Metaheuristic Optimization: Nature-Inspired Algorithms and Applications

Xin-She Yang

Abstract. Turing's pioneer work in heuristic search has inspired many generations of research in heuristic algorithms. In the last two decades, metaheuristic algorithms have attracted strong attention in scientific communities with significant developments, especially in areas concerning swarm intelligence based algorithms. In this work, we will briefly review some of the important achievements in metaheuristics, and we will also discuss key implications in applications and topics for further research.

1 Introduction

Alan Turing pioneered many areas from artificial intelligence to pattern formation. Turing was also the first to use heuristic algorithms during the second World War for his code-breaking work at Bletchley Park. Turing called his search method *heuristic search*, as it could be expected it worked most of time, but there was no guarantee to find the true or correct solution, but it was a tremendous success [6]. In 1945, Turing was recruited to the National Physical Laboratory (NPL), UK where he set out his design for the Automatic Computing Engine (ACE). In an NPL report on *Intelligent machinery* in 1948 [33], he outlined his innovative ideas of machine intelligence and learning, neural networks and evolutionary algorithms [38]. In this chapter, we will review the latest development in metaheuristic methods, especially swarm intelligence based algorithms.

2 Metaheuristics

Metaheuristic algorithms, especially those based on swarm intelligence, form an important part of contemporary global optimization algorithms [21, 40, 2, 3, 4, 24, 26]

Xin-She Yang
Mathematics & Scientific Computing, National Physical Laboratory,
Teddington TW11 0LW, UK

X.-S. Yang (Ed.): Artif. Intell., Evol. Comput. and Metaheuristics, SCI 427, pp. 405–420.
springerlink.com

Good examples are simulated annealing [22], particle swarm optimization [21] and firefly algorithm [40, 41]. They work remarkably efficiently and have many advantages over traditional, deterministic methods and algorithms, and thus they have been applied in almost all area of science, engineering and industry [15, 11, 42, 43, 51].

Despite such a huge success in applications, mathematical analysis of algorithms remains limited and many open problems are still un-resolved. There are three challenging areas for algorithm analysis: complexity, convergence and no-free-lunch theory. Complexity analysis of traditional algorithms such as quick sort and matrix inverse are well-established, as these algorithms are deterministic. In contrast, complexity analysis of metaheuristics remains a challenging task, partly due to the stochastic nature of these algorithms. However, good results do exist, concerning randomization search techniques [2].

Convergence analysis is another challenging area. One of the main difficulties concerning the convergence analysis of metaheuristic algorithms is that no generic framework exists, though substantial studies have been carried out using dynamic systems and Markov processes. However, convergence analysis still remains one of the active research areas with many encouraging results [5, 17].

Along the mathematical analysis of optimization algorithms, another equally challenging, and yet fruitful area is the theory on algorithm performance and comparison, leading to a wide range of no-free-lunch (NFL) theorems [36, 19]. While in well-posed cases of optimization where its functional space forms finite domains, NFL theorems do hold; however, free lunches are possible in continuous domains [2, 37, 34].

In this chapter, we intend to provide a state-of-the-art review of widely used metaheuristic algorithms. We will also briefly highlights some of the convergence studies. Based on these studies, we will summarize and propose a series of recommendations for further research.

3 Metaheuristic Algorithms

There are more than a dozen of swarm-based algorithms using the so-called swarm intelligence. For a detailed introduction, please refer to [43, 26]. In this section, we will focus on the main chararcteristics and the ways that each algorithm generate new solutions, and we will not discuss each algorithm in details. Obviously, not all metaheuristic algorithms are swarm-inspired, for example, harmony search is not a swarm-intelligence-based algorithm. Similarly, genetic algorithms are bio-inspired, or more generally nature-inspired, but they are not based on swarm intelligence. Here we will introduce a few population-based metaheuristic algorithms which are widely used or active research topics. Interested readers can follow the references listed at the end of this chapter and also refer to other chapters of this book.

3.1 Ant Algorithms

Ant algorithms, especially the ant colony optimization [10], mimic the foraging behaviour of social ants. Primarily, it uses pheromone as a chemical messenger and the pheromone concentration as the indicator of quality solutions to a problem of interest. As the solution is often linked with the pheromone concentration, the search algorithms often produce routes and paths marked by the higher pheromone concentrations, and therefore, ants-based algorithms are particular suitable for discrete optimization problems.

The movement of an ant is controlled by pheromone which will evaporate over time. Without such time-dependent evaporation, the algorithms will lead to premature convergence to the (often wrong) solutions. With proper pheromone evaporation, they usually behave very well.

