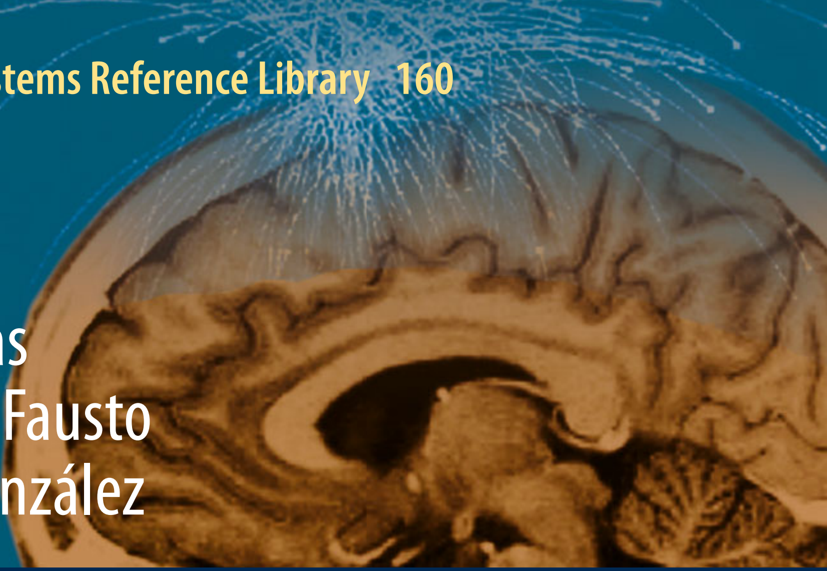
Intelligent Systems Reference Library 160



Erik Cuevas
Fernando Fausto
Adrián González

New Advancements in Swarm Algorithms: Operators and Applications

Springer

Intelligent Systems Reference Library

Volume 160

Series Editors

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland

Lakhmi C. Jain, Faculty of Engineering and Information Technology, Centre for Artificial Intelligence, University of Technology, Sydney, NSW, Australia;
Faculty of Science, Technology and Mathematics, University of Canberra, Canberra, ACT, Australia;
KES International, Shoreham-by-Sea, UK;
Liverpool Hope University, Liverpool, UK

Erik Cuevas · Fernando Fausto ·
Adrián González

New Advancements in Swarm Algorithms: Operators and Applications

Springer

Erik Cuevas
CUCEI, Universidad de Guadalajara
Guadalajara, Mexico

Fernando Fausto
CUCEI, Universidad de Guadalajara
Guadalajara, Mexico

Adrián González
CUCEI, Universidad de Guadalajara
Guadalajara, Mexico

ISSN 1868-4394 ISSN 1868-4408 (electronic)
Intelligent Systems Reference Library
ISBN 978-3-030-16338-9 ISBN 978-3-030-16339-6 (eBook)
https://doi.org/10.1007/978-3-030-16339-6

Library of Congress Control Number: 2019935485

Preface

The most common term for methods that employ stochastic schemes to produce search strategies is metaheuristics. In general, there not exist strict classifications of these methods. However, several kinds of algorithms have been coined depending on several criteria such as the source of inspiration, cooperation among the agents or type of operators.

From the metaheuristic methods, it is considered a special set of approaches which are designed in terms of the interaction among the search agents of a group. Members inside the group cooperate to solve a global objective by using local accessible knowledge that is propagated through the set of members. With this mechanism, complex problems can be solved more efficiently than considering the strategy of single individual. In general terms, this group is referred to as a swarm, where social agents interact with each other in a direct or indirect manner by using local information from the environment. This cooperation among agents produces an effective distributive strategy to solve problems. Swarm intelligence (SI) represents a problem-solving methodology that results from the cooperation among a set of agents with similar characteristics. During this cooperation, local behaviors of simple elements produce the existence of complex collective patterns.

The study of biological entities such as animals and insects which manifest a social behavior has produced several computational models of swarm intelligence. Some examples include ants, bees, locust swarms, spiders and bird flocks. In the swarm, each agent maintains a simple strategy. However, due to its social behavior, the final collective strategy produced by all agents is usually very complex. The complex operation of a swarm is a consequence of the cooperative behavior among the agents generated during their interaction.

The complex operation of the swarm cannot be reduced to the aggregation of behaviors of each agent in the group. The association of all simple agent behaviors is so complex that usually is not easy to predict or deduce the global behavior of the whole swarm. This concept is known as emergence. It refers to the process of produce complex behavioral patterns from the iteration of simple and unsophisticated strategies. Something remarkable is that these behavioral patterns appear without the existence of a coordinated control system but emerge from the exchange of local information among agents. Therefore, there subsists a close relationship between individual and collective behavior. In general, the collective behavior of agents determines the behavior of the swarm. On the other hand, swarm behavior is also strongly influenced by the conditions under which each agent executes its operations.

The operations of each agent can modify its own behavior and the behavior of other neighbor agents, which also alters the global swarm performance. Under such conditions, the most significant element of swarm intelligence is the model of interaction or cooperation among the agents. Cooperation in biological entities that operate as swarm systems happens in different mechanisms from which social interaction represents the most important. This social interaction can be conducted through physical contact, visual information, audio messages, or chemical perceptual inputs. Examples of cooperation models in nature are numerous, and some examples include the dynamical task assignation performed in an ant colony, without any central control or task coordination. The adoption of optimal spatial patterns builds by the self-organization in bird flocks and fish in schools. The hunting strategies developed by predators. The purpose of computational swarm intelligence schemes is to model the simple behaviors of agents and its local interactions with other neighboring agents to perform an effective search strategy for solving optimization problems.

One example is the particle swarm optimization (PSO) which models two simple actions. Each agent (1) moves toward the best agent of the swarm and (2) moves toward the position where the agent has reached its best location. As a consequence, the collective behavior of the swarm produces that all agents are attracted to the best positions experimented by the swarm. Another example is the ant colony optimization (ACO) which models the biological pheromone trail following behavior of ants. Under this mechanism, each ant senses pheromone concentrations in its local position. Then, it probabilistically selects the path with the highest pheromone concentration. Considering this model, the collective effect in the swarm is to find the best option (shortest path) from a group of alternatives available in a decision-making problem.

There exist several features that clearly appear in most of the metaheuristic and swarm approaches, such as the use of diversification to force the exploration of regions of the search space, rarely visited until now, and the use of intensification or exploitation, to investigate thoroughly some promising regions. Another interesting feature is the use of memory to store the best solutions encountered. For these reasons, metaheuristics and swarm methods quickly became popular amongst researchers to solve from simple to complex optimization problems in different areas.

Most of the problems in science, engineering, economics, and life can be translated as an optimization or a search problem. According to their characteristics, some problems can be simple that can be solved by traditional optimization methods based on mathematical analysis. However, most of the problems of practical importance such as system identification, parameter estimation, energy systems, represent conflicting scenarios so that they are very hard to be solved by using traditional approaches. Under such circumstances, metaheuristic and swarm algorithms have emerged as the best alternative to solve this kind of complex formulations. Therefore, swarm techniques have consolidated as a very active research subject in the last ten years. During this time, various new swarm approaches have been introduced. They have been experimentally examined on a set of artificial benchmark problems and in a large number of practical applications. Although metaheuristic and swarm methods represent one of the most exploited research paradigms in computational intelligence, there are a large number of open challenges in the area of swarm intelligence. They range from premature convergence, inability to maintain population diversity and the combination of swarm paradigms with other algorithmic schemes, toward extending the available techniques to tackle ever more difficult problems.

Numerous books have been published tacking in account any of the most widely known swarm methods, namely ant colony algorithms and particle swarm optimization but attempts to consider the discussion of new alternative approaches are always scarce. Initial swarm schemes maintain in their design several limitations such as premature convergence and inability to maintain population diversity. Recent swarm methods have addressed these difficulties providing in general better results. Many of these novel swarm approaches have also been lately introduced. In general, they propose new models and innovative cooperation models for producing an adequate exploration and exploitation of large search spaces considering a significant number of dimensions. Most of the new metaheuristic swarm present promising results. Nevertheless, they are still in their initial stage. To grow and attain their complete potential, new swarm methods must be applied in a great variety of problems and contexts, so that they do not only perform well in their reported sets of optimization problems, but also in new complex formulations. The only way to accomplish this is by making possible the transmission and presentation of these methods in different technical areas as optimization tools. In general, once a scientific, engineering, or practitioner recognizes a problem as a particular instance of a more generic class, he/she can select one of the different swarm algorithms that guarantee an expected optimization performance. Unfortunately, the set of options are concentrated in algorithms whose popularity and high proliferation are better than the new developments.

The excessive publication of developments based on the simple modification of popular swarm methods presents an important disadvantage: They avoid the opportunity to discover new techniques and procedures which can be useful to solve problems formulated by the academic and industrial communities. In the last years, several promising swarm schemes that consider very interesting concepts and operators have been introduced. However, they seem to have been completely overlooked in the literature, in favor of the idea of modifying, hybridizing, or restructuring popular swarm approaches.

The goal of this book is to present advances that discuss new alternative swarm developments which have proved to be effective in their application to several complex problems. The book considers different new metaheuristic methods and their practical applications. This structure is important to us, because we recognize this methodology as the best way to assist researchers, lecturers, engineers, and practitioners in the solution of their own optimization problems.

This book has been structured so that each chapter can be read independently from the others. Chapter 1 describes the main characteristics and properties of metaheuristic and swarm methods. This chapter analyses the most important concepts of metaheuristic and swarm schemes.

Chapter 2 discusses the performance and main applications of each metaheuristic and swarm method in the literature. The idea is to establish the strength and weaknesses of each traditional scheme from practical perspective.

The first part of the book that involves Chaps. 3, 4, 5, and 6 present recent swarm algorithms their operators and characteristics. In Chap. 3, an interesting swarm optimization algorithm called the Selfish Herd Optimizer (SHO) is presented for solving global optimization problems. SHO is based on the simulation of the widely observed selfish herd behavior manifested by individuals within a herd of animals subjected to some form of predation risk. In SHO, individuals emulate the predatory interactions between groups of prey and predators by two types of search agents: the members of a selfish herd (the prey) and a pack of hungry predators. Depending on their classification as either a prey or a predator, each individual is conducted by a set of unique evolutionary operators inspired by such prey–predator relationship. These unique traits allow SHO to improve the balance between exploration and exploitation without altering the population size. The experimental results show the remarkable performance of our proposed approach against those of the other compared methods, and as such SHO is proven to be an excellent alternative to solve global optimization problems.

Chapter 4 considers a recent swarm algorithm called the Social Spider Optimization (SSO) for solving optimization tasks. The SSO algorithm is based on the simulation of cooperative behavior of social spiders. In the proposed algorithm, individuals emulate a group of spiders which interact with each other based on the biological laws of the cooperative colony. The algorithm considers two different search agents (spiders): males and females. Depending on gender, each individual is conducted by a set of different evolutionary operators which mimic different cooperative behaviors that are typically found in the colony. In order to illustrate the proficiency and robustness of the proposed approach, it is compared to other well-known evolutionary methods. The comparison examines several standard benchmark functions that are commonly considered within the literature of evolutionary algorithms. The outcome shows a high performance of the proposed method for searching a global optimum with several benchmark functions.

In Chap. 5, a swarm algorithm called Locust Search (LS) is presented for solving optimization tasks. The LS algorithm is based on the simulation of the behavior presented in swarms of locusts. In the proposed algorithm, individuals emulate a group of locusts which interact with each other based on the biological laws of the cooperative swarm. The algorithm considers two different behaviors: solitary and social. Depending on the behavior, each individual is conducted by a set of evolutionary operators which mimic the different cooperative behaviors that are typically found in the swarm. In order to illustrate the proficiency and robustness of the proposed approach, it is compared to other well-known evolutionary methods. The comparison examines several standard benchmark functions that are commonly considered within the literature of evolutionary algorithms. The outcome shows a high performance of the proposed method for searching a global optimum with several benchmark functions.

Chapter 6 presents an algorithm for global optimization called the collective animal behavior (CAB). Animal groups, such as schools of fish, flocks of birds, swarms of locusts, and herds of wildebeest, exhibit a variety of behaviors including swarming about a food source, milling around a central location, or migrating over large distances in aligned groups. These collective behaviors are often advantageous to groups, allowing them to increase their harvesting efficiency, to follow better migration routes, to improve their aerodynamic, and to avoid predation. In the presented swarm algorithm, the searcher agents emulate a group of animals which interact with each other based on the biological laws of collective motion. The method has been compared to other well-known optimization algorithms. The results show good performance of the proposed method when searching for a global optimum of several benchmark functions.

The second part of the book which involves Chaps. 7, 8, and 9 presents the use of recent swarm algorithms in different domains. The idea is to show the potential of new swarm alternatives algorithms from a practical perspective.

In Chap. 7, an algorithm for the optimal parameter calibration of fractional fuzzy controllers (FCs) is presented. Fuzzy controllers (FCs) based on integer schemes have demonstrated their performance in an extensive variety of applications. However, several dynamic systems can be more accurately controlled by fractional controllers. Under such conditions, there is currently an increasing interest in generalizing the design of FCs with fractional operators. In the design stage of fractional FCs, the parameter calibration process is transformed into a multidimensional optimization problem where fractional orders as well as controller parameters of the fuzzy system are considered as decision variables. To determine the parameters, the proposed method uses the swarm method called Social Spider Optimization (SSO) which is inspired by the emulation of the collaborative behavior of social spiders. In SSO, solutions imitate a set of spiders which cooperate to each other based on the natural laws of the cooperative colony. Different to the most of existent evolutionary algorithms, it explicitly avoids the concentration of individuals in the best positions, avoiding critical flaws such as the premature convergence to suboptimal solutions and the limited exploration–exploitation balance. Numerical simulations have been conducted on several plants to show the effectiveness of the proposed scheme.

Chapter 8 presents an algorithm for the automatic selection of pixel classes for image segmentation. The presented method combines a swarm method with the definition of a new objective function that appropriately evaluates the segmentation quality with respect to the number of classes. The employed swarm algorithm is the Locust Search (LS) which is based on the behavior of swarms of locusts. Different to the most of existent evolutionary algorithms, it explicitly avoids the concentration of individuals in the best positions, avoiding critical flaws such as the premature convergence to suboptimal solutions and the limited exploration–exploitation balance. Experimental tests over several benchmark functions and images validate the efficiency of the proposed technique with regard to accuracy and robustness.

Chapter 9 presents an algorithm for the automatic detection of circular shapes embedded into cluttered and noisy images without considering conventional Hough transform techniques. The approach is based on a swarm technique known as the collective animal behavior (CAB). In CAB, searcher agents emulate a group of animals which interact with each other based on simple biological laws that are modeled as swarm operators. The approach uses the encoding of three non-collinear points embedded into an edge-only image as candidate circles. Guided by the values of the objective function, the set of encoded candidate circles (charged particles) are evolved using the CAB algorithm so that they can fit into actual circular shapes over the edge-only map of the image. Experimental evidence from several tests on synthetic and natural images which provide a varying range of complexity validates the efficiency of our approach regarding accuracy, speed, and robustness.

Finally, In Chap. 10, the swarm optimization algorithm of Locust Search (LS) is applied to a template-matching scheme. In the approach, the LS method is considered as a search strategy in order to find the pattern that better matches in the original image. According to a series of experiments, LS achieves the best results between estimation accuracy and computational load.

As authors, we wish to thank many people who were somehow involved in the writing process of this book. We express our gratitude to Prof. Lakhmi C. Jain, who so warmly sustained this project. Acknowledgments also go to Dr. Thomas Ditzinger and Varsha Prabakaran, who so kindly agreed to its appearance.

Guadalajara, Mexico

Erik Cuevas
Fernando Fausto
Adrián González

Contents

Chapter 1
An Introduction to Nature-Inspired Metaheuristics and Swarm Methods

Abstract Mathematical Optimization is an current problem in many different areas of science and technology; due to this, in the last few years, the interest on the development of methods for solving such kind of problems has increased an unprecedented way. As a result of the intensification in research aimed to the development of more powerful and flexible optimization tools, many different and unique approaches have been proposed and successfully applied to solve a wide array of real-world problems, but none has become as popular as the family of optimization methods known as nature-inspired metaheuristics. This compelling family of problem-solving approaches have become well-known among researchers around the world not only for to their many interesting characteristics, but also due to their ability to handle complex optimization problems, were other traditional techniques are known to fail on delivering competent solutions. Nature-inspired algorithms have become a world-wide phenomenon. Only in the last decade, literature related to this compelling family of techniques and their applications have experienced and astonishing increase in numbers, with hundreds of papers being published every single year. In this chapter, we present a broad review about nature-inspired optimization algorithms, highlighting some of the most popular methods currently reported on the literature as and their impact on the current research.

1.1 Optimization Techniques: A Brief Summary

Mathematical optimization is a branch of applied mathematics and computer sciences which deals with the selection of the optimal solution for a particular mathematical function (or problem) with the purpose of either minimizing or maximizing the output of such function. In more simple terms, optimization could be described as the process of selecting of the best element(s) from among a set of available alternatives to get the best possible results when solving a particular problem [1, 2]. Optimization is a recurring problem for many different areas of application such as robotics, computer networks, security, engineering design, data mining, finances, economics, and many others [2]. Independently of the area of application, optimization problems are

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_1

wide-ranging and numerous, so much that the development of methods for solving such problems has remained a hot topic for many years.

Traditionally, optimization techniques can be roughly classified as either *deterministic* or *stochastic* [3]. Deterministic optimization approaches, which design heavily relies on mathematical formulation and its properties, are known to have some remarkable advantages, such as fast convergence and implementation simplicity [4]. On the other hand, stochastic approaches, which resort to the integration of randomness into the optimization process, stand as promising alternatives to deterministic methods for being far less dependent on problem formulation and due to their ability to thoroughly explore a problems design space, which in turn allow them to overcome local optima more efficiently [5]. While both deterministic and stochastic methods have been successfully applied to solve a wide variety of optimization problems, these classical approaches are known to be subject to some significant limitations; first of all, deterministic methods are often conditioned by problem properties (such as differentiability in the case of gradient-based optimization approaches) [6]. Furthermore, due to their nature, deterministic methods are highly susceptible to get trapped into local optima, which is something undesirable for most (if not all) applications. As for stochastic techniques, while these are far easier to adapt to most black-box formulations or ill-behaved optimization problems, these methods tend to have a notably slower convergence speed in comparison to their deterministic counterparts, which naturally pose as an important limitation for applications where time is critical.

The many shortcomings of classical methods, along with the inherent challenges of real-life optimization problems, eventually lead researchers to the development of *heuristics* as an alternative to tackle such complex problems [1]. Generally speaking, a heuristic could be described as a technique specifically tailored for solving specific problems, often considered too difficult to handle with classic techniques. In this sense, heuristics trade essential qualities such as optimality, accuracy, precision or completeness to, either solve a problem in reasonably less time or to find an approximate solution in situations in which traditional methods fail to deliver an exact solution. However, while heuristic methods have demonstrated to be excellent to handle otherwise hard to solve problems, there are still subject to some issues. Like most traditional approaches, heuristics are usually developed by considering at least some specifications about the target problem, and as such, it is hard to apply them to different problems without changing some or most of their original framework [7].

Recently, the idea of developing methodologies that could potentially solve a wide variety of problems in a generic fashion has caught the attention of many researchers, leading to the development of a new breed of "intelligent" optimization techniques formally known as *metaheuristics* [8]. A metaheuristic is a particular kind of heuristic-based methodology, devised with the idea of being able to solve many different problems without the need of changing the algorithms basic framework. For this purpose, metaheuristic techniques employ a series of generic procedures and abstractions aimed to improve a set of candidate solution iteratively. With that being said, metaheuristics are often praised due to their ability to find adequate solutions for most problems independently of their structure and properties.

1.2 The Rise of Nature-Inspired Metaheuristics

The word "nature" refers to many phenomena observed in the physical world. It comprises virtually everything perceptible to our senses and even some things that are not as easy to perceive. Nature is the perfect example of adaptive problem solving; it has shown countless times how it can solve many different problems by applying an optimal strategy, suited to each particular natural phenomenon. Many researchers around the world have become captivated by how nature can adapt to such an extensive array of situations, and for many years they have tried to emulate these intriguing problem-solving schemes to develop tools with real-world applications. In fact, for the last two decades, nature has served as the most important source of inspiration in the development of metaheuristics. As a result of this, a whole new class of optimization techniques was given birth in the form of the so-called Nature-inspired optimization algorithms. These methods (often referred as bio-inspired algorithms) are a particular kind of metaheuristics, developed with a single idea in mind: mimicking a biological or a physical phenomenon to solve optimization problems. With that being said, depending on their source of inspiration, nature-inspired metaheuristics can be classified in four main categories: evolution-based, swarm-based, physics-based and human-based methods [9, 10]. Evolution-based methods are developed by drawing inspiration in the laws of natural evolution. From these methods, the most popular is without a doubt the Genetic Algorithms approach, which simulates Darwinian evolution [11]. Other popular methods grouped within this category include Evolution Strategy [12], Differential Evolution [13] and Genetic Programming [14]. On the other hand, swarm-based techniques are devised to simulate the social and collective behavior manifested by groups of animals (such as birds, insects, fishes, and others). The Particle Swarm Optimization [15] algorithm, which is inspired in the social behavior of bird flocking, stands as the most representative and successful example within this category, although other relevant methods include Ant Colony Optimization [16], Artificial Bee Colony [17], Firefly Algorithm [18], Social Spider Optimization [19], among others. Also, there are the physics-based algorithms, which are developed with the idea of emulating the laws of physics observed within our universe. Some of the most popular methods grouped within this category are Simulated Annealing [20], Gravitational Search Algorithm [21], Electromagnetism-like Mechanism [22], States of Matter Search [23], to name a few. Finally, we can mention human-based algorithms. These kind of nature-inspired methods are unique due to the fact that they draw inspiration from several phenomena commonly associated with humans' behaviors, lifestyle or perception. Some of the most well-known methods found in the literature include Harmony Search [24], Firework Algorithm [25], Imperialist Competitive Algorithm [26], and many more.

Most nature-inspired methods are modeled as population-based algorithms, in which a group of randomly generated search agents (often referred as individuals) explore different candidate solutions by applying a particular set of rules derived from some specific natural phenomenon. This kind of frameworks offer important advantages in both, the interaction among individuals, which promotes a wider knowledge about different solutions, and the diversity of the population, which is an important aspect on ensuring that the algorithm has the power to efficiently explore the design space while also being able to overcome local optima [8]. Due to this and many other distinctive qualities, nature-inspired methods have become a popular choice among researchers. As a result, literature related to nature-inspired optimization algorithms and its applications for solving otherwise challenging optimization problems has become extremely vast, with hundreds of new papers being published every year.

In this chapter, we analyze some of the most popular nature-inspired optimization methods currently reported on the literature, while also discussing their impact on the current literature. The rest of this chapter is organized as follows: in Sect. 1.2, we analyze the general framework applied by most nature-inspired metaheuristics in terms of design. In Sect. 1.3, we present nature-inspired methods according to their classification while also reviewing some of the most popular algorithms for each case. Finally, in Sect. 1.4, we present a brief study concerning the growth in the number of publications related to nature-inspired methods.

1.3 General Framework of a Nature-Inspired Metaheuristic

With some exceptions, most of the nature-inspired metaheuristics currently reported on the literature are modeled as population-based algorithms, which implies that the general framework employed by most of these methods remains almost identical, independently of the natural phenomenon from which the algorithm is inspired [2].

Usually, the first step of a nature-inspired algorithm involves the definition of a set of N randomly initialized solutions $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ (commonly referred as *population*), and such that:

$$\mathbf{x}_i = \left[x_{i,1}, x_{i,2}, \ldots, x_{i,d}\right] \tag{1.1}$$

where the elements $x_{i,n}$ represent the decision variables (parameters) related to a given optimization problem, while d denotes the dimensionality (number of decision variables) of the target solution space.

From an optimization point of view, each set of parameters $\mathbf{x}_i \in \mathbf{X}$ (also known as an *individual*) is considered as a candidate solution for the specified optimization task; as such, each of these solutions is also assigned with a corresponding quality value (or *fitness*) related to the objective function $f(\cdot)$ that describes the optimization task, such that:

$$f_i = f(\mathbf{x}_i) \tag{1.2}$$

Nature-inspired methods usually follow an iterative search scheme, in which new candidate solutions are generated by modifying currently available individuals; this is achieved by applying some previously specified criteria (usually devised by drawing inspiration from an observed natural phenomenon). For most cases, this process may be illustrated by the following expression:

$$\mathbf{x}_i' = \mathbf{x}_i + \Delta\mathbf{x}_i \tag{1.3}$$

where $\mathbf{x}_i'$ denotes the candidate solution generated by adding up a specified update vector $\Delta\mathbf{x}_i$ to $\mathbf{x}_i$. It is worth noting that the value(s) adopted by the update vector $\Delta\mathbf{x}_i$ depend on the specific operators employed by each individual algorithm.

Finally, most nature-inspired algorithms include some kind of selection process, in which the newly generated solutions are compared against those in the current population $\mathbf{X}^k$ (with k denoting the current iteration) in terms of solution quality, typically with the purpose of choosing the best individual(s) among them. As a result of this process, a new set of solutions $\mathbf{X}^{k+1} = \left\{\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \ldots, \mathbf{x}_n^{k+1}\right\}$, corresponding to the following iteration (or *generation*) '$k + 1$', is generated.

This whole process is iteratively repeated until a particular stop criterion is met (i.e., if a maximum number of iterations is reached). Once this happens, the best solution found by the algorithm is reported as the best approximation for the global optimum [2].

1.4 Classification of Nature-Inspired Metaheuristics

Nature-Inspired optimization algorithms have become so numerous and so varied that illustrating every single method in existence has become an undoubtedly challenging task. However, several algorithms have become widely popular among researchers, either for their fascinating characteristics or their ease of implementation. In this section, we present some of the most popular nature-inspired optimization techniques currently reported on the literature. The algorithms presented in this section were chosen by considering a balance between both classical and modern approaches. Also, in order to give the reader the facility to understand, analyze and compare each of the described methods in the same terms, we have taken some liberties regarding nomenclature and formulation presented on each case so that it is consistent with the general framework of nature-inspired methods presented in Sect. 1.3. While the introduced formulations may look slightly different to those reported on their sources, we have made a special effort to keep the essence and particular traits that distinguish each method unaltered; with that being said, the reader is always invited to refer to the original paper(s) in order to get a deeper understanding of these techniques. All approaches described in this section are presented according to the typical classification of to nature-inspired metaheuristics (see Fig. 1.1).

