, \quad 1.30, \quad 15].
\end{aligned} \tag{28}$$

The best solution found so far is (Cagnina et al., 2008; Yang, 2013)

$$\begin{aligned}
f_{min} &= 0.01266522, \\
\boldsymbol{x}_* &= [0.05169, \quad 0.35673, \quad 11.2885].
\end{aligned} \tag{29}$$

12. For the three-bar truss system design with two cross-section areas $x_1 = A_1$ and $x_2 = A_2$, the objective is to minimize

$$\min f(\boldsymbol{x}) = 100\left(2\sqrt{2}x_1 + x_2\right), \tag{30}$$

subject to

$$g_1(\boldsymbol{x}) = \frac{(\sqrt{2}x_1 + x_2)P}{\sqrt{2}x_1^2 + 2x_1 x_2} - \sigma \le 0, \tag{31}$$

$$g_2(\boldsymbol{x}) = \frac{x_2 P}{\sqrt{2}x_1^2 + 2x_1 x_2} - \sigma \le 0, \tag{32}$$

$$g_3(\boldsymbol{x}) = \frac{P}{x_1 + \sqrt{2}x_2} - \sigma \le 0. \tag{33}$$

where $\sigma = 2\,kN/cm^2$ is the stress limit and $P = 2\,kN$ is the load. In addition, $x_1, x_2 \in [0, 1]$. The best solution so far in the literature is (Bekasş et al., 2018)

$$f_{min} = 263.8958, \quad \boldsymbol{x}_* = (0.78853, 0.40866). \tag{34}$$

13. Beam design. For the beam design to support a vertical load at the free end of the beam, the objective is to minimize

$$\min f(\boldsymbol{x}) = 0.0624(x_1 + x_2 + x_3 + x_4 + x_5), \tag{35}$$

subject to

$$g(\boldsymbol{x}) = \frac{61}{x_1^3} + \frac{37}{x_2^3} + \frac{19}{x_3^3} + \frac{7}{x_4^3} + \frac{1}{x_5^3} - 1 \le 0, \tag{36}$$

with the simple bounds $0.01 \le x_i \le 100$. The best solution found so far in the literature is (Bekasş et al., 2018)

$$f_{min} = 1.33997,$$

$$\boldsymbol{x}_* = (6.0202, 5.3082, 4.5042, 3.4856, 2.1557). \tag{37}$$

14. Pressure vessel design. The main objective is to minimize

$$\begin{aligned}
\min f(\boldsymbol{x}) &= 0.6224 x_1 x_3 x_4 + 1.7781 x_2 x_3^2 \\
&\quad + 3.1661 x_1^2 x_4 + 19.84 x_1^2 x_3,
\end{aligned} \tag{38}$$

subject to

$$g_1(\boldsymbol{x}) = -x_1 + 0.0193 x_3 \le 0, \tag{39}$$

$$g_2(\boldsymbol{x}) = -x_2 + 0.00954 x_3 \le 0, \tag{40}$$

$$g_3(\boldsymbol{x}) = -\pi x_3^2 x_4 - \frac{4\pi}{3} x_3^3 + 1296000 \le 0, \tag{41}$$

$$g_3(\boldsymbol{x}) = x_4 - 240 \le 0. \tag{42}$$

The first two design variables must be the integer multiples of the basic thickness $h = 0.0625$ inches (Cagnina et al., 2008). Therefore, the lower and upper bounds are

$$Lb = [h, \quad h, \quad 10, \quad 10], \tag{43}$$

and

$$Ub = [99h, \quad 99h, \quad 200, \quad 200]. \tag{44}$$

Its true global optimal solution $f_{min} = 6059.714335$ occurs at

$$x_1 = 0.8125, \quad x_2 = 0.4375, \quad x_3$$
$$= 40.098446, \quad x_4 = 176.636596. \quad (45)$$

14. Parameter estimation of an ODE. For a vibration problem with a unit step input, we have its mathematical equation as an ordinary differential equation (Yang, 2023)

$$\frac{\ddot{y}}{\omega^2} + 2\zeta\frac{\dot{y}}{\omega} + y = u(t), \quad (46)$$

where ω and ζ are the two parameters to be estimated. Here, the unit step function is given

$$u(t) = \begin{cases} 0 & \text{if } t < 0, \\ 1 & \text{if } t \geq 0. \end{cases} \quad (47)$$

The initial conditions are $y(0) = y'(0) = 0$. For a given system, we have observed its actual response. The relevant measurements are given Table 1. In order to estimate the two unknown parameter values ω and ζ, we can define the objective function as

$$f(\boldsymbol{x}) = \sum_{i=0}^{10} [y(t_i) - y_s(t_i)]^2, \quad (48)$$

where $y(t_i)$ for $i = 0, 1, ..., 10$ are the observed values and $y_s(t_i)$ are the values obtained by solving the differential equation (46), given a guessed set of values ζ and ω. Here, we have used $\boldsymbol{x} = (\zeta, \omega)$.

