Studies in Computational Intelligence 637

Xin-She Yang *Editor*

Nature-Inspired Computation in Engineering

Springer

Studies in Computational Intelligence

Volume 637

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

About this Series

The series "Studies in Computational Intelligence" (SCI) publishes new developments and advances in the various areas of computational intelligence—quickly and with a high quality. The intent is to cover the theory, applications, and design methods of computational intelligence, as embedded in the fields of engineering, computer science, physics and life sciences, as well as the methodologies behind them. The series contains monographs, lecture notes and edited volumes in computational intelligence spanning the areas of neural networks, connectionist systems, genetic algorithms, evolutionary computation, artificial intelligence, cellular automata, self-organizing systems, soft computing, fuzzy systems, and hybrid intelligent systems. Of particular value to both the contributors and the readership are the short publication timeframe and the worldwide distribution, which enable both wide and rapid dissemination of research output.

More information about this series at http://www.springer.com/series/7092

Xin-She Yang
Editor

Nature-Inspired Computation in Engineering

Springer

Editor
Xin-She Yang
School of Science and Technology
Middlesex University
London
UK

ISSN 1860-949X ISSN 1860-9503 (electronic)
Studies in Computational Intelligence
ISBN 978-3-319-30233-1 ISBN 978-3-319-30235-5 (eBook)
DOI 10.1007/978-3-319-30235-5

Library of Congress Control Number: 2016933797

© Springer International Publishing Switzerland 2016
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

Preface

Nature-inspired computation has become increasingly popular in engineering and is thus leading to a paradigm shift in problem-solving and design approaches. Many design problems in engineering are complex with multiple nonlinear constraints, and their design objectives can often be conflicting under stringent design requirements. Such problems can be very challenging to solve because of their complexity, nonlinearity and potentially high-dimensionality. Traditional algorithms, especially gradient-based algorithms, can struggle to cope and the results are not satisfactory. New and alternative methods are highly needed.

In the last two decades, nature-inspired computation and optimization algorithms have demonstrated their effectiveness as an alternative set of design tools to deal with such complex design problems. As a result, the popularity of nature-inspired optimization algorithms has started to increase significantly in recent years. These nature-inspired algorithms include particle swarm optimization, differential evolution, cuckoo search, firefly algorithm, bat algorithm, bee algorithms and ant colony optimization as well as others. The key features of these algorithms are their simplicity and flexibility, which enable them to highly adaptable and sufficiently efficient to deal with a wide range of optimization problems. In practice, they are also easy to be implemented in almost any programming languages, and all these factors have contributed to their popularity in engineering.

A majority of nature-inspired optimization algorithms can be said to belong to the class of swarm intelligence (SI) based algorithms. Swarm intelligence concerns the high-level behaviour arising from simple interactions among multiple agents. Though the main characteristics of swarming behaviour may be drawn from different sources of inspiration in nature, the procedures and steps of each algorithm can be very different. However, they seem to use the self-organized abilities of complex systems based on simple interaction rules. Since most algorithms treat problems under consideration as a black box, it is possible for such algorithms to deal with complex problems with different properties whether continuous, discrete or mixed. In a broad sense, swarm intelligence is part of evolutionary computation paradigm and bio-inspired computation is part of nature-inspired computation;

however, there is still some confusion about some terminologies in the literature. For example, there are some overlaps and there are no agreed standard definitions about bio-inspired computation, nature-inspired computation, metaheuristic and computational intelligence. Therefore, we will not enter the debate about what the right terminology or subject fields should be, but we will rather focus our attention on nature-inspired computation and its applications in engineering.

The diversity and rapid advances in nature-inspired computation have resulted in a much richer literature. Therefore, a timely review is necessary to summarize the latest developments in terms of algorithm developments and their applications in engineering. Algorithms and topics include discrete firefly algorithm, discrete cuckoo search, plant propagation algorithm, parameter-free bat algorithm, gravitational search, biogeography-based algorithm, differential evolution, particle swarm optimization and others. State-of-the-art applications and case studies include vehicle routing, swarming robots, discrete and combinatorial optimization, clustering of wireless sensor networks, cell formation, economic load dispatch, metamodeling, surrogate-assisted cooperative co-evolution, data fitting and reverse engineering as well as other real-world applications.

As a timely review volume, this book can be an ideal reference for researchers, lecturers, graduates and engineers who are interested in latest developments in nature-inspired computation, artificial intelligence and computational intelligence. It can also serve as a reference for relevant courses in computer science, artificial intelligence, machine learning, natural computation, engineering optimization and data mining.