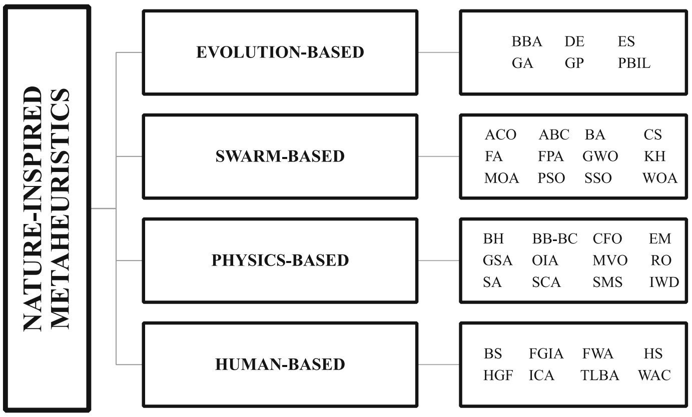

NATURE-INSPIRED METAHEURISTICS
EVOLUTION-BASED
BBA DE ES GA GP PBIL
SWARM-BASED
ACO ABC BA CS FA FPA GWO KH MOA PSO SSO WOA
PHYSICS-BASED
BH BB-BC CFO EM GSA OIA MVO RO SA SCA SMS IWD
HUMAN-BASED
BS FGIA FWA HS HGF ICA TLBA WAC

Fig. 1.1 Classification of nature-inspired metaheuristics

1.4.1 Evolution-Based Methods

Evolution-Based methods comprise a series of optimization algorithms developed by drawing inspiration in the laws of natural evolution. In this kind of techniques, solutions are typically represented by a set of individuals, which compete and combine in ways that allow only the most suitable individuals to prevail. The process for modifying existent solutions in evolution-based techniques often involve the implementation of a series of operators inspired in several processes commonly observed in natural evolution, such as *crossover*, *mutation*, and *selection*.

1.4.1.1 Differential Evolution

The Differential Evolution (DE) approach is an evolutionary algorithm introduced by Rainer Storn and Kenneth Price in 1996 [13] and, along with Genetic Algorithms (GA), is one of the most popular optimization approaches inspired in the evolution phenomena.

At each generation 'k', DE applies a series of mutation, crossover and selection operators in order to allow a population of solutions $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$ to "evolve" toward an optimal solution. For DE's mutation operation, new candidate (mutant) solutions $\mathbf{m}_i^k = \left[m_{i,1}^k, m_{i,2}^k, \ldots, m_{i,d}^k\right]$ are generated for each individual $\mathbf{x}_i$ as illustrated as follows:

$$\mathbf{m}_i^k = \mathbf{x}_{r_3}^k + F\left(\mathbf{x}_{r_1}^k - \mathbf{x}_{r_2}^k\right) \tag{1.4}$$

where $r_1, r_2, r_3 \in \{1, 2, \ldots, N\}$ (and with $r_1 \neq r_2 \neq r_3 \neq i$) each denote a randomly chosen solution index, while the parameter $F \in [0, 2]$ is called differential weight, and is used to control the magnitude of the differential variation $\left(\mathbf{x}_{r_1}^k - \mathbf{x}_{r_2}^k\right)$.

Furthermore, for the crossover operation, DE generates a trial solution vector $\mathbf{u}_i^k = \left[u_{i,1}^k, u_{i,2}^k, \ldots, u_{i,d}^k\right]$ corresponding to each population member 'i'. The components $u_{i,n}^k$ in such a trial vector are given by combining both the candidate solution $\mathbf{x}_i^k$ and its respective mutant solution $\mathbf{m}_i^k$ as follows:

$$u_{i,n}^k = \begin{cases} m_{i,n}^k & \text{if (rand} \leq CR) \text{ or n} = n^* \\ x_{i,n}^k & \text{if(rand} > CR) \text{ otherwise} \end{cases} \quad \text{for } n = 1, 2, \ldots, d \tag{1.5}$$

where $n^* \in \{1, 2, \ldots, d\}$ denotes a randomly chosen dimension index, while rand stand for a random number from within the interval [0, 1]. Furthermore, the parameter $CR \in [0, 1]$ represents the DE's crossover rate which is used to control the probability of an element $u_{i,n}^k$ being given by either a component from the candidate solution $\mathbf{x}_i^k$ $\left(x_{i,n}^k\right)$ or a component from the mutant solution $\mathbf{m}_i^k$ $\left(m_{i,n}^k\right)$.

Finally, for DE's selection process, each trail solution $\mathbf{u}_i^k$ is compared against its respective candidate solution $\mathbf{x}_i^k$ in terms of solution quality (fitness) by applying a greedy criterion. This means that, if the trial solution $\mathbf{u}_i^k$ yields to a better fitness value than $\mathbf{x}_i^k$, then the value of the candidate solution for the next generation '$k+1$' takes the value of $\mathbf{u}_i^k$, otherwise, it remains unchanged. This is:

$$\mathbf{x}_i^{k+1} = \begin{cases} \mathbf{u}_i^k & \text{if} \left(f\left(\mathbf{u}_i^k\right) > \text{if } f\left(\mathbf{x}_i^k\right)\right) \\ \mathbf{x}_i^k & \text{otherwise} \end{cases} \tag{1.6}$$

As one of the most popular Evolution-based algorithms currently reported on the literature, DE has been extensively studied and applied by researchers in many different areas of science [27–32].

1.4.1.2 Evolution Strategies

Evolution Strategies (ES) are a series of optimization techniques which draw inspiration from natural evolution [12]. The first ES approach was introduced by Ingo Rechenberg in the early 1960s and further developed during the 1970s. The most straightforward ES approach is the so-called (1 + 1)-ES (or two-membered ES). This approach considers the existence of only a single parent $\mathbf{x} = [x_1, x_2, \ldots, x_d]$, which is assumed to be able to produce a new candidate solution (offspring) $\mathbf{x}' = \left[x_1', x_2', \ldots, x_d'\right]$ by means of mutation as follows:

$$\mathbf{x}' = \mathbf{x} + \mathbf{N}(0, \sigma) \tag{1.7}$$

where $\mathbf{N}(0, \sigma)$ denotes a d-dimensional random vector whose values are drawn from a Gaussian distribution of mean 0 and fixed standart deviation σ (although later approaches consider a dynamic value based on the number of successful mutations) [12].

Furthermore, the (1 + 1)-ES implements a selection operator which allows excluding the individual with the least performance between the parent $\mathbf{x}$ and its respective offspring $\mathbf{x}'$, so that only the best of these solution is considered as the parent for the next generation (iteration).

In later approaches, Rechemberg introduced the concept of population to ES by proposing the first multimembered ES in the form of the so-called $(\mu + 1)$-ES. In such an approach, a population $\mathbf{P} = \{\mathbf{I}_1, \ldots, \mathbf{I}_\mu\}$ consisting on $\mu > 1$ parents $\mathbf{I}_i = \{\mathbf{x}_i, \boldsymbol{\sigma}_i\}$ (with $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \cdots x_{i,d}]$ and $\boldsymbol{\sigma}_i = [\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,d}]$) is considered. Furthermore, a discrete recombination mechanism which considers information drawn from a pair of randomly chosen parent is implemented to generate a new offspring $\mathbf{I}' = \{\mathbf{x}', \boldsymbol{\sigma}'\}$ as follows:

$$x'_j = \begin{cases} x_{r_1,n} & \text{if (rand} > 1/2) \\ x_{r_2,n} & \text{otherwise} \end{cases} \quad \text{for } n = 1, 2, \ldots, d \tag{1.8}$$

$$\sigma'_j = \begin{cases} \sigma_{r_1,n} & \text{if (rand} > 1/2) \\ \sigma_{r_2,n} & \text{otherwise} \end{cases} \quad \text{for } n = 1, 2, \ldots, d \tag{1.9}$$

where $r_1, r_2 \in \{1, \ldots, \mu\}$ denote two randomly chosen solution indexes corresponding to the parents population, while rand stand for a random number from within the interval [0,1].

Similarly to (1 + 1)-ES, $(\mu + 1)$-ES also implements a mutation operator which generate a new offspring solution by perturbing a currently existing parent. Furthermore, a selection operator which allows to choose the best μ solutions from among the population of parents and offspring (generated through recombination and mutation) is also implemented to define a new parents' population for the next generation.

Later approaches, such as the $(\mu + \lambda)$-ES and the (μ, λ)-ES were further proposed to consider the generation of multiple offspring rather than a single one [12]. Furthermore, several variations to the recombination and mutation processes employed on classical ES have also been proposed, giving birth to some interesting variants such as (μ, λ)-MSC-ES and CMA-ES, the latter of which is considered by many authors as the state-of-the-art in ES [33–36].

1.4.1.3 Genetic Algorithms

Genetic Algorithms (GA) is one of the earliest metaheuristics inspired in the concepts of natural selection and evolution and is among the most successful Evolutionary Algorithms (EA) due to its conceptual simplicity and easy implementation [37]. GA was initially developed by John Henry Holland in 1960 (and further extended in 1975) with the goal to understand the phenomenon of natural adaptation, and how this mechanism could be implemented into computers systems to solve complex problems.

In GA, a population of N solutions $\mathbf{x}_i = \left[x_{i,1}, x_{i,2}, \ldots, x_{i,d}\right]$ is first initialized; each of such solutions (called *chromosomes*) comprises a bitstring (this is, $x_{i,n} \in \{0, 1\}$), which further represents a possible solution for a particular binary problem. At each iteration (also called generation) of GA's evolution process, the chromosome population is modified by applying a set of three evolutionary operators, namely: selection, crossover and mutation. For the selection operation, GA randomly selects a pair of chromosomes $\mathbf{x}_{p_1}$ and $\mathbf{x}_{p_2}$ (with $p_1, p_2 \in \{1, 2, \ldots, N\}$ and $p_1 \neq p_2$) from within the entire chromosome population, based on their individual selection probabilities. The probability P_i for a given chromosome 'i' ($\mathbf{x}_i$) to be selected depends on its quality (fitness value), as given as follows:

$$P_i = \frac{f(\mathbf{x}_i)}{\sum_{j=1}^{N} f\left(\mathbf{x}_j\right)} \tag{1.10}$$

Then, for the crossover operation, the bitstring information of the selected chromosomes (now called parents) is recombined to produce two new chromosomes, $\mathbf{x}_{s_1}$ and $\mathbf{x}_{s_2}$ (referred as *offspring*) as follows:

$$x_{s_1,n} = \begin{cases} x_{p_1,n} & \text{if } (n < l) \\ x_{p_2,n} & \text{otherwise} \end{cases} \quad x_{s_2,n} = \begin{cases} x_{p_2,n} & \text{if} (n < l) \\ x_{p_1,n} & \text{otherwise} \end{cases} \quad \text{for } n = 1, 2, \ldots, d. \tag{1.11}$$

where $l \in \{1, 2, \ldots, d\}$ is a randomly selected pivot index (usually referred as *locus*).

Finally, for the mutation operation, some elements (bits) of the newly generated offspring are flipped (changed from 1 to 0 or vice versa). Mutation can occur over each bit position in the string with a particular probability P_m (typically as low as 0.001), as given as follows:

$$x_{s_r,n} = \begin{cases} \bar{x}_{s_r,n} & \text{if (rand} < P_m) \\ x_{s_r,n} & \text{otherwise} \end{cases} \quad \text{for } n = 1, 2, \ldots, d \tag{1.12}$$

where $x_{s_r,n}$ (with $r \in \{1, 2\}$) stand for the jth element (bit) of the s_rth offspring, while rand stand for a randomumber form within the interval of 0 and 1.

This process of selection, crossover, and mutation of individuals takes place until a population of N new chromosomes (mutated offspring) has been produced, and then, the N best chromosomes among the original and new populations are taken for the next generation, while the remainder individuals are discarded [38–41].

1.4.1.4 Genetic Programming

Genetic Programming (GP) is a unique optimization technique proposed by John R. Koza in 1992 [14]. The development of GP is closely related to popularization gained by evolutionary algorithms between the 1960s and 1970s. In essence, GP can be considered an extension of evolutionary methods such as Rechenberg's Evolution

Strategies (ES) or Holland’s Genetic Algorithms (GA). Different to said traditional methods, however, is the fact that in GP solutions are represented by a set of operations or computer programs; this is that, instead of finding a set of decision variables that optimize a given objective function, the outputs of GP are computer programs specifically tailored (evolved) to perform optimally on predefined tasks. Traditionally, solutions in GP are represented as tree structures which group a set of functions and operands, although other representations are also common. Furthermore, GP is also distinctive due to its variable-length representation of output solutions, which drastically differs to the fixed-length representations adopted by most traditional techniques [42–46].

Typically, most GP approaches are comprised of the following four fundamental steps:

1. Generating an initial population of computer programs, composed by the available functions and terminals (operands).
2. Execute each program in the population and assign it a fitness value according to how well it solves a given problem.
3. Generate a new population of programs by:
    a. Copying the current best computer programs (reproduction).
    b. Creating new offspring programs by randomly changing some parts of a program (mutation).
    c. Creating new offspring programs by recombining parts from two existent programs (crossover).
4. If a specified stop criterion is met, return the single best program in the population as the solution for the pre-specified problem. Otherwise, return to step 2.

1.4.2 Swarm-Based Methods

Swarm-based optimization algorithms comprise a series of techniques which draw inspiration from the collective behavior manifested by a wide range of living organisms, such as birds, insects, fishes, and others. In this kind of techniques, search agents are modeled by a population of individuals (usually from the same species) which are capable of interacting with each other and the environment that surrounds them. While the movement of search agents in swarm algorithms is often based on simplified behavioral rules abstracted from those observed in nature, the collective manifestation of these individual conducts allows the entire population to exhibit global and complex behavioral patterns, thus allowing them to explore an extensive amount of candidate solutions.

1.4.2.1 Ant Colony Optimization

The Ant Colony Optimization (ACO) algorithm is one of the most well-known nature-inspired metaheuristics. The ACO approach was first proposed by Marco Dorigo in 1992 under the name of Ant Systems (AS) and draws inspiration in the natural behavior of ants [47]. In nature, ants move randomly while foraging for food, and when an appropriate source is found, they return to their colony while leaving a pheromone trail behind. Ants are able to guide themselves toward previously found food source by following the path traced by pheromones left by them or other ants. However, as time passes, pheromones start to evaporate; intuitively, the more time an ant takes to travel down a given path back and forth, the more time the pheromones have to dissipate; on the other hand, shorter paths are traversed more frequently, promoting that pheromone density becomes higher in comparison to that on longer routes. In this sense, if an ant finds a good (short) path from the colony to a food source, others members are more likely to follow the route traced by said ant. The positive feedback provided by the increase in pheromone density through paths traversed by an increasing number of ants eventually lead all members of the colony to follow a single optimal route [16].

The first ACO approach was conceived as an iterative process devised to handle the task of finding optimal paths in a graph [47]. For this purpose, ACO considers a population of N ants which move through the nodes and arcs of a graph $G(\mathcal{N}, \mathcal{P})$ (with $\mathcal{N}$ and $\mathcal{P}$ denoting its respective sets of nodes and arcs, respectively). Depending on their current state (node), each ant is able to choose from among a set of adjacent paths (arc) to traverse based on the pheromone density and length associated to each of them. With that being said, at each iteration 'k', the probability for a given ant 'i' to follow a specific path 'xy' (which connects states 'x' and 'y') is given by the following expression:

$$p^k_{i_{(xy)}} = \frac{\left(\alpha \cdot \tau^k_{(xy)}\right)\left(\beta \cdot \eta^k_{(xy)}\right)}{\sum_{z \in \mathbf{Y}_x} \left(\alpha \cdot \tau^k_{(xz)}\right)\left(\beta \cdot \eta^k_{(xz)}\right)} \tag{1.13}$$

where, $\tau^k_{(xy)}$ denotes the pheromone density over the given path 'xy', while $\eta^k_{(xy)}$ stand for the preference for traversing said path, which is relative to its distance (cost). Furthermore, $\mathbf{Y}_x$ represent the set of all adjacent states for the given current state 'x'. Finally, α and β are constant parameters used to control the influence of $\tau^k_{(xy)}$ and $\eta^k_{(xy)}$, respectively.

By applying this mechanism, each ant moves through several paths within the graph until a specific criterion is met (i.e., that a particular destination node has been reached). Once this happens, each ant backtracks its traversed route while releasing some pheromones on each the paths they used. In ACO, the amount of pheromones released by an ant 'i' over any given path 'xy' is given by:

$$\Delta\tau^{k}_{i_{(xy)}} = \begin{cases} Q/L_i & \text{if ''the ant used the path xy in its tour''} \\ 0 & \text{otherwise} \end{cases} \tag{1.14}$$

where, L_i denotes the length (cost) associated to the route taken by the ant 'i', while Q stand for a constant value.

Finally, ACO includes a procedure used to update the pheromone density over all paths in the graph for the following iteration $(k + 1)$. For this purpose, it considers both, the amount of pheromones released by each ant while backtracking its traced route and the natural dissipation of pheromones which takes place as time passes. This is applied by considering the following expression:

$$\tau^{k+1}_{(xy)} = (1-\rho)\cdot\tau^{k}_{(xy)} + \sum_{i=1}^{N}\Delta\tau^{k}_{i_{(xy)}} \tag{1.15}$$

where ρ is a constant value known as pheromone evaporation coefficient, while $\Delta\tau^{k}_{i_{(xy)}}$ stand for the amount of pheromones released by an ant 'i' over a specific path 'xy' (as given by Eq. 1.14).

1.4.2.2 Artificial Bee Colony

Bees are among the most well-known example of insects which manifest a collective behavior, either for food foraging or mating. Based on this premise, many researchers have proposed several different swarm intelligence approaches inspired by the behavior of bees. In particular, the Artificial Bee Colony (ABC) approach proposed by Dervis Karaboga and Bahriye Basturk in 2007 is known to be among the most popular of these bee-inspired methods [48].

In the ABC approach, search agents are represented as a colony of artificial honey bees which explore a d-dimentional search space while looking for optimal food (nectar) sources. The locations of these food sources each represent a possible solutions for a given optimization problem and their amount of nectar (quality) is related to the fitness value associated to each of such solutions. Furthermore, the members of the bee colony are divided in three groups: employed bees, onlooker bees and scout bees. Each of these groups of bees has distinctive functions inspired in the mechanics employed by bees while foraging for food. For example, the employed bees comprises the members of the colony which function is to explore the surroundings of individually-known food sources in the hopes of finding places with greater amounts of nectar. In addition, employed bees are able to share the information of currently known food sources with the rest of the members of the colony, so that they can also exploit them. With that being said, at each iteration 'k' of ABC's search process, each employed bee generates a new candidate solution $\mathbf{v}_i$ around a currently known food source $\mathbf{x}_i$ as follows:

$$\mathbf{v}^{k}_{i} = \mathbf{x}^{k}_{i} + \phi\left(\mathbf{x}^{k}_{i} - \mathbf{x}^{k}_{r}\right) \tag{1.16}$$

where $\mathbf{x}_i^k$ denotes the location of the food source remembered by a particular employed bee 'i' while $\mathbf{x}_r^k$ (with $r \neq i$) stands for the location of any other randomly chosen food source. Furthermore, ϕ is a random number drawn from within the interval $[-1, 1]$.

On the other hand, onlooker bees can randomly visit any food source known by the employed bees. For this purpose, each available food source is assigned with a certain probability of being visited by an onlooker bee as follows:

$$P_i^k = \frac{f\left(\mathbf{x}_i^k\right)}{\sum_{j=1}^{N} f\left(\mathbf{x}_j^k\right)} \tag{1.17}$$

Similarly to the employed bees, once an onlooker bee has decided to visit a particular food source, a new candidate solution $\mathbf{v}_i^k$ is generated around the chosen location $\mathbf{x}_i^k$ by applying Eq. (1.16). Furthermore, any candidate solution $\mathbf{v}_i^k$ generated by either an employed or an onlooker bee is compared against its originating location $\mathbf{x}_i^k$ in terms of solution quality, and then, the best among them is chosen as the new food source location for the following iteration; this is:

$$\mathbf{x}_i^{k+1} = \begin{cases} \mathbf{v}_i^k \text{ if } \left(f\left(\mathbf{x}_i^k\right) < f\left(\mathbf{v}_i^k\right)\right) \\ \mathbf{x}_i^k \text{ otherwise} \end{cases} \tag{1.18}$$

Finally, scout bees are the members of the colony whose function is to explore the whole terrain for new food sources randomly. Scout bees are deployed to look for new solutions only if a currently known food source is chosen to be "abandoned" (a thus forgotten by all members of the colony). In ABC, a solution is considered to be abandoned only if it cannot be improved by either the employed or onlooker bees after a determined number of iterations, indicated by the algorithm's parameter "$limit$". This mechanism is important for the ABC approach since it allows it to keep the diversity of solutions during the search process.

In general, ABC's local search performance may be attributed to the neighborhood exploration and greedy selection mechanisms applied by the employed and onlooker bees, while the global search performance is mainly related to the diversification attributes of scout bees.

1.4.2.3 Bat Algorithm

The Bat Algorithm (BA) is a bio-inspired metaheuristic proposed by Xin-SheYang in 2010. The BA approach draws inspiration on the behavior manifested by certain species of bats (particularly, microbats) [49]. In nature, most bats are equipped with a type of biologic sonar known as echolocation. In simple terms, the echolocation consists of two steps: the emission of loud frequency-modulated sound pulses and the reception (listening) of the echoing sounds that bounce back from surrounding objects, which essentially allows building a three-dimensional scenario of the immediate environment. Typically, bats use this specialized sonar system to assist themselves in several tasks, such as prey detection, obstacle evasion or even locating their roosting places.

In the BA approach, search agents are modeled as a swarm of bats whose position within the d-dimensional search space represent a possible solution for a given optimization problem. Furthermore, each bat is assumed to use echolocation to assist its movement toward a particular prey (modeled as the global best solution). For this purpose, the BA algorithm considers three sets of parameters whose values are constantly adjusted as the search process goes on: *frequencies*, *loudness* and *pulse emission rates*. In the case of the frequencies, these are modeled by a set of d-dimensional vectors, each associated to a given bat 'i' and whose values are randomly adjusted at each iteration 'k', such that:

$$\mathcal{F}_i^k = \mathcal{F}_{\text{min}} + \beta \cdot \left(\mathcal{F}_{\text{max}} - \mathcal{F}_{\text{min}}\right) \tag{1.19}$$

where the parameters $\mathcal{F}_{\text{min}}$ and $\mathcal{F}_{\text{max}}$ denote the minimum and maximum frequencies, respectively. Finally, β is a vector of random numbers each from within the interval [0, 1].

On the other hand, the loudness and pulse emission rates A_i and r_i, respectively, comprise a set of parameters whose initial values A_i^0 and r_i^0 are defined during the algorithms initialization. As the search process evolves, the values of these parameters are modified according to the following expression:

$$A_i^{k+1} = \alpha A_i^k, \quad r_i^{k+1} = r_i^0(1 - \exp(-\gamma k)) \tag{1.20}$$

where $\alpha < 1$ and $\gamma < 1$ are constant parameters.

With regard to the position update operators, the BA algorithm applies the following movement rule to update the location of each bat 'i' for the current iteration 'k':

$$\mathbf{x}_i^k = \mathbf{x}_i^{k-1} + \mathbf{v}_i^k \tag{1.21}$$

where $\mathbf{x}_i^{k-1}$ represents the position of the ith bat at the previous iterations $(k - 1)$, while $\mathbf{v}_i^k$ stands for the velocity of said bat, as given by the following expression:

$$\mathbf{v}_i^k = \mathbf{v}_i^{k-1} + f_i \cdot \left(\mathbf{x}_i^k - \mathbf{x}_{\text{best}}\right) \tag{1.22}$$

where $\mathbf{x}_{\text{best}}$ denotes for the current global best solution found during the search process, while $\mathcal{F}_i^k$ represents frequency vector associated to bat 'i', as given by Eq. (1.19):

Furthermore, BA also includes a local search scheme in which, at each iteration, a randomly chosen individual among the current best solutions is further refined by performing a random walk as follows:

$$\mathbf{x}_*^k = \begin{cases} \mathbf{x}_i^k + \varepsilon A_i^k & \text{if}\left(\text{rand} > r_i^k\right) \\ \mathbf{x}_i^k & \text{otherwise} \end{cases} \tag{1.23}$$

where the parameters A_i^k and r_i^k denote the loudness and pulse emission rates associated to the randomly chosen bat 'i', while rand stand for a random number drawn from within the uniformly distributed interval [0, 1]. Finally, it is worth noting that the newly generated solution $\mathbf{x}_*^k$ is accepted as the new position for bat 'i' $\left(\mathbf{x}_i^k\right)$ only if certain conditions are met; particularly:

$$\mathbf{x}_i^k = \begin{cases} \mathbf{x}_*^k & \text{if}\left(\text{rand} < A_i^k \text{ and } f\left(\mathbf{x}_i^k\right) < f\left(\mathbf{x}_*^k\right)\right) \\ \mathbf{x}_i^k & \text{otherwise} \end{cases} \tag{1.24}$$

1.4.2.4 Crow Search Algorithm

The Crow Search Algorithm (CSA), as proposed by Alireza Askarzadeh in 2016, is a nature-inspired optimization method which draws inspiration in the behavior of flocking crows [50]. Crows are considered by many as the most intelligent of birds; they are mainly known for hiding their excess food in specific locations around their environment, and when needed they can accurately remember the location of their hidden food sources. Crows are also known for its tendency to commit thievery. To do so, they observe and follow other crows to find their hiding places, and then, they steal their resources once its owner leaves. Also, from their experience as thieves, crows are able to develop different tactics to prevent their hiding places from being pilfered by other crows, such as moving their hiding places to other locations or even tricking other birds to follow them to a different place.

In CSA, search agents are modeled as a flock comprised by N crows, each with a particular positions $\mathbf{x}_i = \left[x_{i,1}, x_{i,2}, \ldots, x_{i,d}\right]$ within a feasible d-dimentional solution space. Each of these crows is also assumed to have a memory, which allows them to remember the location of their respective food hiding places. For this purpose, CSA assigns each crow 'i' with a memory solution $\mathbf{m}_i = \left[m_{i,1}, m_{i,2}, \ldots, m_{i,d}\right]$, which represents the best solution found so far by said crow 'i'. The movement operators employed by CSA to update the position of each crow considers two different possibilities: (1) the case where a crow 'i' follows some randomly chosen crow 'j' to their hiding place, and (2) the situation in which said crow 'i' is deceived by crow 'j' into moving to a different location. With that being said, at each iteration 'k', CSA updates the position of each crow as follows:

$$\mathbf{x}_i^{k+1} = \begin{cases} \mathbf{x}_i^k + r_i \cdot fl_i^k \cdot \left(\mathbf{m}_j^k - \mathbf{x}_i^k\right) & \text{if}\left(\text{rand} \geq AP_j^k\right) \\ \mathbf{r}_i & \text{if}\left(\text{rand} < AP_j^k\right) \end{cases} \tag{1.25}$$

where AP_j^k denotes awareness probability of crow 'j' at iteration 'k', whereas rand stand for a random number drawn from a uniform distribution within the interval

[0, 1]. The position update operator applied when rand(0, 1) $\geq AP_j^k$ denotes the situation in which a crow 'j' is not aware that a crow 'i' is following it and, as a result, crow 'i' approaches toward the hiding place of sais crow 'j' $\left(\mathbf{m}_j^k\right)$. In this sense, the parameter fl_i^k denotes the maximum flight length (step size) of crow 'i', while $r_i \in [0, 1]$ stand for a random step factor. On the other hand, if rand $< AP_j^k$, it is assumed that crow 'j' is aware that it is being followed by the crow 'i', and in response, it will fool said pilferer into moving to a different location. With that being said, the position of crow 'i' is updated to $\mathbf{r}_i$, which stand for a randomly generated solution within the feasible decision space.