There are two important issues here: the probability of choosing a route, and the evaporation rate of pheromone. There are a few ways of solving these problems, although it is still an area of active research. Here we introduce the current best method.

For a network routing problem, the probability of ants at a particular node i to choose the route from node i to node j is given by

$$p_{ij} = \frac{\phi_{ij}^{\alpha} d_{ij}^{\beta}}{\sum_{i,j=1}^{n} \phi_{ij}^{\alpha} d_{ij}^{\beta}}, \tag{1}$$

where $\alpha > 0$ and $\beta > 0$ are the influence parameters, and their typical values are $\alpha \approx \beta \approx 2$. ϕ_{ij} is the pheromone concentration on the route between i and j, and d_{ij} the desirability of the same route. Some *a priori* knowledge about the route such as the distance s_{ij} is often used so that $d_{ij} \propto 1/s_{ij}$, which implies that shorter routes will be selected due to their shorter traveling time, and thus the pheromone concentrations on these routes are higher. This is because the traveling time is shorter, and thus the less amount of the pheromone has been evaporated during this period.

3.2 Bee Algorithms

Bees-inspired algorithms are more diverse, and some use pheromone and most do not. Almost all bee algorithms are inspired by the foraging behaviour of honey bees in nature. Interesting characteristics such as waggle dance, polarization and nectar maximization are often used to simulate the allocation of the foraging bees along flower patches and thus different search regions in the search space. For a more comprehensive review, please refer to [26, 40].

In the honeybee-based algorithms, forager bees are allocated to different food sources (or flower patches) so as to maximize the total nectar intake. The colony has to 'optimize' the overall efficiency of nectar collection, the allocation of the bees is thus depending on many factors such as the nectar richness and the proximity to the hive [23, 39, 20, 27].

The virtual bee algorithm (VBA), developed by Xin-She Yang in 2005, is an optimization algorithm specially formulated for solving both discrete and continuous problems [39]. On the other hand, the artificial bee colony (ABC) optimization algorithm was first developed by D. Karaboga in 2005. In the ABC algorithm, the bees in a colony are divided into three groups: employed bees (forager bees), onlooker bees (observer bees) and scouts. For each food source, there is only one employed bee. That is to say, the number of employed bees is equal to the number of food sources. The employed bee of an discarded food site is forced to become a scout for searching new food sources randomly. Employed bees share information with the onlooker bees in a hive so that onlooker bees can choose a food source to forage. Unlike the honey bee algorithm which has two groups of the bees (forager bees and observer bees), bees in ABC are more specialized [1, 20].

Similar to the ants-based algorithms, bee algorithms are also very flexible in dealing with discrete optimization problems. Combinatorial optimization such as routing and optimal paths has been successfully solved by ant and bee algorithms. In principle, they can solve both continuous optimization and discrete optimization problems; however, they should not be the first choice for continuous problems.

3.3 Genetic Algorithms

Genetic algorithms are by far the most widely used [18], and one of the reasons is that the GA appeared as early as in the 1960s, based on the evolutionary features of biological systems. Genetic operators such as crossover and mutation are very powerful in generating diverse solutions or search points, while elitism, adaptation and selection of the fittest help to ensure the proper convergence of genetic algorithms.

Parameter choices are also important, but there are many parametric studies in the literature, and the overall literature of genetic algorithms is vast. In essence, the crossover should be more frequent with the highest probability, often above 0.7 to 0.95. On the other hand, mutation rate should be very low, because if the mutation rate is too high, the solutions generated are too diverse, and thus makes it difficult for the search process to converge properly. Therefore, mutation rate is typically 0.1 to 0.01.

Genetic algorithms have many variants and often combined with other algorithms to form hybrid algorithms, and encode and decoding can be binary, real or even imaginary. Interested readers can refer to the recent books, for example, Goldberg [16] and other relevant books listed in the bibliography.

3.4 Differential Evolution

Differential evolution (DE) can be considered as a vectorized and improved genetic algorithm, though DE has its own unique mutation operator and crossover operation [32]. Mutation is carried out by the donor vector based on the difference of two randomly chosen solution vectors; in this sense, its mutation is like an exploration move in pattern search. Alternatively, we can consider it as a multi-site mutation vector,

based on genetic algorithms. Crossover is more elaborate which can be performed either in a binomial or exponential manner. There are many variants of DE and they are often combined with other algorithms to form efficient hybrid algorithms [28]. DE can also be combined with other methods such as eagle strategy to get even better results [48].