The true values are $\zeta = 1/4$ and $\omega = 2$. The aim of this benchmark is to solve the differential equation iteratively so as to find the best parameter values that minimize the objective or best-fit errors.

4.3. Parameter settings

In our simulations, the parameter settings are: population size $n = 10$ with parameter values of $a = 1$, $b = 0.7$, $c = 1$, $p = 0.7$, $q = r = 1$ and $\Theta = \theta^t$ with $\theta = 0.97$. The maximum number of iterations is set to $t_{max} = 1000$.

For the benchmark functions, $D = 5$ is used for f_1, f_2, f_3, f_4, and f_{10}. For f_9, $D = 4$ is used so as to give f_{min} as a nice integer. For all other problems, their dimensionality has been given earlier in this section.

Table 1. Measured data for a vibration problem.

Time t_i	1	2	3	4	5
$y(t_i)$	1.0706	1.3372	0.8277	0.9507	1.0848
Time t_i	6	7	8	9	10
$y(t_i)$	0.9814	0.9769	1.0169	1.0012	0.9933

Table 2. Numerical experiments for ten test functions.

	Best (f_{min})	Worst	Mean (f_{min})	True f_{min}
Fun 1	$+1.4073e - 28$	$+1.3985e - 27$	$+5.2722e - 28$	0
Fun 2	$+1.0192e - 26$	$+4.3595e - 05$	$+4.3595e - 06$	0
Fun 3	$+2.6645e - 15$	$+2.3981e - 14$	$+9.7700e - 15$	0
Fun 4	$+9.8139e - 27$	$+6.6667e - 01$	$+2.0000e - 01$	0
Fun 5	$-3.4560e + 03$	$-3.4560e + 03$	$-3.4560e + 03$	-3456
Fun 6	$+3.1554e - 30$	$+1.8302e - 28$	$+7.9359e - 29$	0
Fun 7	$-1.9209e + 01$	$-1.6268e + 01$	$-1.8620e + 01$	-19.2085
Fun 8	$+1.9586e - 29$	$+7.6207e - 01$	$+7.6207e - 02$	0
Fun 9	$-1.6000e + 01$	$-1.6000e + 01$	$-1.6000e + 01$	-16
Fum 10	$+0.0000e + 00$	$+1.9899e + 00$	$+1.1940e + 00$	0

Table 3. Pressure vessel design benchmarks.

	x_1	x_2	x_3	x_4	f_{min}
Run 1	1.2500	0.6250	64.7668	11.9886	7332.8419
Run 2	0.7500	0.3750	38.8601	221.3657	**5850.3851**
Run 3	1.1875	0.6250	61.5285	26.9310	7273.5127
Run 4	1.0000	0.5000	51.8135	84.5785	6410.0869
Run 5	0.8750	0.4375	45.3367	140.2544	6090.5325
Run 6	0.8750	0.4375	45.3368	140.2538	6090.5262
Run 7	0.7500	0.3750	38.8601	221.3655	5850.3831
Run 8	1.0000	0.5625	51.8135	84.5785	6708.4337
Run 9	0.8125	0.4375	42.0984	176.6366	6059.7143
Run 10	0.7500	0.4375	38.8601	221.3655	6018.2036

5. Numerical experiments and results

After implementing the algorithm using MATLAB, we have carried out various numerical experiments. This section summarizes the results.

5.1. Results for function benchmarks

For the first ten functions, the algorithm has been run 20 times so that the best f_{min}, mean values and other statistics can be calculated. The results have been summarized in Table 2. As we can see, the algorithm can find all the true optimal solutions even with a small population size $n = 10$. This shows that the unified GEM algorithm is very efficient for solving function optimization problems.

5.2. Five case studies

To test the proposed algorithm further, we have used it to solve five different case studies, subject to various constraints. The constraints are handled by the standard penalty method with a penalty coefficient $\lambda = 1000$ to 10^5. For the pressure vessel problem, $\lambda = 10^5$ is used, whereas all other problems use $\lambda = 1000$.