1.4.2.5 Cuckoo Search

Cuckoo birds are well-known due to they're aggressive, yet fascinating reproduction strategies. In particular, a good number of cuckoo species are known to resort to brood parasitism as part of their life-cycle. Essentially, cuckoos secretly lay their eggs in the nest of other birds (typically of a different species) in the hopes of deceiving the host into thinking that such eggs are their own. Should these alien eggs succeed to be undetected by the host bird, they are almost guaranteed to hatch into new cuckoo chicks; otherwise, the host bird may opt to remove such aliens eggs of their nest or even abandon the nest to build a new one somewhere else. Inspired by this intriguing behavior, in 2009, Xin-She Yang and Suash Deb proposed the nature-inspired algorithm known as Cuckoo Search (CS) [51].

In the CS approach, solutions are modeled as a set of N host nests. To simulate the situation in which a cuckoo bird chooses a host nest to lay their own eggs, at each iteration 'k', new candidate solutions are generated around randomly selected host nests 'i' by applying a random walk as follows:

$$\mathbf{x}_{\text{new}}^k = \mathbf{x}_i^k + \alpha \cdot \text{Lévy}(\lambda) \tag{1.26}$$

where Lévy(λ) denotes a random walk via Leví flights [8], while $\alpha > 0$ stand for a vector of step sizes related to the scale of the objective solution space.

Once a new candidate solution has been generated, its quality (fitness) is compared to that of the solution represented by the randomly selected host nest. If the quality of a candidate solution $\mathbf{x}_{\text{new}}^k$ is better than $\mathbf{x}_i^k$ related to the host nest 'i', then $\mathbf{x}_{\text{new}}^k$ replaces $\mathbf{x}_i^k$ for the next iteration. This is:

$$\mathbf{x}_i^{k+1} = \mathbf{x}_{\text{new}}^k \tag{1.27}$$

Furthermore, after new solutions have been generated and evaluated for each randomly selected host nest, a fraction p_a of the worst remaining solutions are eliminated, and then, replaced by an equal amount of randomly generated solutions. This mechanism is intended to simulate the nest abandonment behavior manifested by host birds once they discover the presence of cuckoo eggs within their nest, which further promotes them to build a new nest on a different place.

An important trait about the CS approach is that the definition of host nest can further be modified so that each of them represents a set of solutions (multiple eggs within each nest) instead of a single one which could effectively allow CS to be extended into a type meta-population algorithm, and thus, be applied to solve even more complex optimization problems.

1.4.2.6 Firefly Algorithm

The Firefly Algorithm (FA), as proposed by Xin-SheYang in 2010, is a swarm intelligence approach inspired by the light-emitting behavior observed in fireflies [49]. In the FA approach, search agents, modeled as a swarm of fireflies, are assumed to be able to interact with each other through its characteristic bioluminescent glowing. In particular, such interactions are modeled as attractions toward other conspecific individuals within the target solution space. The attractiveness of each firefly is considered to be proportional to its light intensity, which in turn is also said to be equivalent to their quality (fitness). Furthermore, it is also assumed that fireflies are only attracted toward brighter individuals; this is, that for any two flashing fireflies, the less bright one will be attracted toward the brighter one (regardless of their gender). Also, it is also assumed that the attractiveness "perceived" by a particular firefly 'i' toward any other individual source 'j' decreases as the distance that separates them increases. Such phenomenon is simplified by the following expression:

$$\beta_{ij} = \begin{cases} \beta_{0j} \cdot e^{-\gamma r_{ij}^2} & \text{if } f(\mathbf{x}_j) > f(\mathbf{x}_i) \\ 0 & \text{otherwise} \end{cases} \tag{1.28}$$

where $r_{ij} = \|\mathbf{x}_j - \mathbf{x}_i\|$ stand for the Euclidian distance computed with regard to the pair of fireflies 'i' and 'j', while β_{0j} denotes the *attractiveness* of individual 'j'. Furthermore, the parameter γ stand for the so-called *light absorption coefficient* which is used to further vary the overall attractiveness toward the individual 'j'.

Finally, the attraction movement performed by a firefly 'i' toward any given individual 'j' may be given by the following expression:

$$\Delta\mathbf{x}_{ij} = \mathbf{x}_i + \beta_{ij} \cdot (\mathbf{x}_j - \mathbf{x}_i) + \alpha \cdot \boldsymbol{\epsilon}_i \tag{1.29}$$

where $\boldsymbol{\epsilon}_i$ denotes a d-dimensional vector denoting a random movement, while α stands for a randomization parameter, which values are typically within the interval [0, 1].

As noted by its author, the attraction behaviors represented in FA enables search agents to explore the solution space of a given optimization problem more effectively. Another interesting property of FA is that it can effectively locate both global and local optima, which could be an important advantage on certain implementations.

1.4.2.7 Flower Pollination Algorithm

The Flower Pollination Algorithm (FPA) is another popular optimization method in Yang's showcase of nature-inspired metaheuristics. First proposed in 2012, the FPA approach is well-known for drawing inspiration in the intriguing pollination process observed in most species of flowers [52].

In the FPA approach, solutions are modeled as individual flowers (or pollen gametes). Furthermore, FPA considers two natural pollination methods in order to establish a set of valid movement rules: cross-pollination, which refers to the flower reproduction process in which pollen is carried over long distances by pollinators (such as insects, birds, or other animals) as a way to ensure the reproduction of fittest plants, and self-pollination, which emphasize the fertilization process that occurs among flowers in the absence of viable pollinators. In the context of swarm intelligence, FPA considers cross-pollination as a global search process, while self-pollination is seen as a local search approach. Furthermore, it also assumes that at each iteration 'k' each particular flower is only able to produce a single pollen gamete per iteration. By considering these idealized characteristics, the FPA's position update operators may be illustrated as follows:

$$\mathbf{x}_i^{k+1} = \begin{cases} \mathbf{x}_i^k + L\left(\mathbf{x}_* - \mathbf{x}_i^k\right) & \text{if (rand} > P) \\ \mathbf{x}_i^k + \epsilon\left(\mathbf{x}_j^k - \mathbf{x}_q^k\right) & \text{if (rand} \leq P) \end{cases} \tag{1.30}$$

where rand denotes a random number from within the interval [0, 1].

The position update rule applied when rand $> P$ (with P denoting a probability threshold) stand for a global pollination movement rule. In this case $\mathbf{x}_*$ denotes the current global best solution, while L represents a scaling parameter known as pollination strength, which value is given from a Levy distribution, as given by:

$$L \sim \frac{\lambda\Gamma(\lambda)\cdot\sin(\pi\lambda/2)}{\pi\left(s^{1+\lambda}\right)}, \quad (s \gg s_0 > 0) \tag{1.31}$$

where $\Gamma(\lambda)$ stand for the standard gamma function (with regard to the constant parameter λ), while s stand for a user-defined step size.

On the other hand, the movement operator corresponding to the case when rand $\leq p$ represents a local pollination movement rule, which can be essentially considered as a random walk. In such a case, $\mathbf{x}_j^k$ and $\mathbf{x}_q^k$ (with $j \neq q$) represent the positions of two randomly chosen flowers (different to $\mathbf{x}_i^k$), while ϵ is a random value drawn from within the interval [0, 1].

1.4.2.8 Grey Wolf Optimizer

The Grey Wolf Optimizer (GWO) is a nature-inspired metaheuristic proposed by Mirjalili et al. in 2014. As its name may imply, GWO's design is inspired by the distinctive hunting behaviors and social hierarchy observed in packs of grey wolves [53]. Grey wolves are known to be organized in a very strict social dominant hierarchy composed of alpha, beta delta, and omega wolves. The alpha wolves (typically comprised by a male and a female member) are the leaders of the pack and are responsible for all decisions related to hunting, resting and moving. The beta wolves, which comprise the second level on the pack's hierarchy, are subordinate to the alpha wolves but can give commands to lower-level wolves. Similarly, delta wolves are subordinate to both alpha and beta members, yet they dominate over the all other wolves. Finally, the omega wolves, which are the lowest-ranked members, play as subordinates to the dominant wolves (alpha, beta, and delta), and as such follow their instructions (especially while hunting). Another important trait observed in grey wolves is related to their cooperative hunting tactics in which wolves start to chase an identified prey until it is encircled, and then they proceeded to attack the prey until it is finally killed. These distinctive traits are mathematically modeled in GWO for the task of solving global optimization problems [53].

Analogous to the social hierarchy observed in real grey wolves, GWO identifies each search agent as either alpha, beta, delta or omega wolf depending on their current fitness; in particular, the individual $\mathbf{x}_i$ which represent the fittest solution is considered as the alpha wolf (α). Similarly, the second and third best solutions are designated as beta (β) and delta (δ) respectively, while all remaining wolves are labeled as omega members (ω). Also, WOA considers a hierarchy-based search scheme in which lower-ranked wolves move based on information shared by higher-ranked individuals. As such, at each iteration 'k', each wolf updates their position for the following iteration '$k+1$' as follows:

$$\mathbf{x}_i^{k+1} = \frac{\boldsymbol{\alpha}_i^k + \boldsymbol{\beta}_i^k + \boldsymbol{\delta}_i^k}{3} \tag{1.32}$$

where $\boldsymbol{\alpha}_i^k$, $\boldsymbol{\beta}_i^k$ and $\boldsymbol{\delta}_i^k$ each denote the position update applied to wolf 'i' $\left(\mathbf{x}_i^k\right)$ with regard to the positions occupied by the α, β and δ wolves, respectively, and are given by:

$$\boldsymbol{\alpha}_i^k = \mathbf{x}_\alpha^k - \mathbf{A}^k \cdot \left|\mathbf{C}^k \cdot \mathbf{x}_\alpha^k - \mathbf{x}_i^k\right| \tag{1.33}$$

$$\boldsymbol{\beta}_i^k = \mathbf{x}_\beta^k - \mathbf{A}^k \cdot \left|\mathbf{C}^k \cdot \mathbf{x}_\beta^k - \mathbf{x}_i^k\right| \tag{1.34}$$

$$\boldsymbol{\delta}_i^k = \mathbf{x}_\delta^k - \mathbf{A}^k \cdot \left|\mathbf{C}^k \cdot \mathbf{x}_\delta^k - \mathbf{x}_i^k\right| \tag{1.35}$$

where $\mathbf{x}_\alpha^k$, $\mathbf{x}_\beta^k$ and $\mathbf{x}_\delta^k$ denote the current positions of the α, β and δ wolves, respectively, and where:

$$\mathbf{A}^k = 2 \cdot \mathbf{a}^k \cdot \mathbf{r}_1^k - \mathbf{a}^k \tag{1.36}$$

$$\mathbf{C}^k = 2 \cdot \mathbf{r}_2^k \tag{1.37}$$

where $\mathbf{a}^k$ denotes a coefficient vector whose values linearly decreases from 2 to 0 over the course of iterations, while $\mathbf{r}_1^k$ and $\mathbf{r}_2^k$ represent random vector whose values are given by the uniformly distributed interval [0, 1].

1.4.2.9 Krill Herd Algorithm

The optimization techniques known as Krill Herd Algorithm (KHA) was devised by Amir Hossein Gandomi and Alavi in 2012. As its name implies, the KHA method is inspired by the natural herding behavior commonly observed on small crustaceans known as krill [54].

In KHA, solutions are modeled as a group of krill individuals which move through a feasible solution space while herding and foraging for food. Furthermore, KHA considers three distinctive kinds of herding movements: (1) The motion induced by other krill individuals, which represents a tendency to move toward other members of the aggregation aimed to keep a high density of individuals; (2) A foraging movement, which purpose is to guide krill toward the estimated location of a given food source; and (3) A physical diffusion movement, which represents a random process mainly used to keep the diversity of solutions. With that being said, at each time-step 't', the position of each individual krill is updated as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \frac{d\mathbf{x}_i^t}{dt} \tag{1.38}$$

where $\frac{d\mathbf{x}_i^t}{dt}$ denote a speed vector corresponding to the ith krill individual. In the KHA approach, such a speed vector is represented by the following Lagrangian model:

$$\frac{d\mathbf{x}_i^t}{dt} = N_i + F_i + D_i \tag{1.39}$$

with N_i, F_i and D_i denoting each of the three distinctive krill herding movement, namely: (1) The motion induced by other krill individuals (N_i), (2) Foraging movement (F_i), and (3) Physical diffusion movement (D_i) [54]. Furthermore, the value Δt stand for a speed scale factor, as given by the following expression:

$$\Delta t = C^t \sum_{n=1}^{d} (ub_n - lb_n) \tag{1.40}$$

where lb_n and ub_n each denote the lower and upper bounds of the nth variable (dimension), respectively, whereas $C^t \in [0, 2]$ is a constant parameter used to control the intensity of the search process.

Another important trait regarding the implementation of the KHA approach is the incorporation of genetic operators (such as crossover and mutation) which, according to the authors, aids to further improve the algorithms overall performance [54].

1.4.2.10 Moth-Flame Optimization Algorithm

The Moth-flame Optimization Algorithm (MFO) is novel nature-inspired metaheuristic proposed by Seyedali Mirjalili in 2015. The MFO approach is inspired by the night-navigation mechanism employed by moths in nature [55]. Such a mechanism, known as transversal orientation, consist of moths flying by maintaining a fixed angle to a particular light source. While for distant light sources (such as the moon) this method allow moths to fly in a straight line for very long distances effectively, nearby artificial lights (such as light bulbs or even the flame of a candle) are easily able to hamper such navigation method, causing moths to fly in a spiraling pattern toward such light sources. Such distinctive behaviors are mathematically modeled in MFO to perform global optimization tasks.

In MFO, search agents are represented by a population of N_M moths, each with a particular positions $\mathbf{M}_i = \left[m_{i,1}, m_{i,2}, \ldots, m_{i,d}\right]$ within a given solution space. Furthermore, the MFO approach also considers a set of N_F flames (or artificial lights) randomly distributed around such solution space, so that each of them also has a particular position $\mathbf{F}_j = \left[f_{j,1}, f_{j,2}, \ldots, f_{j,d}\right]$. Akin to how moths are "attracted" toward nearby light sources, each moth 'i' is assumed to fly in a spiraling pattern toward a given flame 'j'. In this sense, while the positions of both moths and flames represent solutions, only the moths are actual search agents while the flames stand for the best N_F solutions found so far by MFO search process. By considering this, at each iteration 'k', each moth is first assigned to a particular flame, and then, a movement operator modeled after a logarithmic spiral is applied in order to update the position of each search agent as follows:

$$\mathbf{M}_i^{k+1} = \mathbf{D}_{ij}^k \cdot e^{bl} \cdot \cos(2\pi l) + \mathbf{F}_j^k \tag{1.41}$$

where $\mathbf{D}_{ij}^k = \left|\mathbf{F}_j^k - \mathbf{M}_i^k\right|$. Furthermore, b denotes a constant parameter, while l stand for a random number drawn from within the interval $[r, 1]$ (with r being linearly decreased from -1 to -2 over the course of iterations).

Also, MFO employs a mechanism in which the number of available flames N_F within the search space is reduced as the iterative process goes on. With that being said, at each iteration 'k', the number of available flames is updated by considering the following expression.

$$N_F^{k+1} = \text{round}\left(N_F^0 - k \cdot \frac{N_F^0 - 1}{K}\right) \tag{1.42}$$

where N_F^0 denotes the initial (maximum) number of flames, while K stand for the maximum number of iterations which comprise the whole search process.

Initially, since moths move around by considering the positions of N_F best solutions (flames), exploration over the search space is highly promoted, while exploitation is minimal. However, as the number of available flames is reduced, exploration intensity is slowly decreased, while exploitation is gradually favored, thus, balancing the exploration and exploitation of solutions.

1.4.2.11 Particle Swarm Optimization

Devised by James Kennedy and Russell C. Eberhart in 1995, the Particle Swarm Optimization (PSO) method draws inspiration in the behavior of flocking birds, collectively foraging for adequate food sources [56]. In PSO, search agents (also referred as *particles*) are each composed by a set of three d-dimensional vectors: the particle's current position, its previous best position and its velocity. Also, each member within the swarm of particles is assumed to have knowledge of the global best position reached by its immediate neighborhood during the search process. With that being said, in the traditional PSO, the position update for each particle 'i' is given by the following expressions:

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \mathbf{v}_i^{k+1} \tag{1.43}$$

where k denotes the current iteration. Furthermore, $\mathbf{v}_i^{k+1}$ stand for the velocity of particle 'i' at iteration '$k + 1$', and is given as follows:

$$\mathbf{v}_i^{k+1} = \mathbf{v}_i^k + c_1 \cdot \left(\mathbf{r}_1^k \cdot \left(\mathbf{p}_i^k - \mathbf{x}_i^k\right)\right) + c_2 \cdot \left(\mathbf{r}_2^k \cdot \left(\mathbf{g}_i^k - \mathbf{x}_i^k\right)\right) \tag{1.44}$$

where $\mathbf{p}_i^k$ denotes the previous best position of particle 'i' (also called the personal best of 'i'), while $\mathbf{g}_i^k$ stand for the current global best position within an specific neighborhood of particles, from which individual 'i' belongs; here, the word "neighborhood" makes reference to a specific subset (topology) of particles (although, in its most simple form, it may refer to the whole swarm of particles) [15]. Furthermore, $\mathbf{r}_1^k$ and $\mathbf{r}_2^k$ each denote a d-dimensional vector composed by random numbers drawn from the interval of [0, 1], while the values c_1 and c_2 are known as cognitive and social parameters, respectively.

Due to its simplicity and easy implementation, the PSO method (as well as its many variations) has been extensively studied and applied into a wide variety of engineering areas and, as a result, has become one the most popular swarm intelligence approaches currently available for solving complex optimization problems.

1.4.2.12 Social Spider Optimization

The Social Spider Optimization (SSO) is a swarm intelligence approach proposed by Cuevas et al. in 2013. The SSO approach draws inspiration into several collective behaviors observed within a colony of social spiders [19]. A social spider colony is mainly composed by two components: its members, which may be further distinguished by their gender (either male or female) and a communal web which serves as a medium for both interaction and communication among such members of the colony. One important characteristic of most social spider colonies is the predominance of female spiders within its population. Furthermore, in a social spider colony, each member, depending on their gender, is assigned to cooperate in several different activities, such as building and maintaining the communal web, capturing prey, mating, etc. Another important trait about social spiders lies in their capacity to perceive vibrations transmitted through the communal web. Social spiders employ these vibrations to gain specific information, such as the size of trapped preys or the characteristics of neighboring members.

In the SSO approach, candidate solutions are modeled as a group of N spiders (each with a corresponding position $\mathbf{s}_i = [s_1, s_2, \ldots, s_d]$), interacting within a d-dimensional solution space (properly referred as the communal web). Furthermore, spiders are assumed to be able to communicate through a series of vibrations, emitted by each member and transmitted though the threads which conform the communal web. In this sense, the stimulus perceived by a particular spider 'i' as a result of the vibrations transmitted by some other spider 'j' are modeled as follows:

$$\text{Vib}_{i,j} = w_{\mathbf{s}_j} \cdot e^{-r_{ij}^2} \tag{1.45}$$

where $r_{ij} = \mathbf{s}_i - \mathbf{s}_j$ denotes for the Euclidian distance between the spiders 'i' and 'j'. Furthermore, $w_{\mathbf{s}_j}$ represents the weight corresponding to the jth spider as given by:

$$w_{\mathbf{s}_j} = \frac{f(\mathbf{s}_j) - f_{\text{worst}}}{f_{\text{best}} - f_{\text{worst}}} \tag{1.46}$$

where f_{best} and f_{worst} each denote the current best and worst fitness values from among all spiders within the communal web.

Also, each spider within the communal web is designated with a specific gender (either male or female). Depending on their gender, spiders are able to manifest several different behaviors. In the case of female spiders, for example, an attraction or dislike toward other members of the colony is displayed (irrespective of their gender), which depend on several factors such as reproduction cycle, curiosity and other random phenomena. In SSO, such behavior is modeled as either an attraction or repulsion movement toward other prominent individuals within the communal web. With that being said, at each iteration 'k', the position of a given female spider $\mathbf{f}_i^k$ (with $\mathbf{f}_i^k = \mathbf{s}_j^k$, $j \in [1, 2, \ldots, N]$) is updated by applying the following movement rules:

$$\mathbf{f}_i^{k+1} = \begin{cases} \mathbf{f}_i^k + \alpha \cdot \text{Vib}_{c_i}\left(\mathbf{s}_{c_i}^k - \mathbf{f}_i^k\right) + \beta \cdot \text{Vib}_{b_i}\left(\mathbf{s}_b^k - \mathbf{f}_i^k\right) + \delta \cdot \left(\gamma - \frac{1}{2}\right) & \text{if } P > P_f \\ \mathbf{f}_i^k - \alpha \cdot \text{Vib}_{c_i}\left(\mathbf{s}_{c_i}^k - \mathbf{f}_i^k\right) - \beta \cdot \text{Vib}_{b_i}\left(\mathbf{s}_b^k - \mathbf{f}_i^k\right) + \delta \cdot \left(\gamma - \frac{1}{2}\right) & \text{if } P \leq P_f \end{cases} \tag{1.47}$$

where $\mathbf{s}_{c_i}^k$ denotes the position of the nearest best (nearest heavier) member to the spider 'i', while $\mathbf{s}_b^k$ stand for the position of the best (heaviest) spider within the communal web. Furthermore, Vib_{c_i} and Vib_{b_i} denote the vibrations perceived by the spider 'i' with regard to $\mathbf{s}_{c_i}^k$ and $\mathbf{s}_b^k$, respectively (see Eq. 1.45), while α, β, δ, γ and P each denote a random number within the interval [0, 1]. Finally, P_f stands for a probability threshold used to define the kind of movement the female spider will perform (either an attraction or a repulsion).

On the other hand, male spiders, which are said to manifest an exclusive attraction toward female members, are further classified as either dominant or non-dominant male spiders. Typically, dominant male spiders have better qualities (i.e., a greater size or weight) in comparison to non-dominant male spiders. Furthermore, while dominant male spiders are usually attracted toward their closest female spider within the communal web, non-dominant male spiders tend to concentrate toward the center of the male population as a strategy to take advantage of resources that are wasted by dominant males. In SSO, these male-characteristic behaviors are modeled by first considering the median weight of all male spiders. At each iteration 'k', the weight of each male spider is compared to such median value in order to classify them as either dominant (weight above the median) or non-dominant (weight equal or lower to the median) male spiders, an then, an appropriate movement rule is applied to update the position $\mathbf{m}_i^k$ (with $\mathbf{m}_i^k = \mathbf{s}_j^k$, $j \in [1, 2, \ldots, N]$) of each of such male spiders, as illustrated as follows:

$$\mathbf{m}_i^{k+1} = \begin{cases} \mathbf{m}_i^k + \alpha \cdot \text{Vib}_{f_i}\left(\mathbf{s}_{f_i}^k - \mathbf{m}_i^k\right) + \delta \cdot \left(\gamma - \frac{1}{2}\right) & \text{if } w_{\mathbf{m}_i}^k > M(\mathbf{w_m}) \\ \mathbf{m}_i^k + \alpha \cdot \left(\frac{\sum_{h=1}^{N_m} \mathbf{m}_h^k \cdot w_{\mathbf{m}_h}^k}{\sum_{h=1}^{N_m} w_{\mathbf{m}_h}^k} - \mathbf{m}_i^k\right) & \text{if } w_{\mathbf{m}_i}^k \le M(\mathbf{w_m}) \end{cases} \tag{1.48}$$

where $w_{\mathbf{m}_i}^k$ denotes the weight of the male spider 'i', whereas $M(\mathbf{w_m})$ denotes the median value with regard to the weights of all male spiders. Furthermore, $\mathbf{s}_{f_i}^k$ stand for the position of the nearest female spider to $\mathbf{m}_i^k$, while Vib_{f_i} stand for the stimulus perceived by the male spider 'i' as a result of the vibrations emitted by its nearest female member (see Eq. 1.45). Also, the values α, δ, and γ each denote a random number within the interval [0, 1].

Finally, the SSO approach employs a mating mechanism in which female spiders and dominant male spiders are used to construct new candidate solutions. For such a procedure, a dominant male spider is first selected, and then, a set female spiders within a particular "range of mating" (which depends on the size of the target solution space) are further selected to perform a mating operation, in which a new individual is formed by combining the position information of each involved member. In this case, individuals possessing heavier weights (or higher fitness values) are more likely to influence the newly produced solution while those with lower weights tend to be of less relevance [19].

1.4.2.13 Whale Optimization Algorithm

In 2016, Seyedali Mirjalili and Andrew Lewis proposed a novel bio-inspired metaheuristic called Whale Optimization Algorithm (WOA). The WOA approach draws inspiration on the predatory behavior observed in humpback whales which, different to other species of whales, distinguish themselves for employing a unique cooperative hunting maneuver known as bubble-net feeding [10]. In such a hunting method, a group of whales (typically formed by two or three individuals) coordinate their efforts to encircle a group of small prey (such as schools of krill or small fishes) by swimming in a spiraling fashion around their quarry. While moving beneath the surface in such a distinctive pattern, each whale starts to exhale a burst of bubbles from their blowhole to form encircling bubble barrier (also referred as bubble-net), which prevents prey from escaping. As prey gets corralled into a tighter circle, one whale sounds a feeding call to the other whales, at which point all individuals simultaneously swim to the surface with their mouths open to feed on the trapped prey.

In WOA, such unique bubble-net feeding behavior is mathematically modeled and applied to solve global optimization problems by first considering a group of search agents, represented by N whales. Moreover, WOA's search process is divided in two phases: exploration and exploitation phase. For the exploration phases, whales are assumed to be randomly searching for prey. This process is modeled as movements toward a randomly chosen member. With that being said, at each iteration 'k', the position of each whale is updated by applying the following equation:

$$\mathbf{x}_i^{k+1} = \mathbf{x}_{\text{rand}}^k - \mathbf{A}^k \cdot \mathbf{D}^k \tag{1.49}$$

where $\mathbf{x}_{\text{rand}}^k$ denotes the position of a randomly chosen whale and where:

$$\mathbf{A}^k = 2 \cdot \mathbf{a}^k \cdot \mathbf{r}_1^k - \mathbf{a}^k \tag{1.50}$$

$$\mathbf{D}^k = \left|\left(2 \cdot \mathbf{r}_2^k \cdot \mathbf{x}_{\text{rand}}^k\right) - \mathbf{x}_i^k\right| \tag{1.51}$$

where $\mathbf{a}^k$ is a coefficient vector whose values linearly decreases from 2 to 0 over the course of iterations, while $\mathbf{r}_1^k$ and $\mathbf{r}_2^k$ denote random vectors whose values are drawn from within the uniformly distributed interval [0, 1].