I thank our editors, Drs. Thomas Ditzinger and Holger Schaepe, and staff at Springer for their help and professionalism. Last but not least, I thank my family for the help and support.

London Xin-She Yang
December 2015

Contents

Contributors

Ali R. Al-Roomi Dalhousie University, Electrical and Computer Engineering, Halifax, NS, Canada

Roberto Carballedo Deusto Institute of Technology (DeustoTech), University of Deusto, Bilbao, Spain

Fernando Diaz Deusto Institute of Technology (DeustoTech), University of Deusto, Bilbao, Spain

Mohamed E. El-Hawary Dalhousie University, Electrical and Computer Engineering, Halifax, NS, Canada

Bouazza Elbenani Department of Computer Science, Mohammed V University, Rabat, Morocco

Eid Emary Faculty of Computers and Information, Cairo University, Giza, Egypt

Iztok Fister Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

Iztok Fister Jr. Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

Akemi Gálvez Department of Applied Mathematics and Computational Sciences, University of Cantabria, Santander, Spain

Xingshi He College of Science, Xi'an Polytechnic University, Xi'an, People's Republic of China

Andrés Iglesias Department of Applied Mathematics and Computational Sciences, University of Cantabria, Santander, Spain; Department of Information Science, Faculty of Sciences, Toho University, Funabashi, Japan

Alan Kwan School of Engineering, Cardiff University, Cardiff, UK

Haijiang Li School of Engineering, Cardiff University, Cardiff, UK

Yang Liu School of Engineering, Cardiff University, Cardiff, UK

Salvatore Marano Department of Computer Engineering, Modeling, Electronics, and Systems Science, University of Calabria, Rende, CS, Italy

Uroš Mlakar Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

Eneko Osaba Deusto Institute of Technology (DeustoTech), University of Deusto, Bilbao, Spain

Aziz Ouaarab LRIT, Associated Unit to the CNRST (URAC 29), Mohammed V-Agdal University, Rabat, Morocco

Nunzia Palmieri Department of Computer Engineering, Modeling, Electronics, and Systems Science, University of Calabria, Rende, CS, Italy; School of Science and Technology, Middlesex University, London, UK

Yacine Rezgui School of Engineering, Cardiff University, Cardiff, UK

Abdellah Salhi Department of Mathematical Sciences, University of Essex, Colchester, UK

Birsen İ. Selamoğlu University of Essex, Colchester, UK

Marwa Sharawi Faculty of Computer Studies, Arab Open University, Cairo, Egypt

Muhammad Sulaiman Department of Mathematics, Abdul Wali Khan University Mardan, Mardan, Khyber Pakhtunkhwa, Pakistan

Giuseppe A. Trunfio DADU, University of Sassari, Alghero, Italy

Xin-She Yang School of Science and Technology, Middlesex University, London, UK

Manal Zettam Department of Computer Science, Mohammed V University, Rabat, Morocco

Nature-Inspired Optimization Algorithms in Engineering: Overview and Applications

Xin-She Yang and Xingshi He

Abstract Nature-inspired computation has become popular in engineering applications and nature-inspired algorithms tend to be simple and flexible and yet sufficiently efficient to deal with highly nonlinear optimization problems. In this chapter, we first review the brief history of nature-inspired optimization algorithms, followed by the introduction of a few recent algorithms based on swarm intelligence. Then, we analyze the key characteristics of optimization algorithms and discuss the choice of algorithms. Finally, some case studies in engineering are briefly presented.

1 Introduction

Optimization is everywhere and important in many applications such as engineering, business activities and industrial designs. The aims of optimization can be very diverse—to minimize the energy consumption and costs, to maximize the profit, outputs, performance and efficiency. Thus, optimization concerns almost every application from engineering design to business planning and from holiday planning to vehicle routing. In reality, because resources, time and money are always limited in real-world applications, the optimal use these valuable resources under various constraints should be sought. As most real-world applications are often highly nonlinear, it requires sophisticated optimization tools so as to find quality and feasible solutions.