For example, in the pressure vessel design problem, the four constraints are incorporated as $P(\boldsymbol{x})$ so that the new objective becomes

$$F_p(\boldsymbol{x}) = f(\boldsymbol{x}) + \lambda P(\boldsymbol{x}), \quad (49)$$

where

Table 4. Results for the five design benchmarks.

	Best (f_{min})	Worst	Mean (f_{min})	Best so far
Spring design	$1.2665e - 02$	$1.2873e - 02$	$1.2729e - 02$	$1.2665e - 02$
Truss system	$2.6389e + 02$	$2.6390e + 02$	$2.6390e + 02$	$2.6389e + 02$
Beam design	$1.3400e + 00$	$5.1657e + 00$	$2.3158e + 00$	$1.3400e + 00$
Pressure vessel	$5.8504e + 03$	$7.3328e + 03$	$6.3685e + 03$	$6.0597e + 03$
ODE parameters	$6.9479e - 09$	$1.5850e - 02$	$3.1700e - 03$	$6.8900e - 09$

$$P(x) = \sum_{i=1}^{4} \max\{0, g_i(\boldsymbol{x})\}, \quad \lambda = 10^5. \qquad (50)$$

The pressure vessel design problem is a mixed integer programming problem because the first two variables x_1 and x_2 can take only integer multiples of the basic thickness $h = 0.0625$ due to manufacturing constraints. The other two variables x_3 and x_4 can take continuous real values. For each case study, the algorithm has been run 10 times. For example, the 10 runs of the pressure vessel design are summarized in Table 3. As we can see, Run 2 finds a much better solution $f_{min} = 5850.3851$, highlighted in bold, than the best solution known so far in the literature 6059.7143. All the constraints are satisfied, which means that this is a valid new solution.

Following the exact same procedure, each case study has been simulated 20 times. The results for the five design case studies are summarized in Table 4. As we can see, all the best known optimal solutions have been found by the algorithm with a population size $n = 10$.

The above simulations and results have shown that the unified GEM algorithm performs well for all the 15 test benchmarks, and in some cases it can achieve even better results. This indicates that this unified algorithm is effective and can potentially be applied to solve a wide range of optimization problems. This will form part of our further research.

6. Conclusion and discussion

In this paper, we have proposed a unified algorithm, called generalized evolutionary metaheuristic (GEM), which represents more than 20 different algorithms in the current literature. We have validated the proposed GEM with 15 different test benchmarks and optimal solutions have been obtained with a small population size and fixed parameters.

From the parameter tuning perspective, it can be expected that if the parameters can be tuned systematically, it may be possible to enhance the algorithm's performance further. In fact, a systematical parameter study and parameter tuning is needed for this new algorithm, which will be carried out in our future work.

In addition, apart from more than 20 algorithms as special cases of this unified algorithm when setting different parameter values, it can be expected that the vast majority of the 540 algorithms can also be rightly represented in this unified algorithm if certain parameter are allowed to vary in certain ways and some minor differences in some algorithms can be ignored. Obviously, for some algorithms such as the genetic algorithm, the algorithm is mainly described as an algorithmic procedure without explicit mathematical formulas. Such type of algorithm may not be easily categorized into the generalized framework. However, the procedure and algorithmic flow may in essence be quite similar to some of the main framework. In addition, a systematic comparison of this general framework can be carried out with various component algorithms. This can be a useful topic for further research.

Declaration of data

No datasets were received/used. All results are simulated and reproducible from the proposed algorithms. MATLAB codes related to this work are available upon request, by emailing the author.

Disclosure statement

The author confirms that there are no relevant financial or non-financial competing interests to report.

Funding

No funding was received for this research work.

About the author

Xin-She Yang obtained his DPhil in Applied Mathematics from the University of Oxford. He then worked at Cambridge University and National Physical Laboratory (UK) as a Senior Research Scientist. Now he is Reader at Middlesex University London, and a co-Editor of the Springer Tracts in Nature-Inspired Computing. He is also an elected Fellow of the Institute of Mathematics and its Applications. He was the IEEE Computational Intelligence Society (CIS) chair for the Task Force on Business Intelligence and Knowledge Management (2015 to 2020). He has published more than 300 peer-reviewed research

papers with more than 90,270 citations, and he has been on the prestigious list of highly-cited researchers (Web of Sciences) for eight consecutive years (2016-2023).