On the other hand, the exploitation phase emphasizes the situation in which the group of whales has already identified their prey. To represent such behavior, the WOA approach considers two distinctive movement rules: prey encircling and bubble-net attacking method. For the prey encircling behavior, whales are assumed to move to positions around the target prey, while for the bubble-net attacking method step each whale moves in a spiraling fashion around such targeted prey to corral it. These two behaviors are represented by the following position update operators:

$$\mathbf{x}_i^{k+1} = \begin{cases} \mathbf{x}_*^k - \mathbf{A}^k \cdot \mathbf{D}^k & \text{if } p < 0.5 \\ \mathbf{D}_*^k \cdot \left(e^{b \cdot l} \cdot \cos(2\pi l)\right) + \mathbf{x}_*^k & \text{if } p \geq 0.5 \end{cases} \tag{1.52}$$

where $\mathbf{x}_*^k$ denotes the position of the current best solution from among all search agents (which further represent the targeted prey), whereas $\mathbf{D}_*^k = \left|\mathbf{x}_*^k - \mathbf{x}_i^k\right|$. Furthremore p stands for a random number drawn from within the interval [0, 1], while the value '0.5' stands for a probability threshold. The operator applied for $p < 0.5$ correspond to the prey encircling movement rule, while the operator applied when $p \geq 0.5$ represent the bubble-net attacking method operator, fittingly represented by the model of a logarithmic spiral whose shape is controlled by the constant parameters b and a random value $l \in [-1, 1]$.

1.4.3 Physics-Based Methods

The nature-inspired techniques known physics-based methods comprise a series of optimization algorithms which design is inspired by the laws of physics that govern our universe. In this sense, the movement that search agents can manifest in this kind of algorithmic structures are usually based on some observable physical phenomenon, such as the movement caused by gravitational forces, the interaction between electrically charged particles, thermodynamical processes, light refraction, among others.

1.4.3.1 Electromagnetism-like Mechanism

In 2003, Ş. Ilker Birbil and Shu Chering Fang proposed a population-based metaheuristic known as Electromagnetism-like Mechanism (EM) to solve global optimization problems. The EM approach was designed based on the laws of physics which govern the movement of charged particles within a given space (notably, the Coulomb's laws) [22].

In EM, search agents are represented as electrically charged particles, interacting within a feasible solution space. Each of these particles is assigned with an individual charge value, associated to the quality (fitness) of the solution they represent. With that being said, for a given iteration 'k' the charge value associated to a specific particle 'i' is given by the following expression:

$$q_i^k = \exp\left(-d \frac{f\left(\mathbf{x}_i^k\right) - f\left(\mathbf{x}_{\text{best}}^k\right)}{\sum_{j=1}^{N}\left(f\left(\mathbf{x}_j^k\right) - f\left(\mathbf{x}_{\text{best}}^k\right)\right)}\right) \tag{1.53}$$

where $\mathbf{x}_{\text{best}}^k$ denotes the current best solution found so far by the algorithm's search process, while d stand for the dimensionality of the target solution space.

Furthermore, as illustrated by the Coulomb's law, each of these particles is assumed to be subjected to a series of electrostatic forces, which depend on the magnitude of their individual charges and the distance between them. Furthermore, these electrostatic forces may be either of attraction or repulsive, depending on the sign of each charge. In the EM approach, the total electrostatic force experimented by a given particle 'i' with regard to all other particles is modeled as:

$$\mathbf{F}_i^k = \sum_{j \neq i}^{N} \begin{Bmatrix} \left(\mathbf{x}_j^k - \mathbf{x}_i^k\right) \frac{q_i^k q_j^k}{r_{ij}^k} & \text{if } f\left(\mathbf{x}_j^k\right) < f\left(\mathbf{x}_i^k\right) \\ \left(\mathbf{x}_i^k - \mathbf{x}_j^k\right) \frac{q_i^k q_j^k}{r_{ij}^k} & \text{if } f\left(\mathbf{x}_j^k\right) \geq f\left(\mathbf{x}_i^k\right) \end{Bmatrix} \tag{1.54}$$

where $r_{ij}^k = \left\|\mathbf{x}_j^k - \mathbf{x}_i^k\right\|$ denotes the Euclidian distance between the particles 'i' and 'j.

Intuitively, as a result of such attraction/repulsion forces, each particle is forced to change their position at each time instant. In EM, the position update rule applied to each of such particles is given by the following equation:

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \lambda \cdot \hat{\mathbf{F}}_i^k \cdot \mathbf{m}_i^k \tag{1.55}$$

where λ denotes a d-dimensional vector of random numbers drawn from within the interval [0, 1], while $\widehat{\mathbf{F}}_i^k = \mathbf{F}_i^k / \left\|\mathbf{F}_i^k\right\|$ stand for the normalized electrostatic force of to the ith particle's at iteration 'k'. Finally, $\mathbf{m}_i^k$ denotes a movement vector whose components $\left(\mathbf{m}_i^k = \left[m_{i,1}^k, m_{i,2}^k, \ldots, m_{i,d}^k\right]\right)$ depend on those of $\hat{\mathbf{F}}_i^k$ $\left(\hat{\mathbf{F}}_i^k = \left[\hat{F}_{i,1}^k, \hat{F}_{i,2}^k, \ldots, \hat{F}_{i,d}^k\right]\right)$, as illustrated as follows:

$$m_{i,n}^k = \begin{cases} ub_n - x_{i,n}^k \ if \ \hat{F}_{i,n}^k > 0 \\ x_{i,n}^k - lb_n \ \ if \ \hat{F}_{i,n}^k \leq 0 \end{cases} \tag{1.56}$$

with $x_{i,n}^k$ denoting the nth component of the particle's position $\mathbf{x}_i^k$, while lb_n and ub_n stand for the lower and upper fitness function bounds at the nth dimension, respectively.

1.4.3.2 Gravitational Search Algorithm

The Gravitational Search Algorithm (GSA) is a population-based metaheuristic proposed by Rashedi et al. in 2009. The GSA design is mainly inspired by the laws of gravity, which establishes the inherent interaction between different objects (or masses) as a result of the gravitational forces experienced by them [21].

In GSA, search agents are modeled as N individual masses, subjected to constant interaction within d-dimensional solution space (also referred as a *system*). At each iteration 't' (also referred as *time*), each individual mass 'i' is assigned with a particular mass M_i, which value depends on its current solution quality (fitness), such that:

$$M_i^t = \frac{m_i^t}{\sum_{j=1}^{N} m_j^t} \tag{1.57}$$

where m_i^t denotes the normalized fitness value corresponding to the ith mass, as given by the following expression:

$$m_i^t = \frac{f\left(\mathbf{x}_i^t\right) - f_{\text{worst}}^t}{f_{\text{best}}^t - f_{\text{worst}}^t} \tag{1.58}$$

where f_{best}^t and f_{best}^t each denote the current best and worst fitness values at time 't', respectively.

Furthermore, akin to the law of gravity, each of these masses is assumed to be in constant interaction with each other as a result of the gravitational forces exerted by each of them. In GSA, the total gravitational force experienced by a particular mass 'i' with regard to all other masses is given by the following equation:

$$\mathbf{F}_i^t = \sum_{j \neq i}^{N} \left(G(t) \frac{M_i^t \cdot M_j^t}{r_{ij}^t + \varepsilon} \left(\mathbf{x}_j^t - \mathbf{x}_i^t\right) \cdot \mathbf{rand} \right) \tag{1.59}$$

where $r_{ij}^t = \left\| \mathbf{x}_j^t - \mathbf{x}_i^t \right\|$ stand for the Euclidian distance between masses 'i' and 'j', while ε is a small value used to prevent singularities. Furthermore, **rand** denote a d-dimensional vector of random numbers drawn from within the interval [0, 1]. Finally, $G(t)$ represents the so-called gravitational constant, whose value depends on the current time 't' as follows:

$$G(t) = G_0 e^{\left(-\alpha \frac{t}{T}\right)} \tag{1.60}$$

with G_0 denoting the initial gravitational constant value, α standing for a constant parameter and T representing the total iteration number which comprises the whole GSA search process.

Finally, as a result of such gravitational interactions, masses are also assumed to be able to experience a movement. With that being said, in GSA, the following position update rule is applied:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{1.61}$$

where $\mathbf{v}_i^{t+1}$ denotes the velocity of the ith mass at the time '$t + 1$', as expressed by the following equation:

$$\mathbf{v}_i^{t+1} = \mathbf{rand} \cdot \mathbf{v}_i^t + \mathbf{a}_i^t \tag{1.62}$$

where $\mathbf{a}_i^t = \frac{1}{M_i^t}\mathbf{F}_i^t$ stand for the acceleration experienced by the mass 'i' at such time 't'.

1.4.3.3 Simulated Annealing

Simulated Annealing (SA) comprises one of the earliest and most successful local search methods specially devised to tackle complex discrete (and to a lesser extend continuous) optimization problems. As first introduced by Kirkpatrick et al. in 1982, the SA approach is mainly inspired in the physical process of annealing (a process typically aimed to achieve an alteration on the physical properties of crystalline solids through heat treatment). In particular, a solid is first heated until it reaches a certain temperature, and then it is allowed to cool down slowly; if properly done, this process enables said material's microstructure to achieve a better crystal lattice configuration, and as such, superior structural integrity [57].

SA is essentially an iterative local search process, in which a given candidate solution is iteratively modified by implementing a computational procedure inspired by the temperature transition scheme modeled on a typical annealing process. As such, SA starts by first defining an initial solution (state) $\mathbf{s}$ and cooling schedule $\mathbf{T} = \{t_0, t_1, \ldots, t_n\}$, where the elements t_k each represent a finite transition temperature in the annealing process and such that:

$$t_i > 0 \text{ and } \lim_{k \to +\infty} t_i = 0 \tag{1.63}$$

Each temperature t_i in the cooling schedule is applied for a finite number of iterations of the SA's search process. With that being said, SA also defines a repetition schedule $\mathbf{M} = \{M_0, M_1, \ldots, M_n\}$, where the elements M_i dictate the number of iterations a given temperature t_i will be applied.

Once the SA algorithm completes its initialization step, it starts an iterative search process in which, at each iteration 'k', a neighboring solution $\mathbf{s}'$ around the current best solution $\mathbf{s}^k$ is generated, either randomly or by following a particular criterion. After that, both solutions are compared in terms of fitness value and, depending on the outcome of such a comparison, there is a certain probability to accept said neighbor solution as the current best solution; for a minimization problem, for example, this probability is given by:

$$P^k = \begin{cases} \exp\left(-\frac{(f(\mathbf{s}')-f(\mathbf{s}^k))}{t^k}\right) & \text{if } f(\mathbf{s}') - f(\mathbf{s}^k) > 0 \\ 1 & \text{if } f(\mathbf{s}') - f(\mathbf{s}^k) \leq 0 \end{cases} \tag{1.64}$$

where $t^k \in \mathbf{T}$ denotes the transition temperature that is applied at given iteration 'k'.

As one of the earliest metaheuristics, the SA algorithm has been a subject of constant study and improvements. Some of the most well-known variants of SA involve changes to either the cooling schedule model, the neighborhood selection method, and even its learning mechanism [58].

1.4.3.4 Sine Cosine Algorithm

The Sine Cosine Algorithm (SCA) is a population-based metaheuristic developed by Seyedali Mirjalili in 2016, which design is based on the properties of sinusoidal functions [59].

SCA consider a set of N search agents each with a corresponding position $\mathbf{x}_i = \left[x_{i,1}, x_{i,2}, \ldots, x_{i,d}\right]$ within a given d-dimensional solution space. In SCA, each available search agent updates their positions by applying one of two different movement operators, each modeled after some particular sinusoidal function (namely, the sine and cosine functions). With that being said, at each iteration 'k', the position update operator applied to each individual $\mathbf{x}_i^k$ is given by:

$$\mathbf{x}_i^{k+1} = \begin{cases} \mathbf{x}_i^k + r_1 \cdot \sin(r_2) \cdot \left|r_3 \cdot \mathbf{p}_i^t - \mathbf{x}_i^k\right| & \text{if } r_4 < 0.5 \\ \mathbf{x}_i^k + r_1 \cdot \cos(r_2) \cdot \left|r_3 \cdot \mathbf{p}_i^t - \mathbf{x}_i^k\right| & \text{if } r_4 \geq 0.5 \end{cases} \tag{1.65}$$

where $r_1 = a - k(a/T)$ (with a being a constant value and K denoting the maximum number of iterations for the whole search process), while r_2, r_3 and r_4 are random numbers drawn from within the uniformly distributed intervals $[0, 2\pi]$, $[0, 2]$ and $[0, 1]$, respectively. Furthermore, $\mathbf{p}_i^t$ stand for the current *destination point*, which is given by the position of best solution found so far by SCA's search process.

1.4.3.5 States of Matter Search

In 2014, Cuevas et al. proposed a novel metaheuristic approach coined States of Matter Search (SMS). The SMS approach draws inspiration on the physical principles of thermal-energy motion, manifested by the molecules of a substance as it transitions from one state of matter to another [23].

In SMS, search agents are represented by individual molecules, in constant motion within the target d-dimentional solution space. A unique trait of SMS is that the whole evolutionary process is divided into three different stages, inspired by the three classic states of matter: (1) a gas state in which molecules are assumed to experiment constant movement and collision with other molecules; (2) a liquid state in which a significant reduction of molecular movement occurs as a result of a descent on the available thermal-energy; and (3) a solid state in which the bonding force among the molecules becomes so strong that their movement is almost completely inhibited. Each of such stages occurs in sequence as the SMS iteration process goes on, with each of them taking place for a finite number iterations. Under this considerations, each molecule moves with a particular "intensity", which depends on the current transitory state (gas, solid or liquid). In general, at each iteration 'k', each molecule updates its position by applying the following movement rule:

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \mathbf{v}_i^k \cdot \mathbf{rand} \cdot (\mathbf{ub} - \mathbf{lb}) * \rho \tag{1.66}$$

where $\mathbf{lb} = [lb_1, lb_2, \ldots, lb_d]$ and $\mathbf{ub} = [ub_1, ub_2, \ldots, ub_d]$ each denote the lower and upper bound vectors of the target solution space, respectively, while **rand** stand for a random vector whose values are drawn from the interval [0, 1]. Furthermore, $\rho \in [0, 1]$ denotes a scalar factor which value depend on the current SMS stage [23]. Finally, $\mathbf{v}_i^t$ denotes the velocity of the ith molecule at iteration 'k', as given by the following expressions:

$$\mathbf{v}_i^k = v_{init} * \mathbf{d}_i^k \tag{1.67}$$

where v_{init} denotes the magnitude of the initial velocity, as given as follows:

$$v_{init} = \beta * \frac{\sum_{n=1}^{d} (ub_n - lb_n)}{d} \tag{1.68}$$

where lb_n and ub_n each denote the lower and upper objective function bounds at the nth dimension, respectively, while $\beta \in [0, 1]$ denotes a scalar factor which, which similarly to ρ in Eq. (1.66), depends on the current SMS stage [23]. On the other hand, $\mathbf{d}_i^k$ stand for a direction vector corresponding to the ith, and is given by the following equation:

$$\mathbf{d}_i^k = \mathbf{d}_i^{k-1} * 0.5\left(1 - \frac{k}{K}\right) + \mathbf{a}_i^k \tag{1.69}$$

with K denoting the maximum number of iterations which compose the whole SMS's search process. Also, $\mathbf{a}_i^k$ denotes a unit vector oriented toward the current global best solution $\left(\mathbf{x}_{\text{best}}^k\right)$, as given by the following expression:

$$\mathbf{a}_i^k = \frac{\left(\mathbf{x}_{\text{best}}^k - \mathbf{x}_i^k\right)}{\left\|\mathbf{x}_{\text{best}}^k - \mathbf{x}_i^k\right\|} \tag{1.70}$$

Another important trait present of SMS's is the inclusion of a collision mechanism, which causes molecules to exchange their directions. Essentially, a collision is said to occur among a pair of molecules 'i' and 'j' (with $i \neq j$) if their Euclidian distance $\left\|\mathbf{x}_i - \mathbf{x}_j\right\|$ is shorter than a given distance threshold. Once a collision occurs, the directions $\mathbf{d}_i$ and $\mathbf{d}_j$ corresponding to such molecules 'i' and 'j' are exchanged, such that:

$$\text{if } \left(\left\|\mathbf{x}_i^k - \mathbf{x}_i^k\right\| < r\right) \rightarrow \begin{matrix} \mathbf{d}_i^k = \mathbf{d}_j^k \\ \mathbf{d}_j^k = \mathbf{d}_j^k \end{matrix} \tag{1.71}$$

where the distance threshold r (also known as *collision radius*), and is given by the following expression:

$$r = \alpha \cdot \frac{\sum_{n=1}^{d}(ub_n - lb_n)}{d} \tag{1.72}$$

where $\alpha \in [0, 1]$ is a scalar factor which, similarly to β and ρ in Eqs. (1.66) and (1.68), depends on the current SMS stage [23].

1.4.4 Human-Based Methods

Human-based optimization algorithms are a special class of methodologies which design draws inspiration from several phenomena related to the behavior and the lifestyle of human beings. With that being said, this kind of techniques may be designed based on, either some cognitive process applied by humans to solve problems, or even on certain activities commonly related to the way in which human beings live.

1.4.4.1 Fireworks Algorithm

In 2010, Ying Tan and Yuanchun Zhu proposed a novel optimization framework known as the Fireworks Algorithm (here referred as FWA). Interestingly, the inspiration for the FWA approach comes from the observation of several properties observed in the flashy explosions created by fireworks [25]. In general, once a firework is set off, the resulting explosion creates a shower of sparks which spreads around a local radius around the denotation's location. In the context of swarm intelligence approaches, the way in which such sparks spread around the firework's explosion radius is seen as a specific case of a local search process around the location in which the firework was set off, and thus, is presented as a useful mechanism to perform optimization.

In the FWA approach, at each iteration 'k', a set of N locations within the feasible d-dimensional search space are chosen to set off individual fireworks. In the context of the fireworks explosion metaphor, explosions can be distinctively identified as either good or bad explosions. In this sense, good explosions create numerous centralized sparks and are assumed to be the result of well manufactured fireworks. On the other hand, bad explosions, which result from badly manufactured fireworks, generate fewer sparks and these scatter around a much larger local space. With that being said, after a firework is set off, several sparks are assumed to be spread within a fixed local area around the explosion's location. Each generated spark is treated as a local search agent and evaluated with regard to the target objective function. Furthermore, for the given iteration 'k', the number of generated sparks s_i^k and the amplitude of explosion A_i^k for each deployed firework 'i' are said to depend on the quality of its manufacture, which is further represented by the quality (fitness) at its respective explosion location $\mathbf{x}_i$, as given by the following expressions:

$$s_i^k = m \cdot \frac{f_{\text{worst}}^k - f\left(\mathbf{x}_{\text{i}}^k\right) + \xi}{\sum_{i=1}^{N}\left(f_{\text{worst}}^k - f\left(\mathbf{x}_{\text{i}}^k\right)\right) + \xi} \tag{1.73}$$

$$A_i^k = \hat{A}.\frac{f\left(\mathbf{x}_i^k\right) - f_{\text{best}}^k + \xi}{\sum_{i=1}^{N}\left(f\left(\mathbf{x}_i^k\right) - f_{\text{best}}^k\right) + \xi} \tag{1.74}$$

where m and $\hat{A}$ each denote the fireworks' maximum number of sparks and amplitude of explosion, respectively. Furthermore f_{best}^k and f_{worst}^k each denote the fitness values corresponding to the current best and worst solutions among the N fireworks, respectively, while ξ stand for a small value used to prevent singularities while computing either s_i^k or A_i^k. Furthermore, in order to avoid the overwhelming effects of "splendid" fireworks, bounds are defined for s_i^k with regard to a set of constant parameters a and b as follows:

$$\hat{s}_i^k = \begin{cases} \text{round}(a \cdot m) \text{ if } s_i < a \cdot m \\ \text{round}(b \cdot m) \text{ if } s_i > b \cdot m, a < b < 1 \\ \text{round}\left(s_i^k\right) \quad \text{otherwise} \end{cases} \tag{1.75}$$

Finally, for each firework 'i', each of the $\hat{s}_i^k$ generated sparks are randomly distributed around the local area defined by the amplitude of explosion A_i^k. In FWA, this is achieved by randomly selecting a number of affected directions (dimensions) for each individual spark, and then, a displacement magnitude is calculated within the explosion amplitude A_i^k. This could be effectively seen as a type of random walk, and as such, other similar methods could may be applied to define the locations of the generated sparks.

1.4.4.2 Harmony Search

The Harmony Search (HS) method, as proposed by Geem et al. in 2001, is a metaheuristic approach inspired on the principles behind the process of harmony improvisation, in which musicians are said to compose a harmony by combining different music pitches stored in their memory, this with the purpose of finding the perfect harmony [24]. In HS, the process of finding the perfect harmony is seen as an analogy to finding the optimal solution in an optimization problem.

HS is initialized by considering a set of N randomly generated solutions, collectively referred as the HS Memory. At each iteration 'k' of the HS search process, a new candidate solution $\mathbf{x}_c = \left[x_{c,1}, x_{c,2}, \ldots, x_{c,d}\right]$ is generated (improvised) and then evaluated against currently existing solutions; in particular, if the quality (fitness) of the proposed candidate solution $\mathbf{x}_c$ is better than that of the current worst solution in the HS Memory, then such worst solution is replaced by $\mathbf{x}_c$. Each component $x_{c,n}$

of the improvised solution $\mathbf{x}_c$ is generated depending on the value of the Harmony Memory Considering Rate ($HMCR$), as illustrated as follows:

$$x_{c,n} = \begin{cases} x_{r,n} + \text{rand}(-1, 1) \cdot bw & \text{if}\,\text{rand}(0, 1) < HMCR \\ \text{rand}(lb_n, ub_n) & \text{if}\,\text{rand}(0, 1) \geq HMCR \end{cases} \tag{1.76}$$

where $x_{r,n}$ denotes the nth component corresponding to a randomly chosen solution $\mathbf{x}_r$ within the HS Memory, whereas the parameter bw represent the so called distance bandwidth (essentially a step size value). Furthermore, rand(a, b) stand for a random number from within the interval $[a, b]$ (i.e., rand(0, 1) correspond to a random number between 0 and 1), while lb_n and ub_n are the objective function's lower and upper bounds at the nth dimension, respectively.

1.4.4.3 Imperialist Competitive Algorithm

In 2007, Esmail Atashpaz-Gargari and Caro Lucas proposed a novel population-based metaheuristic known as Imperialist Competitive Algorithm (ICA). Said optimization technique is inspired by imperialism (or neocolonialism); that is the actions taken by individual countries to extend their power (typically through the acquisition of other territories) [26].

At the initialization step, ICA starts by randomly generating a set of N search agents (called countries), each with an individual solution $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,d}\}$ representing as a set of socio-political characteristics (such as culture, language, economy, religion, etc.). After generating such set of solutions, these are then classified as either imperialists or colonies. For this purpose, the best N_{imp} countries (according to fitness quality) are designated as imperialists countries $(\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_{N_{\text{imp}}})$, while the remaining $N_{\text{col}} = N - N_{\text{imp}}$ countries are labeled as colonies $(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{N_{\text{col}}})$. Afterwards, each of the available N_{col} colonies are proportionally distributed among the N_{imp} imperialists in order to form the empires. The number of colonies assigned to each imperialist is proportional to their respective imperialist power, as given by the following expression:

$$power_i = \left| \frac{cost_i}{\sum_{j=1}^{N_{\text{imp}}} cost_i} \right| \tag{1.77}$$

where c_i denotes the normalized imperialist cost of the ith imperialist, as given by:

$$cost_i = \max_j \{ f(\mathbf{i}_j) \} - f(\mathbf{i}_i) \tag{1.78}$$

By considering the previous, the total number of colonies that are assigned to each imperialist is given by:

$$NC_i = \text{round}(power_i \cdot N_{\text{col}}) \tag{1.79}$$

Once ICA's initialization process has been performed, the algorithm starts an iterative search process comprised of four main steps (1) Assimilation, (2) Revolution, (3) Intra-imperialistic competition and (4) Inter-imperialistic competition. For the assimilation process, each colony belonging to an empire is assumed to be influenced by the socio-political elements (culture, economy, religion, etc.) of its respective imperialist country. Said influence is represented as a movement from said colony $\mathbf{c}_{ij}$ toward is respective imperialist $\mathbf{i}_j$, as given as follows:

$$\mathbf{c}_{ij}^{*} = \mathbf{c}_{ij} + \text{rand}(0, \beta) \cdot \left(\mathbf{i}_j - \mathbf{c}_{ij}\right) \tag{1.80}$$

where the value β is typically set to be between 1 and 2.

Then, in the revolution step, some randomly chosen colonies are assumed to experience some changes in their socio-political characteristics. Akin to the mutation operator in methods such as Genetic Algorithms (GA), the purpose of the revolution step is to apply sudden changes to some of the colonies in order to favor diversity of solutions, and thus preventing premature convergence or being trapped into local optima.

Furthermore, at the intra-imperialist, colonies that have performed a movement (either by assimilation or revolution) compete for the position of imperialist. Essentially, if any country $\mathbf{c}_{ij}$ within an empire has a better quality than that of their respective imperialist $\mathbf{i}_j$, then $\mathbf{c}_{ij}$ is labeled as the new imperialist, while $\mathbf{i}_j$ is considered as a colony. Finally, for the inter-imperialist step, one of weaker the colonies within the weakest empire is given to another empire based on competition. For this purpose, a probability for possessing said weak colony is first assigned to each empire as follows:

$$P_j = \left| \frac{NTC_j}{\sum_{i=1}^{N_{\text{imp}}} NTC_i} \right| \tag{1.81}$$

where NTC_j denotes the normalized total cost of the jth empire as given by:

$$NTC_j = \max_i\{TC_i\} - TC_j,\; TC_j = f\left(\mathbf{i}_j\right) + \zeta \cdot \text{mean}\left\{f\left(\mathbf{c}_{ij}\right)\right\} \tag{1.82}$$

where the value $\zeta \in [0, 1]$ represents the influence that the mean power of the colonies has when determining the empire's total power.

After a possessing probability P_j has been assigned to each empire, an appropriate selection method, (such as the roulette selection approach) is applied to decide which empire will take charge of the disputed colony. This process is applied at each iteration of ICA's search process. As a result of the inter-imperialistic competition, weaker empires suffer a gradually decrease in power as they lose their colonies over time, while stronger empires increase their power as they take possession of said colonies.

This eventually lead weaker empires to collapse over time until only a single strongest empire remains.

1.5 Nature-Inspired Metaheuristics on the Literature

In the last few years, works related to applications or modifications of nature-inspired metaheuristics for solving a wide range of optimization problems have become so numerous that mentioning every single paper in existence has become an overwhelming task. In Fig. 1.2, the number annual publications (as reported by IEEE and Elsevier) related to nature-inspired metaheuristics for the last 36 years is shown. As evidenced by the data shown in said figure, the number of publications related to this kind of techniques started to increase at a considerable rate around the 1990s. Nature-inspired optimization methods have become so successful and so well-known on the literature and, as a result, the number of publications per-year related to these techniques has reached astonishing levels, with essentially more than 1000 new papers being published annually since 2010. While nature-inspired techniques have been well received in the literature, some particular methods stand as popular choices among researchers around the world. In Fig. 1.3, the annual publication statistics (as given by IEEE and Elsevier publishers) related to the ten most cited nature-inspired optimization methods is shown. In particular, some of the earliest nature-inspired methods such as GA, SA, PSO, DE, and ACO stand as the most representative examples among these techniques, while latest methods such as ABC, HS, CS, FA, and GSA have recently experienced an increase on popularity. Furthermore, Table 1.1 shows specific data regarding each of these methods, including its year of publication and its total number of citations up to the year 2016.

