Engineering Optimization and Industrial Applications

Xin-She Yang

Abstract Design optimization is important in engineering and industrial applications. It is usually very challenging to find optimum designs, which require both efficient optimization algorithms and high-quality simulators that are often time-consuming. To some extent, an optimization process is equivalent to a self-organizing system, and the organized states are the optima that are to be searched for. In this chapter, we discuss both optimization and self-organization in a unified framework, and we use three metaheuristic algorithms, the firefly algorithm, the bat algorithm and cuckoo search, as examples to see how this self-organized process works. We then present a set of nine design problems in engineering and industry. We also discuss the challenging issues that need to be addressed in the near future.

Keywords Bat algorithm · Cuckoo search · Firefly algorithm · Optimization · Metaheuristic · Self-organizaion

1 Introduction

Optimization is ubiquitous in many applications in engineering and industry. In essence, optimization is a process of searching for the optimal solutions to a particular problem of interest, and this search process can be carried out using multiple agents which essentially form a system of evolving agents. This system can evolve by iterations according to a set of rules or mathematical equations. Consequently, such systems will show some emergent characteristics, leading to self-organizing states which correspond to some optima in the search space. Once the self-organized states are reached, we say the system has converged. Therefore, the design of an efficient optimization algorithm is equivalent to mimicking the evolution of a self-organizing system.

In almost all applications in engineering and industry, we are always trying to optimize something—whether to minimize the cost and energy consumption, or to

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

maximize the profit, output, performance and efficiency. In reality, resources, time and money are always limited; consequently, optimization is far more important in practice [2, 13, 30, 32, 36, 39, 41].

It is worth pointing out that computational efforts are a main issue in many optimization problems in engineering and industry, because the most time-consuming part of the optimization process is the evaluations of objective functions [17, 39]. The use of the most efficient algorithms is just one way of tackling the problem, while an alternative is to use surrogate-based models which can often be more efficient if the number of evaluating high-fidelity models is significantly reduced [18–20]. Such surrogate-based optimization using a combination of low-fidelity and high-fidelity models can be a powerful tool for many real-world applications. This book contains many examples of surrogate-based modelling and optimization. In this chapter, our focus is mainly on the introduction of some widely used new algorithms and a well-chosen set of design benchmarks.

2 Optimization Algorithms and Self-organization

2.1 Self-organizing Systems

Self-organization exists in many systems, from physical and chemical to biological and artificial systems. Emergent phenomena such as Rayleigh–Bénard convection, Turing pattern formation [26] and organisms and thunderstorms [15] can all be called self-organizing. Though there is no universal theory for self-organizing processes, some aspects of self-organization can be partly understood using theories based on nonlinear dynamical systems, far-from-equilibrium [23] multiple interacting agents, and closed systems under unchanging laws [3]. As pointed out by cyberneticist and mathematician Ross Ashby, every isolated determinate dynamic system, obeying unchanging laws, will ultimately develop some sort of ‘organisms’ that are adapted to their ‘environments’ [3].

For simple systems, going to equilibrium is trivial, but, for a complex system, if its size is so large that its equilibrium states are just a fraction of the vast number of possible states, and if the system is allowed to evolve long enough, some self-organized structures may emerge. Changes in environment can apply pressure on the system to re-organize and adapt to such changes. If the systems have sufficient perturbations or noise, often working at the edge of the chaos, some spontaneous formation of structures will emerge as the systems move, far from equilibrium, and select some states, thus reducing the uncertainty or entropy.

The state set S of a complex system such as a machine may change from initial states $S(\psi)$ to other states $S(\phi)$, subject to the change of a parameter set $\alpha(t)$ which can be time-dependent. That is,

$$S(\psi) \xrightarrow{\alpha(t)} S(\phi), \tag{1}$$

where $\alpha(t)$ must come from external conditions such as the heat flow in Rayleigh–Bénard convection, not from the states S themselves. Obviously, $S + \alpha(t)$ can be considered as a larger, closed system [3]. In this sense, self-organization is equivalent to a mapping from some high-entropy states to low-entropy states.

An optimization algorithm can be viewed as a complex, dynamical system. If we can consider the convergence process as a self-organizing process, then there are strong similarities and links between self-organizing systems and optimization algorithms.

2.2 Algorithms for Self-organization

Mathematically speaking, an algorithm is a procedure to generate outputs for given inputs. From the optimization point of view, an optimization algorithm generates a new solution $\boldsymbol{x}^{t+1}$ to a given problem from a known solution $\boldsymbol{x}^t$ at iteration or time t. That is

$$\boldsymbol{x}^{t+1} = \boldsymbol{A}\big(\boldsymbol{x}^t, \boldsymbol{p}(t)\big), \tag{2}$$

where $\boldsymbol{A}$ is a nonlinear mapping from a given solution, or d-dimensional vector, $\boldsymbol{x}^t$ to a new solution vector $\boldsymbol{x}^{t+1}$. The algorithm $\boldsymbol{A}$ has k algorithm-dependent parameters $\boldsymbol{p}(t) = (p_1, \dots, p_k)$ which can be time-dependent and can thus be tuned if necessary.

To find the optimal solution $\boldsymbol{x}_*$ to a given optimization problem S with an often infinite number of states is to select some desired states ϕ from all states ψ, according to some predefined criterion D. We have

$$S(\psi) \xrightarrow{A(t)} S\big(\phi(x_*)\big), \tag{3}$$

where the final converged state ϕ corresponds to an optimal solution $\boldsymbol{x}_*$ to the problem of interest. The selection of the system states in the design space is carried out by running the optimization algorithm $\boldsymbol{A}$. The behaviour of the algorithm is controlled by $\boldsymbol{p}$, the initial solution $\boldsymbol{x}^{t=0}$ and the stopping criterion D. We can view the combined $S + \boldsymbol{A}(t)$ as a complex system with a self-organizing capability.

