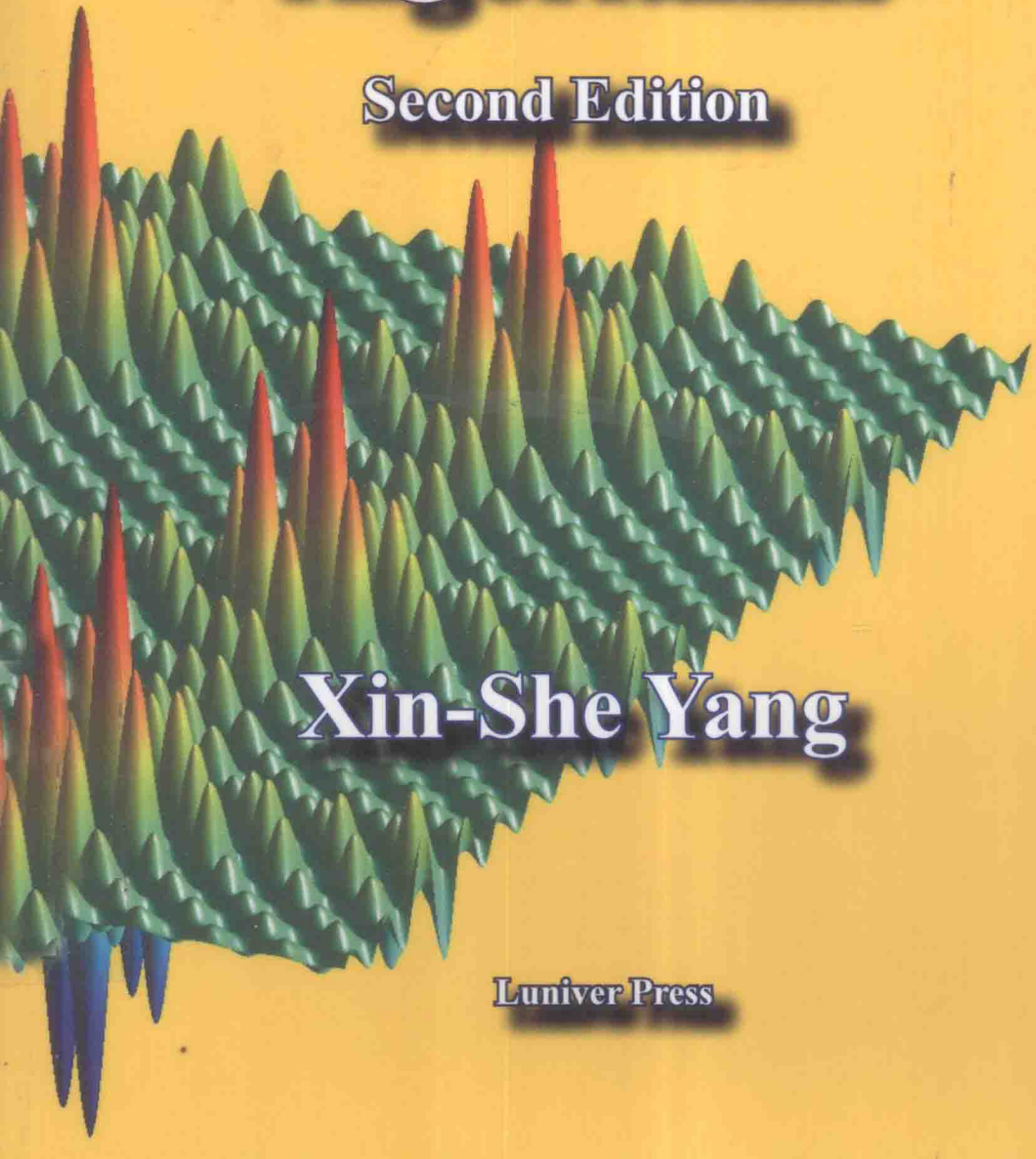
Nature-Inspired Metaheuristic Algorithms

Second Edition



Xin-She Yang

Luniver Press

Nature-Inspired Metaheuristic Algorithms

Second Edition

常州大学山书馆
藏书章

Xin-She Yang

University of Cambridge, United Kingdom

LUNIVER PRESS

Published in 2010 by Luniver Press
Frome, BA11 6TT, United Kingdom
www.luniver.com

Copyright ©Luniver Press 2010
Copyright ©Xin-She Yang 2010

All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without permission in writing from the copyright holder.