1.6 Conclusions

Nature is often praised by researchers as the perfect example of adaptive problem solving, and as such, it is not surprising to see why metaheuristics optimization algorithms inspired in natural phenomena have become so popular. These kinds of methods are designed with the idea of mimicking some biological or physical phenomenon observed in nature with the purpose of developing powerful tools that could be applied to solve optimization problems. The main advantage of nature-inspired metaheuristics over traditional optimization methods, however, lies on their ability to handle a wide variety of problems independently of their structure and properties. Due to this distinctive trait, these methods have become popular choices for solving otherwise complex problems. As a result, these techniques have found application in virtually every single area of science, including robotics, computer networks, security, engineering design, data mining, finances, economics, and many others. In the last few years, literature related to nature-inspired algorithms and its applications

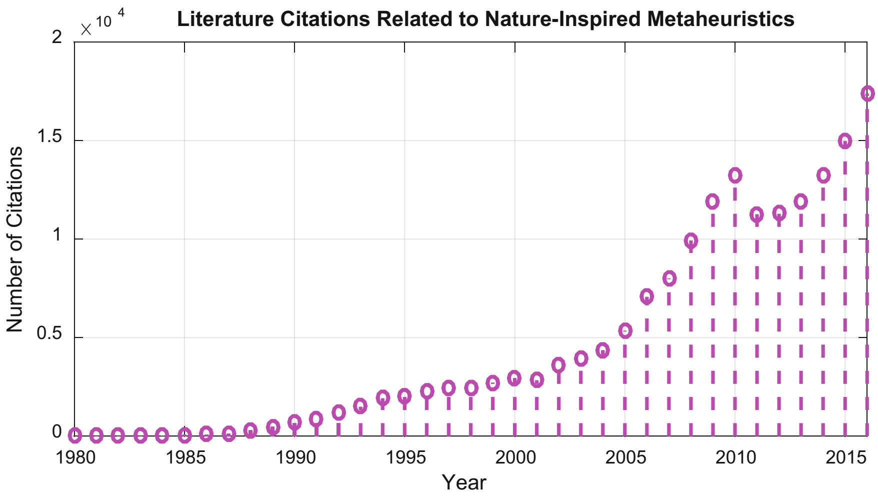


Fig. 1.2 Annual number of publications in IEEE and Elsevier related nature-inspired metaheuristics

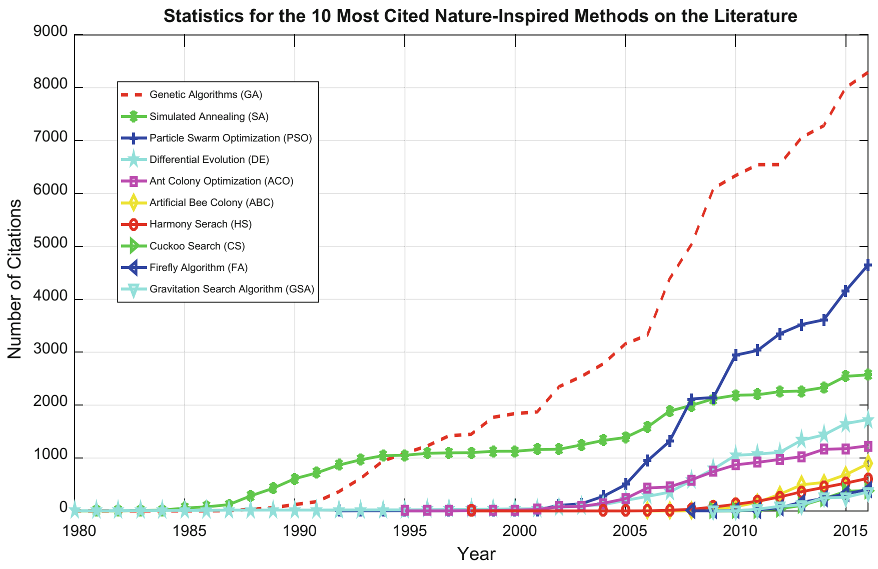


Fig. 1.3 Annual number of publications in IEEE and Elsevier related to the ten most cited nature-inspired metaheuristics: GA, SA, PSO, DE, ACO, ABC, HS, CS, FA, and GSA

Table 1.1 Total number of citations (up to the year 2016) for some of the most popular nature-inspired metaheuristics

Algorithm	Author(s)	Year of publication	Total number of citations
Genetic Algorithms (GA)	J. H. Holland	1960	102,190
Simulated Annealing (SA)	S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi	1982	44,711
Particle Swarm Optimization (PSO)	J. Kennedy and R. Eberhart	1995	38,634
Differential Evolution (DE)	R. Storn and K. Price	1996	38,632
Ant Colony Optimization (ACO)	M. Dorigo	1992	14,472
Artificial Bee Colony (ABC)	D. Karaboga	2005	4174
Harmony Search (HS)	Z. W. Geem, J. H. Kim and G. V. Loganathan	2001	3466
Cuckoo Search (CS)	X.-S. Yang and S. Deb	2009	1843
Firefly Algorithm (FA)	X.-S. Yang	2008	1807
Gravitational Search Algorithm (GSA)	E. Rashedi, H. Nezamabadi-pour and S. Saryazdi	2009	1639

for solving optimization problems has experienced an almost exponential increase, with tons of new papers being published every year. As suggested by statistical data collected from publishers such as IEEE and Elsevier, methods such as GA, SA, PSO, DE, and ACO are among the most successful and most cited optimization approaches currently reported on the literature. The popularity gained by these optimization techniques is related not only to their interesting individual characteristics, but also due to their easiness of implementation and capabilities to be applied for solving numerous real-world problems. In any case, there is no doubt that nature-inspired metaheuristics have rightfully earned their place as powerful tools for optimization, and as such, this line of investigation is expected to keep growing in the near future.

References

1. Galinier, P., Hamiez, J.P., Hao, J.K., Porumbel, D.: Handbook of Optimization, vol. 38 (2013)
2. Cuevas, E., Díaz Cortés, M.A., Oliva Navarro, D.A.: Advances of Evolutionary Computation: Methods and Operators, 1st edn. Springer International Publishing (2016)
3. Cavazzuti, M.: Optimization Methods: From Theory to Design (2013)
4. Lin, M., Tsai, J., Yu, C.: A review of deterministic optimization methods in engineering and management. Math. Probl. Eng. **2012**, 1–15 (2012)
5. Schneider, J.J., Kirkpatrick, S.: Stochastic optimization (2006)
6. Cuevas, E., Osuna, V., Oliva, D.: Evolutionary Computation Techniques: A Comparative Perspective, vol. 686 (2017)
7. Díaz-Cortés, M.-A., Cuevas, E., Rojas, R.: Engineering Applications of Soft Computing (2017)
8. Yang, X.: Nature-Inspired Metaheuristic Algorithms, 2nd edn. Luniver Press, Beckington, UK (2008)
9. Binitha, S., Sathya, S.S.: A Survey of Bio inspired Optimization Algorithms. Int. J. Soft Comput. Eng. **2**(2), 137–151 (2012)
10. Mirjalili, S., Lewis, A.: The whale optimization algorithm. Adv. Eng. Softw. **95**, 51–67 (2016)
11. Mitchell, M.: Genetic algorithms: an overview. Complexity **1**(1), 31–39 (1995)
12. Bäck, T., Hoffmeister, F., Schwefel, H.-P.: A survey of evolution strategies. In: Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, vol. 9, no. 3, p. 8.
13. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
14. Sette, S., Boullart, L.: Genetic programming: principles and applications. Eng. Appl. Artif. Intell. **14**(6), 727–736 (2001)
15. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. Swarm Intell. **1**(1), 33–57 (2007)
16. Dorigo, M., Stützle, T.: Ant Colony Optimization (2004)
17. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. J. Glob. Optim. **39**(3), 459–471 (2007)
18. Yang, X.: Firefly algorithm, Lévy flights and global optimization (2010)
19. Cuevas, E., Cienfuegos, M., Zaldívar, D., Pérez-cisneros, M.: A swarm optimization algorithm inspired in the behavior of the social-spider. Expert Syst. Appl. **40**(16), 6374–6384 (2013)
20. Rutenbar, R.A.: Simulated annealing algorithms: an overview. IEEE Circuits Devices Mag. **5**(1), 19–26 (1989)
21. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: a gravitational search algorithm. Inf. Sci. (Ny) **179**(13), 2232–2248 (2009)
22. Birbil, Ş.I., Fang, S.C.: An electromagnetism-like mechanism for global optimization. J. Glob. Optim. **25**(3), 263–282 (2003)
23. Cuevas, E., Echavarría, A., Ramírez-Ortegón, M.A.: An optimization algorithm inspired by the States of Matter that improves the balance between exploration and exploitation. Appl. Intell. **40**(2), 256–272 (2013)
24. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: harmony search. Simulation **76**(2), 60–68 (2001)
25. Tan, Y., Zhu, Y.: Fireworks algorithm for optimization. In: First International Conference, ICSI 2010—Proceedings, Part I, 2010, June, pp. 355–364
26. Atashpaz-Gargari, E., Lucas, C.: Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: 2007 IEEE Congress on Evolutionary Computation, CEC 2007, 2007, pp. 4661–4667
27. Opara, K.R., Arabas, J.: Differential evolution: a survey of theoretical analyses. Swarm Evol. Comput. June 2017, pp. 1–13 (2018)
28. Padhye, N., Mittal, P., Deb, K.: Differential evolution: performances and analyses. 2013 IEEE Congr. Evol. Comput. CEC 2013 (no. i), 1960–1967 (2013)
29. Mohamed, A.W., Sabry, H.Z., Khorshid, M.: An alternative differential evolution algorithm for global optimization. J. Adv. Res. **3**(2), 149–165 (2012)

30. Das, S., Suganthan, P.N.: Differential evolution: a survey of the state-of-the-art. IEEE Trans. Evol. Comput. **15**(1), 4–31 (2011)
31. Das, S., Mullick, S.S., Suganthan, P.N.: Recent advances in differential evolution—an updated survey. Swarm Evol. Comput. **27**, 1–30 (2016)
32. Piotrowski, A.P.: Review of differential evolution population size. Swarm Evol. Comput. **32**, 1–24 (2017)
33. Bäck, T., Foussette, C., Krause, P.: Contemporary evolution strategies, vol. 47 (2013)
34. Beyer, H.G., Sendhoff, B.: Covariance matrix adaptation revisited—the CMSA evolution strategy. Lecture Notes on Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5199 LNCS, pp. 123–132 (2008)
35. Auger, A., Schoenauer, M., Vanhaecke, N.: {LS-CMA-ES}: a second-order algorithm for covariance matrix adaptation. Parallel Probl Solving from Nat. PPSN VIII **3242**(1), 182–191 (2004)
36. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864v2, pp. 1–13 (2017)
37. Mitchell, M.: An introduction to genetic algorithms. The MIT Press, Cambridge, MA (1996)
38. Sayed, G.I., Hassanien, A.E., Nassef, T.M.: Genetic and Evolutionary Computing, vol. 536, no. Mci (2017)
39. McCall, J.: Genetic algorithms for modelling and optimisation. J. Comput. Appl. Math. **184**(1), 205–222 (2005)
40. Yadav, P.K., Prajapati, N.L.: An Overview of Genetic Algorithm and Modeling, vol. 2, no. 9, pp. 1–4 (2012)
41. Pham, D.T., Huynh, T.T.B., Bui, T.L.: A survey on hybridizing genetic algorithm with dynamic programming for solving the traveling salesman problem. In: 2013 International Conference on Soft Computing and Pattern Recognition, SoCPaR 2013, pp. 66–71 (2013)
42. Khu, S.T., Liong, S.Y., Babovic, V., Madsen, H., Muttil, N.: Genetic programming and its application in real-time runoff forecasting. J. Am. Water Resour. Assoc. **37**(2), 439–451 (2001)
43. Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: Genetic programming an introductory tutorial and a survey of techniques and applications. Tech Rep CES475, vol. 18, pp. 1–112 (Oct. 2007)
44. Harman, M., Langdon, W.B., Weimer, W.: Genetic Programming For Reverse Engineering. In: 20th Working Conference on Reverse Engineering WCRE 2013, pp. 1–10 (2013)
45. Gerules, G., Janikow, C.: A survey of modularity in genetic programming. 2016 IEEE Congr. Evol. Comput. CEC 2016, pp. 5034–5043 (2016)
46. Vanneschi, L., Castelli, M., Silva, S.: A survey of semantic methods in genetic programming. Genet. Program Evolvable Mach. **15**(2), 195–214 (2014)
47. Dorigo, M., Blum, C.: Ant colony optimization theory: a survey. Theor. Comput. Sci. **344**(2–3), 243–278 (2005)
48. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Appl. Soft Comput. J. **8**(1), 687–697 (2008)
49. Yang, X.-S.: A new metaheuristic bat-inspired algorithm. Stud. Comput. Intell. **284**, 65–74 (2010)
50. Askarzadeh, A.: A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput. Struct. **169**, 1–12 (2016)
51. Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: 2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009—Proceedings, pp. 210–214 (2009)
52. Yang, X.S.: Flower Pollination Algorithm for Global Optimization. Lecture Notes in Computer Science, vol. 7445 LNCS, pp. 240–249, 2012
53. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Adv. Eng. Softw. **69**, 46–61 (2014)
54. Gandomi, A.H., Alavi, A.H.: Krill herd: a new bio-inspired optimization algorithm. Commun. Nonlinear Sci. Numer. Simul. **17**(12), 4831–4845 (2012)
55. Mirjalili, S.: Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. Knowl. Based Syst. **89**, 228–249 (2015)

56. Kennedy, J., Eberhart, R.: Particle swarm optimization. IEEE Int. Conf. Neural Networks **4**, 1942–1948 (1995)
57. Kirkpatrick, S., Gelatt, C.D., Vecch, M.P.: Optimization by simulated annealing. *Science (80-.)* **220**(4598), 671–680 (2007)
58. Siddique, N., Adeli, H.: Simulated annealing, its variants and engineering applications. Int. J. Artif. Intell. Tools **25**(06), 1630001 (2016)
59. Mirjalili, S.: SCA: A Sine Cosine Algorithm for solving optimization problems. Knowl. Based Syst. **96**, 120–133 (2016)

Chapter 2
Metaheuristics and Swarm Methods: A Discussion on Their Performance and Applications

Abstract Nature-inspired metaheuristics are easily the largest family of optimization techniques currently on existence and they have become widely-known among researchers from virtually every single area of scientific application. While the rapid development of this area of science has originated a vast amount of novel problem-solving schemes, it has also brought many interesting questions. Nowadays, researchers are centering their attention on studying the properties on nature-inspired methods that have a direct impact on their performance, and how these properties contribute on better solving particular optimization problems. In this chapter, we present a discussion centered on several observable characteristics in nature-inspired methods and their influence on its overall performance. Furthermore, we also present a survey on some of the most important areas science and technology where nature-inspired algorithms have found applications. Finally, we expose some of the current research gaps regarding to the development and application of nature-inspired metaheuristics, as well as some of the potential directions that this area of science may take in the future.

2.1 On the Performance of Nature-Inspired Metaheuristics

Nowadays, nature-inspired metaheuristics have become so numerous and varied in terms of design and applications. From such an abundant variety of techniques, the main question to be addressed is: Which metaheuristic technique performs the best overall? This question, present since the formulation of the first of such algorithms, has been surprisingly hard to answer, and up to this day, it remains as an open concern in this area of science [1]. One widely accepted theory in this research field is that metaheuristic algorithms perform best over a broad spectrum of problems when a proper balance between exploration and exploitation is present in their mechanism. In general terms, exploration refers to the ability of search agents to visit entirely new regions of a search space, while exploitation emphasizes on the capabilities of these agents refine currently known “good” solutions [2]. While the importance of such a balance is recognized as essential in most new proposals, it is often loosely implied given the lack of appropriate analysis tools that allow understanding how

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_2

an algorithm search mechanisms affect it. Another factor that highly affects this balance is the selection of parameters (tuning) designed to control the exploration and exploitation pressure of each algorithm. However, this process is not straightforward, and it highly depends on the algorithm and the problem to solve. Also, there is the so-called 'No-free-lunch (NFL) theorem' proposed by Wolpert and Macready in 1997, which states that any algorithm will on average perform equally well as a random search algorithm over all possible functions. For this reason, it is assumed that statistical methods can be applied to find the dominance of one algorithm over others on a specific problem; however, as of today, there is no objective way to point out which algorithm is the overall best and the reasons of such superiority, if existent [4]. In this section, we open a discussion about the several observable characteristics of metaheuristic algorithms and how these characteristics impact the performance of these methods (see Table 2.1).

2.1.1 Computational Complexity

An in-depth analysis of the mechanisms implemented on nature-inspired metaheuristics allows pointing on several characteristics which have a direct impact on the expected computational complexity of these methods [3]. Depending on their search strategy, for example, some algorithms may require to sort the available candidate solutions with regard to their fitness value, either to find the best members from the population (as in GA), select several good solutions to implement their search strategy (as done in GWO), or even as a tool for efficient implementation (as in the case of FA). Although population sorting can be implemented using several sort algorithms, for most cases it is also important to consider their implied computational cost. In the case of the default sort function employed by MatLab® (QuickSort), for example, the computational complexity of the sorting operation is $O(P\log(P))$ in the average case scenario. The additional complexity these operations add breaks the desired linear complexity that is often sough when developing optimization algorithms, especially when sorting is required on each iteration (although this becomes a real burden only if the population size is too large). On the other hand, some algorithms may require to calculate some kind of population-related measurement(s) as part of their search mechanisms. A measurement commonly computed in several nature-inspired algorithms is the Euclidian distance between individuals within the search space. As seen in the case of GSA, for example, such distance measurements are used to compute a sum of weighted attraction between solutions. On the other hand, in algorithms such as SMS, the Euclidian distance between individuals is compared with some sort of distance threshold in order to establish conditions for the movement of search agents. While the Euclidian distance often proves to be useful for the purpose of modeling complex search strategies, the added computational complexity that this implies must always be considered. Under the supposition that Euclidian distances are needed to be calculated with regard to every single pair of individuals in the population (worst case scenario), this will require at least as it requires $O\left(P^\wedge 2\right)$ root

Table 2.1 Observable performance-influencing characteristics on several popular nature-inspired metaheuristics

Algorithms	Exploration/exploitation			Computational complexity			Required additional memory	Parameter tuning process	Overall implementation difficulty
	Selection mechanism	Attraction operators	Iteration dependent	Population sorting	Population-related measurements	Variable FFC			
DE	Ind. greedy	N/A	No	No	No	No	No	Easy	Low
ES	Non-greedy	Global best	No	No	No	No	No	Hard	Low
GA	Greddy	N/A	No	Yes	No	No	No	Easy	LOW
ABC	Ind. greedy	Multiple	No	No	No	Yes	Yes	Tuneless	Medium
BA	Ind. greedy	Global best	No	No	No	No	Yes	Easy	Low
CSA	Non-greedy	Personal	No	No	No	No	Yes	Easy	Low
CS	Ind. greedy	Global best	No	No	No	No	No	Easy	Medium
FA	Non-greedy	Multiple	No	No	Yes	Yes	Yes	Hard	Medium
FPA	Non-greedy	Global best	No	No	No	No	No	Easy	Low
GWO	Non-greedy	Multiple	No	Yes	No	No	No	Tuneless	Low
KH	Ind. greedy	Global best	No	No	Yes	No	Yes	Hard	Medium
MFO	Greedy	Multiple	Yes	Yes	No	No	Yes	Tuneless	Medium
PSO	Non-greedy	Multiple	No	No	No	No	Yes	Easy	Low
SSO	Ind. greedy	Multiple	No	No	Yes	No	Yes	Hard	High
WOA	Non-greedy	Global best	No	No	No	No	No	Tuneless	Low
EMO	Non-greedy	Multiple	No	No	Yes	No	Yes	Tuneless	Medium
GSA	Non-greedy	Multiple	Yes	No	Yes	NO	Yes	Easy	Medium
SCA	Non-greedy	Global best	Yes	No	No	No	No	Tuneless	Low
SMS	Non-greedy	Global best	Yes	No	Yes	No	No	Hard	High
FWA	Greedy	Multiple	No	Yes	Yes	Yes	Yes	Hard	High
HS	Ind. greedy	N/A	No	No	No	No	No	Easy	Low
ICA	Non-greedy	Multiple	No	No	Yes	Yes	Yes	Hard	High

square calculations (although this can be reduced under certain conditions). Finally, while it is common to find optimization algorithms that perform a fixed number of fitness function computations (FFCs), there are several metaheuristic algorithms that break this rule. In methods such as ABC, for example, scout bees are deployed to explore new solutions within the search space once it is determined that any of the currently known solutions cannot be improved, hence requiring an additional FFC for each abandoned solution. Similarly, in the CS algorithm, a secondary mechanism devised to generate new random solutions is implemented as a mean to increase diversity, adding additional FFCs in the process.

2.1.2 Memory Efficiency

All the population-based metaheuristic required a minimum memory with size $O(N * (D + 1))$ (with N denoting the number of elements D their dimensionality) to store each of the available solution vector solution plus their correspondently fitness value, and $(N * (D + 1) + a * (D + 1))$ (with $a \in \{1, 2\}$) if memorizing the best and/or worst solution(s) is required. Furthermore, some nature-inspired algorithms are known to require extra memory space to store some kind of additional data that is required as part of their search strategy. Some examples of algorithms that required such extra information include SMS, KH and GSA, where measurements such as the Euclidian distances are constantly computed. In these algorithms, such distance measurements are store in memory in order to reduce the number root squares calculations that are needed at each iteration, instead of performing the calculation as they're required [4]. Another algorithm with a more complex demand of memory is the ICA that require the calculation of the distance between colonies and empires, and their proposed costs on each iteration. While these computational requirements may represent no issue for today standard computers, this can severally limit their application on hardware with limited resources (such as those developed with the idea of portability in mind).

2.1.3 Exploration Versus Exploitation

Although seemingly trivial, questions about how exploration and exploitation is achieved are still an open subject, and it's often a source of disagreement among researchers [5, 6]; however, it is commonly accepted among the research community that a good ratio between exploration and exploitation is essential to ensure good performance in metaheuristic search algorithms. The question here is: how can we find a proper balance between these two crucial characteristics? Answering this question is not trivial given that metaheuristic optimization algorithms can be very different in terms of search strategy, so it is necessary to understand first what type of mechanisms are implemented in these methods. In the case of population-based metaheuristics, for example, these usually include selection mechanisms that allow them to collect prominent solutions from among the entire population, either to make them prevail for the next iteration of the search process or to implement some solution update strategy. Although selection operators do not modify or generate new solutions in the population, they play an important role in balancing elitism and diversity [7]. In the case of a greedy selection mechanism, for example, this ensures that the most fitting individuals among candidate solutions remain intact for the next generation. Thus, this type of selection is often used to improve fast convergence on metaheuristic algorithms toward promising solutions. Another type of elitism (individual greedy selection) can be observed in algorithms like DE or HS, where a new solution is accepted only if it directly improves the solution that originates them. This strategy has the appeal of potentially improves the exploration-exploitation ratio by forcing a highly diverse initial population to improve individually from its starting point, and as such this kind of selection method is often preferred in metaheuristic optimization methods, especially when these are meant to deal with multimodal problems [8]. Also, some algorithms do not implement any kind of selection strategy at all. In the case of methods like PSO or GWO, these do not enforce the selection of prominent individuals, accepting every new solution independently of its quality. In this case, the lack of elitism is preferred to accent exploration over exploitation; however, this methods also implement additional behaviors in order balance the exploration-exploitation rate as is, for example, "attractions" toward the best know solution. Speaking of attractions, it is not uncommon to find some kind of attraction strategy implemented as a part of the search mechanisms of nature-inspired algorithms. As previously implied, attractions are usually applied as an exploitation mechanism that seeks to improve currently known solutions by moving other solutions toward the location of seemingly "good" individuals. Choosing which solutions are to be considered as attractors, and how other solutions will be attracted to these attractors is entirely dependent on the design of the algorithm itself. In the case of PSO, for example, particles are attracted toward the global best solution at the current iteration of the search process; however, individuals modeled in PSO also implement a series of attractions toward the best solutions recorded by each particle as the search process evolves (personal best solution). This approach is usually considered as a more balanced attraction mechanism regarding convergence and solutions diversity; however, implementing this kind of strategy requires the allocation of additional memory, which could be undesired depending on the intended application(s). Also, some algorithms apply attraction mechanisms which consider the composite effect of more than one attractor to discover new potential solutions. These attractors can be comprised by a subset of members from the current population of solutions, or even, by the whole set of available candidate solutions. As previously noted, in PSO both the best-known solution of the population and the best personal record of each solution are considered to modify the velocity of each particle; this could be considered an example of such multiple-attractors phenomenon. On the other hand, there are methods that implement more complex attraction schemes, which consider very specific members of the entire population as well as other particular properties. In FA, for example, the attractiveness between individuals is set to be inversely proportional to the distance that separates them, hence, the longer the distance, the lower the attraction [6]. Similarly, GSA attractiveness is dictated by "gravitation force" exerted among particles in the available search space, but the magnitude of such attraction also depends on the fitness value of each particular solution. While these mechanisms have proven to be effective for maintaining higher diversity on the population, it is worth noting that modeling these behaviors requires to constantly calculate of the distance between several pairs of solutions, increasing the computational complexity to these the algorithms. Also, it's been observed that this kind of mechanisms tend to slow-down the convergence of the algorithm, a fact that must be considered is execution time is a priority [6]. Furthermore, some algorithms do not contemplate any type of attraction as part of their search strategy; instead, these methods generate new solutions by means pure random walks (as in the case of HS) or by considering other criterions (such as the distance between solutions, as in the case of DE). In evolutionary algorithms like GA, for example, some solutions are generated by "mixing" the information of randomly chosen solution (crossover), whereas others are generated by adding perturbations to currently existing solutions (mutation). In this regard, it's also worth mentioning that, while crossover and mutation operators in evolutionary algorithms are often perceived as exploration and exploitation operators, respectively, a deeper observation of crossover behaviors suggest that at the beginning of the evolutionary process (where the population is still diverse), crossover operators favor the exploration of solutions, whereas, toward the end of the search process (where population has lost diversity), exploration capabilities are dramatically reduced. Similarly, mutation operators that consider large amounts of perturbations for modifying existing solution could very easily be considered as exploration mechanisms, as solutions could be generated over much larger proportions of the feasible solution space. Under this considerations, it is hard to roughly classify crossover and mutation as either exploration or exploitation operators, as they intended behavior could be altered by adjusting their crossover and mutation rates, respectively. Finally, it worth noting that there are several algorithms which consider the iterations stop criterion (maximum number of iterations), as well as the iterations progress as part of their search strategy. Such mechanisms are mostly used to adjust the exploration-exploitation rate as the search process evolves with the purpose of avoiding premature convergence. However, this strategy often prevents the algorithm to converge quickly toward currently known best solutions, which could be undesired depending on the situation.