The change of states or solutions of the problem of interest is controlled by the algorithm $\boldsymbol{A}$. In many classical algorithms such as hill-climbing, gradient information is often used to select states, say, the minimum value of the landscape, and the stopping criterion can be a given tolerance or accuracy, or zero gradient etc. Alternatively, an algorithm can act like a tool to tune a complex system. If an algorithm does not use any state information of the problem, then it is more likely to be versatile to deal with many types of problems. However, such black-box approaches can also imply that the algorithm may not be as efficient as it could be for a given type of problem. For example, if the optimization problem is convex, algorithms that use such convexity information will be more efficient than those that do not use such information. In order to select states/solutions efficiently, the information from the search process should be used to enhance the search process. In many cases, such information is often fed into the selection mechanism of an algorithm. By far the most widely used selection mechanism is to identify and keep the best solution found so far. That is, some form of ‘survival of the fittest’ is used.

Optimization algorithms can be very diverse. There are dozens of widely used algorithms. The main characteristics of different algorithms will only depend on the actual, often highly nonlinear or implicit, forms of $\boldsymbol{A}(t)$ and their parameters $\boldsymbol{p}(t)$.

In many situations concerning optimization, the generation and verification of the new solutions can often involve computationally expensive computer simulations or even measurements of the physical system. In such cases, the expensive model of the system under consideration is often replaced by its cheaper representation, called a surrogate model, and the algorithm A uses that model to produce a new solution. The parameters $p(t)$ may then include variables that are used to align the surrogate with the expensive model to make it a reliable representation of the latter.

3 Three New Algorithms

In this chapter, we illustrate the concept of a self-organizing optimization algorithm using a specific class of algorithms called metaheuristics. Metaheuristics have some important characteristics that uses stochastic components to enable an algorithm to escape the possibility of being trapped in a local optimum. This often makes the search process more ergodic, and thus such algorithms can generate high-quality solutions over the search space during iterations, which may ultimately converge towards the true optimality of the problem of interest.

There are well over two dozen metaheuristic algorithms now in use for optimization [16, 30, 34]. All metaheuristic algorithms have to balance exploration and exploitation during the search process by using some sort of algorithm-dependent parameter setting. From the viewpoint of a self-organizing system, parameter settings will affect the way and routes by which the optimization process converges to an organized state. Here we analyse three relatively new nature-inspired algorithms and see the ways in which they can quickly converge towards optimality.

3.1 Firefly Algorithm

The first algorithm to be discussed is the firefly algorithm, which is essentially a dynamical system. The firefly algorithm (FA), first developed by Xin-She Yang in 2008 [29, 30], was based on the flashing patterns and behaviour of fireflies. In essence, FA uses the following three idealized rules:

- Fireflies are unisex, so one firefly can be attracted to any other one.
- The attractiveness is proportional to the brightness and they both decrease as their distance increases. Thus for any two flashing fireflies, the less brighter one will move towards the brighter one. If there is no brighter one than a particular firefly, it will move randomly.

- The brightness of a firefly is determined by the landscape of the objective function.

As a firefly's attractiveness is proportional to the light intensity seen by adjacent fireflies, we can now define the variation of attractiveness β with the distance r by

$$\beta = \beta_0 e^{-\gamma r^2}, \tag{4}$$

where β_0 is the attractiveness at $r = 0$.

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

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

where the second term is due to the attraction. The third term is randomization with α being the randomization parameter, and $\boldsymbol{\epsilon}_i^t$ is a vector of random numbers drawn from a Gaussian distribution or uniform distribution at time t. If $\beta_0 = 0$, it becomes a simple random walk. Furthermore, the randomization $\boldsymbol{\epsilon}_i^t$ can easily be extended to other distributions such as Lévy flights. A Lévy flight essentially provides a random walk whose random step length is drawn from a Lévy distribution

$$\text{Lévy} \sim u = t^{-\lambda} \quad (1 < \lambda \le 3), \tag{6}$$

which has an infinite variance with an infinite mean. Here the steps essentially form a random walk process with a power-law step-length distribution with a heavy tail. Some of the new solutions should be generated by the Lévy walk around the best solution obtained so far, and this will speed up the local search. A demo version of firefly algorithm implementation, without Lévy flights, can be found at the Mathworks file exchange web site.[1] FA has attracted much attention recently [1, 11, 25].

A discrete version of FA can efficiently solve NP-hard scheduling problems [25], while a detailed analysis has demonstrated the efficiency of FA over a wide range of test problems, including multiobjective load dispatch problems [1]. Highly nonlinear and non-convex global optimization problems can be solved efficiently using FA [11, 42].

From the self-organization point of view, FA acts as a simple dynamic system with diverse characteristics that can automatically subdivide the entire population into subgroups, and each subgroup can swarm around a local mode. Among all the local modes, there is always a global optimum, and thus FA can find the global optimality and local optima simultaneously if the number of fireflies is sufficiently higher than the number of modes.

[1] http://www.mathworks.com/matlabcentral/fileexchange/29693-firefly-algorithm.

3.2 Cuckoo Search

Cuckoo search (CS) is one of the latest nature-inspired metaheuristic algorithms, developed in 2009 by Xin-She Yang and Suash Deb [34]. CS is based on the brood parasitism of some cuckoo species. In addition, this algorithm is enhanced by Lévy flights, rather than by simple isotropic random walks. Recent studies show that CS is potentially far more efficient than particle swarm optimization (PSO) and genetic algorithms [35].

Cuckoos are fascinating birds, not only because of the beautiful sounds they can make, but also because of their aggressive reproduction strategy. Some species such as the *Ani* and *Guira* cuckoos lay their eggs in communal nests, though they may remove others' eggs to increase the hatching probability of their own eggs. Quite a number of species engage the obligate brood parasitism by laying their eggs in the nests of other host birds (often other species).

For simplicity in describing CS, we now use the following three idealized rules:

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest.
- The best nests with high-quality eggs will be carried over to the next generations.
- The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0, 1]$. In this case, the host bird can either get rid of the egg, or simply abandon the nest and build a completely new nest.