3.5 Particle Swarm Optimization

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart in 1995 [21], based on the swarm behaviour such as fish and bird schooling in nature. Since then, PSO has generated much wider interests, and forms an exciting, ever-expanding research subject, called swarm intelligence. PSO has been applied to almost every area in optimization, computational intelligence, and design/scheduling applications.

The movement of a swarming particle consists of two major components: a stochastic component and a deterministic component. Each particle is attracted toward the position of the current global best g^* and its own best location x_i^* in history, while at the same time it has a tendency to move randomly.

Let x_i and v_i be the position vector and velocity for particle i, respectively. The new velocity and location updating formulas are determined by

$$v_i^{t+1} = v_i^t + \alpha\varepsilon_1[g^* - x_i^t] + \beta\varepsilon_2[x_i^* - x_i^t]. \tag{2}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}, \tag{3}$$

where ε_1 and ε_2 are two random vectors, and each entry taking the values between 0 and 1. The parameters α and β are the learning parameters or acceleration constants, which can typically be taken as, say, $\alpha \approx \beta \approx 2$.

There are at least two dozen PSO variants which extend the standard PSO algorithm, and the most noticeable improvement is probably to use inertia function $\theta(t)$ so that v_i^t is replaced by $\theta(t)v_i^t$ where $\theta \in [0,1]$. This is equivalent to introducing a virtual mass to stabilize the motion of the particles, and thus the algorithm is expected to converge more quickly.

3.6 Firefly Algorithm

Firefly Algorithm (FA) was developed by Xin-She Yang at Cambridge University [40, 41], which was based on the flashing patterns and behaviour of fireflies. In essence, each firefly will be attracted to brighter ones, while at the same time, it explores and searches for prey randomly. In addition, the brightness of a firefly is determined by the landscape of the objective function.

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

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}(x_j^t - x_i^t) + \alpha_t\, \varepsilon_i^t, \tag{4}$$

where the second term is due to the attraction. The third term is randomization with α_t being the randomization parameter, and ε_i^t is a vector of random numbers drawn from a Gaussian distribution or uniform distribution. Here is $\beta_0 \in [0,1]$ is the attractiveness at $r = 0$, and $r_{ij} = ||x_i^t - x_j^t||$ is the Cartesian distance. For other problems such as scheduling, any measure that can effectively characterize the quantities of interest in the optimization problem can be used as the 'distance' r. For most implementations, we can take $\beta_0 = 1$, $\alpha = O(1)$ and $\gamma = O(1)$.

Ideally, the randomization parameter α_t should be monotonically reduced gradually during iterations. A simple scheme is to use

$$\alpha_t = \alpha_0 \delta^t, \quad \delta \in (0,1), \tag{5}$$

where α_0 is the initial randomness, while δ is a randomness reduction factor similar to that used in a cooling schedule in simulated annealing. It is worth pointing out that (4) is essentially a random walk biased towards the brighter fireflies. If $\beta_0 = 0$, it becomes a simple random walk. Furthermore, the randomization term can easily be extended to other distributions such as Lévy flights. A basic implementation can be obtained from this link.[1] High nonlinear and non-convex global optimization problems can be solved using firefly algorithm efficiently [14, 49]).

3.7 Harmony Search

Harmony Search (HS) is a music-inspired metaheuristic algorithm and it was first developed by Z. W. Geem et al. in 2001 and a recent summary can be found at Geem [12]. Harmony search has three components: usage of harmony memory, pitch adjusting, and randomization.

The usage of harmony memory is similar to choose the best fit individuals in the genetic algorithms, while pitch adjustment is similar to the mutation operator in genetic algorithms. Further more, randomization is used to increase the diversity of the solutions.

3.8 Bat Algorithm

Bat algorithm is a relatively new metaheuristic, developed by Xin-She Yang in 2010 [44]. It 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 the surrounding objects. Their pulses vary in 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. Their signal bandwidth varies depends on the species, and often increased by using more harmonics.

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

Inside the bat algorithm, it uses three idealized rules:

1. All bats use echolocation to sense distance, and they also 'know' the difference between food/prey and background barriers in some magical way;
2. Bats fly randomly with velocity v_i at position 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;
3. Although the loudness can vary in many ways, we assume that the loudness varies from a large (positive) A_0 to a minimum constant value $A_{\min}$.

BA has been extended to multiobjective bat algorithm (MOBA) by [47], and preliminary results suggested that it is very efficient.