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from
the British Library

ISBN-13: 978-1-905986-28-6
ISBN-10: 1-905986-28-9

While every attempt is made to ensure that the information in this publication is correct, no liability can be accepted by the authors or publishers for loss, damage or injury caused by any errors in, or omission from, the information given.

Preface to the Second Edition

Since the publication of the first edition of this book in 2008, significant developments have been made in metaheuristics, and new nature-inspired metaheuristic algorithms emerge, including cuckoo search and bat algorithms. Many readers have taken time to write to me personally, providing valuable feedback, asking for more details of algorithm implementation, or simply expressing interests in applying these new algorithms in their applications.

In this revised edition, we strive to review the latest developments in metaheuristic algorithms, to incorporate readers' suggestions, and to provide a more detailed description to algorithms. Firstly, we have added detailed descriptions of how to incorporate constraints in the actual implementation. Secondly, we have added three chapters on differential evolution, cuckoo search and bat algorithms, while some existing chapters such as ant algorithms and bee algorithms are combined into one due to their similarity. Thirdly, we also explained artificial neural networks and support vector machines in the framework of optimization and metaheuristics. Finally, we have been trying in this book to provide a consistent and unified approach to metaheuristic algorithms, from a brief history in the first chapter to the unified approach in the last chapter.

Furthermore, we have provided more Matlab programs. At the same time, we also omit some of the implementation such as genetic algorithms, as we know that there are many good software packages (both commercial and open course). This allows us to focus more on the implementation of new algorithms. Some of the programs also have a version for constrained optimization, and readers can modify them for their own applications.

Even with the good intention to cover most popular metaheuristic algorithms, the choice of algorithms is a difficult task, as we do not have the space to cover every algorithm. The omission of an algorithm does not mean that it is not popular. In fact, some algorithms are very powerful and routinely used in many applications. Good examples are Tabu search and combinatorial algorithms, and interested readers can refer to the references provided at the end of the book. The effort in writing this little book becomes worth while if this book could in some way encourage readers' interests in metaheuristics.

Xin-She Yang

August 2010

Preface to the First Edition

Modern metaheuristic algorithms such as the ant colony optimization and the harmony search start to demonstrate their power in dealing with tough optimization problems and even NP-hard problems. This book reviews and introduces the state-of-the-art nature-inspired metaheuristic algorithms in optimization, including genetic algorithms (GA), particle swarm optimization (PSO), simulated annealing (SA), ant colony optimization (ACO), bee algorithms (BA), harmony search (HS), firefly algorithms (FA), photosynthetic algorithm (PA), enzyme algorithm (EA) and Tabu search. By implementing these algorithms in Matlab/Octave, we will use worked examples to show how each algorithm works. This book is thus an ideal textbook for an undergraduate and/or graduate course. As some of the algorithms such as the harmony search and firefly algorithms are at the forefront of current research, this book can also serve as a reference book for researchers.

I would like to thank my editor, Andy Adamatzky, at Luniver Press for his help and professionalism. Last but not least, I thank my wife and son for their help.

Xin-She Yang

Cambridge, 2008

此为试读,需要完整PDF请访问: www.ertongbook.com

CONTENTS

Chapter 1

INTRODUCTION

It is no exaggeration to say that optimization is everywhere, from engineering design to business planning and from the routing of the Internet to holiday planning. In almost all these activities, we are trying to achieve certain objectives or to optimize something such as profit, quality and time. As resources, time and money are always limited in real-world applications, we have to find solutions to optimally use these valuable resources under various constraints. Mathematical optimization or programming is the study of such planning and design problems using mathematical tools. Nowadays, computer simulations become an indispensable tool for solving such optimization problems with various efficient search algorithms.