As a further approximation, this last assumption can be approximated by a fraction p_a of the n host nests that are replaced by new nests (with new random solutions). For a maximization problem, the quality or fitness of a solution can simply be proportional to the value of the objective function. Other forms of fitness can be defined in a similar way to the fitness function in genetic algorithms.

This algorithm uses a balanced combination of a local random walk and a global explorative random walk, controlled by a switching parameter p_a. The local random walk can be written as

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha s \otimes H(p_a - \epsilon) \otimes \left(\boldsymbol{x}_j^t - \boldsymbol{x}_k^t\right), \tag{7}$$

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

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \alpha L(s, \lambda), \tag{8}$$

where

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}} \quad (s \gg s_0 > 0). \tag{9}$$

Here $\alpha > 0$ is the step size scaling factor, which should be related to the scales of the problem of interest. In most cases, we can use $\alpha = O(L/10)$, where L is the characteristic scale of the problem of interest, while in some case $\alpha = O(L/100)$ can be more effective and avoid flying too far.

The above equation is essentially the stochastic equation for a random walk. In general, a random walk is a Markov chain whose next status/location only depends on the current location (the first term in the above equation) and the transition probability (the second term). However, a substantial fraction of the new solutions should be generated by far-field randomization, and their locations should be far enough from the current best solution to make sure that the system will not be trapped in a local optimum [34].

Though the pseudo-code given in many papers is sequential, vectors should be used from the implementation point of view, as vectors are more efficient than loops. A Matlab implementation is given by the author, and it can be downloaded.[2]

The literature on CS is expanding rapidly. Much attention and many recent studies have used CS with a diverse range of applications [7, 11, 27, 37]. Walton et al. improved the algorithm by formulating a modified CS algorithm [27], while Yang and Deb extended it to multiobjective optimization problems [37].

Looking at CS in terms of self-organization, we can see that this swarm-intelligence-based algorithm uses multiple interacting Markov chains by switching between two key branches of global search and local search using Lévy flights as well as random walks so that a balance between global exploration and local exploitation can be achieved during the optimization process.

3.3 Bat Algorithm

A third way of looking at an algorithm, apart from dynamic systems and Markov chains, is by using a varying parameter setting. This idea is used in the bat algorithm; the parameter tuning is essentially achieved by frequency tuning and mimicking the hunting strategy of microbats.

The bat algorithm (BA) is a relatively new metaheuristic, developed by Xin-She Yang in 2010 [40], which was inspired by the echolocation behaviour of microbats. Microbats use a type of sonar, called echolocation, to detect prey, avoid obstacles and locate their roosting crevices in the dark. These bats emit a very loud sound pulse and listen for the echo that bounces back from surrounding objects. Their pulses have varying properties and can be correlated with their hunting strategies, depending on the species. Most bats use short, frequency-modulated signals to sweep through about an octave, while others more often use constant-frequency signals for echolocation. The signal bandwidth varies depending on the species, and is often increased by using more harmonics.

The bat algorithm has three idealized rules:

- All bats use echolocation to sense distance, and they also 'know' the difference between food/prey and background barriers in some magical way.

[2] www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm.

- Bats fly randomly with velocity $\boldsymbol{v}_i$ at position $\boldsymbol{x}_i$ with a fixed frequency $f_{\min}$, varying wavelength λ and loudness A_0 to search for prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r \in [0, 1]$, depending on the proximity of their target.
- Although the loudness can vary in many ways, we assume that it varies from a large (positive) A_0 to a minimum constant value $A_{\min}$.

Obviously, we have to define the rules of how their positions $\boldsymbol{x}_i$ and velocities $\boldsymbol{v}_i$ in a d-dimensional search space are updated. The new solutions $\boldsymbol{x}_i^t$ and velocities $\boldsymbol{v}_i^t$ at time step t are given by

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{10}$$

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + \left(\boldsymbol{x}_i^{t-1} - \boldsymbol{x}_*\right) f_i, \tag{11}$$

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

where $\beta \in [0, 1]$ is a random vector drawn from a uniform distribution. Here $\boldsymbol{x}_*$ is the current global best location (solution) which is located after comparing all the solutions among all the n bats at each iteration t. As the product $\lambda_i f_i$ is the velocity increment, we can use f_i (or λ_i) to adjust the velocity change while fixing the other factor λ_i (or f_i), depending on the type of problem of interest. In our implementation, we will use $f_{\min} = 0$ and $f_{\max} = O(1)$, depending on the domain size of the problem of interest. Initially, each bat is randomly assigned a frequency which is drawn uniformly from $[f_{\min}, f_{\max}]$.

For the local search part, once a solution is selected among the current best solutions, a new solution for each bat is generated locally using random walk

$$\boldsymbol{x}_{\text{new}} = \boldsymbol{x}_{\text{old}} + \boldsymbol{\epsilon} A^t, \tag{13}$$

where $\boldsymbol{\epsilon}$ is a random number vector drawn from $[-1, 1]$, while $A^t = \langle A_i^t \rangle$ is the average loudness of all the bats at this time step.

Furthermore, the loudness A_i and the rate r_i of pulse emission have to be updated accordingly as the iterations proceed. As the loudness usually decreases once a bat has found its prey, while the rate of pulse emission increases, the loudness can be chosen as any value of convenience. For simplicity, we can also use $A_0 = 1$ and $A_{\min} = 0$, assuming $A_{\min} = 0$ means that a bat has just found the prey and will temporarily stop emitting any sound. Now we have

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

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

where α and γ are constants. In fact, α is similar to the cooling factor of a cooling schedule in the simulated annealing. For any $0 < \alpha < 1$ and $\gamma > 0$, we have

$$A_i^t \to 0, \qquad r_i^t \to r_i^0, \quad \text{as } t \to \infty. \tag{16}$$

In the simplest case, we can use $\alpha = \gamma$, and we have used $\alpha = \gamma = 0.95$ to 0.97 in our simulations.