3.9 Cuckoo Search

Cuckoo search (CS) is one of the latest nature-inspired metaheuristic algorithms, developed in 2009 by Xin-She Yang and Suash Deb [45, 46]. CS is based on the brood parasitism of some cuckoo species. In addition, this algorithm is enhanced by the so-called Lévy flights, rather than by simple isotropic random walks. This algorithm was inspired by the aggressive reproduction strategy of some cuckoo species such as the *ani* and *Guira* cuckoos. These 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).

In the standard cuckoo search, the following three idealized rules are used:

- 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 are replaced by new nests (with new random solutions). Recent studies suggest that cuckoo search can outperform particle swarm optimization and other algorithms [46].

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

$$x_i^{t+1} = x_i^t + s \otimes H(p_a - \varepsilon) \otimes (x_j^t - x_k^t), \tag{6}$$

where x_j^t and 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

$$x_i^{t+1} = x_i^t + \alpha L(s,\lambda), \quad L(s,\lambda) = \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi}\frac{1}{s^{1+\lambda}}, \quad (s \gg s_0 > 0). \quad (7)$$

A vectorized implementation can be obtained from this link here.[2]

The literature on cuckoo search is expanding rapidly. Interestingly, cuckoo search was originally published in 2009 and our matlab program was in the public domain in 2010, while some authors later in 2011 used a different name, cuckoo optimization algorithm, to essentially talk about the same inspiration from cuckoo behaviour.

There have been a lot of attention and recent studies using cuckoo search with diverse range of applications [13, 35, 50]. Walton et al. improved the algorithm by formulating a modified cuckoo search algorithm [35], while Yang and Deb extended it to multiobjective optimization problems [50]. Durgun and Yildiz applied it to structural design optimization [9].

There are other metaheuristic algorithms which have not been introduced here, and interested readers can refer to more advanced literature [43, 26].

4 Artificial Neural Networks

Artificial neural networks in essence belong to optimization algorithms, though they may work in a different context.

The basic mathematical model of an artificial neuron was first proposed by W. McCulloch and W. Pitts in 1943, and this fundamental model is referred to as the McCulloch-Pitts model. Other models and neural networks are based on it.

An artificial neuron with n inputs or impulses and an output y_k will be activated if the signal strength reaches a certain threshold θ. Each input has a corresponding weight w_i. The output of this neuron is given by

$$y_l = \Phi\Big(\sum_{i=1}^{n} w_i u_i\Big), \quad (8)$$

where the weighted sum $\xi = \sum_{i=1}^{n} w_i u_i$ is the total signal strength, and Φ is the so-called activation function, which can be taken as a step function. That is, we have

$$\Phi(\xi) = \begin{cases} 1 \text{ if } \xi \geq \theta, \\ 0 \text{ if } \xi < \theta. \end{cases} \quad (9)$$

We can see that the output is only activated to a non-zero value if the overall signal strength is greater than the threshold θ.

The step function has discontinuity, sometimes, it is easier to use a nonlinear, smooth function, called a Sigmoid function

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

$$S(\xi) = \frac{1}{1+e^{-\xi}}, \tag{10}$$

which approaches 1 as $U \to \infty$, and becomes 0 as $U \to -\infty$. An interesting property of this function is

$$S'(\xi) = S(\xi)[1 - S(\xi)]. \tag{11}$$

4.1 Neural Networks

A single neuron can only perform a simple task – on or off. Complex functions can be designed and performed using a network of interconnecting neurons or perceptrons. The structure of a network can be complicated, and one of the most widely used is to arrange them in a layered structure, with an input layer, an output layer, and one or more hidden layer (see Fig. 1). The connection strength between two neurons is represented by its corresponding weight. Some artificial neural networks (ANNs) can perform complex tasks, and can simulate complex mathematical models, even if there is no explicit functional form mathematically. Neural networks have developed over last few decades and have been applied in almost all areas of science and engineering.

The construction of a neural network involves the estimation of the suitable weights of a network system with some training/known data sets. The task of the training is to find the suitable weights w_{ij} so that the neural networks not only can best-fit the known data, but also can predict outputs for new inputs. A good artificial neural network should be able to minimize both errors simultaneously – the fitting/learning errors and the prediction errors.

The errors can be defined as the difference between the calculated (or predicated) output o_k and real output y_k for all output neurons in the least-square sense

$$E = \frac{1}{2}\sum_{k=1}^{n_o}(o_k - y_k)^2. \tag{12}$$

Here the output o_k is a function of inputs/activations and weights. In order to minimize this error, we can use the standard minimization techniques to find the solutions of the weights.