ORCID

Xin-She Yang http://orcid.org/0000-0001-8231-5556

References

Abdel-Basset, M., & Shawky, L. A. (2019). Flower pollination algorithm: A comprehensive review. *Artificial Intelligence Review*, *52*(4), 2533–2557. https://doi.org/10.1007/s10462-018-9624-4

Al-Roomi, A. R. (2015). Unconstrained single-objective benchmark function repository.

Auger, A., & Teytaud, O. (2010). Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica*, *57*(1), 121–146. https://doi.org/10.1007/s00453-008-9244-5

Beer, D. (2016). The social power of algorithms. *Information, Communication & Society*, *20*(1), 1–13. https://doi.org/10.1080/1369118X.2016.1216147

Bekasş, G., Nigdeli, M., & Yang, X.-S. (2018). A novel bat algorithm based optimum tuning of mass dampers for improving the seismic safety of structures. *Engineering Structures*, *159*(1), 89–98.

Bertsekas, D. P., Nedic, A., & Ozdaglar, A. (2003). *Convex analysis and optimization* (2nd ed.). Athena Scientific.

Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Cagnina, L. C., Esquivel, S. C., & Coello Coello, A. C. (2008). Solving engineering optimization problems with the simple constrained particle swarm optimizer. *Informatica*, *32*(2), 319–326.

Chabert, J. L. (1999). *A history of algorithms: From the pebble to the microchips*. Springer-Verlag.

Chen, S., Peng, G.-H., He, X.-S., & Yang, X.-S. (2018). Global convergence analysis of the bat algorithm using a Markovian framework and dynamic system theory. *Expert Systems with Applications*, *114*(1), 173–182. https://doi.org/10.1016/j.eswa.2018.07.036

Civicioglu, P. (2013). Artificial cooperative search algorithm for numerical optimization problems. *Information Sciences*, *229*(1), 58–76. https://doi.org/10.1016/j.ins.2012.11.013

Clerc, M., & Kennedy, J. (2002). The particle swarm: Explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, *6*(1), 58–73. https://doi.org/10.1109/4235.985692

Coello, C. A. (2000). Use of self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, *41*(2), 113–127. https://doi.org/10.1016/S0166-3615(99)00046-9

Corne, D., & Knowles, J. (2003). Some multiobjective optimizers are better than others. *Evolutionary Computation*, *4*(2), 2506–2512.

Eiben, A. E., & Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, *1*(1), 19–31. https://doi.org/10.1016/j.swevo.2011.02.001

Erol, O., & Eksin, I. (2006). A new optimization method: Big bang-big crunch. *Advances in Engineering Software*, *37*(2), 106–111. https://doi.org/10.1016/j.advengsoft.2005.04.005

Eskandar, H., Sadollah, A., Bahreininejad, A., & Hamdi, M. (2012). Water cycle algorithm – A novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers & Structures*, *110–111*(1), 151–166. https://doi.org/10.1016/j.compstruc.2012.07.010

Gavvala, S. K., Jatoth, C., Gangadharan, G. R., & Buyya, R. (2019). QoS-aware cloud service composition using eagle strategy. *Future Generation Computer Systems*, *90*, 273–290. https://doi.org/10.1016/j.future.2018.07.062

Geem, Z. W., Kim, J. H., and Loganathan, G. V. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation*, *76*(2):60–68. https://doi.org/10.1177/003754970107600201

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

Glover, F., & Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Greenhalgh, D., & Marshal, S. (2000). Convergence criteria for genetic algorithm. *SIAM Journal Comput*, *30*(1), 269–282.

Hashim, F. A., Houssein, E. H., Mabrouk, M. S., Al-Atabany, W., & Mirjalili, S. (2019). Henry gas solubility optimization: A novel physics-based algorithm. *Future Generation Computer Systems*, *101*, 646–667. https://doi.org/10.1016/j.future.2019.07.015

Heidari, A. A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., & Chen, H. (2019). Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems*, *97*, 849–872. https://doi.org/10.1016/j.future.2019.02.028

Holland, J. (1975). *Adaptation in nature and artificial systems*. University of Michigan Press.

James, J., & Li, V. (2015). A social spider algorithm for global optimization. *Applied Soft Computing*, *30*, 614–627. https://doi.org/10.1016/j.asoc.2015.02.014

Jamil, M., & Yang, X.-S. (2013). A literature survey of benchmark functions for global optimization problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, *4*(2), 150–194.