BA has been extended to the multiobjective bat algorithm (MOBA) by Yang [31], and preliminary results suggest that it is very efficient.

Again looking at BA from the self-organization point of view, the convergence is controlled by loudness and pulse emission rate so that it can explore the vast search space in the earlier stage and then focus on the local exploitation in the more promising regions. Compared with FA and CS, where fixed parameters are used in terms of balancing exploration and exploitation, BA uses a more dynamic approach to balance exploration and exploitation.

4 Engineering Optimization and Applications

Engineering optimization is very diverse with vast collections of case studies, and some case studies require lengthy descriptions to provide sufficient details [6, 10–12, 24]. Here we provide nine case studies as a subset of design optimization benchmarks in engineering and industrial applications.

4.1 Bending Beam Design

Probably the simplest design problem with engineering relevance is the design of a cantilever beam with five different square cross sections with heights/widths of $x_1, x_2, \ldots, x_5$, respectively. The thickness is fixed with $t = 2/3$, and the objective is to minimize [5, 9]

$$f(\boldsymbol{x}) = 0.0624(x_1 + x_2 + x_3 + x_3 + x_4 + x_5), \tag{17}$$

subject to

$$g(\boldsymbol{x}) = \frac{61}{x_1^3} + \frac{37}{x_2^3} + \frac{19}{x_3^3} + \frac{7}{x_4^3} + \frac{1}{x_5^3} - 1 \le 0. \tag{18}$$

It is straightforward to use all three algorithms discussed earlier to find the best solution

$$\boldsymbol{x} = (6.0089,\ 5.3049,\ 4.5023,\ 3.5077,\ 2.1504), \tag{19}$$

which gives

$$f_{\min} = 1.33999. \tag{20}$$

4.2 Spring Design

Tensional and/or compressional springs are used widely in engineering. A standard spring design problem has three design variables: the wire diameter w, the mean coil diameter d and the length (or number of coils) L.

The objective is to minimize the weight of the spring, subject to various constraints such as maximum shear stress, minimum deflection and geometrical limits. For a detailed description, please refer to earlier studies [2, 4]. This problem can be written compactly as

$$\text{Minimize} \quad f(\boldsymbol{x}) = (L+2)w^2 d, \tag{21}$$

subject to

$$\begin{aligned} g_1(\boldsymbol{x}) &= 1 - \frac{d^3 L}{71785 w^4} \leq 0, \\ g_2(\boldsymbol{x}) &= 1 - \frac{140.45 w}{d^2 L} \leq 0, \\ g_3(\boldsymbol{x}) &= \frac{2(w+d)}{3} - 1 \leq 0, \\ g_4(\boldsymbol{x}) &= \frac{d(4d-w)}{w^3(12566d - w)} + \frac{1}{5108 w^2} - 1 \leq 0, \end{aligned} \tag{22}$$

with the following limits:

$$0.05 \leq w \leq 2.0, \qquad 0.25 \leq d \leq 1.3, \qquad 2.0 \leq L \leq 15.0. \tag{23}$$

Using any of the algorithms discussed earlier, we can easily obtain the same or slightly better solutions than the best solution obtained by [4]:

$$f_* = 0.012665 \quad \text{at } (0.051690, 0.356750, 11.287126), \tag{24}$$

but both CS and FA use significantly fewer evaluations.

4.3 Three-Bar Truss Design

The three-bar truss design is a simple but practical example first presented by Nowcki [22], which requires one to find the optimal cross-sectional areas A_1 and A_2. The problem can be formulated as

$$\text{Minimize} \quad f(A_1, A_2) = (\sqrt{8A_1} + A_2)L, \tag{25}$$

subject to

$$g_1 = \frac{(\sqrt{2}A_1 + A_2)P}{\sqrt{2}A_1^2 + 2A_1A_2} - \sigma \leq 0, \tag{26}$$

$$g_2 = \frac{A_2 P}{\sqrt{2}A_1^2 + 2A_1A_2} - \sigma \leq 0, \tag{27}$$

$$g_3 = \frac{P}{A_1 + \sqrt{2}A_2} - \sigma \leq 0, \tag{28}$$

where $\sigma = 2{,}000$ N/cm^2 is the stress constraint, and $P = 2{,}000$ N/cm^2 is the load. The simple limits are

$$0 \leq A_1, A_2 \leq 1. \tag{29}$$

Using CS and BA, it is easy to find the optimal solution

$$\boldsymbol{x}_* = (A_1, A_2) = (0.78867, 0.40902), \tag{30}$$

and

$$f_{\min} = 263.97156. \tag{31}$$

4.4 Discrete Beam Design

Reinforced concrete beam designs are relevant in many applications in engineering and construction. One class of beam designs is the discrete beam design, where the dimensions and some design variables can only take discrete values [11, 21]. For example, a very simple design benchmark of a reinforced concrete beam can be written as

$$\text{Minimize} \quad f(A_s, b, h) = 0.6bh + 2.9A_s, \tag{32}$$

subject to

$$g_1 = \frac{h}{b} - 4 \leq 0, \tag{33}$$

$$g_2 = \frac{7.375A_s^2}{b} + 180 - A_s h \leq 0. \tag{34}$$

However, the area A_s only take values of $\{6.0, 6.16,\ 6.32, 6.6,\ 7.0, 7.11, 7.2, 7.8, 7.9,\ 8.0, 8.4\}$ in^2, and b only takes a value from $\{28, 29, 30, \ldots, 39, 40\}$ and $5 \leq h \leq 10$ in the continuous domain [21].

By using FA and CS, we have found the best solution

$$f_{\min} = 359.2080, \tag{35}$$

with

$$(A_s, b, h) = (6.32, 34, 8.5), \tag{36}$$

which is better than any solutions found so far in the literature [11].