Nature-inspired computation is now playing an increasingly important role in many areas, including artificial intelligence, computational intelligence, data mining, machine learning and optimization [1]. Good examples of such algorithms are bat algorithm, cuckoo search, particle swarm optimization, firefly algorithm, bee algorithms and others [1–4]. Obviously, their popularity can be attributed to many factors

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

X. He
College of Science, Xi'an Polytechnic University, Jinhua South Road, Xi'an, People's Republic of China

© Springer International Publishing Switzerland 2016

X.-S. Yang (ed.), *Nature-Inspired Computation in Engineering*,
Studies in Computational Intelligence 637, DOI 10.1007/978-3-319-30235-5_1

and one key reason is that these algorithms are simple, flexible, efficient and highly adaptable. In addition, from the implementation point of view, they are very simple to be implemented in any programming language. As a result, these algorithms have been applied in a wide spectrum of problems in real-world applications.

The main objective of this chapter is to provide an overview of the history of the nature-inspired computation and review some of the recent nature-inspired algorithms for optimization. Some applications in engineering will also be highlighted. Therefore, the chapter is organized as follows. Section 2 outlines the basic formulation of optimization problems and the search for optimality, and then Sect. 3 presents the brief history of the recent developments in nature-inspired computation. Section 4 highlights the key features of metaheuristic algorithms, followed by the introduction to some recent algorithms based on swarm intelligence in Sect. 5. Section 6 discusses the choice of algorithms and then Sect. 7 highlights some recent applications of bio-inspired computation in engineering.

2 Optimization and Optimality

Many problems in engineering can be formulated as an optimization problem, though how to formulate and the way of formulations usually depend on the perspective that we are looking at the problem. Though there are many different ways of formulations, the exact formulations may also depend on the subject area and thus such formulations may directly or indirectly affect the choice of the solution techniques. Thus, this may be a subject matter, however, we will focus on the general context of optimization and search for optimality.

2.1 Optimization

From a mathematical point of view, all almost optimization problems can be formulated in the following generic form:

$$\underset{x \in \Re^d}{\text{minimize}} \quad f_i(x), \qquad (i = 1, 2, \ldots, M), \tag{1}$$

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

$$g_k(x) \leq 0, \quad (k = 1, 2, \ldots, K), \tag{3}$$

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

$$x = (x_1, x_2, \ldots, x_d)^T. \tag{4}$$

Here the components x_i of x are called design or decision variables, and they can be real continuous, discrete or the mixed (partly continuous and partly discrete). Here,

the functions $f_i(x)$ where $i = 1, 2, \ldots, M$ are called the objective functions or simply cost functions, and in the case of $M = 1$, there is only a single objective and the optimization process become a single objective optimization. The space spanned by the decision variables is called the design space or search space $\Re^d$, 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 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. For some difficult design problems with multiple, complex, potentially conflicting constraints, to find even a feasible solution may be a challenging task.

Obviously, 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 chapter are equally applicable to multiobjective optimization with some modifications, we will mainly focus 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 (i.e., $J = K = 0$), then it is called an unconstrained optimization problem. If $K = 0$ and $J \geq 1$, it is called an equality-constrained problem, while the case for $J = 0$ and $K \geq 1$ becomes an inequality-constrained problem.

It is worth pointing out that equality constraints have special properties, and require special care. One drawback is that the volume of satisfying an equality is essentially zero in the search space, thus very difficult to get sampling points that satisfy the equality exactly. Some tolerance or allowance is used in practice.

Furthermore, we can also use the actual function forms for classifying optimization problems. The objective functions can be either linear or nonlinear. If the constraints h_j and g_k are all linear, then the problem becomes a linearly constrained optimization 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.

2.2 Search for Optimality

Once an optimization problem is formulated correctly, the main task is to find the optimal solutions by a proper solution procedure using the right mathematical

techniques. However, there may not be any right solution procedure for some classes of problems, at least, it is not easy to find a right technique to solve a particular type of problem.

In many ways, searching for the optimal solution is very similar to treasure hunting. Imagine a scenario that we are trying to hunt for a hidden treasure in a hilly landscape within a time limit. In one extreme, suppose we are blind-fold without any guidance, the search process is essentially a pure random search, which is usually not efficient. 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 techniques. In most cases, the search is between these extremes. We are not blind-fold, and we do not know where to look for. Obviously, it is physically impossible to search every single square inch of an extremely large hilly region so as to find the treasure.

In reality, the most likely search 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 a random walk is a main characteristic of modern search algorithms [1]. 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 algorithms such as particle swarm optimization and firefly algorithm, 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).

Going with this analogy even further, 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.