A simple and yet efficient technique is the steepest descent method. For any initial random weights, the weight increment for w_{hk} is

$$\Delta w_{hk} = -\eta\frac{\partial E}{\partial w_{hk}} = -\eta\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial w_{hk}}, \tag{13}$$

where η is the learning rate. Typically, we can choose $\eta = 1$.

From

$$S_k = \sum_{h=1}^{m} w_{hk}o_h, \quad (k = 1,2,...,n_o), \tag{14}$$

and

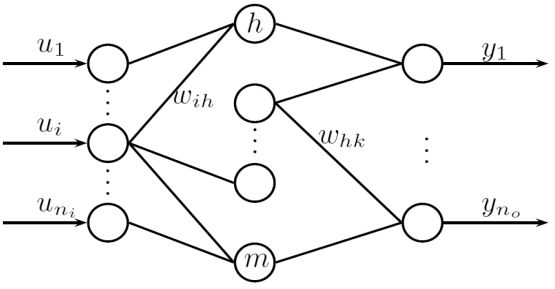

Fig. 1 Schematic representation of a three-layer neural networks with n_i inputs, m hidden nodes and n_o outputs.

$$o_k = f(S_k) = \frac{1}{1+e^{-S_k}}, \tag{15}$$

we have

$$f' = f(1-f), \tag{16}$$

$$\frac{\partial o_k}{\partial w_{hk}} = \frac{\partial o_k}{\partial S_k}\frac{\partial S_k}{\partial w_{hk}} = o_k(1-o_k)o_h, \tag{17}$$

and

$$\frac{\partial E}{\partial o_k} = (o_k - y_k). \tag{18}$$

Therefore, we have

$$\Delta w_{hk} = -\eta\,\delta_k o_h, \quad \delta_k = o_k(1-o_k)(o_k-y_k). \tag{19}$$

4.2 Back Propagation Algorithm

There are many ways of calculating weights by supervised learning. One of the simplest and widely used methods is to use the back propagation algorithm for training neural networks, often called back propagation neural networks (BPNNs).

The basic idea is to start from the output layer and propagate backwards so as to estimate and update the weights.

From any initial random weighting matrices w_{ih} (for connecting the input nodes to the hidden layer) and w_{hk} (for connecting the hidden layer to the output nodes), we can calculate the outputs of the hidden layer o_h

$$o_h = \frac{1}{1+\exp[-\sum_{i=1}^{n_i} w_{ih}u_i]}, \quad (h = 1,2,...,m), \tag{20}$$

and the outputs for the output nodes

$$o_k = \frac{1}{1+\exp[-\sum_{h=1}^{m} w_{hk}o_h]}, \quad (k = 1,2,...,n_o). \tag{21}$$

The errors for the output nodes are given by

$$\delta_k = o_k(1-o_k)(y_k - o_k), \quad (k = 1,2,...,n_o), \tag{22}$$

where $y_k(k = 1,2,...,n_o)$ are the data (real outputs) for the inputs $u_i(i = 1,2,...,n_i)$. Similarly, the errors for the hidden nodes can be written as

$$\delta_h = o_h(1-o_h)\sum_{k=1}^{n_o} w_{hk}\delta_k, \quad (h = 1,2,...,m). \tag{23}$$

The updating formulae for weights at iteration t are

$$w_{hk}^{t+1} = w_{hk}^{t} + \eta\,\delta_k o_h, \tag{24}$$

and

$$w_{ih}^{t+1} = w_{ih}^{t} + \eta\,\delta_h u_i, \tag{25}$$

where $0 < \eta \leq 1$ is the learning rate.

Here we can see that the weight increments are

$$\Delta w_{ih} = \eta\,\delta_h u_i, \tag{26}$$

with similar updating formulae for w_{hk}. An improved version is to use the so-called weight momentum α to increase the learning efficiency

$$\Delta w_{ih} = \eta\,\delta_h u_i + \alpha w_{ih}(\tau - 1), \tag{27}$$

where τ is an extra parameter.

5 Characteristics of Metaheuristics

Metaheuristics can be considered as an efficient way to produce acceptable solutions by trial and error to a complex problem in a reasonably practical time. The complexity of the problem of interest makes it impossible to search every possible solution or combination, the aim is to find good feasible solution in an acceptable timescale. There is no guarantee that the best solutions can be found, and we even do not know whether an algorithm will work and why if it does work. The idea is to have an efficient but practical algorithm that will work most the time and is able to produce good quality solutions. Among the found quality solutions, it is expected some of them are nearly optimal, though there is often no guarantee for such optimality.