4.5 Heat Exchanger Design

The heat exchanger design is a problem with six constraints [38], which can be expressed in the simplest case as the following minimization problem with eight design variables:

$$\text{Minimize} \quad f(\boldsymbol{x}) = x_1 + x_2 + x_3, \tag{37}$$

subject to

$$g_1(\boldsymbol{x}) = 0.0025(x_4 + x_6) - 1 \leq 0, \tag{38}$$

$$g_2(\boldsymbol{x}) = 0.0025(x_5 + x_7 - x_5) - 1 \leq 0, \tag{39}$$

$$g_3(\boldsymbol{x}) = 0.01(x_8 - x_5) - 1 \leq 0, \tag{40}$$

$$g_4(\boldsymbol{x}) = 833.33252x_4 + 100x_1 - x_1x_6 - 83333.333 \leq 0, \tag{41}$$

$$g_5(\boldsymbol{x}) = 1250x_5 + x_2x_4 - x_2x_7 - 125x_4 \leq 0, \tag{42}$$

$$g_6(\boldsymbol{x}) = x_3x_5 - 2500x_5 - x_3x_8 + 1250000 \leq 0. \tag{43}$$

For example, using CS with $n = 20$ cuckoos, we can easily find the optimal solution for these eight design variables as

$$x^* = (579.3068, 1359.9708, 5109.9705, 182.0177, \\ 295.6012, 217.9823, 286.4165, 395.6012). \tag{44}$$

4.6 Welded Beam Design

The welded beam design is another standard test problem for constrained design optimization [4, 38]. The problem has four design variables: the width w and length L of the welded area, and the depth d and thickness h of the main beam. The objective is to minimize the overall fabrication cost, under the appropriate constraints of shear stress τ, bending stress σ, buckling load P and maximum end deflection δ.

The problem can be written as

$$\text{minimize} \quad f(\boldsymbol{x}) = 1.10471w^2L + 0.04811dh(14.0 + L), \tag{45}$$

subject to

$$\begin{aligned}
g_1(\boldsymbol{x}) &= w - h \le 0,\\
g_2(\boldsymbol{x}) &= \delta(\boldsymbol{x}) - 0.25 \le 0,\\
g_3(\boldsymbol{x}) &= \tau(\boldsymbol{x}) - 13{,}600 \le 0,\\
g_4(\boldsymbol{x}) &= \sigma(\boldsymbol{x}) - 30{,}000 \le 0,\\
g_5(\boldsymbol{x}) &= 0.10471w^2 + 0.04811hd(14+L) - 5.0 \le 0,\\
g_6(\boldsymbol{x}) &= 0.125 - w \le 0,\\
g_7(\boldsymbol{x}) &= 6000 - P(\boldsymbol{x}) \le 0,
\end{aligned} \tag{46}$$

where

$$\begin{aligned}
&\sigma(\boldsymbol{x}) = \frac{504{,}000}{hd^2}, && Q = 6000\left(14 + \frac{L}{2}\right),\\
&D = \frac{1}{2}\sqrt{L^2 + (w+d)^2}, && J = \sqrt{2}wL\left[\frac{L^2}{6} + \frac{(w+d)^2}{2}\right],\\
&\delta = \frac{65{,}856}{30{,}000hd^3}, && \beta = \frac{QD}{J},\\
&\alpha = \frac{6000}{\sqrt{2}wL}, && \tau(\boldsymbol{x}) = \sqrt{\alpha^2 + \frac{\alpha\beta L}{D} + \beta^2},\\
&P = 0.61423 \times 10^6 \frac{dh^3}{6}\left(1 - \frac{d\sqrt{30/48}}{28}\right).
\end{aligned} \tag{47}$$

The simple limits or bounds are $0.1 \le L, d \le 10$ and $0.1 \le w, h \le 2.0$. For example, using both CS and FA, we have obtained the following optimal solution:

$$\begin{aligned}
\boldsymbol{x}_* &= (w, L, d, h)\\
&= (0.20572963978, 3.47048866563, 9.03662391036, 0.20572963979),
\end{aligned} \tag{48}$$

with

$$f(\boldsymbol{x}*)_{\min} = 1.72485230859. \tag{49}$$

This solution is exactly the same as those in the literature [4]

$$f_* = 1.724852 \quad \text{at } (0.205730, 3.470489, 9.036624, 0.205729). \tag{50}$$

We have also solved this problem using BA, and we got exactly the same solution.

4.7 Pressure Vessel Design

Pressure vessels are literally everywhere; some examples are champagne bottles and gas tanks. For a given volume and working pressure, the basic aim of designing a cylindrical vessel is to minimize the total cost. Typically, the design variables are the thickness d_1 of the head, the thickness d_2 of the body, the inner radius r and the length L of the cylindrical section [4, 38]. This is a well-known test problem for optimization and it can be written as

$$\text{minimize} \quad f(\boldsymbol{x}) = 0.6224 d_1 r L + 1.7781 d_2 r^2 + 3.1661 d_1^2 L + 19.84 d_1^2 r, \tag{51}$$

subject to the following constraints:

$$\begin{aligned} g_1(\boldsymbol{x}) &= -d_1 + 0.0193 r \leq 0, \\ g_2(\boldsymbol{x}) &= -d_2 + 0.00954 r \leq 0, \\ g_3(\boldsymbol{x}) &= -\pi r^2 L - \frac{4\pi}{3} r^3 + 1296000 \leq 0, \\ g_4(\boldsymbol{x}) &= L - 240 \leq 0. \end{aligned} \tag{52}$$

The simple bounds are

$$0.0625 \leq d_1, d_2 \leq 99 \times 0.0625, \tag{53}$$

and

$$10.0 \leq r, \ L \leq 200.0. \tag{54}$$

We have used all three algorithms (FA, CS, BA) to solve this problem, and they all found the same solution $f_* \approx 6,059.714$ at

$$\boldsymbol{x}_* \approx (0.8125,\ 0.4375,\ 42.0984,\ 176.6366), \tag{55}$$

which is the same as the one obtained by Cagnina et al. [4]. This means that the lowest price is about \$6,059.71.