1.1 OPTIMIZATION

Mathematically speaking, it is possible to write most optimization problems in the generic form

$$\underset{\boldsymbol{x}\in\Re^n}{\text{minimize}} \quad f_i(\boldsymbol{x}), \qquad (i = 1, 2, ..., M), \tag{1.1}$$

$$\text{subject to} \quad h_j(\boldsymbol{x}) = 0, \quad (j = 1, 2, ..., J), \tag{1.2}$$

$$g_k(\boldsymbol{x}) \leq 0, \quad (k = 1, 2, ..., K), \tag{1.3}$$

where $f_i(\boldsymbol{x}), h_j(\boldsymbol{x})$ and $g_k(\boldsymbol{x})$ are functions of the design vector

$$\boldsymbol{x} = (x_1, x_2, ..., x_n)^T. \tag{1.4}$$

Here the components x_i of $\boldsymbol{x}$ are called design or decision variables, and they can be real continuous, discrete or the mixed of these two.

The functions $f_i(\boldsymbol{x})$ where $i = 1, 2, ..., M$ are called the objective functions or simply cost functions, and in the case of $M = 1$, there is only a single objective. The space spanned by the decision variables is called the design space or search space $\Re^n$, while the space formed by the objective function values is called the solution space or response space. The equalities for h_j and inequalities for g_k are called constraints. It is worth pointing

Copyright © 2010 Luniver Press

out that we can also write the inequalities in the other way ≥ 0, and we can also formulate the objectives as a maximization problem.

In a rare but extreme case where there is no objective at all, there are only constraints. Such a problem is called a feasibility problem because any feasible solution is an optimal solution.

If we try to classify optimization problems according to the number of objectives, then there are two categories: single objective $M = 1$ and multiobjective $M > 1$. Multiobjective optimization is also referred to as multicriteria or even multi-attributes optimization in the literature. In real-world problems, most optimization tasks are multiobjective. Though the algorithms we will discuss in this book are equally applicable to multiobjective optimization with some modifications, we will mainly place the emphasis on single objective optimization problems.

Similarly, we can also classify optimization in terms of number of constraints $J + K$. If there is no constraint at all $J = K = 0$, then it is called an unconstrained optimization problem. If $K = 0$ and $J \geq 1$, it is called an equality-constrained problem, while $J = 0$ and $K \geq 1$ becomes an inequality-constrained problem. It is worth pointing out that in some formulations in the optimization literature, equalities are not explicitly included, and only inequalities are included. This is because an equality can be written as two inequalities. For example $h(\boldsymbol{x}) = 0$ is equivalent to $h(\boldsymbol{x}) \leq 0$ and $h(\boldsymbol{x}) \geq 0$.

We can also use the actual function forms for classification. The objective functions can be either linear or nonlinear. If the constraints h_j and g_k are all linear, then it becomes a linearly constrained problem. If both the constraints and the objective functions are all linear, it becomes a linear programming problem. Here 'programming' has nothing to do with computing programming, it means planning and/or optimization. However, generally speaking, all f_i, h_j and g_k are nonlinear, we have to deal with a nonlinear optimization problem.

1.2 SEARCH FOR OPTIMALITY

After an optimization problem is formulated correctly, the main task is to find the optimal solutions by some solution procedure using the right mathematical techniques.

Figuratively speaking, searching for the optimal solution is like treasure hunting. Imagine we are trying to hunt for a hidden treasure in a hilly landscape within a time limit. In one extreme, suppose we are blindfold without any guidance, the search process is essentially a pure random search, which is usually not efficient as we can expect. In another extreme, if we are told the treasure is placed at the highest peak of a known region, we will then directly climb up to the steepest cliff and try to reach to the highest peak, and this scenario corresponds to the classical hill-climbing

此为试读,需要完整PDF请访问: www.ertongbook.com

techniques. In most cases, our search is between these extremes. We are not blind-fold, and we do not know where to look for. It is a silly idea to search every single square inch of an extremely large hilly region so as to find the treasure.

The most likely scenario is that we will do a random walk, while looking for some hints; we look at some place almost randomly, then move to another plausible place, then another and so on. Such random walk is a main characteristic of modern search algorithms. Obviously, we can either do the treasure-hunting alone, so the whole path is a trajectory-based search, and simulated annealing is such a kind. Alternatively, we can ask a group of people to do the hunting and share the information (and any treasure found), and this scenario uses the so-called swarm intelligence and corresponds to the particle swarm optimization, as we will discuss later in detail. If the treasure is really important and if the area is extremely large, the search process will take a very long time. If there is no time limit and if any region is accessible (for example, no islands in a lake), it is theoretically possible to find the ultimate treasure (the global optimal solution).