The main components of any metaheuristic algorithms are: intensification and diversification, or exploitation and exploration [4, 40, 43]. Diversification means to generate diverse solutions so as to explore the search space on the global scale, while intensification means to focus on the search in a local region by exploiting the information that a current good solution is found in this region. This is in combination with the selection of the best solutions.

As seen earlier, an important component in swarm intelligence and modern metaheuristics is randomization, which enables an algorithm to have the ability to jump out of any local optimum so as to search globally. Randomization can also be used for local search around the current best if steps are limited to a local region. When the steps are large, randomization can explore the search space on a global scale. Fine-tuning the randomness and balancing local search and global search are crucially important in controlling the performance of any metaheuristic algorithm.

Randomization techniques can be a very simple method using uniform distributions and/or Gaussian distributions, or more complex methods as those used in Monte Carlo simulations. They can also be more elaborate, from Brownian random walks to Lévy flights.

6 No-Free-Lunch Theorems

The seminal paper by Wolpert and Mcready in 1997 essentially proposed a framework for performance comparison of optimization algorithms [36], using a combination of Bayesian statistics and Markov random field theories. Let us sketch Wolpert and Macready's original idea. Assuming that the search space is finite (though quite large), thus the space of possible objective values is also finite. This means that objective function is simply a mapping $f : \mathscr{X} \mapsto \mathscr{Y}$, with $\mathscr{F} = \mathscr{Y}^{\mathscr{X}}$ as the space of all possible problems under permutation.

As an algorithm tends to produce a series of points or solutions in the search space, it is further assumed that these points are distinct. That is, for k iterations, k distinct visited points forms a time-ordered set

$$\Omega_k = \left\{ \left(\Omega_k^x(1), \Omega_k^y(1) \right), ..., \left(\Omega_k^x(k), \Omega_k^y(k) \right) \right\}. \tag{28}$$

There are many ways to define a performance measure, though a good measure still remains debatable [30]. Such a measure can depend on the number of iteration k, the algorithm a and the actual cost function f, which can be denoted by $P(\Omega_k^y \| f, k, a)$. Here we follow the notation style in the seminal paper by Wolpert and Mcready (1997). For any pair of algorithms a and b, the NFL theorem states

$$\sum_f P(\Omega_k^y | f, k, a) = \sum_f P(\Omega_k^y | f, k, b). \tag{29}$$

In other words, any algorithm is as good (bad) as a random search, when the performance is averaged *o*ver all possible functions.

Along many relevant assumptions in proving the NFL theorems, two fundamental assumptions are: finite states of the search space (and thus the objective values), and the non-revisiting time-ordered sets.

The first assumption is a good approximation to many problems, especially in finite-digit approximations. However, there is mathematical difference in countable finite, and countable infinite. Therefore, the results for finite states/domains may not directly applicable to infinite domains. Furthermore, as continuous problem are

uncountable, NFL results for finite domains will usually not hold for continuous domains [2].

The second assumption on non-revisiting iterative sequence is often considered as an over-simplification, as almost all metaheuristic algorithms are revisiting in practice, some points visited before will possibly be re-visited again in the future. The only possible exception is the Tabu algorithm with a very long Tabu list [15]. Therefore, results for non-revisiting time-ordered iterations may not be true for the cases of revisiting cases, because the revisiting iterations break an important assumption of 'closed under permutation' (c.u.p) required for proving the NFL theorems [25].

Furthermore, optimization problems do not necessarily concern the whole set of all possible functions/problems, and it is often sufficient to consider a subset of problems. It is worth pointing out active studies have carried out in constructing algorithms that can work best on specific subsets of optimization problems, in fact, NFL theorems do not hold in this case [8].

These theorems are vigorous and thus have important theoretical values. However, their practical implications are a different issue. In fact, it may not be so important in practice anyway, we will discuss this in a later section.

7 Search for Free Lunches

The validity of NFL theorems largely depends on the validity of their fundamental assumptions. However, whether these assumptions are valid in practice is another question. Often, these assumptions are too stringent, and thus free lunches are possible.

One of the assumptions is the non-revisiting nature of the k distinct points which form a time-ordered set. For revisiting points as they do occur in practice in real-world optimization algorithms, the 'closed under permutation' does not hold, which renders NFL theorems invalid [29, 25, 31]. This means free lunches do exist in practical applications.