4.8 Gearbox Design

Another important benchmark is the design of a speed reducer which is commonly used in many mechanisms such as a gearbox [14]. This problem involves the optimization of seven variables, including the face width, the number of teeth, the diameter of the shaft and others. All variables are continuous within some limits, except x_3 which only takes integer values. We have

$$\begin{aligned} f(\boldsymbol{x}) = {} & 0.7854 x_1 x_2^2 \big(3.3333 x_3^2 + 14.9334 x_3 - 43.0934\big) \\ & - 1.508 x_1 \big(x_6^2 + x_7^2\big) + 7.4777 \big(x_6^3 + x_7^3\big) + 0.7854 \big(x_4 x_6^2 + x_5 x_7^2\big), \end{aligned} \tag{56}$$

subject to

$$g_1(\boldsymbol{x}) = \frac{27}{x_1 x_2^2 x_3} - 1 \leq 0, \tag{57}$$

$$g_2(\boldsymbol{x}) = \frac{397.5}{x_1 x_2^2 x_3^2} - 1 \leq 0, \tag{58}$$

$$g_3(\boldsymbol{x}) = \frac{1.93 x_4^3}{x_2 x_3 x_6^4} - 1 \leq 0, \tag{59}$$

$$g_4(\boldsymbol{x}) = \frac{1.93 x_5^3}{x_2 x_3 x_7^4} - 1 \leq 0, \tag{60}$$

$$g_5(\boldsymbol{x}) = \frac{1.0}{110 x_6^3} \sqrt{\left(\frac{745.0 x_4}{x_2 x_3}\right)^2 + 16.9 \times 10^6} - 1 \leq 0, \tag{61}$$

$$g_6(\boldsymbol{x}) = \frac{1.0}{85 x_7^3} \sqrt{\left(\frac{745.0 x_5}{x_2 x_3}\right)^2 + 157.5 \times 10^6} - 1 \leq 0, \tag{62}$$

$$g_7(\boldsymbol{x}) = \frac{x_2 x_3}{40} - 1 \leq 0, \tag{63}$$

$$g_8(\boldsymbol{x}) = \frac{5 x_2}{x_1} - 1 \leq 0, \tag{64}$$

$$g_9(\boldsymbol{x}) = \frac{x_1}{12 x_2} - 1 \leq 0, \tag{65}$$

$$g_{10}(\boldsymbol{x}) = \frac{1.5 x_6 + 1.9}{x_4} - 1 \leq 0, \tag{66}$$

$$g_{11}(\boldsymbol{x}) = \frac{1.1 x_7 + 1.9}{x_5} - 1 \leq 0, \tag{67}$$

where the simple bounds are $2.6 \leq x_1 \leq 3.6$, $0.7 \leq x_2 \leq 0.8$, $17 \leq x_3 \leq 28$, $7.3 \leq x_4 \leq 8.3$, $7.8 \leq x_5 \leq 8.4$, $2.9 \leq x_6 \leq 3.9$, $5.0 \leq x_7 \leq 5.5$.

The best result in the literature [4] is

$$\boldsymbol{x}_* = (3.5, 0.7, 17, 7.3, 7.8, 3.350214, 5.286683), \tag{68}$$

with $f_{\min} = 2996.348165$.

By using FA and BA as well as CS, we have obtained a new best result:

$$\boldsymbol{x}_* = (3.5, 0.7, 17, 7.3, 7.8, 3.34336449, 5.285351) \tag{69}$$

with the best objective $f_{\min} = 2993.7495888$.

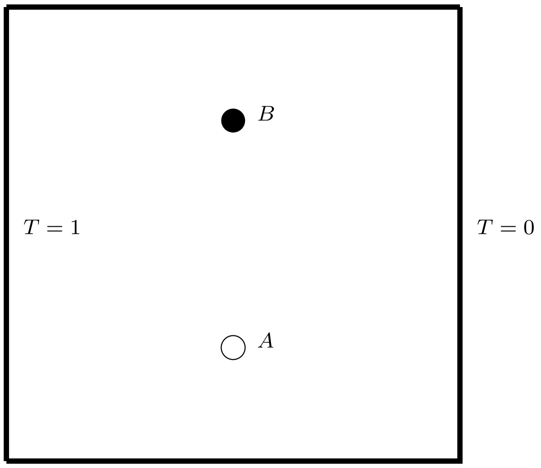

Fig. 1 The topology optimization benchmark to maximize $|T_A - T_B|$

4.9 Simulation-Driven Shape Optimization

Heat management, thus heat transfer modelling, is very important for many electronic applications, especially those using large-scale integrated circuits. In fact, nanoscale heat transfer is a challenging area, and topological optimization for designing nanoscale device is even more challenging [8, 28, 43]. For example, Evgrafov et al. proposed a topology optimization benchmark for a nanoscale heat-conducting system with a size of 150 nm by 150 nm [8]. Heat transfer can occur at many different scales, though smaller scales may be more difficult to control. Now we extend this to a unit of area of 1 mm by 1 mm, and the aim is to distribute two different materials so as to maximize the temperature difference $|T_A - T_B|$ at these two points A and B under the boundary conditions given in Fig. 1 where the top and bottom boundaries are symmetric. Obviously, if there is only one type of material, then $T_A = T_B$ can be expected at the steady state, due to symmetry in the system configuration. However, two types of different materials will change this into a tough shape optimization problem.

Two materials used in the design of the unit area have heat diffusivities of K_1 and K_2, respectively. In addition, $K_1 \gg K_2$. For example, for Si and Mg_2Si, $K_1/K_2 \approx 10$. The domain is continuous, and the objective is to distribute the two materials such that the difference $|T_A - T_B|$ is as large as possible.

By dividing the domain into 40×40 small grids and using CS to search for possible design solutions, an optimal shape and distribution of materials is obtained, as shown in Fig. 2, where Si is shown in light blue and Mg_2Si is shown in red.