Joy, G., Huyck, C., & Yang, X.-S. (2023). *Review of Parameter tuning methods for nature-inspired algorithms* (pp. 33–47). Springer Nature Singapore.

Karaboga, D., & Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, *8*(1), 687–697. https://doi.org/10.1016/j.asoc.2007.05.007

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, *4*(4), 373–395. https://doi.org/10.1007/BF02579150

Kaveh, A., & Talatahari, S. (2010). A novel heuristic optimization method: Charged system search. *Acta Mechanica*, *213*(3–4), 267–289. https://doi.org/10.1007/s00707-009-0270-4

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1942–1948). IEEE.

Kennedy, J., Eberhart, R. C., & Shi, Y. (2001). *Swarm intelligence*. Academic Press.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science (New York, N.Y.)*, *220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Kuo, R., & Zulvia, F. (2015). The gradient evolution algorithm: A new metaheuristic. *Information Sciences*, *316*(2), 246–265. https://doi.org/10.1016/j.ins.2015.04.031

Mirjalili, S. (2015). The ant lion optimizer. *Advances in Engineering Software*, *83*(1), 80–98.

Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software*, *95*(1), 51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008

Mirjalili, S., Mirjalili, S., & Hatamlou, A. (2016). Multi-verse optimizer: A nature-inspired algorithm for global optimization. *Neural Computing and Applications*, *27*(2), 495–513. https://doi.org/10.1007/s00521-015-1870-7

Mohamed, A., Hadi, A., & Mohamed, A. (2020). Gaining-sharing knowledge based algorithm for solving optimization problems: A novel nature-inspired algorithm. *International Journal of Machine Learning and Cybernatics*, *11*(7), 1501–1529.

Osaba, E., Yang, X.-S., Diaz, F., Lopez-Garcia, P., & Carballedo, R. (2016). An improved discrete bat algorithm for symmetric and assymmetric travelling salesman problems. *Engineering Applications of Artificial Intelligence*, *48*(1), 59–71. https://doi.org/10.1016/j.engappai.2015.10.006

Osaba, E., Yang, X.-S., Diaz, F., Onieva, E., Masegosa, A., & Perallos, A. (2017). A discrete firefly algorithm to solve a rich vehicle routing problem modelling a newspaper distribution system with recycling policy. *Soft Computing*, *21*(18), 5295–5308. https://doi.org/10.1007/s00500-016-2114-1

Ouaarab, A., Ahiod, B., & Yang, X.-S. (2014). Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Computing and Applications*, *24*(7–8), 1659–1669. https://doi.org/10.1007/s00521-013-1402-2

Palmieri, N., Yang, X.-S., Rango, F. D., & Santamaria, A. F. (2018). Self-adaptive decision-making mechanisms to balance the execution of multiple tasks for a multi-robots team. *Neurocomputing*, *306*(1), 17–36. https://doi.org/10.1016/j.neucom.2018.03.038

Pham, D. T., & Castellani, M. (2009). The bees algorithm: Modelling foraging behaviour to solve continuous optimization problems. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, *223*(12), 2919–2938. https://doi.org/10.1243/09544062JMES1494

Pham, D. T., & Castellani, M. (2014). Benchmarking and comparison of nature-inspired population-based continuous optimisation algorithms. *Soft Computing*, *18*(5), 871–903. https://doi.org/10.1007/s00500-013-1104-9

Pham, D. T., & Castellani, M. (2015). A comparative study of the bees algorithm as a tool for function optimisation. *Cogent Engineering*, *2*(1), 1091540. https://doi.org/10.1080/23311916.2015.1091540

Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. (2005). *The bees algorithm, technical note*. Technical report. Cardiff University.

Price, K., Storn, R., & Lampinen, J. (2005). *Differential evolution: A practical approach to global optimization*. Springer.

Rajwar, K., Deep, K., & Das, S. (2023). An exhaustive review of the metaheuristic algorithms for search and optimization: Taxonomy, applications, and open challenges. *Artificial Intelligence Review*, *56*(11), 1–71. https://doi.org/10.1007/s10462-023-10470-y

Rango, F. D., Palmieri, N., Yang, X.-S., & Marano, S. (2018). Swarm robotics in wireless distributed protocol design for coordinating robots involved in cooperative tasks. *Soft Computing*, *22*(13), 4251–4266. https://doi.org/10.1007/s00500-017-2819-9

Rashedi, E., Nezamabadi-Pour, H. H., & Saryazdi, S. (2009). GSA: A gravitational search algorithm. *Information Sciences*, *179*(13), 2232–2248. https://doi.org/10.1016/j.ins.2009.03.004

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *Computer Journal*, *3*(3), 175–184. https://doi.org/10.1093/comjnl/3.3.175

Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). In Aardal, K., Nemhauser, G. L., & Weismantel, R. (Eds.), *Handbook of discrete optimization* (pp. 1–68). Elsevier.