2.1.4 Implementation

As mention in Sect. 2.2, most metaheuristic algorithms share a general framework independently of the inspiration. Therefore the main difference is present in their mechanism to generating new solutions and selecting those that will remain to the next iteration. Among the algorithms presented in this work, some have a high level of simplicity that can be translated onto computational code with relative ease, while others may be relatively complex to program given the kind of behaviors and rules these intend to model. In other words, it could be said that the number of lines of code required to program a given metaheuristic algorithm increases as more sophisticated mechanisms are integrated to these methods. In this sense, algorithms such as SCA can be considered among the most straightforward algorithms to code as all of the population is directly attracted toward the best solution, without the need of sorting the population or excessive memory assignation. HS may be as well considered ease to code given how simple is for it to generate new candidate solutions (generating random values, values from known solutions, or slight perturbations of currently existing ones). However, the fact that HS resort to a greedy selection mechanism demands it to perform population sorting, slightly increasing its coding difficulty, and by extension its computational complexity. Leaving aside the performance that a given algorithm could have when applied to solve problems, the degree of coding complexity that relates to these computational approaches is plagued by two particular concerns [9]: 1. A higher risk for some of its elements to introduce structural biases, making the algorithm more prone to explore several parts of the optimization landscape more frequently than others without an actual justification [10]; and 2. The discouragement this coding complexity could cause to its potential users. Besides, most (if not all) metaheuristic algorithms are designed to work as black box models, thus being problem independent. However, regarding implementation, users often require to invest additional time to specify other essential characteristics, such as the formulation of the fitness function that describes the problem, the codification (representation) of the solutions and, if necessary, the parameters tuning for the algorithm in question. Speaking of parameter tuning, it is important to remember that its relevance lies on the fact that it allows to change the exploration-exploitation ratio of the algorithm, potentially allowing it to perform better over certain tasks. The main problem here though, is that there is no universal agreement on how metaheuristic algorithms should be tuned to work optimally over specific optimization problems (which is not surprising given that most algorithms are comparably different); in fact scientists and practitioners are for the most part used to tune metaheuristics by hand, guided only by their experience and by some rules of thumb, hence, tuning metaheuristics is often considered to be more of an art than a science. Although, there are several efforts aimed to provide frameworks to mechanically and objectively select these parameters finding the optimal tuning for these parameters is still an open problem [11, 12]. Another issue related to parameter tuning in metaheuristic algorithms is the number tuning parameters itself. While some algorithms like GA or PSO have only a reduced number of parameters (two in both cases), other methods such as ICA or SMS requires the user to set several more parameters, needed by the algorithm in order to control their behavior. However, as the number of parameters to set increases, understanding how these values influence the performance of the algorithm becomes more complex, thus, making the algorithm harder to analyze overall. Finally, it is worth noting that there are algorithms that do not have any parameters to tune at all. Methods such as WOA and SCA integrate mechanisms devised to automatically adapt the exploration-exploitation rate as they progress over the available iterations, but these do not need the user to set any parameters for it to work. Methods like these offer inexperienced users the ability to easily implement and understand the mechanisms behind metaheuristic algorithms, although these private more experimented users from studying its behavior in a more complex perspective.

2.2 Nature-Inspired Metaheuristics and Their Applications

In recent years, nature-inspired metaheuristics have become popular choices for solving a wide-range of optimization-related problems in many different areas of application such as engineering design, digital image processing, and computer vision, networks and communications, power, and energy management, data analysis and machine learning, robotics, medical diagnosis, and many others [13]. In this section, we will discuss several implementations of nature-inspired algorithms for solving real-world problems in different areas of application.

2.2.1 Engineering Design

Applications of nature-inspired metaheuristics to engineering design are as varied as the multidisciplinary areas of science currently on existence. For example, in the area of networks and communications, a popular design problem is the design of antennas. Notably, in Goudos (2017) the author presented a study where this design problem is tackled by applying several discrete-coded metaheuristics, including GA, DE, and PSO, demonstrating competence in all cases [14]. Another representative area of applications is aeronautics, where the most common design problems are related to aircrafts design. In Keshtegar et al. (2017), for example, the authors proposed an optimization framework for the design of aircraft panels based on an adaptive dynamic HS (ADHS). This design approach was compared in terms of performance against those found in other variants of HS, ultimately demonstrating ADHS to be the superior method [15]. Another common design application is the design of truss structures. As examples, we can mention the works presented in Bekdaş et al. (2015) and Khatibinia and Yazdani (2017), where algorithms such as FPA and GSA were successfully applied to solve this design problem [16, 17]. Another interesting application is reported in Shukla and Singh (2016), where the FA algorithm is implemented to aid on the selection of parameters for advanced machining processes with good results [18]. Other classical engineering design problems where nature-inspired algorithms have been successfully applied include the design of tension/compression springs, welded beams, pressure vessels, gear trains, to name a few [19–23].

2.2.2 Digital Image Processing and Computer Vision

Nature-inspired metaheuristics have also found interesting applications in the area of digital image processing and computer vision. One typical application in this regard is image segmentation by multilevel thresholding, in which the optimization algorithm is applied to found a set of optimal gray-level threshold that maximizes some kind of image measurement [24]. Prominent examples this kind of applications includes those presented in Horng and Jiang (2010), Ouadfel and Taleb-Ahmed (2016), Aziz et al. (2017), He and Huang (2017), Khairuzzaman and Chaudhury (2017), were algorithms such as ABC, SSO/FPA, WOA/MFO, FA, GWO have been successfully implemented [25–30]. Also, in Cuevas et al. (2013), Oliva et al. (2014) and Zhang and Zhou (2017), the task of template matching based on nature-inspired metaheuristics have been explored, where techniques such SMS, EMO, GWO have been implemented to good results [25, 27–30]. Also, in Olague and Trujillo (2012), the authors proposed to use a Multi-Objective GP (MO-GP) approach for the task of synthesizing operators for the detection of interest points in digital images, where the optimization problem is represented regarding three properties (stability, point dispersion, and information content). The experimental results presented by the authors suggest that their proposed approach is able to construct interest point detection operators adapted to different performance criteria, thus making it promising for a wide variety of computer vision applications [31]. Another interesting application is reported in Kiranyaz et al. (2015), where PSO is applied to assist on the task of perceptual dominant color extraction, presenting promising results when compared to some traditional methods [32].

2.2.3 Networks and Communications

Applications of nature-inspired metaheuristics to this area include the Optimal Sensor Deployment (OSD) for Wireless Sensor Networks (WSNs), a task that consists on finding an optimal distribution for a set of sensing devices designed to collect some kind of physical data. One recent application for this is reported in Zou et al. (2017), where the authors proposed an optimal sensor deployment scheme based on the SSO algorithm. The reported experiments suggest that the performance of SSO, when applied for OSD in WSNs, is superior to that of methods based on Virtual Force Algorithm (VFA) [33], GA, and PSO [34]. Another similar application is reported in Deif et al. (2017), where ACO is applied as the optimization method of choice. In this work, there is a special emphasis not only on maximizing the sensor networks' coverage area, but also other important requirements in WSNs, such as minimum level of reliability and deployment cost [35]. In similar terms, in Alia and Al-Ajouri (2017), the HS algorithm is applied for solving the problem of minimum cost OSD, comparing its performance in this case with that of a random deployment scheme [36]. In Mann and Singh (2017), the authors managed to improve the performance of the ABC algorithm by implementing the Student's t-distribution as a sampling method and applied it to perform energy-efficient clustering in Wireless Sensor Networks, yielding to promising results [37]. Other similar applications of nature-inspired methods for OSD in WSNs can be found in Goyal and Patterh (2015) and in Cao et al. (2012), where the techniques such as BA and FA have been studied [38, 39]. Community detection in complex networks is another application that has caught the interest of researchers as a candidate to be solved by metaheuristic techniques. In Rahimi et al. (2018), for example, a novel multi-objective community detection scheme based on PSO is proposed. The authors presented experimental results that consider both synthetic and real-world environments and compared their results against those of other similar approaches, showing an outstanding performance [40]. Also, in Guerrero et al. (2017), the problem of adaptive community detection is handled by applying a modification of the GA algorithm coined Generational Genetic Algorithm (GGA+), which involves efficient initialization methods and modularity-guided search operators, is applied and compared against other GA variants, demonstrating superior performance [41]. Another interesting application of nature-inspired methods to this area is documented in Bhardwaj et al. (2014), were the ABC algorithm is implemented for the detection of malicious URLs [42]. Also, in Din et al. (2016), the CS algorithm is applied to perform Left Feedback Shift Registers (LFSR) cryptanalysis, a tool mainly employed in information security [43]. Finally, E-mail Spam detection via nature-inspired methods has also been a subject study. In Idris et al. (2015), for example, the authors proposed an e-mail spam detection system based on a combination between Negative Selection Algorithm (NSA) [44] and PSO, where the latter is applied to improve the random detector generation phase in the former. The reported statistical data reported suggest a significant improvement in performance when compared to a framework based exclusively in NSA, thus proving the significance of the proposed modification [45].

2.2.4 Power and Energy Management

Applications of nature-inspired metaheuristics to the area of energy applications are quite numerous. In Mesbahi et al. (2017), for example, the authors proposed an optimal energy management strategy for hybrid energy storage systems based on PSO and the Nelder-Mead simplex method, which show to have a significant improvement in both battery usage and lifetime when compared to other conventional methods [46]. In You et al. (2017), a home energy management system based on the SSO algorithm was proposed, showing interesting results when applied for the task of appliances load management [47]. In 2016, Guha et al. presented an approach based on the GWO algorithm aimed to solve the problem of load frequency control in interconnected power systems networks by tuning the parameters of a PI/PID controller, showing to outperform schemes based on other similar metaheuristic techniques [48]. In Prasad et al. (2017), a modification of the KHA, known as Chaotic Krill Herd Algorithm (C-KHA) was implemented to solve the problem of Optimal Power Flow (OPF) while also comparing its performance against other state-of-the-art metaheuristics, yielding to favorable results [49]. Another interesting application was proposed in Sickel et al. (2007), where the DE algorithm is used for the intelligent control of power plants. In this work, DE is applied to both generate a set of optimal points for the monitoring of a reference power governor and the parameter tuning of the same power plant controller [50]. Also in Al-Betar et al. (2018), an approach aimed to solve the problem of Economic Load Dispatch (ELD) was proposed, in which an approach called β-Hill Climbing is applied to minimize the total fuel cost of operation and emission from a set of generating units. The proposed method was evaluated by considering several real-world ELD systems and compared against other similar approaches (including some based in metaheuristics such as GA, ACO, HS and PSO), demonstrating β-Hill Climbing to be superior for solving this particular problem [51]. Furthermore, in Babu et al. (2017), the authors proposed the use of PSO for the reconfiguration of photovoltaic (PV) panels aimed to maximize power extraction, presenting promising results when compared to other similar approaches [52]. Related to the subject of PV-based power generation, another interesting application represented by parameter identification in PV cells/modules. In Oliva et al. (2014), for example, the Artificial Bee Colony (ABC) algorithm was implemented and compared against other similar techniques in terms of performance, showing promising results [53]. Other similar applications for PV cells parameter identification can be found in Askarzadeh and Rezazadeh (2012), Ma et al. (2013), Han et al. (2014), and Sarjila et al. (2016), where algorithms such as FA, CS, AFSA, GSA where successfully implemented [54–57]. In Valdivia-Gonzales et al. (2017), an intelligent power allocation scheme for plug-in hybrid vehicles (PHEVs) based on SMS was proposed, in which the objective is to adjust the power distribution provided by PHEV's charging infrastructures by taking into account customer characteristics and restrictions [58]. In Prakash and Lakshminarayana (2016), the authors prosed to tackle the problem of optimal capacitor placement in Radial Distribution Networks by applying an optimization scheme based in WOA, demonstrating to be much more effective than most traditional techniques applied for this purpose [59]. Other applications include those presented in Massan et al. (2015) and Tolabi and Ayob (2016), where nature inspired metaheuristics such FA and a hybridization between SA and GA are applied for the task of optimal wind turbines allocation [60] and solar radiation forecasting [61], respectively, yielding to interesting results in both cases.

2.2.5 Data Analysis and Machine Learning

As for the area of data analysis and machine learning, some prominent application related to these areas include feature selection. In Mafarja and Mirjalili (2016) proposed a series of wrapper feature selection scheme based on hybridization between the WOA and SA algorithms, where the former is used to promote exploration while the latter is applied to enhance exploitation. The proposed approach was compared in terms of performance against other similar methods based on metaheuristics, such as ALO, PSO, and GA, proving to have the best performance regarding accuracy and average selection size [62]. In the same year, Hafez et al. presented a feature selection approach based on the SCA algorithm, and compared it in terms of performance against methods based in PSO and GA, leading to favorable results. Similarly, in the authors published a feature selection methodology based on a binary GWO which was also successful regarding performance against PSO and GA. Also, in 2017, Moayedikia et al. proposed a feature selection algorithm called SYMON which applies both symmetrical uncertainties along with the HS algorithm to develop a strategy to rank features in high dimensional imbalanced class datasets, proving to be superior when compared to other similar techniques [63]. Other interesting feature selection applications can be found in [64, 65], Another application related to data analysis is data clustering, where metaheuristics have also been applied for good results. In 2018, for example, Alswaitti et al. proposed a clustering method based in PSO which integrates a density estimation technique based on Gaussian kernels developed to address premature convergence as well as a method to estimate the best learning coefficients. The proposed approach was compared against other techniques commonly used for data clustering and proved to be significantly better in terms of accuracy [66]. Also, in Zhou et al. (2017) the SSO algorithm was modified to implement a Simplex method in order to improve data clustering on higher dimensions, proving to be better than the original SSO in terms of performance, while also showing its superiority over other similar methods. Furthermore, in Abualigah et al. (2017, 2018) the authors proposed a clustering technique based on a hybrid KHA that integrates the mechanisms of the HS algorithm as a way enhance both exploration and exploitation of solutions. The algorithm was compared regarding performance against traditional clustering methods and approaches based on metaheuristic optimization algorithms demonstrating a higher performance in general [67]. Other similar works are reported in [68–71], where algorithms like the standard KH, GA, and PSO has been applied to improve text document clustering. Then, in Han et al. (2017) proposed a clustering method based on a variant of GSA that integrates an update mechanism devised to increase the diversity of solutions. This variant, called Bird Flocking GSA (BFGSA) was compared against the standard GSA as well as methods based on ABC, PSO, and FA, proving to be much more competent for the aforementioned task [72]. Other approaches for data clustering include those reported in Shukla and Nanda (2016) and Jadhav and Gomathi (2016), where methods such as SSO and a hybrid between GWO and WOA were implemented to good results [73, 74]. Other interesting applications of nature-inspired metaheuristics in this area involve their use as an alternative to train Artificial Neural Networks (ANN). In Sahlol et al. (2009), for example, the authors developed a feedforward ANN based on the SCA algorithm to improve the prediction accuracy of liver enzymes on fishes fed with certain compounds. In this approach the SCA algorithm is applied to find the configuration of weights/biases that allows the proposed NN to achieve optimal performance, yielding to much better results than those of previous prediction models [75]. In Rere et al. (2015), for example, the authors proposed to use the SA algorithm to train a Convolutional Neural Network (CNN) for the classification of handwritten digits. Although the proposed method comes with a significant increase in computation time, it also yields a substantial increase in performance when compared to other methods commonly used to train CNNs [76]. Another interesting application is reported in Pereira et al. (2016), where the SSO algorithm is used for both, feature selection and parameter tuning for a Support Vector Machine (SVM) designed to aid on energy theft detection. The authors compared the performance of their proposed approach against those based on PSO and a variant of the HS algorithm known as Novel Global Harmony Search (NGHS), highlighting their respective advantages [77].

2.2.6 Robotics

Implementations of nature-inspired optimization algorithms to the area of robotics usually include tasks such as path planning and trajectory optimization. Perhaps the most popular application in this regard is the autonomous navigation of Unmanned Aerial Vehicles (UAVs). Interesting work is reported in Li and Duan (2012), where an improved GSA (IGSA) approach is applied to develop a path planning strategy for Unmanned Aerial Vehicles (UAVs) devised for military applications. The proposed I-GSA based path planning method was compared against those based on the classic GSA and PSO, demonstrating much better performance [78]. Similarly, in Oz et al. (2013), path planning strategies for 3D environments based on GA and Hyper-Heuristics (HH) [79] that considers technical constraints and mission-specific objectives are proposed. Both algorithms were evaluated by considering other classical studies developed for UAV navigation, yielding to favorable results [80]. Then, in Behnck et al. (2015), the authors proposed a path planning strategy based on a modified SA algorithm. The reported results demonstrate that the proposed approach can calculate minimum distance paths for a pair of UAVs while also being sufficiently simple to be implemented in an embedded platform [81]. Also, in Xie and Zheng (2016), the problem of path planning in UAVs is handled by applying a search strategy based on a hybrid CS algorithm that integrates the mutation and crossover operators of GA, called Improved CS (ICS). As illustrated by the results presented in this work, the ICS notably outperform the standard CS algorithm, demonstrating its competence for the task in question [82]. Outside of UAVs applications, metaheuristic optimization approaches have also been successfully applied to solve the problem of navigation in other kinds of robotic platforms. In Tsai et al. (2016), for example, a path planning scheme based on a multi-objective GWO (MOGWO) approach is applied to aid on robot navigation over environments consisting on fixed positions and obstacles. The proposed path planning scheme was compared regarding performance against that based on a multi-objective GA (MOGA), ultimately proving MOGWO to be slightly superior [83]. Furthermore, in Contreras-Cruz et al. (2017), a distributed path planned method for multi-robot systems based on ABC (DPABC) was simulated and compared against a classic priority planner scheme and another approach also based on ABC (PPABC). As suggested by their experimental results, the proposed DPABC approach is favored as the better alternative for solving this problem due to it being able to solve the task in a lower time and with a better performance than that of the other compared methods [84]. Another interesting application documented in the literature is the development of controllers for robotic platforms. In Silva et al. (2014), for example, GP was applied as an automatic search method for motion primitives in a bipedal robot based on the exploration and exploitation of its particular characteristics. Experimental results demonstrated a significant improvement in performance on the applied robot's locomotion, especially when compared to that of a hard-tuned system [85]. More recently, in Benkhoud and Bouallègue (2017), the authors proposed a series of metaheuristic-based tuning strategies for a Linear Quadratic Gaussian (LQG) controller, with application to a special class of UAV. The algorithms studied in this work include the HS algorithm, Water Cycle Algorithm (WCA) [86], and Fractional PSO-based Memetic Algorithm (FPSOMA). Furthermore, comparisons which consider tuning methods based on the standard PSO and ABC algorithms where also developed, yielding to interesting results [87].

2.2.7 Medical Diagnosis

Interestingly, nature-inspired metaheuristics have found a plethora of applications aimed to develop tools to assist on medical diagnosis. Typical applications to this area include diagnostic applications based on the digital image processing medical images. In Ibrahim et al. (2012), for example, the authors proposed an approach to automatically measure ventricular heart volumes on cardiovascular magnetic resonance (CMR) images by applying an ACO-based approach that integrates iterative salient isolated thresholding (ACOISIT) to segment blood-myocardium borders and demonstrated promising results [88]. Furthermore, in Ouaddah and Boughaci (2016), the authors proposed a methodology to perform image reconstruction from projections based on the HS algorithm. In this approach, the HS algorithm is hybridized with a local search method to enhance its performance and improve the quality of images reconstructed from tomographic images tomographies. Both the original and hybrid HS, as well as two other traditional techniques implemented for image reconstruction, were compared regarding reconstruction quality, demonstrating, in the end, the proficiency of the proposed techniques [89]. Later, in Chen (2017), an image segmentation approach based on the ACO algorithm is proposed for the detection of Lung Lesions in chest computed tomographies. To validate the proposed systems the authors analyzed its accuracy by considering a specific database of lung patients, obtaining favorable results [90]. Similarly, in Oliva et al. (2017), the authors proposed an image segmentation method based on both cross entropy thresholding and the CSA algorithm, with applications to the analysis of magnetic resonance brain images. Said approach was compared against cross entropy segmentation techniques based on DE and HS, ultimately demonstrating to be superior in terms of performance [91]. Other applications of nature-inspired methods to medical diagnosis include the development of tools aimed to aid on the analysis of specific medical data. As an example, we have the work reported in Wang et al. (2015), where an

Improved Electromagnetism-like Algorithm (IEA) is proposed to develop a feature selection method for the prediction of diabetes mellitus. The proposed approach was tested by considering an extensive number of pertinent datasets, and its performance was compared with several other benchmark metaheuristic techniques reported in the literature, yielding to interesting results [65]. Furthermore, in Kora and Kalva (2015), the authors developed and improved BA approach to extract features from Electrocardiogram (ECG) signals, with the purpose of feeding them to a neural network classifier trained for the prediction of myocardial infarction on heart patients. Both the improved and the standard BA algorithm were compared regarding performance by also considering other possible classifiers, including NN, KNN, SVM, and others [92]. Also, in Nagpal et al. (2017), a feature selection approach based on GSA and k-nearest neighborhood classification is proposed for the task of efficient feature selection. According to the experimental results presented by the authors, the proposed method is able to reduce the number of significant features in the processed data to an average of 66%, while also showing a better performance than technique based on algorithms such as PSO and GA [93]. Then, in Sahoo and Chandra (2017), a variant of the GWO algorithm known as Non-dominated Sorting GWO (NSGWO) and the Multi-Objective GWO (MOGWO) are proposed to address the problem of feature selection to aid on the classification/detection of cervix lesions. The authors compared their proposed feature selection methods with those based on several multi-objective implementations of GA and FA, demonstrating the GWO variants to be superior in all cases [94]. Another interesting application is reported in Alshamlan et al. (2015), were the ABC algorithm is applied to aid on the task of gene selection for cancer classification using microarray datasets. The proposed approach combines a filtering stage based on a Minimum Redundancy Maximum Relevancy (mRMR) method and a wrapper method based in both BA and SVM, and it was compared with two other similar approaches based in GA (mRMR-GA) and PSO (mRMR-PSO), showing impressive results [95]. Similarly, in Alomari et al. (2017), a gene selection methodology based in BA was presented and compared with other popular gene selection methods, demonstrating to be more than competent for the task mentioned above [96].

2.3 Nature-Inspired Metaheuristics: Research Gaps and Future Directions

Undoubtedly, research on nature-inspired metaheuristic algorithms and their applications has grown at an accelerated rate. However, while there exists an overwhelming amount of related works reported on the literature, this research area is yet to reach the maturity level that other areas of science currently have. There are still several research gaps and areas of opportunity that are still to be explored by researchers on this rather young area of science. One of this areas of opportunity arise due to the fact that, to this date, there is no single metaheuristic optimization algorithm with the ability to effectively handle all existing problems [97]; in fact, the literature suggest that there exist several techniques that perform significantly better than other methods when applied to specific problems. In this regard, we can mention the widely known 0–1 knapsack optimization problem [98], on which numerous optimization techniques have been successfully applied with good results; in the case of the work presented in Sapra et al. (2017), for example, a comparative study of several metaheuristic algorithms applied to solve the knapsack problem is presented, where methods such as Tabu Search [99], Scatter Search [100] and Local Search [101] are the center of discussion. The experimental results presented in this work suggest that Tabu Search has the least deviation from the best known solution, suggesting it to be more reliable in this regard. However, this work also suggest that Scatter Search presents the least time complexity, being the best option among the three tested methods when execution time is crucial [102]. Another example is given by Feng et al. (2017), where a novel Binary Monarch Butterfly Optimization (BMBO) algorithm was proposed to solve this problem, and was further validated by comparing it with binary-coded implementations of ABC, CS, DE, and GA, demonstrating BMBO to have greater accuracy and convergence speed [103]. Other commonly studied optimization problem is represented by the famous Traveling Salesman Problem (TSP) [104]. An interesting study can be found in the work present by Saji and Riffi (2016), where a comparison between a novel Discrete Bat Algorithm (DBA) and discrete coded modifications for PSO, CS, and GSA-ACS-PSOT (a hybrid approach based on several metaheuristics) for solving the TSP is presented, demonstrating DBA to be the overall best method to handle this problem [105]. Similarly, in Zhou et al. (2017), the authors propose to solve several spherical TSP by applying a Discrete Greedy Flower Pollination Algorithm (DGFPA), and compared it with several variants of GA and Tabu Search, finally concluding that DGFPA performs the best in most cases [106]. Finally, we have the Vehicle Routing Problem (VRP) [107], a problem that nowadays could be considered a benchmark real-problem for validating the performance of optimization algorithms. While there are several variants to this particular problem, nature-inspired metaheuristic methods have been extensively applied for solving each of these [108–111]; in particular, it is common to see implementations to this kind of problems based on enhanced/improved implementations of well-known bio-inspired algorithms. One example of this is reported in Xu et al. (2018), where Dynamic VRP is handled by applying an Enhanced ACO algorithm (E-ACO). The proposed method was compared in terms of performance against the standard ACO algorithm and another improved variant known as K-means ACO (K-ACO), concluding E-ACO to be slightly superior [112]. Similarly, In Zhang and Lee (2015), an Improved ABC algorithm is applied to solve Capacitated VRP and was compared in terms of performance against the standard ABC approach, yielding to a superior performance [113]. Another example is presented in Xiang et al. (2015), where an improved PSO algorithim (NPSO) which integrates Gauss Mutation is developed for solving the VRP and compared in terms of performance with the original PSO, virtually outperforming it in both performance and efficiency [114]. From what was previously discussed, it must be concluded that the different degrees of performance on nature-inspired metaheuristics over certain problems is not only heavily influenced by the specific search mechanisms and "adaptations" applied by each applied method, but also by the particular challenges offered by each of these problems, hence there is no way to establish a particular method as the absolute best, and as such there is still a plenty of space for proposing new and innovative methodologies for solving these problems or to modify existing ones with the purpose of enhancing their performance and efficiency. In this sense, many researchers have proposed interesting ideas aimed to improve the performance of nature-inspired metaheuristics. Some researchers, for example, suggest that a hybridization of exact methods and metaheuristic-based techniques could lead to the development of algorithms with enhanced efficiency and convergence capabilities [115, 116]. While research in this regard is still somewhat limited, there are several works on the literature which serve as examples of successful applications of both kind of methods [117–121]. Other interesting ideas can also be found on the work presented by Zelinka (2015), were the author explore the possibility of improving that performance and diversity of search operators by implementing specific control mechanisms (such as deterministic chaos models) to alter the dynamics of swarm and evolutionary algorithms [122]. An example of a metaheuristic-based application that implement this kind of mechanism is presented in Valdivia-Gonzalez et al. (2017), where different chaos models were combined with the search operators of the GSA algorithm in order to enhance its performance [123]. Similarly, in Cuevas et al. (2017), deterministic chaos models where integrated to the search operators of novel swarm-based algorithm known as Locust Search (LS) [124–126], with the purpose enhancing its performance for the identification of parameters in fractional-order systems [127]. Also, in Hinojosa et al. (2017), a Multi-objective implementation of the CSA approach (MOCSA) was modified to integrate chaotic behaviors in order to enhance its solution diversity, demonstrating a notable increase on performance [128]. In the last few years, techniques aimed to improve the efficiency of metaheuristic optimization algorithms by implementing surrogated models are gaining popularity among researchers from this area of science. In Regis (2014), for example, the author proposed a surrogated-assisted optimization framework based on PSO coined Optimization by particle swarm Using Surrogates (OPUS) with the purpose of efficiently solving high-dimensional blackbox optimization problems, and implemented it to solve several real world problems, including groundwater bioremediation and watershed calibration [129]. Similarly, in Liu et al. (2016), an optimization approach based on Differential Evolution which implement Gaussian Process (GP) regressions in conjunction with Optimization by Radial Basis functions Interpolation in Trust-regions (ORBIT) [130] was developed to efficiently and reliably solve optimization problems. The performance of this method, called Multi-fidelity Gaussian Process and radial basis Memetic Differential Evolution (MGPMDE), was validated by considering a wide set of benchmark test functions, as well as with its applications for data mining [131]. While all of the previous suggests that there are still plenty of areas of opportunity for the development and application of nature-inspired metaheuristic optimization algorithms, perhaps the greatest gap in this area of science is the absence of theoretical foundations that allows to objectively analyze important characteristics of these techniques, such as convergence rate and efficiency. While there is a general agreement in that the good performance of nature-inspired algorithms is attributed to a proper balance between exploration and exploitation of solutions, the truth is that there is barely a clear definition of what these two concepts truly represent. In fact, classifying the search operators and strategies applied by nature-inspired methods is often an ambiguous task, as many of these seems to contribute in some way to both the exploration and exploitation of solutions, and even then, there is currently no clear way to objectively measure the rate of exploration/exploitation provided by these operators [2]. In the absence of mathematical analysis methods that could be applied to measure these properties, the performance of nature-inspired metaheuristics is mostly evaluated by applying ad hoc methodologies based on the quantitative analysis of certain validation metrics such as error, mean, median, standard deviation, and so on [132]. A recent attempt to measure these factors over the most popular swarm metaheuristic is presented in [5]. They proposed a modified equation from [133] that calculates the diversity of the population using the distance between the solutions in the search space. This diversity measurement calculates the averaged sum distance between all the solution to the median of each dimension and solutions. Then, the exploration percent of each iteration is calculated by dividing the diversity value between the maximum diversity value encounter in the process, and its inverse is considered as the exploitation rate. However, this idea is oversimplified, as it doesn't provide any information related to the optimization landscape itself, thus making it impossible to get objective conclusions in this regard. Also, it is of common knowledge that the performance of these methods is often evaluated over well-known benchmark test functions, developed to represent in some way important characteristics of real-world problems such as search space scale and imbalance. However, there is no theoretical evidence to prove if these test problems truly reflect on this important characteristics, and as such, these performance evaluation frameworks are often criticized as well [134, 135]. Finally, while the absence of theoretical foundations for proving the prowess of metaheuristic algorithms is a major issue, it is also worth noting that the absence of standardized frameworks for the implementation and comparison of this kind algorithms is virtually absent as well [132]; this means that most of the time, researchers are forced to implement algorithms coded by other researchers or, in the worst case scenario, code the algorithm themselves. Whichever the case, the main problem here is the fact that algorithms coded by different people tend to be somewhat different in terms of implementation and by extension they also tend to manifest different degrees of performance and efficiency (even in those algorithms meant to represent the same optimization technique); with that being said, for nature-inspired algorithms to be implemented and compared in fair terms, software platforms devised to allow the implementation and evaluation of this methods, all in the same terms, are also necessary.