3 A Brief History of Nature-Inspired Computation

Humans' approach to problem-solving has always been heuristic or metaheuristic—by trial and error, especially at the early periods of human history. Many important discoveries were done by 'thinking outside the box', and often by accident; that is heuristics. In fact, our daily learning experience during our childhood 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 the 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; however, it was a tremendous success. In his National Physical Laboratory report on *Intelligent machinery* in 1948, Turing outlined his innovative ideas of machine intelligence and learning, neural networks and evolutionary algorithms [7].

The first golden period is the 1960 s and 1970 s for the development of evolutionary algorithms. John Holland and his collaborators at the University of Michigan first developed the genetic algorithms in 1960 s and 1970s. As far back to 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 [8]. 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. The essence of a genetic algorithm (GA) is 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.

During the same decade, 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. 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. On the other hand, 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 [9]. The above innovative ideas and methods have evolved into a much wider discipline, called evolutionary algorithms and/or evolutionary computation [10, 11].

It may not be entirely clear how bio-inspired computation is also relevant to artificial neural network, support vector machine, and other many learning techniques in the context of optimization. However, these methods all intend to minimize their learning errors and prediction (capability) errors via iterative trials and errors. For example, 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' [7, 12]. Significant developments were carried out from the 1940 s and 1950 s to the 1990 s with more than 60 years of history. Another example is the so-called support vector machine as a classification technique that 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 [13].

Another important period for metaheuristic algorithms is the two decades of 1980 s and 1990s. First, 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. 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 [14]. Marco Dorigo finished his PhD thesis in 1992 on optimization and natural algorithms [2], 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. At the same time 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 [15]. 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.

In 1995, the particle swarm optimization (PSO) was developed by American social psychologist James Kennedy, and engineer Russell C. Eberhart [3], based on the swarming behaviour of fish and birds. 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.

Then, slightly later in 1996 and 1997, R. Storn and K. Price developed their vector-based evolutionary algorithm, called differential evolution (DE) [16], and this algorithm proves more efficient than genetic algorithms in many applications. In 1997, the 'no free lunch theorems for optimization' were proved by D.H. Wolpert and W.G. Macready [17, 18]. 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. However, researchers 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 may now focus on finding the best and most efficient algorithm(s) for a given set of

problems. The objective is to design better algorithms for most types of problems, not for all the problems. Therefore, the search is still on.

Another very exciting period for developing metaheuristic algorithms is the first decade of the 21st century. In 2001, Zong Woo Geem et al. developed the harmony search (HS) algorithm [19], 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 [20], which followed by the development of virtual bee algorithm by Xin-She Yang in 2005. At the same time, the bees algorithm was developed by D.T. Pham et al. in 2005 and the artificial bee colony (ABC) was developed by D. Karaboga in 2005. Then, in late 2007 and early 2008, the firefly algorithm (FA) was developed by Xin-She Yang [1, 4], which has generated a wide range of interests. In 2009, Xin-She Yang at Cambridge University, UK, and Suash Deb at Raman College of Engineering, India, proposed an efficient cuckoo search (CS) algorithm [21, 22], and it has been demonstrated that CS can be far more effective than some existing metaheuristic algorithms. In 2010, the bat algorithm was developed by Xin-She Yang for continuous optimization, based on the echolocation behaviour of microbats [23]. In 2012, the flower pollination algorithm was developed by Xin-She Yang, and its efficiency is very promising.

As the literature is expanding rapidly, the number of nature-inspired algorithms has increased dramatically [1, 24] As we can see, more and more metaheuristic algorithms are being developed. Such a diverse range of algorithms necessitates a systematic summary of various metaheuristic algorithms, however, we only briefly outline a few of these recent algorithms in the rest of the chapter. But before we proceed, let us pause and highlight the key characteristics of nature-inspired algorithms.

4 Main Characteristics of Nature-Inspired Algorithms

The key characteristics of nature-inspired optimization algorithms can be analyzed from many different perspectives. From the algorithmic development point of view, we can analyze algorithms in terms of algorithm search behaviour, adaptation and diverse as well as algorithmic components in terms of mathematical equations for iterations.

4.1 Exploration and Exploitation

The key aim of an algorithm is to generate new solutions that should be better than previous solutions. For an ideal algorithm, new solutions should be always better than existing solutions and it can be expected that the most efficient algorithms are to find the best solutions with the least minimum efforts (ideally in one move). However, such ideal algorithms may not exist at all.