Obviously, we can refine our search strategy a little bit further. Some hunters are better than others. We can only keep the better hunters and recruit new ones, this is something similar to the genetic algorithms or evolutionary algorithms where the search agents are improving. In fact, as we will see in almost all modern metaheuristic algorithms, we try to use the best solutions or agents, and randomize (or replace) the not-so-good ones, while evaluating each individual's competence (fitness) in combination with the system history (use of memory). With such a balance, we intend to design better and efficient optimization algorithms.

Classification of optimization algorithm can be carried out in many ways. A simple way is to look at the nature of the algorithm, and this divides the algorithms into two categories: deterministic algorithms, and stochastic algorithms. Deterministic algorithms follow a rigorous procedure, and its path and values of both design variables and the functions are repeatable. For example, hill-climbing is a deterministic algorithm, and for the same starting point, they will follow the same path whether you run the program today or tomorrow. On the other hand, stochastic algorithms always have some randomness. Genetic algorithms are a good example, the strings or solutions in the population will be different each time you run a program since the algorithms use some pseudo-random numbers, though the final results may be no big difference, but the paths of each individual are not exactly repeatable.

Furthermore, there is a third type of algorithm which is a mixture, or a hybrid, of deterministic and stochastic algorithms. For example, hill-climbing with a random restart is a good example. The basic idea is to use the deterministic algorithm, but start with different initial points. This has certain advantages over a simple hill-climbing technique, which may be

stuck in a local peak. However, since there is a random component in this hybrid algorithm, we often classify it as a type of stochastic algorithm in the optimization literature.

1.3 NATURE-INSPIRED METAHEURISTICS

Most conventional or classic algorithms are deterministic. For example, the simplex method in linear programming is deterministic. Some deterministic optimization algorithms used the gradient information, they are called gradient-based algorithms. For example, the well-known Newton-Raphson algorithm is gradient-based, as it uses the function values and their derivatives, and it works extremely well for smooth unimodal problems. However, if there is some discontinuity in the objective function, it does not work well. In this case, a non-gradient algorithm is preferred. Non-gradient-based or gradient-free algorithms do not use any derivative, but only the function values. Hooke-Jeeves pattern search and Nelder-Mead downhill simplex are examples of gradient-free algorithms.

For stochastic algorithms, in general we have two types: heuristic and metaheuristic, though their difference is small. Loosely speaking, *heuristic* means 'to find' or 'to discover by trial and error'. Quality solutions to a tough optimization problem can be found in a reasonable amount of time, but there is no guarantee that optimal solutions are reached. It hopes that these algorithms work most of the time, but not all the time. This is good when we do not necessarily want the best solutions but rather good solutions which are easily reachable.

Further development over the heuristic algorithms is the so-called metaheuristic algorithms. Here *meta-* means 'beyond' or 'higher level', and they generally perform better than simple heuristics. In addition, all metaheuristic algorithms use certain tradeoff of randomization and local search. It is worth pointing out that no agreed definitions of heuristics and metaheuristics exist in the literature; some use 'heuristics' and 'metaheuristics' interchangeably. However, the recent trend tends to name all stochastic algorithms with randomization and local search as metaheuristic. Here we will also use this convention. Randomization provides a good way to move away from local search to the search on the global scale. Therefore, almost all metaheuristic algorithms intend to be suitable for global optimization.

Heuristics is a way by trial and error to produce acceptable solutions 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 no guarantee for such optimality.

Two major components of any metaheuristic algorithms are: intensification and diversification, or exploitation and exploration. 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 with the selection of the best solutions. The selection of the best ensures that the solutions will converge to the optimality, while the diversification via randomization avoids the solutions being trapped at local optima and, at the same time, increases the diversity of the solutions. The good combination of these two major components will usually ensure that the global optimality is achievable.