Another basic assumption is the finiteness of the domains. For continuous domains, Auger and Teytaud in 2010 have proven that the NFL theorem does not hold [2], and therefore they concluded that 'continuous free lunches exist'. Indeed, some algorithms are better than others [7]. For example, for a 2D sphere function, they demonstrated that an efficient algorithm only needs 4 iterations/steps to reach the global minimum.

No-free-lunch theorems may be of theoretical importance, and they can also have important implications for algorithm development in practice, though not everyone agrees the real importance of these implications.

There are three kinds of opinions concerning the implications. The first group may simply ignore these theorems, as they argue that the assumptions are too stringent, and the performance measures based on average overall functions are irrelevant in practice. Therefore, no practical importance can be inferred, and research just carries on.

The second kind is that NFL theorems can be true, and they can accept that the fact there is no universally efficient algorithm. But in practice some algorithms do performance better than others for a specific problem or a subset of problems. Research effort should focus on finding the right algorithms for the right type of problem. Problem-specific knowledge is always helpful to find the right algorithm(s).

The third kind of opinion is that NFL theorems are not true for other types of problems such as continuous problems and NP-hard problems. Theoretical work concerns more elaborate studies on extending NFL theorems to other cases or on finding free lunches [2]. On the other hand, algorithm development continues to design better algorithms which can work for a wider range of problems, not necessarily all types of problems. As we have seen from the above analysis, free lunches do exist, and better algorithms can be designed for a specific subset of problems [41, 46].

Thus, free lunch or no free lunch is not just a simple question, it has important and yet practical importance. There is certain truth in all the above arguments, and their impacts on optimization community are somehow mixed. Obviously, in reality, the algorithms with problem-specific knowledge typically work better than random search, and we also realized that there is no universally generic tool that works best for all the problems. Therefore, we have to seek balance between speciality and generality, between algorithm simplicity and problem complexity, and between problem-specific knowledge and capability of handling black-box optimization problems.

References

1. Afshar, A., Haddad, O.B., Marino, M.A., Adams, B.J.: Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation. J. Franklin Institute 344, 452–462 (2007)
2. Auger, A., Teytaud, O.: Continuous lunches are free plus the design of optimal optimization algorithms. Algorithmica 57, 121–146 (2010)
3. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific (2010)
4. Blum, C., Roli, A.: Metaheuristics in combinatorial optimisation: Overview and conceptural comparision. ACM Comput. Surv. 35, 268–308 (2003)
5. Clerc, M., Kennedy, J.: The particle swarm - explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. Evolutionary Computation 6, 58–73 (2002)
6. Copeland, B.J.: The Essential Turing. Oxford University Press (2004)
7. Corne, D., Knowles, J.: Some multiobjective optimizers are better than others. In: Evolutionary Computation, CEC 2003, vol. 4, pp. 2506–2512 (2003)
8. Christensen, S., Oppacher, F.: Wath can we learn from No Free Lunch? In: Proc. Genetic and Evolutionary Computation Conference (GECCO 2001), pp. 1219–1226 (2001)
9. Durgun, I., Yildiz, A.R.: Structural design optimization of vehicle components using cuckoo search algorithm. Materials Testing 3, 185–188 (2012)
10. Dorigo, M., Stütle, T.: Ant Colony Optimization. MIT Press (2004)
11. Floudas, C.A., Pardolos, P.M.: Encyclopedia of Optimization, 2nd edn. Springer (2009)