For each configuration generated during the search process, the temperature distribution is estimated using the finite difference method by solving the heat conduction equation with varied material conductivities so that the temperature difference at the two fixed points is as large as possible.

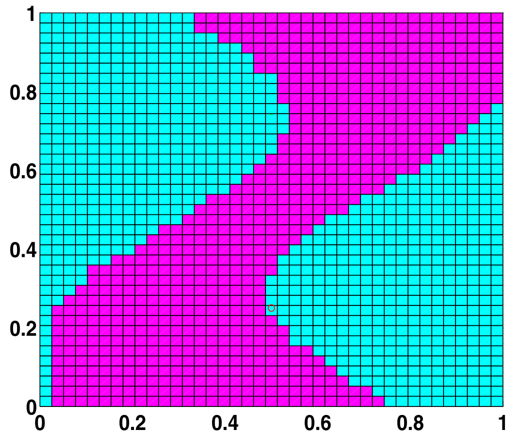

Fig. 2 Optimal topology and distribution of two different materials

5 Challenges and Further Research Topics

Despite the huge success of optimization and extensive applications, many challenging issues must be addressed in the near future. As an optimization process typically involves an optimization algorithm and a simulator, two main issues naturally arise: the efficiency of the algorithm and the efficiency and accuracy of the numerical simulator. Obviously, we try to use the most efficient algorithms available, but the actual efficiency of an algorithm may depend on many factors such as the inner working of the algorithm, the available information (such as objective functions and their derivatives) and implementation details. The associated issue is how to assign the right algorithms to a given problem, which is not easy to solve. In fact, for some highly nonlinear problems, there may not be any efficient algorithm at all. One well-known case is the travelling salesman problem, which is hard in the non-deterministic polynomial-time (NP) sense, that is, NP-hard. There is no efficient algorithm to deal with these types of problems.

The efficiency of a simulator or solver is even more complicated, depending on the actual numerical methods used and the complexity of the problem of interest. Straightforward optimization of a given objective function is not always practical. If the objective function comes from a computer simulation, it may be computationally expensive, noisy or non-differentiable. In such cases, a surrogate-based optimization algorithm may be a very useful alternative [18]. The surrogate model can be typically constructed from the sampled data of the original objective function; however, this surrogate model should reasonably be cheap, smooth and easy to optimize, and yet accurate enough so that it can produce a good prediction of the function's optimum [17]. A challenging issue is how to construct good surrogate models that have good fidelity and yet can save sufficient computational efforts.

On the other hand, it may be no exaggeration to say that metaheuristics have had great success in solving various tough optimization problems. However, there are many important questions which remain unanswered.

First, an important issue to be addressed in any metaheuristic algorithm is how to provide a good balance between local intensification and global diversification [29, 30]. At present, different algorithms uses various techniques and mechanisms with various parameters to control this, which may be far from optimal. Is there any optimal way to achieve this balance? If yes, how? If not, what is the best we can achieve?

Second, it is still only partly understood why different components of heuristics and metaheuristics interact in a coherent and balanced way so that they produce efficient algorithms which converge under the given conditions. For example, why does a balanced combination of randomization and a deterministic component lead to a much more efficient algorithm (than a purely deterministic and/or a purely random algorithm)? How can we measure or test if a balance is reached? How can we prove that the use of memory can significantly increase the search efficiency of an algorithm? Under what conditions?

Finally, most applications in the current literature have been tested against toy problems or small-scale benchmarks with a few design variables or at most problems with several dozen variables. In real-world applications, many design problems in engineering, business and industry may involve thousands or even millions of variables. We have not seen case studies for such large-scale problems in the literature. A crucial issue is that there is no indication that the methodology that works for such toy benchmarks will work equally well for large-scale problems. Apart from the difference in the problem size, there may be other fundamental differences for large-scale problems, and thus the methodology could be very different [33].

Such challenges still remain unresolved, both in theory and in practice. These important issues also provide golden opportunities for researchers to rethink the existing methodology and approaches, perhaps more fundamentally. We can expect that some significant progress will be made in the next ten years.

References

1. Apostolopoulos, T., Vlachos, A.: Application of the firefly algorithm for solving the economic emissions load dispatch problem. Int. J. Comb. **2011**, 523806 (2011). http://www.hindawi.com/journals/ijct/2011/523806.html
2. Arora, J.: Introduction to Optimum Design. McGraw-Hill, New York (1989)
3. Ashby, W.R.: Principles of the self-organizing system. In: Von Foerster, H., Zopf, G.W. Jr. (eds.) Principles of Self-organization: Transactions of the University of Illinois Symposium, pp. 255–278. Pergamon Press, London (1962)
4. Cagnina, L.C., Esquivel, S.C., Coello, C.A.: Solving engineering optimization problems with the simple constrained particle swarm optimizer. Informatica **32**, 319–326 (2008)
5. Chickermane, H., Gea, H.C.: Structural optimization using a new local approximation method. Int. J. Numer. Methods Eng. **39**, 829–846 (1996)
6. Deb, K.: Optimization for Engineering Design. Prentice-Hall, New Delhi (1995)
7. Durgun, I., Yildiz, A.R.: Structural design optimization of vehicle components using cuckoo search algorithm. Mater. Test. **3**, 185–188 (2012)
8. Evgrafov, A., Maute, K., Yang, R.G., Dunn, M.L.: Topology optimization for nano-scale heat transfer. Int. J. Numer. Methods Eng. **77**, 285–300 (2009)