For stochastic algorithms, solutions do not always get better. In fact, it can be advantageous to select not-so-good solutions, which can help the search process escape from being trapped at any local optima. Though this may be counter-intuitive, such stochastic nature now forms the essential component of modern metaheuristic algorithms [4]. Exploitation typically uses any information obtained from the problem of interest so as to help to generate new solutions that are better than existing solutions. However, this process is typically local, and information (such as gradients) is also local. Therefore, it is mainly for local search. For example, hill-climbing is a method that uses derivative information to guide the search procedure. In fact, new steps always try to climb up the local gradient. The advantage of exploitation is that it usually leads to very high convergence rates, but its disadvantage is that it can get stuck in a local optimum because the final solution point largely depends on the starting point.

On the other hand, exploration makes it possible to explore the search space in far away regions more efficiently [25], and it can generate solutions with enough diversity and far from the current solutions. Therefore, the search is typically on a global scale. The advantage of exploration is that it is less likely to get stuck in a local mode, and the global optimality can be more accessible. However, its disadvantages are slow convergence and waste of lot computational efforts because many new solutions can be far from global optimality.

However, whether local search or global search, all depends on the actual search mechanisms within an algorithm. Sometimes, there is no clear cut between local or global. In addition, a fine balance may be required so that an algorithm can achieve the good performance. Too much exploitation and too little exploration means the system may converge more quickly, but the probability of finding the true global optimality may be low. Conversely, too little exploitation and too much exploration can cause the search path meander with very slow convergence. The optimal balance should mean the right amount of exploration and exploitation, which may lead to the optimal performance of an algorithm. Therefore, a proper balance is crucially important to ensure the good performance of an algorithm.

4.2 Diversity and Adaptation

Looking from a different perspective, nature-inspired algorithms can have both diversity and adaptation. Adaptation in nature-inspired algorithms can take many forms. For example, the ways to balance exploration and exploitation are the key form of adaptation [26]. As diversity can be intrinsically linked with adaptation, it is better not to discuss these two features separately. For example, in genetic algorithms, representations of solutions are usually in binary or real-valued strings [8, 26], while in swarm-intelligence-based algorithms, representations mostly use real number solution vectors. As another example, the population size used in an algorithm can be fixed or varying. Adaptation in this case may mean to vary the population size so as to maximize the overall performance.

For a given algorithm, adaptation can also occur to adjust its algorithm-dependent parameters. As the performance of an algorithm can largely depend on its parameters, the choice of these parameter values can be very important. Similarly, diversity in metaheuristic algorithms can also take many forms. The simplest diversity is to allow the variations of solutions in the population by randomization. For example, solution diversity in genetic algorithms is mainly controlled by the mutation rate and crossover mechanisms, while in simulated annealing, diversity is achieved by random walks. In addition, adaptation can also be in the form of self-tuning in terms of parameters so as to achieve better performance automatically [27].

In most swarm-intelligence-based algorithms, new solutions are generated according to a set of deterministic equations, which also include some random variables. Diversity is represented by the variations, often in terms of the population variance. Once the population variance is getting smaller (approaching zero), diversity also decreases, leading to converged solution sets. However, if diversity is reduced too quickly, premature convergence may occur. Therefore, a right amount of randomness and the right form of randomization can be crucial.

Both diversity and adaptation are important to ensure the effectiveness of an algorithm. A good diversity will ensure the population can explore different regions and can thus maintain a non-zero probability of finding the global optimality of the problem. In addition, good adaption will enable the algorithm to adapt to suit the problem and landscape under consideration, and thus may ensure a potentially better convergence than non-adaptive approaches. However, it is not yet clear what a good degree of diversity should be and what kind of adaptation mechanisms can be used.

4.3 Key Operators in Algorithms

Algorithms can also be analyzed by studying in detail the key algorithmic operators used in the construction of the algorithms. For example, in the well-established class of genetic algorithms [8], genetic operators such as crossover (or recombination), mutation and selection are used [26].

In genetic algorithms, crossover is the operation of generating two new solutions (offsprings) from two existing solutions (parents) by swapping relevant/corresponding parts of their solutions. This is similar to the main crossover feature in the biological systems. Crossover usually provides good mixing of solution characteristics and can usually generate completely new solutions if the two parents are different. Obviously, when the two parents are identical, offspring solutions will also be identical, and thus provides a good mechanism to maintain good convergence. It is worth pointing out that crossover in contemporary algorithms may take different forms, though its essence remains the same.

Mutation is a mechanism to generate a new solution from a single solution by changing a single site or multiple sites. As in the evolution in nature, mutation often