12. Geem, Z.W.: Music-Inspired Harmony Search Algorithm: Theory and Applications. Springer (2009)
13. Gandomi, A.H., Yang, X.S., Alavi, A.H.: Cuckoo search algorithm: a meteheuristic approach to solve structural optimization problems. In: Engineering with Computers, July 29 (2011), doi:10.1007/s00366-011-0241-y
14. Gandomi, A.H., Yang, X.S., Talatahari, S., Deb, S.: Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization. Computers & Mathematics with Applications 63(1), 191–200 (2012)
15. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
16. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Addison-Wesley, Reading (2002)
17. Gutjahr, W.J.: Convergence Analysis of Metaheuristics. Annals of Information Systems 10, 159–187 (2010)
18. Holland, J.: Adaptation in Natural and Artificial systems. University of Michigan Press, Ann Anbor (1975)
19. Igel, C., Toussaint, M.: On classes of functions for which no free lunch results hold. Inform. Process. Lett. 86, 317–321 (2003)
20. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Turkey (2005)
21. Kennedy, J., Eberhart, R.: Particle swarm optimisation. In: Proc. of the IEEE Int. Conf. on Neural Networks, Piscataway, NJ, pp. 1942–1948 (1995)
22. Kirkpatrick, S., Gellat, C.D., Vecchi, M.P.: Optimisation by simulated annealing. Science 220, 671–680 (1983)
23. Nakrani, S., Tovey, C.: On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers. Adaptive Behaviour 12(3-4), 223–240 (2004)
24. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity. Springer (2010)
25. Marshall, J.A., Hinton, T.G.: Beyond no free lunch: realistic algorithms for arbitrary problem classes. In: WCCI 2010 IEEE World Congress on Computational Intelligence, Barcelona, Spain, July 18-23, pp. 1319–1324 (2010)
26. Parpinelli, R.S., Lopes, H.S.: New inspirations in swarm intelligence: a survey. Int. J. Bio-Inspired Computation 3, 1–16 (2011)
27. Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., Zaidi, M.: The Bees Algorithm A Novel Tool for Complex Optimisation Problems. In: Proceedings of IPROMS 2006 Conference, pp. 454–461 (2006)
28. Price, K., Storn, R., Lampinen, J.: Differential Evolution: A Practical Approach to Global Optimization. Springer (2005)
29. Schumacher, C., Vose, M., Whitley, D.: The no free lunch and problem description length. In: Genetic and Evolutionary Computation Conference, GECCO 2001, pp. 565–570 (2001)
30. Shilane, D., Martikainen, J., Dudoit, S., Ovaska, S.J.: A general framework for statistical performance comparison of evolutionary computation algorithms. Information Sciences 178, 2870–2879 (2008)
31. Spall, J.C., Hill, S.D., Stark, D.R.: Theoretical framework for comparing several stochastic optimization algorithms. In: Probabilistic and Randomized Methods for Design Under Uncertainty, pp. 99–117. Springer, London (2006)
32. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359 (1997)
33. Turing, A.M.: Intelligent Machinery. Technical Report, National Physical Laboratory (1948)

34. Villalobos-Arias, M., Coello Coello, C.A., Hernández-Lerma, O.: Asymptotic convergence of metaheuristics for multiobjective optimization problems. Soft Computing 10, 1001–1005 (2005)
35. 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)
36. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimisation. IEEE Transaction on Evolutionary Computation 1, 67–82 (1997)
37. Wolpert, D.H., Macready, W.G.: Coevolutonary free lunches. IEEE Trans. Evolutionary Computation 9, 721–735 (2005)
38. Turing Archive for the History of Computing, `www.alanturing.net`
39. Yang, X.-S.: Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2005. LNCS, vol. 3562, pp. 317–323. Springer, Heidelberg (2005)
40. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press (2008)
41. Yang, X.-S.: Firefly Algorithms for Multimodal Optimization. In: Watanabe, O., Zeugmann, T. (eds.) SAGA 2009. LNCS, vol. 5792, pp. 169–178. Springer, Heidelberg (2009)
42. Yang, X.S.: Firefly algorithm, stochastic test functions and design optimisation. Int. J. Bio-Inspired Computation 2, 78–84 (2010a)
43. Yang, X.S.: Engineering Optimization: An Introduction with Metaheuristic Applications. John Wiley and Sons, USA (2010b)
44. Yang, X.-S.: A New Metaheuristic Bat-Inspired Algorithm. In: González, J.R., Pelta, D.A., Cruz, C., Terrazas, G., Krasnogor, N., et al. (eds.) NICSO 2010. Studies in Computational Intelligence, vol. 284, pp. 65–74. Springer, Heidelberg (2010c)
45. Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: Proceeings of World Congress on Nature & Biologically Inspired Computing, NaBIC 2009, pp. 210–214. IEEE Publications, USA (2009)
46. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. Int. J. Math. Modelling & Num. Optimisation 1, 330–343 (2010)
47. Yang, X.S.: Bat algorithm for multi-objective optimisation. Int. J. Bio-Inspired Computation 3(5), 267–274 (2011)
48. Yang, X.S., Deb, S.: Two-stage eagle strategy with differential evolution. Int. J. Bio-Inspired Computation 4(1), 1–5 (2012)
49. Yang, X.S., Hossein, S.S., Gandomi, A.H.: Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. Applied Soft Computing 12(3), 1180–1186 (2012)
50. Yang, X.S., Deb, S.: Multiobjective cuckoo search for design optimization. Computers and Operations Research (October 2011) (accepted), doi:10.1016/j.cor.2011.09.026
51. Yu, L., Wang, S.Y., Lai, K.K., Nakamori, Y.: Time series forecasting with multiple candidate models: selecting or combining? Journal of Systems Science and Complexity 18(1), 1–18 (2005)