9. Fleury, C., Braibant, V.: Structural optimization: a new dual method using mixed variables. Int. J. Numer. Methods Eng. **23**, 409–428 (1986)
10. Gandomi, A.H., Yang, X.S.: Benchmark problems in structural optimization. In: Koziel, S., Yang, X.S. (eds.) Computational Optimization, Methods and Algorithms. Study in Computational Intelligence, SCI, vol. 356, pp. 259–281. Springer, Berlin (2011)
11. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Cuckoo search algorithm: a meteheuristic approach to solve structural optimization problems. Eng. Comput. doi:10.1007/s00366-011-0241-y (2011). Online first 29 July 2011
12. Gandomi, A.H., Yang, X.S., Talatahari, S., Deb, S.: Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization. Comput. Math. Appl. **63**(1), 191–200 (2012)
13. Gill, P.E., Murray, W., Wright, M.H.: Practical Optimization. Academic Press Inc., London (1981)
14. Golinski, J.: An adaptive optimization system applied to machine synthesis. Mech. Mach. Theory **8**(4), 419–436 (1973)
15. Keller, E.F.: Organisms, machines, and thunderstorms: a history of self-organization, part two. Complexity, emergence, and stable attractors. Hist. Stud. Nat. Sci. **39**, 1–31 (2009)
16. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proc. of IEEE International Conference on Neural Networks, Piscataway, NJ, pp. 1942–1948 (1995)
17. Koziel, S., Yang, X.S.: Computational Optimization and Applications in Engineering and Industry. Springer, Berlin (2011)
18. Koziel, S., Bandler, J.W., Madsen, K.: Quality assessment of coarse models and surrogates for space mapping optimization. Optim. Eng. **9**(4), 375–391 (2008)
19. Koziel, S., Yang, X.S., Zhang, Q.J.: Simulation-Driven Design Optimization and Modeling for Microwave Engineering. Imperial College Press, London (2013)
20. Leifsson, L., Koziel, S.: Multi-fidelity design optimization of transonic airfoils using physics-based surrogate modeling and shape-preserving response prediction. J. Comput. Sci. **1**(2), 98–106 (2010)
21. Liebman, J.S., Khachaturian, N., Chanaratna, V.: Discrete structural optimization. J. Struct. Div. **107**(ST11), 2177–2197 (1981)
22. Nowcki, H.: Optimization in pre-contract ship design. In: Fujita, Y., Lind, K., Williams, T.J. (eds.) Computer Applications in the Automation of Shipyard Operation and Ship Design, vol. 2, pp. 327–338. North-Holland, Elsevier, New York (1974)
23. Prigogine, I., Nicolois, G.: On symmetry-breaking instabilities in dissipative systems. J. Chem. Phys. **46**, 3542–3550 (1967)
24. Ravindran, A., Ragsdell, K.M., Reklaitis, G.V.: Engineering Optimization: Methods and Applications, 2nd edn. Wiley, Hoboken (2006)
25. Sayadi, M.K., Ramezanian, R., Ghaffari-Nasab, N.: A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. Int. J. Ind. Eng. Comput. **1**, 1–10 (2010)
26. Turing, A.M.: The chemical basis of morphogenesis. Phys. Today **237**, 37–72 (1952)
27. Walton, S., Hassan, O., Morgan, K., Brown, M.R.: Modified cuckoo search: a new gradient free optimization algorithm. Chaos Solitons Fractals **44**(9), 710–718 (2011)
28. Yang, X.S.: Modelling heat transfer of carbon nanotubes. Model. Simul. Mater. Sci. Eng. **13**, 893–902 (2005)
29. Yang, X.S.: Introduction to Computational Mathematics. World Scientific Publishing, Singapore (2008)
30. Yang, X.S.: Engineering Optimization: An Introduction with Metaheuristic Applications. Wiley, New York (2010)
31. Yang, X.S.: Bat algorithm for multi-objective optimisation. Int. J. Bio-Inspir. Comput. **3**(5), 267–274 (2011)
32. Yang, X.S.: Review of meta-heuristics and generalised evolutionary walk algorithm. Int. J. Bio-Inspir. Comput. **3**(2), 77–84 (2011)

33. Yang, X.S.: Nature-inspired metaheuristic algorithms: success and new challenges. J. Comput. Eng. Inf. Technol. **1**, 1–3 (2012). doi:10.4172/2324-9307.1000e101
34. Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: Proc. of World Congress on Nature & Biologically Inspired Computing (NaBic 2009), pp. 210–214. IEEE Publications, New York (2009)
35. Yang, X.S., Deb, S.: Engineering optimization by cuckoo search. Int. J. Math. Model. Numer. Optim. **1**(4), 330–343 (2010)
36. Yang, X.S., Deb, S.: Eagle strategy using Lévy walk and firefly algorithms for stochastic optimization. In: Cruz, C., González, R.J., Krasnogor, N., Terrazas, G. (eds.) Nature Inspired Cooperative Strategies for Optimization (NICSO2010). Studies in Computational Intelligence (SCI), vol. 284, pp. 101–111. Springer, New York (2010)
37. Yang, X.S., Deb, S.: Multiobjective cuckoo search for design optimization. Comput. Oper. Res. **40**(6), 1616–1624 (2013). doi:10.1016/j.cor.2011.09.026
38. Yang, X.S., Gandomi, A.H.: Bat algorithm: a novel approach for global engineering optimization. Eng. Comput. **29**(5), 464–483 (2012)
39. Yang, X.S., Koziel, S.: Computational optimization, modelling and simulation—a paradigm shift. Proc. Comput. Sci. **1**(1), 1291–1294 (2010)
40. Yang, X.S.: A new metaheuristic bat-inspired algorithm. In: Gonzalez, J.R., et al. (eds.) Nature-Inspired Cooperative Strategies for Optimization (NICSO 2010), vol. 284, pp. 65–74. Springer, Berlin (2010)
41. Yang, X.S., Deb, S., Fong, S.: Accelerated particle swarm optimization and support vector machine for business optimization and applications. In: Networked Digital Technologies 2011. Communications in Computer and Information Science, vol. 136, pp. 53–66 (2011)
42. Yang, X.S., Hossein, S.S., Gandomi, A.H.: Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. Appl. Soft Comput. **12**(3), 1180–1186 (2012)
43. Zhirnov, V.V., Cavin, R.K., Hutchby, J.A., Bourianoff, G.I.: Limits to binary logic switch scaling—a gedanken model. Proc. IEEE **91**, 1934–1939 (2003)