Metaheuristic algorithms can be classified in many ways. One way is to classify them as: population-based and trajectory-based. For example, genetic algorithms are population-based as they use a set of strings, so is the particle swarm optimization (PSO) which uses multiple agents or particles.

On the other hand, simulated annealing uses a single agent or solution which moves through the design space or search space in a piecewise style. A better move or solution is always accepted, while a not-so-good move can be accepted with a certain probability. The steps or moves trace a trajectory in the search space, with a non-zero probability that this trajectory can reach the global optimum.

Before we introduce all popular meteheuristic algorithms in detail, let us look at their history briefly.

1.4 A BRIEF HISTORY OF METAHEURISTICS

Throughout history, especially at the early periods of human history, we humans' approach to problem-solving has always been heuristic or metaheuristic – by trial and error. Many important discoveries were done by 'thinking outside the box', and often by accident; that is heuristics. Archimedes's Eureka moment was a heuristic triumph. In fact, our daily learning experience (at least as a child) is dominantly heuristic.

Despite its ubiquitous nature, metaheuristics as a scientific method to problem solving is indeed a modern phenomenon, though it is difficult to pinpoint when the metaheuristic method was first used. Alan Turing was probably the first to use heuristic algorithms during the second World War when he was breaking German Enigma ciphers 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 correct solution, but it was a tremendous success. 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, he outlined his innovative ideas of machine intelligence and learning, neural networks and evolutionary algorithms.

The 1960s and 1970s were the two important decades for the development of evolutionary algorithms. First, John Holland and his collaborators at the University of Michigan developed the genetic algorithms in 1960s and 1970s. As early as 1962, Holland studied the adaptive system and was the first to use crossover and recombination manipulations for modeling such system. His seminal book summarizing the development of genetic algorithms was published in 1975. In the same year, De Jong finished his important dissertation showing the potential and power of genetic algorithms for a wide range of objective functions, either noisy, multimodal or even discontinuous.

In essence, a genetic algorithm (GA) is a search method based on the abstraction of Darwinian evolution and natural selection of biological systems and representing them in the mathematical operators: crossover or recombination, mutation, fitness, and selection of the fittest. Ever since, genetic algorithms become so successful in solving a wide range of optimization problems, there have several thousands of research articles and hundreds of books written. Some statistics show that a vast majority of Fortune 500 companies are now using them routinely to solve tough combinatorial optimization problems such as planning, data-fitting, and scheduling.

During the same period, Ingo Rechenberg and Hans-Paul Schwefel both then at the Technical University of Berlin developed a search technique for solving optimization problem in aerospace engineering, called evolutionary strategy, in 1963. Later, Peter Bienert joined them and began to construct an automatic experimenter using simple rules of mutation and selection. There was no crossover in this technique, only mutation was used to produce an offspring and an improved solution was kept at each generation. This was essentially a simple trajectory-style hill-climbing algorithm with randomization. As early as 1960, Lawrence J. Fogel intended to use simulated evolution as a learning process as a tool to study artificial intelligence. Then, in 1966, L. J. Fogel, together A. J. Owen and M. J. Walsh, developed the evolutionary programming technique by representing solutions as finite-state machines and randomly mutating one of these machines. The above innovative ideas and methods have evolved into a much wider discipline, called *evolutionary algorithms* and/or *evolutionary computation*.

Although our focus in this book is metaheuristic algorithms, other algorithms can be thought as a heuristic optimization technique. These includes artificial neural networks, support vector machines and many other machine learning techniques. Indeed, they all intend to minimize their learning errors and prediction (capability) errors via iterative trials and errors.

Artificial neural networks are now routinely used in many applications. In 1943, W. McCulloch and W. Pitts proposed the artificial neurons as simple information processing units. The concept of a neural network was probably first proposed by Alan Turing in his 1948 NPL report concerning 'intelligent machinery'. Significant developments were carried out from the 1940s and 1950s to the 1990s with more than 60 years of history.

The support vector machine as a classification technique can date back to the earlier work by V. Vapnik in 1963 on linear classifiers, and the nonlinear classification with kernel techniques were developed by V. Vapnik and his collaborators in the 1990s. A systematical summary in Vapnik's book on the Nature of Statistical Learning Theory was published in 1995.