Schwefel, H. (1995). *Evolution and optimum seeking*. John Wiley Sons.

Ser, J. D., Osaba, E., Molina, D., Yang, X.-S., Salcedo-Sanz, S., Camacho, D., Das, S., Suganthan, P. N., Coello, C. A. C., & Herrera, F. (2019). Bio-inspired computation: Where we stand and what's next. *Swarm and Evolutionary Computation*, *48*, 220–250.

Storn, R., & Price, K. (1997). Differential evolution: A simple and efficient heuristic for global optimization. *Journal of Global Optimization*, *11*(4), 341–359. https://doi.org/10.1023/A:1008202821328

Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y., Auger, A., & Tiwar, S. (2005). *Problem definitions and evaluation criteria for CEC 2005, special session on real-parameter optimization*. Technical Report. Nanyang Technological University (NTU), Singapore.

Wang, G. (2018). Moth search algorithm: A bio-inspired metaheuristic algorithm for global optimization problems. *Memetic Computing*, *10*(2), 151–164. https://doi.org/10.1007/s12293-016-0212-3

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.

Wolpert, D. H., & Macready, W. G. (2005). Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, *9*(6), 721–735. https://doi.org/10.1109/TEVC.2005.856205

Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. In Watanabe, O., & Zeugmann, T. (Eds.), *Proceedings of Fifth symposium on stochastic algorithms, foundations and applications* (Vol. 5792, pp. 169–178). Lecture Notes in Computer Science Springer.

Yang, X.-S. (2013). *Cuckoo search and firefly algorithm: Theory and applications, volume 516 of studies in computational intelligence*. Springer.

Yang, X.-S. (2020a). *Nature-inspired optimization algorithms* (2nd ed.). Academic Press.

Yang, X.-S. (2020b). Nature-inspired optimization algorithms: Challenges and open problems. *Journal of Computational Science*, *46*, 101104. https://doi.org/10.1016/j.jocs.2020.101104

Yang, X.-S. (2023). Ten new benchmarks for optimization. In Yang, X.-S. (Ed.), *Benchmarks and hybrid algorithms in optimization and applications* (pp. 19–32). Springer.

Yang, X.-S., & Deb, S. (2010). Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, *1*(4), 330–343. https://doi.org/10.1504/IJMMNO.2010.035430

Yang, X.-S., Deb, S., Loomes, M., & Karamanoglu, M. (2013). A framework for self-tuning optimization algorithm. *Neural Computing and Applications*, *23*(7–8), 2051–2057. https://doi.org/10.1007/s00521-013-1498-4

Yang, X.-S., Deb, S., Zhao, Y.-X., Fong, S., & He, X. (2018). Swarm intelligence: Past, present and future. *Soft Computing*, *22*(18), 5923–5933. https://doi.org/10.1007/s00500-017-2810-5

Yang, X.-S., & He, X.-S. (2019). *Mathematical foundations of nature-inspired algorithms*. Springer Briefs in Optimization. Springer.

Yazdani, M., & Jolai, F. (2016). Lion optimization algorithm (LOA): A nature-inspired metaheuristic algorithm. *Journal of Computational Design and Engineering*, *3*(1), 24–36. https://doi.org/10.1016/j.jcde.2015.06.003

Zaharie, D. (2009). Influence of crossover on the behavior of the differential evolution algorithm. *Applied Soft Computing*, *9*(3), 1126–1138. https://doi.org/10.1016/j.asoc.2009.02.012

Zdenek, D. (2009). *Optimal quadratic programming algorithms: With applications to variational inequalities*. Springer.

Zelinka, I. (2015). A survey on evolutionary algorithm dynamics and its complexy-mutual relations, past, present and future. *Swarm and Evolutionary Computation*, *25*(1), 2–14. https://doi.org/10.1016/j.swevo.2015.06.002

Zervoudakis, K., & Tsafarakis, S. (2020). A mayfly optimization algorithm. *Computers & Industrial Engineering*, *145*, 106559. https://doi.org/10.1016/j.cie.2020.106559

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Fonseca, V. G. D. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transaction on Evolutionary Computation*, *7*(2), 117–132.