2.4 Concluding Remarks

Nature-inspired metaheuristics have become widely popularized due to their capabilities for solving an ever-increasing amount of complex real-world problems. Their flexibility for application is mainly related to several observable characteristics that are present on each of these method, such as search strategy, population management, fitness function computations, memory usage, and so on. These individual traits has a strong impact on each algorithm's individual performance, hence giving them different capabilities in terms of exploration and exploitation of solutions; as a result, there are methods which are better suited for solving some specific problems than others, a fact that people looking to implement this methods must always have in consideration. Thanks to their unique qualities, nature-inspired metaheuristics have been successfully implemented to solve a plethora of optimization problems within several areas of application, including engineering design, digital image processing, and computer vision, networks and communications, power, and energy management, data analysis and machine learning, robotics, medical diagnosis, and many others. While there are still several research gaps that remain to be explored for this area of science to reach full maturity, there is no doubt that nature-inspired metaheuristics have earned a rightful place among many researchers as powerful tools for optimization. In fact, all of these remaining questions does nothing but serve as inspiration for the development of newer and better techniques. In either case, this rather young but interesting area of science is expected to keep growing in the following years not only on the number of proposed approaches, but also in terms of complexity and quality.

References

1. Neumann, F., Witt, C.: Bioinspired computation in combinatorial optimization—algorithms and their computational complexity
2. Črepiňsek, M., Liu, S.-H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput. Surv. Art. **45**(33), 1–33 (2013)
3. Avigad, J., Donnelly, K.: Formalizing O notation in Isabelle/HOL. In: International Joint Conference on Automated Reasoning, pp. 357–371 (2004)
4. Yang, X.S., He, X.: Firefly algorithm: recent advances and applications. Int. J. Swarm Intell. **1**(1), 1–14 (2013)
5. Ghazali, R., Deris, M.M., Nawi, N.M., Abawajy, J.H. (eds.) Recent Advances on Soft Computing and Data Mining, vol. 700, no. Scdm (2018)
6. Yang, X.S., Deb, S., Hanne T., He, X.: Attraction and diffusion in nature-inspired optimization algorithms. Neural Comput. Appl. **19** (2015)
7. Du, H., Wang, Z., Zhan, W.E.I.: Elitism and distance strategy for selection of evolutionary algorithms. IEEE Access **6**, 44531–44541 (2018)
8. Huang, T., Jia, X., Yuan, H., Jiang, J.: Niching community based differential evolution for multimodal optimization problems (2017)
9. Piotrowski, A.P., Napiorkowski, J.J.: Some metaheuristics should be simplified. Inf. Sci. (Ny) **427**, 32–62 (2018)

10. Piotrowski, A.P., Napiorkowski, J.J.: Searching for structural bias in particle swarm optimization and differential evolution algorithms. Swarm Intell. **10**(4), 307–353 (2016)
11. Yang, X.-S.: Swarm-based metaheuristic algorithms and no-free-lunch theorems. Intech Open **2**, 64 (2018)
12. Sipper, M., Fu, W., Ahuja, K., Moore, J.H.: Investigating the parameter space of evolutionary algorithms. 1–14 (2018)
13. Osman, I.H., Laporte, G.: Metaheuristics: a bibliography. Ann. Oper. Res. **63**(5), 511–623 (1996)
14. Goudos, S.K.: Antenna design using binary differential evolution. IEEE Antennas Propag. Mag. February (2017)
15. Keshtegar, B., Hao, P., Wang, Y., Li, Y.: Optimum design of aircraft panels based on adaptive dynamic harmony search. Thin-Walled Struct. **118**(May), 37–45 (2017)
16. Bekdaş, G., Nigdeli, S.M., Yang, X.S.: Sizing optimization of truss structures using flower pollination algorithm. Appl. Soft Comput. J. **37**, 322–331 (2015)
17. Khatibinia, M., Yazdani, H.: Accelerated multi-gravitational search algorithm for size optimization of truss structures. Swarm Evol. Comput. December 2016, 0–1 (2017)
18. Shukla, R., Singh, D.: Selection of parameters for advanced machining processes using firefly algorithm. Eng. Sci. Technol. Int. J. **20**(1), 1–10 (2016)
19. Kaveh, A., Khayatazad, M.: A new meta-heuristic method: ray optimization. Comput. Struct. **112–113**, 283–294 (2012)
20. Askarzadeh, A.: A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput. Struct. **169**, 1–12 (2016)
21. Mirjalili, S., Lewis, A.: The whale optimization algorithm. Adv. Eng. Softw. **95**, 51–67 (2016)
22. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Adv. Eng. Softw. **69**, 46–61 (2014)
23. Camarena, O., Cuevas, E., Pérez-cisneros, M., Fausto, F., González, A., Valdivia, A.: Ls-II : an improved locust search algorithm for solving constrained optimization problems (2018)
24. Mesejo, P., Ibáñez, Ó., Cordón, Ó., Cagnoni, S.: A survey on image segmentation using metaheuristic-based deformable models: state of the art and critical analysis. Appl. Soft Comput. J. **44**, 1–29 (2016)
25. Khairuzzaman, A.K.M., Chaudhury, S.: Multilevel thresholding using grey wolf optimizer for image segmentation. Expert Syst. Appl. **86**, 64–76 (2017)
26. Khairuzzaman, A.K.M., Chadhury, S.: Moth-flame optimization algorithm based multilevel thresholding for image segmentation. Int. J. Appl. Metaheuristic Comput. **8**(4), 58–83 (2017)
27. Horng, M.-H., Jiang, T.-W.: Multilevel image thresholding selection using the artificial bee colony algorithm. Artif. Intell. Comput. Intell. **6320**, 318–325 (2010)
28. Ouadfel, S., Taleb-Ahmed, A.: Social spiders optimization and flower pollination algorithm for multilevel image thresholding: a performance study. Expert Syst. Appl. **55**, 566–584 (2016)
29. El Aziz, M.A., Ewees, A.A., Hassanien, A.E.: Whale optimization algorithm and moth-flame optimization for multilevel thresholding image segmentation. Expert Syst. Appl. **83**, 242–256 (2017)
30. He, L., Huang, S.: Modified firefly algorithm based multilevel thresholding for color image segmentation. Neurocomputing **240**, 152–174 (2017)
31. Olague, G., Trujillo, L.: Interest point detection through multiobjective genetic programming. Appl. Soft Comput. J. **12**(8), 2566–2582 (2012)
32. Kiranyaz, S., Uhlmann, S., Ince, T., Gabbouj, M..: Perceptual dominant color extraction by multidimensional particle swarm optimization. EURASIP J. Adv. Signal Process **2009** (2015)
33. Zou, Y., Chakrabarty, K.: Sensor deployment and target localization based on virtual forces. In: Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 2, no. C, pp. 1293–1303 (2003)
34. Zhou, Y., Zhao, R., Luo, Q., Wen, C.: Sensor deployment scheme based on social spider optimization algorithm for wireless sensor networks. Neural Process. Lett. (2017)
35. Deif, D.S., Member, S., Gadallah, Y., Member, S.: An ant colony optimization approach for the deployment of reliable wireless sensor networks. IEEE Access **5**, 10744–10756 (2017)

36. Alia, O.M., Al-Ajouri, A.: Maximizing wireless sensor network coverage with minimum cost using harmony search algorithm. IEEE Sens. J. **17**(3), 882–896 (2017)
37. Mann, P.S., Singh, S.: Improved metaheuristic based energy-efficient clustering protocol for wireless sensor networks. Eng. Appl. Artif. Intell. **57**(November 2016), 142–152 (2017)
38. Goyal, S., Patterh, M.S.: Performance of BAT algorithm on localization of wireless sensor network. Wireless Pers. Commun. **6**(3), 351–358 (2015)
39. Cao, S., Wang, J., Gu, X.: A wireless sensor network location algorithm based on firefly algorithm. In: Asia Simulation Conference 2012, pp. 18–26 (2012)
40. Rahimi, S., Abdollahpouri, A., Moradi, P.: A multi-objective particle swarm optimization algorithm for community detection in complex networks. Swarm Evol. Comput. **39**(February 2017), 297–309 (2018)
41. Guerrero, M., Montoya, F.G., Baños, R., Alcayde, A., Gil, C.: Adaptive community detection in complex networks using genetic algorithms. Neurocomputing **266**, 101–113 (2017)
42. Bhardwaj, T., Sharma, T.K., Pandit, M.R.: Social engineering prevention by detecting malicious URLs using artificial bee colony algorithm. In: 3rd International Conference on Soft Computing for Problem Solving, Advances in Intelligent Systems, pp. 355–363 (2014)
43. Din, M., Pal, S.K., Muttoo, S.K., Jain, A.: Applying Cuckoo search for analysis of LFSR based cryptosystem. Perspect. Sci. **8**, 435–439 (2016)
44. Johny, D.C., Assistant, A.J.S.: Negative selection algorithm: a survey. Int. J. Sci. Eng. Technol. Res. **6**(4), 711–715 (2017)
45. Idris, I., et al.: A combined negative selection algorithm-particle swarm optimization for an email spam detection system. Eng. Appl. Artif. Intell. **39**, 33–44 (2015)
46. Mesbahi, T., Rizoug, N., Bartholomeus, P., Sadoun, R., Khenfri, F., Lemoigne, P.: Optimal energy management for a Li-ion battery/supercapacitor hybrid energy storage system based on particle swarm optimization incorporating Nelder-Mead simplex approach. IEEE Trans. Intell. Veh. **2**(2), 1–1 (2017)
47. You, I., Yim,K., Barolli, L.: A social spider optimization based home energy management system. In: International Conference on Network-Based Information Systems, pp. 771–778 (2017)
48. Guha, D., Roy, P.K., Banerjee, S.: Load frequency control of interconnected power system using grey wolf optimization. Swarm Evol. Comput. **27**, 97–115 (2016)
49. Prasad, D., Mukherjee, A., Mukherjee, V.: Application of chaotic krill herd algorithm for optimal power flow with direct current link placement problem. Chaos, Solitons Fractals **103**, 90–100 (2017)
50. Van Sickel, J.H., Lee, K.Y., Heo, J.S.: Differential evolution and its applications to power plant control. In: 14th International Conference on Intelligent Systems Applications to Power Systems, no. 2, pp. 560–565 (2007)
51. Al-Betar, M.A., Awadallah, M.A., Abu Doush, I., Alsukhni, E., ALkhraisat, H.: A non-convex economic dispatch problem with valve loading effect using a new modified $$\beta $$β-Hill climbing local search algorithm. Arab. J. Sci. Eng. (2018)
52. Babu, T.S., Ram, J.P., Dragicevic, T., Miyatake, M., Blaabjerg, F., Rajasekar, N.: Particle swarm optimization based solar PV array reconfiguration of the maximum power extraction under partial shading conditions. IEEE Trans. Sustain. Energy **3029**(c) (2017)
53. Oliva, D., Cuevas, E., Pajares, G.: Parameter identification of solar cells using artificial bee colony optimization. Energy **72**, 93–102 (2014)
54. Han, W., Wang, H., Chen, L.: Parameters identification for photovoltaic module based on an improved artificial fish swarm algorithm. **2014** (2014)
55. Askarzadeh, A., Rezazadeh, A.: Parameter identification for solar cell models using harmony search-based algorithms. Sol. Energy **86**(11), 3241–3249 (2012)
56. Sarjila, K., Ravi, K., Edward, J.B., Kumar, K.S., Prasad, A.: Parameter extraction of solar photovoltaic modules using gravitational search algorithm. **2016** (2016)
57. Ma, J., Ting, T.O., Man, K.L., Zhang, N., Guan, S.U., Wong, P.W.H.: Parameter estimation of photovoltaic models via cuckoo search. J. Appl. Math. **2013**, 10–12 (2013)

58. Valdivia-Gonzalez, A., Zaldívar, D., Fausto, F., Camarena, O., Cuevas, E., Perez-Cisneros, M.: A states of matter search-based approach for solving the problem of intelligent power allocation in plug-in hybrid electric vehicles. Energies **10**(1) (2017)
59. Prakash, D.B., Lakshminarayana, C.: Optimal siting of capacitors in radial distribution network using whale optimization algorithm. Alexandria Eng. J. (2016)
60. Massan, S.U.R., Wagan, A.I., Shaikh, M.M., Abro, R.: Wind turbine micrositing by using the firefly algorithm. Appl. Soft Comput. J. **27**, 450–456 (2015)
61. Tolabi, H.B., Ayob, S.M.: New technique for global solar radiation forecasting by simulated annealing and genetic algorithms using. Appl. Sol. Energy **50**(3), 202–206 (2014)
62. Mafarja, M.M., Mirjalili, S.: Hybrid whale optimization algorithm with simulated annealing for feature selection. Neurocomputing **260**, 302–312 (2016)
63. Moayedikia, A., Ong, K.-L., Boo, Y.L., Yeoh, W.G., Jensen, R. Feature selection for high dimensional imbalanced class data using harmony search. Eng. Appl. Artif. Intell. **57**(May 2016), 38–49 (2017)
64. Wu, J., Qiu, T., Wang, L., Huang, H.: An Approach to feature selection based on ant colony optimization and rough set, pp. 466–471 (2011)
65. Wang, K.J., Adrian, A.M., Chen, K.H., Wang, K.M.: An improved electromagnetism-like mechanism algorithm and its application to the prediction of diabetes mellitus. J. Biomed. Inform. **54**, 220–229 (2015)
66. Alswaitti, M., Albughdadi, M., Isa, N.A.M.: Density-based particle swarm optimization algorithm for data clustering. Expert Syst. Appl. **91**, 170–186 (2018)
67. Abualigah, L.M., Khader, A.T., Hanandeh, E.S., Gandomi, A.H.: A novel hybridization strategy for krill herd algorithm applied to clustering techniques. Appl. Soft Comput. J. **60**, 423–435 (2017)
68. Abualigah, L.M., Khader, A.T., Al-Betar, M.A., Awadallah, M.A.: A krill herd algorithm for efficient text documents clustering. In: 2016 IEEE symposium on computer applications and industrial electronics, pp. 67–72 (2016)
69. Mohammad, L., Abualigah, Q., Hanandeh, E.S.: Applying genetic algorithms to information retrieval using vector space model. Int. J. Comput. Sci. Eng. Appl. **5**(1), 19–28 (2015)
70. Abualigah, L.M., Khader, A.T., Al-Betar, M.A., Alomari, O.A.: Text feature selection with a robust weight scheme and dynamic dimension reduction to text document clustering. Expert Syst. Appl. **84**, 24–36 (2017)
71. Abualigah, L.M., Khader, A.T.: Unsupervised text feature selection technique based on hybrid particle swarm optimization algorithm with genetic operators for the text clustering. J. Supercomput. **73**(11), 4773–4795 (2017)
72. Han, X., Quan, L., Xiong, X., Almeter, M., Xiang, J., Lan, Y.: A novel data clustering algorithm based on modified gravitational search algorithm. Eng. Appl. Artif. Intell. **61**(September 2016) 1–7 (2017)
73. Shukla, U.P., Nanda, S.J.: Parallel social spider clustering algorithm for high dimensional datasets. Eng. Appl. Artif. Intell. **56**, 75–90 (2016)
74. Jadhav, A.N., Gomathi, N.: WGC: hybridization of exponential grey wolf optimizer with whale optimization for data clustering. Alexandria Eng. J. (2016)
75. Sahlol, A.T., Ewees, A.A., Hemdan, A.M., Hassanien, A.E.: Training of feedforward neural networks using sine-cosine algorithm to improve the prediction of liver enzymes on fish farmed on nano-selenite. In: Computer Engineering Conference (ICENCO), 2016 12th International Conference, pp. 35–40 (2009)
76. Rere, L.M.R., Fanany, M.I., Arymurthy, A.M.: Simulated annealing algorithm for deep learning. Procedia Comput. Sci. **72**, 137–144 (2015)
77. Pereira, D.R., et al.: Social-spider optimization-based support vector machines applied for energy theft detection. Comput. Electr. Eng. **49**, 25–38 (2016)
78. Li, P., Duan, H.: Path planning of unmanned aerial vehicle based on improved gravitational search algorithm. Sci. Chin Technol. Sci. **55**(10), 2712–2719 (2012)
79. Burke, E.K., et al.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)

80. Oz, I., Topcuoglu, H.R., Ermis, M.: A meta-heuristic based three-dimensional path planning environment for unmanned aerial vehicles. Simulation **89**(8), 903–920 (2013)
81. Behnck, L.P., Doering, D., Pereira, C.E., Rettberg, A.: A modified simulated annealing algorithm for SUAVs path planning. IFAC-PapersOnLine **28**(10), 63–68 (2015)
82. Xie, C., Zheng, H.: Application of improved Cuckoo search algorithm to path planning unmanned aerial vehicles. In: Intelligent Computing Theories and Application, 12th International Conference, ICIC 2016, pp. 722–729 (2016)
83. Tsai, P., Nguyen, T., Dao, T.: Genetic and evolutionary robot path planning optimization based on multiobjective grey wolf optimizer. In: Genetic and Evolutionary Computing Proceedings of the Tenth International Conference on Genetic and Evolutionary Computing, pp. 166–173 (2016)
84. Contreras-Cruz, M.A., Lopez-Perez, J.J., Ayala-Ramirez, V.: Distributed path planning for multi-robot teams based on Artificial Bee Colony. In: IEEE Congress on Evolutionary Computation (CEC) 2017—Proceeding, pp. 541–548 (2017)
85. Silva, P., Santos, C.P., Matos, V., Costa, L.: Automatic generation of biped locomotion controllers using genetic programming. Rob. Auton. Syst. **62**(10), 1531–1548 (2014)
86. Eskandar, H., Sadollah, A., Bahreininejad, A., Hamdi, M.: Water cycle algorithm—a novel metaheuristic optimization method for solving constrained engineering optimization problems. Comput. Struct. **110–111**, 151–166 (2012)
87. Benkhoud, K., Bouallègue, S.: Dynamics modeling and advanced metaheuristics based LQG controller design for a Quad Tilt Wing UAV. Int. J. Dyn. Control **6**(2), 630–651 (2017)
88. Ibrahim, E., Birchell, S., Elfayoumy, S.: Automatic heart volume measurement from CMR images using ant colony optimization with iterative salient isolated thresholding. J. Cardiovasc. Magn. Reson. **14**(1), 1–2 (2012)
89. Ouaddah, A., Boughaci, D.: Harmony search algorithm for image reconstruction from projections. Appl. Soft Comput. J. **46**, 924–935 (2016)
90. Chen, C.: Image segmentation for lung lesions using ant colony optimization classifier in chest CT. In: Advances in Intelligent Information Hiding and Multimedia Signal Processing, pp. 283–289 (2017)
91. Oliva, D., Hinojosa, S., Cuevas, E., Pajares, G., Avalos, O., Gálvez, J.: Cross entropy based thresholding for magnetic resonance brain images using crow search algorithm. Expert Syst. Appl. **79**, 164–180 (2017)
92. Kora, P., Kalva, S.R.: Improved Bat algorithm for the detection of myocardial infarction. Springerplus **4**(1), 666 (2015)
93. Nagpal, S., Arora, S., Dey, S., Shreya.: Feature selection using gravitational search algorithm for biomedical data. Procedia Comput. Sci. **115**, 258–265 (2017)
94. Sahoo, A., Chandra, S.: Multi-objective grey wolf optimizer for improved cervix lesion classification. Appl. Soft Comput. J. **52**, 64–80 (2017)
95. Alshamlan, H., Badr, G., Alohali, Y.: MRMR-ABC: a hybrid gene selection algorithm for cancer classification using microarray gene expression profiling. Biomed Res. Int. **2015** (2015)
96. Alomari, O.A., Khader, A.T., Al Betar, M.A., Abualigah, L.M.: Gene selection for cancer classification by combining minimum redundancy maximum relevancy and bat-inspired algorithm. Int. J. Data Min. Bioinform. **19**(1), 32 (2017)
97. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
98. Vocking, B., et al.: Algorithms Unplugged. Springer, Berlin Heidelberg (2011)
99. Pardalos, P.M., Du, D.-Z., Graham, R. L.: Handbook of Combinatorial Optimization. Springer US (2013)
100. Laguna, M., Martí, R.: Scatter Search, Methodology and Implementations in C. Springer US (2003)
101. Galinier, P., Hamiez, J.P., Hao, J.K., Porumbel, D.: Handbook of Optimization, vol. 38 (2013)
102. Sapra, D., Sharma, R., Agarwal, A.P.: Comparative study of metaheuristic algorithms using Knapsack Problem. In: Proceedings of 7th International Conference on Cloud Computing, Data Science and Engineering-Confluence, pp. 134–137 (2017)

103. Feng, Y., Wang, G.G., Deb, S., Lu, M., Zhao, X.J.: Solving 0–1 knapsack problem by a novel binary monarch butterfly optimization. Neural Comput. Appl. **28**(7), 1619–1634 (2017)
104. Gutin, G., Punnen, A.P.: The Traveling Salesman Problem and Its Variations. Springer US (2007)
105. Saji, Y., Riffi, M.E.: A novel discrete bat algorithm for solving the travelling salesman problem. Neural Comput. Appl. **27**(7), 1853–1866 (2016)
106. Zhou, Y., Wang, R., Zhao, C., Luo, Q., Metwally, M.A.: Discrete greedy flower pollination algorithm for spherical traveling salesman problem. Neural Comput. Appl. 1–16 (2017)
107. Pereira, F.B., Tavares, J.: Bio-inspired Algorithms for the Vehicle Routing Problem. Springer US (2009)
108. Yurtkuran, A., Emel, E.: A new hybrid electromagnetism-like algorithm for capacitated vehicle routing problems. Expert Syst. Appl. **37**(4), 3427–3433 (2010)
109. Wei, L., Zhang, Z., Zhang, D., Leung, S.C.H.: A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. Eur. J. Oper. Res. 1–17 (2017)
110. Marinaki, M., Marinakis, Y.: A glowworm swarm optimization algorithm for the vehicle routing problem with stochastic demands. Expert Syst. Appl. **46**(4), 145–163 (2016)
111. Potvin, J.Y.: A review of bio-inspired algorithms for vehicle routing. Stud. Comput. Intell. **161**(July), 1–34 (2009)
112. Xu, H., Pu, P., Duan, F.: Dynamic vehicle routing problems with enhanced ant colony optimization. Discret. Dyn. Nat. Soc. **2018**, 1–13 (2018)
113. Zhang, S.Z., Lee, C.K.M.: An improved artificial bee colony algorithm for the capacitated vehicle routing problem. In: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics—SMC 2015, pp. 2124–2128 (2016)
114. Xiang, T.: Vehicle routing problem based on particle swarm optimization algorithm with gauss mutation. Am. J. Softw. Eng. Appl. **5**(1), 1 (2016)
115. Jourdan, L., Basseur, M., Talbi, E.G.: Hybridizing exact methods and metaheuristics: a taxonomy. Eur. J. Oper. Res. **199**(3), 620–629 (2009)
116. Puchinger, J.: Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification, pp. 1–12 (2006)
117. Plateau, A., Tachat, D., Tolla, P.: A hybrid search combining interior point methods and metaheuristics for 0–1 programming. Int. Trans. Oper. Res. **9**(6), 731–746 (2002)
118. Yan, L., Yujuan, Q., Zujian, W., Wang, L., Yan, J.: A hybrid method combining genetic algorithm and Hooke-Jeeves method for 4PLRP. In: International Conference on Communications in China-Workshops (CIC/ICCC) 2014, vol. 10, no. 4, pp. 36–40 (2015)
119. Portmann, M.C., Vignier, A., Dardilhac, D., Dezalay, D.: Branch and bound crossed with GA to solve hybrid flowshops. Eur. J. Oper. Res. **107**(2), 389–400 (1998)
120. Basseur, M., Lemesre, J., Dhaenens, C., Talbi, E.-G.: Cooperation between branch and bound and evolutionary approaches to solve a bi-objective flow shop problem, vol. 2632 (2004)
121. Gomes, A.M., Oliveira, J.F.: Solving Irregular Strip Packing problems by hybridising simulated annealing and linear programming. Eur. J. Oper. Res. **171**(3), 811–829 (2006)
122. Zelinka, I.: A survey on evolutionary algorithms dynamics and its complexity—mutual relations, past, present and future. Swarm Evol. Comput. **25**, 2–14 (2015)
123. Valdivia-Gonzalez, A., Zaldívar, D., Fausto, F., Camarena, O., Cuevas, E., Perez-Cisneros, M.: A States of matter search-based approach for solving the problem of intelligent power allocation in plug-in hybrid electric vehicles. Energies **10**(1), 92 (2017)
124. Cuevas, E., González, A., Fausto, F., Zaldívar, D., Cisneros, M.P.: An optimisation algorithm based on the behaviour of locust swarms. Int. J. Bio-Inspired Comput. **7**(6), 402 (2015)
125. González, A., Cuevas, E., Fausto, F., Valdivia, A., Rojas, R.: A template matching approach based on the behavior of swarms of locust. Appl. Intell. **47**(4) (2017)
126. Cuevas, E., González, A., Fausto, F., Zaldívar, D., Pérez-Cisneros, M.: Multithreshold segmentation by using an algorithm based on the behavior of locust swarms. Math. Probl. Eng. **2015**, 26 (2015)

127. Cuevas, E., Gálvez, J., Avalos, O.: Parameter estimation for chaotic fractional systems by using the locust search algorithm. Comput. Sist. **21**(2), 369–380 (2017)
128. Hinojosa, S., Oliva, D., Cuevas, E., Pajares, G., Avalos, O., Gálvez, J.: Improving multi-criterion optimization with chaos: a novel multi-objective chaotic crow search algorithm. Neural Comput. Appl. **29**(8), 319–335 (2018)
129. Regis, R.G.: Particle swarm with radial basis function surrogates for expensive black-box optimization. J. Comput. Sci. **5**(1), 1–12 (2013)
130. Wild, S.M., Regis, R.G., Shoemaker, C.A.: ORBIT: optimization by radial basis function interpolation in trust-regions. SIAM J. Sci. Comput. **30**(6), 3197–3219 (2008)
131. Liu, B., Koziel, S., Zhang, Q.: A multi-fidelity surrogate-model-assisted evolutionary algorithm for computationally expensive optimization problems. J. Comput. Sci. **12**, 28–37 (2016)
132. Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Inf. Sci. (Ny) **237**, 82–117 (2013)
133. Cheng, S., Shi, Y., Qin, Q., Ting, T.O., Bai, R.: Maintaining population diversity in brain storm optimization algorithm. In: Proceedings—2014 IEEE Congress Evolutionary Computation (CEC), pp. 3230–3237 (2014)
134. Yang, X.S.: Metaheuristic optimization: algorithm analysis and open problems. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6630, pp. 21–32. LNCS (2011)
135. Yang, X.S.: Nature-inspired algorithms: success and challenges. Comput. Methods Appl. Sci. **38**, 129–143 (2015)

Chapter 3
The Selfish Herd Optimizer

Abstract In this chapter, a swarm optimization algorithm called Selfish Herd Optimizer (SHO) is presented. The SHO algorithm's design is based on the emulation of the widely-observed selfish herd behavior, manifested by individuals living in aggregations while exposed to some kind of predation risk. An interesting trait that distinguish the SHO algorithm from other similar approaches is the division of the entire population of search agents in two opposite groups: the members of a selfish herd (the prey), and a pack of starving predators. These two types of search agents interact with each other in ways that allows to emulate the intriguing interaction between prey and predators that arise from the unique behaviors manifested by the members of the so-called selfish herds. This chapter also presents a series of experiments done with the purpose of comparing the performance of the SHO algorithm against other similar swarm optimization approaches, showing remarkable results.