The two decades of 1980s and 1990s were the most exciting time for metaheuristic algorithms. The next big step is the development of simulated annealing (SA) in 1983, an optimization technique, pioneered by S. Kirkpatrick, C. D. Gellat and M. P. Vecchi, inspired by the annealing process of metals. It is a trajectory-based search algorithm starting with an initial guess solution at a high temperature, and gradually cooling down the system. A move or new solution is accepted if it is better; otherwise, it is accepted with a probability, which makes it possible for the system to escape any local optima. It is then expected that if the system is cooled down slowly enough, the global optimal solution can be reached.

The actual first usage of memory in modern metaheuristics is probably due to Fred Glover's Tabu search in 1986, though his seminal book on Tabu search was published later in 1997.

In 1992, Marco Dorigo finished his PhD thesis on optimization and natural algorithms, in which he described his innovative work on ant colony optimization (ACO). This search technique was inspired by the swarm intelligence of social ants using pheromone as a chemical messenger. Then, in 1992, John R. Koza of Stanford University published a treatise on genetic programming which laid the foundation of a whole new area of machine learning, revolutionizing computer programming. As early as in 1988, Koza applied his first patent on genetic programming. The basic idea is to use the genetic principle to breed computer programs so as to gradually produce the best programs for a given type of problem.

Slightly later in 1995, another significant progress is the development of the particle swarm optimization (PSO) by American social psychologist James Kennedy, and engineer Russell C. Eberhart. Loosely speaking, PSO is an optimization algorithm inspired by swarm intelligence of fish and birds and by even human behavior. The multiple agents, called particles, swarm around the search space starting from some initial random guess. The swarm communicates the current best and shares the global best so as to focus on the quality solutions. Since its development, there have been about 20 different variants of particle swarm optimization techniques, and have been applied to almost all areas of tough optimization problems. There is

some strong evidence that PSO is better than traditional search algorithms and even better than genetic algorithms for many types of problems, though this is far from conclusive.

In around 1996 and later in 1997, R. Storn and K. Price developed their vector-based evolutionary algorithm, called differential evolution (DE), and this algorithm proves more efficient than genetic algorithms in many applications.

In 1997, the publication of the 'no free lunch theorems for optimization' by D. H. Wolpert and W. G. Macready sent out a shock way to the optimization community. Researchers have been always trying to find better algorithms, or even universally robust algorithms, for optimization, especially for tough NP-hard optimization problems. However, these theorems state that if algorithm A performs better than algorithm B for some optimization functions, then B will outperform A for other functions. That is to say, if averaged over all possible function space, both algorithms A and B will perform on average equally well. Alternatively, there is no universally better algorithms exist. That is disappointing, right? Then, people realized that we do not need the average over all possible functions for a given optimization problem. What we want is to find the best solutions, which has nothing to do with average over all possible function space. In addition, we can accept the fact that there is no universal or magical tool, but we do know from our experience that some algorithms indeed outperform others for given types of optimization problems. So the research now focuses on finding the best and most efficient algorithm(s) for a given problem. The objective is to design better algorithms for most types of problems, not for all the problems. Therefore, the search is still on.

At the turn of the 21st century, things became even more exciting. First, Zong Woo Geem *et al.* in 2001 developed the harmony search (HS) algorithm, which has been widely applied in solving various optimization problems such as water distribution, transport modelling and scheduling. In 2004, S. Nakrani and C. Tovey proposed the honey bee algorithm and its application for optimizing Internet hosting centers, which followed by the development of a novel bee algorithm by D. T. Pham *et al.* in 2005 and the artificial bee colony (ABC) by D. Karaboga in 2005. In 2008, the author of this book developed the firefly algorithm (FA)[1]. Quite a few research articles on the firefly algorithm then followed, and this algorithm has attracted a wide range of interests. In 2009, Xin-She Yang at Cambridge University, UK, and Suash Deb at Raman College of Engineering, India, introduced an efficient cuckoo search (CS) algorithm, and it has been demonstrated that CS is far more effective than most existing metaheuristic algorithms

[1]X. S. Yang, *Nature-Inspired Meteheuristic Algorithms*, Luniver Press, (2008)

此为试读,需要完整PDF请访问: www.ertongbook.com