3.1 Introduction

The intelligent collective behavior of many species of animals and insects, have attracted the attention of researches for many years. Many animal species such as birds, ants, and fishes, which live in social animal groups such as flocks, colonies and schools respectively, exhibit particular aggregative conducts widely known as swarm behavior. Such collective phenomenon has been studied by entomologist in order to model the behavior of the many biological swarms. Computer science researchers have studied and adapted these models as frameworks for solving complex real-world problems, giving birth to a branch of artificial intelligence commonly addressed as swarm intelligence. As a result of this, many unique swarm optimization algorithms, which mimic the collective behavior of groups of animals or insects, have been developed to solve a wide variety of optimization problems. Some of these methods include well know state-of-the-art techniques such as Particle Swarm Optimization (PSO), which emulates the social behavior of bird flocking and fish schooling [1], Artificial Bee Colony (ABC), which is based on the cooperative behavior of bee colonies [2], Firefly Algorithm (FA) which mimics the mating behavior of firefly insects [3], and Cuckoo Search (CS), which draws inspiration from the cuckoo bird

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_3

lifestyle [4]. Although most of these methods are widely used to solving complex optimization problems, they are known to suffer from some serious flaws, such as premature convergence and the difficulty to overcome local optima [5, 6], which prevent them from finding optimal solutions. The cause of such issues is usually related to the operators used to modify each individual's position. In the case of PSO, for example, the position of each search agent for the next iteration is updated yielding an attraction towards the best particle position seen so-far, while in the case of ABC, positions are updated with respect of some other randomly chosen individuals. As the algorithm evolves, those behaviors allow the entire population to, either rapidly concentrates around the current best particle or to diverge without control, which in return favors the premature convergence or a misbalance between exploration and exploitation respectively [7, 8]. In addition, most state of the art swarm algorithms only model individual entities that perform virtually the same behavior. Under such circumstances, the possibility of adding new and selective operators based on individual unique characteristics (such as task-responsibility, strength, size, sex, etc.) that could improve several important algorithm characteristics such as population diversity and searching capabilities.

While it is true that a wide range of organisms living in aggregations show distinctive cooperative behaviors, this is not true for every single animal species living in social units. In contrast to the popular hypothesis that social behavior is based on mutual benefits for the entire population, the widely accepted selfish herd theory proposed by Hamilton in 1971 illustrates that actions among individuals within aggregations (referred as herds) exhibit an unusual degree of selfishness, particularly when members of such aggregations are endangered by the presence of predators [9]. In fact, the selfish herd theory establishes that decisions made by any member of such herds do not only benefit the individual itself but also, in exchange, there are usually some negative repercussions for other members on said aggregation.

In this chapter, a novel swarm optimization algorithm called Selfish Herd Optimizer (SHO), designed for solving optimization problems, is presented. Such algorithm is inspired in the behaviors described on Hamilton's selfish herd theory. The algorithm considers two different kinds of search agents: predators and prey. Each of these agents movements are conducted by a set of unique rules and operators based the observed natural behavior of individuals on a selfish herd while they are endangered by a pack of hungry predators. The rest of this chapter is organized as follows: in Sect. 3.2, we address the selfish herd theory, as proposed by [9]; in Sect. 3.3, the main characteristics of the SHO algorithm are addressed; in Sect. 3.4, we summarize all steps regarding to the implementation of the SHO algorithm; in Sect. 3.5, a discussion about SHO and its most distinctive traits in comparison to other similar methods is given; in Sect. 3.6, a series of comparative experiments competing to SHO and other similar approaches are presented; finally, in Sect. 3.7, conclusions are drawn.

3.2 The Selfish Herd Theory

The selfish herd theory, as proposed by Hamilton in [9], is an antithesis to the common view of gregarious behavior as a way of seeking mutual benefits among members of a population or group of organism. In his paper, Hamilton proposed that gregarious behavior may be considered a form of cover-seeking, in which each individual attempts to reduce their chance of being caught by a predator. It is also stated that, during a predator attack, individuals within a population will attempt to reduce their predation risk by putting other conspecifics between themselves and the predator(s). The basic principle governing the selfish herd theory is that, in aggregations, predation risk increases among individuals in the periphery and decreases toward the center of such aggregation. It is also proposed that more dominant (stronger) animals within the population are easily able to obtain low-risk central positions among the aggregation, whereas subordinate (weaker) animals are usually forced into higher risk positions.

Hamilton illustrated his theory by modeling a circular lily pond in which a population of frogs (a group of prey) and a water snake (a predator) are sheltered. Upon appearance of the water snake, it is supposed that the frogs will scatter to the rim of the pond and that the water snake will most certainly attack the one nearest to it. In this model Hamilton suggests that the predation risk of each frog is related not only to how close they are from the attacking predator, but also with the relative position of all other frogs on the pond. Under these considerations, Hamilton proposes that each frog has a better chance of not being closest to, and thus, vulnerable to attack by the water snake, if other frogs are between them. As a result, modeled frogs attempt to reduce their predation risk by jumping to smaller gaps between other neighboring frogs in an attempt to use them as a "shield". Hamilton also went on to model two-dimensional predation by considering lions as examples. He proposed that movements which would lower an individual's domain of danger are largely based on the theory of marginal predation, which states that predators attack the closest prey (which are typically those at the periphery of an aggregation). From this, Hamilton suggested that, in the face of predation, there should be a strong movement of individuals toward the center of an aggregation. Research has also revealed that there exist several factors which may influence chosen movement rules, such as initial spatial position, population density, the predator's attack strategy, and vigilance. In particular, it has been observed that individuals holding initially central positions are more likely to be successful at remaining in the center of the aggregation, increasing their chances of surviving a predator attack [10].

The selfish herd theory may also be applied to the situation of group escape, in which, the safest position, relative to predation risk, is not the central position but rather that in the front of the herd. In this sense, members at the back of the aggregation have the greatest domain of danger, and thus, suffer the highest predation risk. As the most likely targets for predation, these slower members must choose whether to stay with the herd, or to desert it, which may in turn entice the pursuit of the predator to such vulnerable individuals. This strategy, formally known as herd desertion, is mainly used by slower individuals among the aggregation in an attempt to escape from the sight of predators, although this in turn may signal their vulnerability and thus promote the predators to pursuit such individual [11].

By considering this, it could be assumed that the escape route chosen by members in front of the herd may be greatly affected by the actions of the slowest members. For example, if the herd's leader chooses an escape route that promotes the dispersal of the slowest members of the group it may endanger itself due to the dissipations of its protective buffer. Also, it is known that the leader's chosen escape strategy is often affected by terrain particularities [11].

Many examples of selfish herd behavior have been witnessed in nature. One of the most extensively studied examples is that of aggregations of fiddler crab, in which, dispersed groups are more likely to form an aggregate when subjected to a danger while at the same time individual members attempt to move toward the center of the forming group [12]. Other selfish herd behavior examples include that of mammals living in open plains, such as wildebeest and zebras (which aggregations are likely associated with predation risk reduction), many species of fishes (such as minnows, which school to reduce their individual predation risk) [13], the Adelie Penguins (which frequently wait to jump into the water until they have formed an aggregate to form protective buffers against seal predation) [14], and the Forest Tent Caterpillar (famous for foraging in groups as a strategy to reduce predation risk) [15].

3.3 The Selfish Herd Optimizer Algorithm

The selfish herd theory establishes that, in the face of predation, each individual within a herd of possible prey pursues to increases their chance of survival by aggregating with other conspecifics in ways which could potentially increase their chances of surviving a predator attack without regard of how such behavior affects other individuals' chances of survival [9]. Based on these observations, Fausto et al. developed the swarm optimization algorithm known as the Selfish Herd Optimizer (SHO) [16]. This optimization approach assumes that the entire search space is an open plain where groups of animals interact. This algorithm models two different types of search agents: a herd of prey living in aggregation (also referred as a selfish herd) and a pack of predators which hunts for the prey within said aggregation. Both kinds of search agents are individually conducted by a set of different evolutionary operators based on the unique behavioral aspects observed in such prey-predator relationship.

3.3.1 *Initializing the Population*

Similar to other evolutionary algorithms, SHO is an iterative process which first step is to randomly initialize a population of animals (prey and predators). The algorithm starts by initializing a set $\mathbf{A}$ of N individual positions $\mathbf{a}_i$ $(\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_N\})$, with N representing the total population size. Each of such positions is represented as an n-dimensional vector $\mathbf{a}_i = \left[a_{i,1}, a_{i,2}, \ldots, a_{i,n}\right]$, which further represents a possible solution for a given optimization problem. These positions are initialized by considering a random uniform distribution between a pair of pre-specified parameter bounds as given as follows:

$$\begin{gathered} a_{i,j}^{0} = x_j^{\text{low}} + \text{rand}(0,1) \cdot \left(x_j^{\text{high}} - x_j^{\text{low}}\right) \\ i = 1, 2, \ldots, N; \quad j = 1, 2, \ldots, n \end{gathered} \tag{3.1}$$

where x_j^{low} and x_j^{high} represent the decision space's lower and upper bounds, respectively. Furthermore, i and j denote the individual and parameter indexes corresponding to each animal, respectively. Also, rand(0, 1) stand for randomly generated number, drawn from within the interval [0, 1].

The selfish herd theory establishes a series of distinctive behaviors, resulting from the interaction between a group of prey and predators. With this in mind, SHO employs two different types of search agents: a group of prey (known as a herd while in an aggregation) and a group of predators (collectively known as a pack). With that being said, the entire population of animals $\mathbf{A}$ is divided in two sub-groups: a group $\mathbf{H} = \left\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{N_h}\right\}$ formed by all individuals which belong to the herd of prey (with N_h denoting the number of individuals of $\mathbf{H}$) and a group $\mathbf{P} = \left\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{N_p}\right\}$ represented by the members of the pack of predators (with N_p denoting the number of individuals of $\mathbf{P}$), and such that $\mathbf{A} = \mathbf{H} \cup \mathbf{P}$ ($\mathbf{A} = \{\mathbf{a}_1 = \mathbf{h}_1, \mathbf{a}_2 = \mathbf{h}_2, \ldots, \mathbf{a}_{N_h} = \mathbf{h}_{N_h}, \mathbf{a}_{N_h+1} = \mathbf{p}_1, \mathbf{a}_{N_h+2} = \mathbf{p}_2, \ldots, \mathbf{a}_N = \mathbf{p}_{N_p}\}$). In nature, the number of animals within a herd of prey usually outnumbers those within most packs of predators. In SHO, the number of prey (herd's size) N_h is randomly selected within a range of between 70 and 90% of the total population N, while the remainder individuals are labeled as predators. As such, N_h and N_p are calculated by the following equations:

$$N_h = \text{floor}(N \cdot \text{rand}(0.7, 0.9)) \tag{3.2}$$

$$N_p = N - N_h \tag{3.3}$$

where rand(0.7, 0.9) denotes a random number from within the interval [0.7, 0.9], while floor($\cdot$) maps a real number to an integer number. Furthermore, the number of predators (pack's size) N_p is simply computed as the complement between the total population N and the herd's size N_h.

3.3.2 Survival Value Assignation

In the biological metaphor of a common prey-predator interaction, each individual within a herd of prey or a pack of predator, depending on their survival capabilities, has a chance of surviving an attack or succeed on killing an animal, respectively. In the proposed approach, each animal $\mathbf{a}_i$ (irrespective of it being a prey or a predator) is assigned with a survival value SV_{a_i} which represents its survival aptitude (solution quality) relative to its current position within the solution space. In SHO, it is assumes that each animal's survival value $\mathrm{SV}_{\mathbf{a}_i}$ is related to both the safest and the riskiest positions currently known by all members of the population, which in the context of a global optimization problem are represented by the current best and worst solutions found so far by the optimization process. As such, the survival value assigned to of each individual animal is calculated as follows:

$$\mathrm{SV}_{\mathbf{a}_i} = \frac{f(\mathbf{a}_i) - f_{\text{best}}}{f_{\text{best}} - f_{\text{worst}}} \tag{3.4}$$

where $f(\mathbf{a}_i)$ denotes the fitness value corresponding to the evaluation of the objective function $f(\cdot)$ with regard to the individual's position $\mathbf{a}_i$, while f_{best} and f_{worst} stand for the best and worst fitness values found so far by the SHO algorithm's evolutionary process which, considering a maximization problem, are defined as follows:

$$f_{\text{best}} = \max_{j \in \{0,1,\ldots,k\}} \left(\left(\max_{i \in \{1,2,\ldots,N\}} (f(\mathbf{a}_i)) \right)_j \right) \tag{3.5}$$

$$f_{\text{worst}} = \min_{j \in \{0,1,\ldots,k\}} \left(\left(\min_{i \in \{1,2,\ldots,N\}} (f(\mathbf{a}_i)) \right)_j \right) \tag{3.6}$$

with k denoting the current iteration of SHO's evolutionary process.

3.3.3 Structure of a Selfish Herd

In most selfish herds, a dominant member, known as the herd's leader, may be identified. Such individual distinguishes itself from the rest of the herd as the individual with the greatest survival aptitudes. During a predator attack, the herd's leader performs an important task in choosing the escape route or strategy to be employed by all other members of the herd, and as such, its leadership behavior heavily influences the movement patterns of the entire aggregation [11]. On the other hand, it is known that animals among an aggregation of prey will try to reduce their predation risk by putting other conspecifics between them and the attacking predators [9]. Intuitively, since a given individual's chance of surviving a predator attack is related to its relative position among the herd aggregation, such selfish behavior is also necessarily related to the herd's current internal structure and movement patterns.

By considering this, SHO models several distinctive decision making behaviors by first dividing the herd's population with regard to three distinctive roles: 1. a herd leader tasked to guide the movement of the prey aggregation; 2. a group of herd followers which guide their moves by considering the positions and survival aptitudes of other herd members; and 3. a group of herd deserters which move independently of other prey individuals.

3.3.3.1 Leader of a Selfish Herd

The leader of a selfish herd is the strongest, wisest and most capable individual for survival among the members of the herd, and as such has an important role on guiding the movement of all of its conspecifics. The herd's leader is usually the individual whose position inside the herd aggregation promotes it to have the highest chances of surviving a predator attack [11]. Analogous to this, at each iteration k, the SHO algorithm designates a single individual $\mathbf{h}_i$ among the herd's population ($\mathbf{H}$) as the leader of the selfish herd. Such individual (designated as $\mathbf{h}_L$) is chosen by considering the current survival values possessed by each individual within the herd's aggregation as follows:

$$\mathbf{h}_L^k = \left(\mathbf{h}_i^k \in \mathbf{H}^k | \mathrm{SV}_{\mathbf{h}_i^k} = \max_{j \in \{1,2,\ldots,N_h\}} \left(\mathrm{SV}_{\mathbf{h}_j^k} \right) \right) \tag{3.7}$$

In other words, the prey individual $\mathbf{h}_i^k$ (with $i \in [1, 2, \ldots, N_h]$) possessing highest survival value among all other members of $\mathbf{H}^k$ (with k denoting the current iteration) is assigned as the herd's leader (see Fig. 3.1).

3.3.3.2 Nearest Best Neighbors Within a Selfish Herd

As previously stated, individuals within a selfish herd aims to increase their chances of surviving a predator attack by putting other members of the herd between them

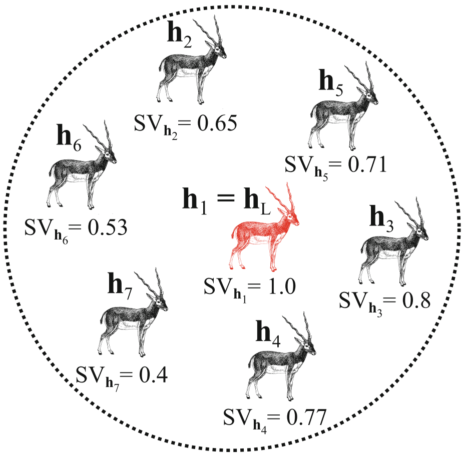
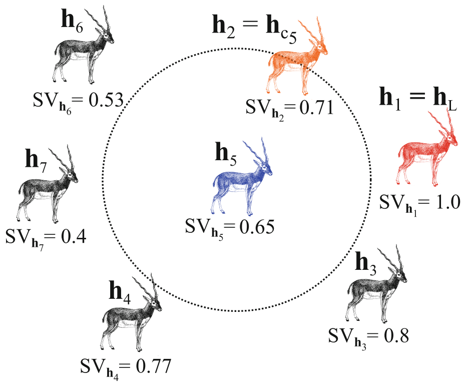

$\mathbf{h}_2$
$\mathrm{SV}_{\mathbf{h}_2} = 0.65$
$\mathbf{h}_5$
$\mathrm{SV}_{\mathbf{h}_5} = 0.71$
$\mathbf{h}_6$
$\mathrm{SV}_{\mathbf{h}_6} = 0.53$
$\mathbf{h}_1 = \mathbf{h}_L$
$\mathrm{SV}_{\mathbf{h}_1} = 1.0$
$\mathbf{h}_3$
$\mathrm{SV}_{\mathbf{h}_3} = 0.8$
$\mathbf{h}_7$
$\mathrm{SV}_{\mathbf{h}_7} = 0.4$
$\mathbf{h}_4$
$\mathrm{SV}_{\mathbf{h}_4} = 0.77$

Fig. 3.1 The individual $\mathbf{h}_i$ holding the highest survival value from among all other members within the herd's population is designated as the herd's leader ($\mathbf{h}_L$)



$\mathbf{h}_6$
$SV_{\mathbf{h}_6} = 0.53$
$\mathbf{h}_2 = \mathbf{h}_{c5}$
$SV_{\mathbf{h}_2} = 0.71$
$\mathbf{h}_1 = \mathbf{h}_L$
$SV_{\mathbf{h}_1} = 1.0$
$\mathbf{h}_7$
$SV_{\mathbf{h}_7} = 0.4$
$\mathbf{h}_5$
$SV_{\mathbf{h}_5} = 0.65$
$\mathbf{h}_3$
$SV_{\mathbf{h}_3} = 0.8$
$\mathbf{h}_4$
$SV_{\mathbf{h}_4} = 0.77$

Fig. 3.2 The nearest best neighbor $\mathbf{h}_{c_i}$ to any given prey individual $\mathbf{h}_i$ (in this case illustrated as $\mathbf{h}_5$) is the herd member $\mathbf{h}_j$ (different to the herd's leader $\mathbf{h}_L$) which is both closer to $\mathbf{h}_i$ and that possess a higher survival value than $\mathbf{h}_i$ (represented as $\mathbf{h}_2$ in this case)

and the attacking predators [9]. In this sense, individuals within a herd of prey move toward closer conspecifics among the aggregation, which they could potentially use to protect themselves from the attacking predators. Intuitively, in order to motivate such movement pattern, such closer individuals must first possess a relatively safer position respective to the predator's location. In this context, at each iteration k, SHO considers that the nearest best neighbor to any prey individual within the selfish herd is that which possess two important traits: 1. it is the nearest herd member to $\mathbf{h}_i$ (other than the herd's leader $\mathbf{h}_L$, as illustrated in Sect. 3.3.3.1); and 2. it has better survival aptitudes than said individual $\mathbf{h}_i$ (see Fig. 3.2). By considering this, the nearest best neighbor of $\mathbf{h}_i$ may be defined as follows:

$$\mathbf{h}_{c_i}^k = \left(\mathbf{h}_j^k \in \mathbf{H}^k, \mathbf{h}_j^k \neq \left[\mathbf{h}_i^k, \mathbf{h}_L^k\right] | SV_{\mathbf{h}_j^k} > SV_{\mathbf{h}_i^k}, r_{i,j} = \min_{j \in \{1,2,\ldots,N_h\}} \left(\left\|\mathbf{h}_i^k - \mathbf{h}_j^k\right\|\right)\right) \tag{3.8}$$

where $r_{i,j}$ denotes the Euclidean distance between the indexed herd members i and j ($\mathbf{h}_i^k$ and $\mathbf{h}_k^k$ respectively), and with k denoting the current iteration number.

3.3.3.3 Herd Followers and Herd Deserters

Perhaps the most interesting behaviors observed in selfish herds is the decision taken by its members for either following the group's movement or to desert the aggregation and move independently of it [11]. The decision making criteria behind these behaviors is strongly related to the degree of safety experimented by each individual among the herd during a predators attack, which in turn is also related to each individual's relative position among the herd. By considering this, the SHO algorithm models a set of unique individual decision making operators which consider the individual survival capabilities of each member of the herd. Such behaviors are modeled by further dividing the herd's population $\mathbf{H}$ in two subgroups: 1. a group of herd followers ($\mathbf{H}_F$), formed by those members who opt to follow the aggregation and 2. a group of herd deserters ($\mathbf{H}_D$), which comprises all prey individuals which decided to move independently of other members of the herd. In SHO, these two groups are defined for each iteration k as follows:

$$\mathbf{H}_F^k = \left\{\mathbf{h}_i^k \neq \mathbf{h}_L^k | \mathrm{SV}_{\mathbf{h}_i^k} \geq \mathrm{rand}(0,1)\right\} \tag{3.9}$$

$$\mathbf{H}_D^k = \left\{\mathbf{h}_i^k \neq \mathbf{h}_L^k | \mathrm{SV}_{\mathbf{h}_i^k} < \mathrm{rand}(0,1)\right\} \tag{3.10}$$

where $\mathbf{H}_F^k$ denotes the groups of herd followers whereas $\mathbf{H}_D^k$ stands for the group of herd deserters. Furthermore, rand(0, 1) denotes a random number from the interval [0, 1].

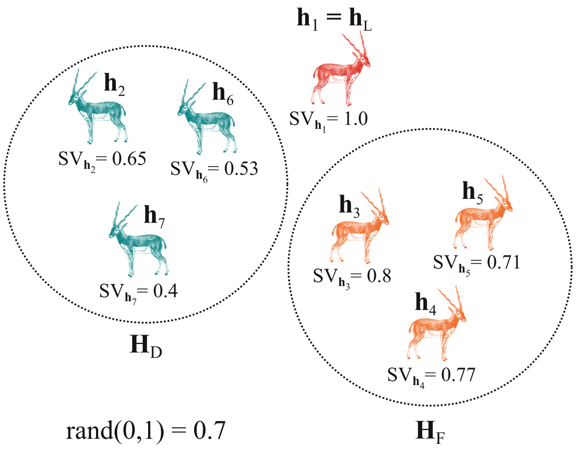
In other words, each individual $\mathbf{h}_i$ on a selfish herd (other than the herd's leader $\mathbf{h}_L$ as illustrated in Sect. 3.3.3.1) is grouped in either $\mathbf{H}_F$ or $\mathbf{H}_D$ depending on its current survival value $\mathrm{SV}_{\mathbf{h}_i}$. In this sense, it is clear that for any given prey individual $\mathbf{h}_i \neq \mathbf{h}_L$, having a higher survival value $\mathrm{SV}_{\mathbf{h}_i}$ yields to higher chances of following the herd, whereas lower values increases the chance of deserting such aggregation (see Fig. 3.3).

Furthermore, prey individuals within $\mathbf{H}_F$ may be further divided into a set of dominant herd members ($\mathbf{H}_d$) and a set of subordinate herd members ($\mathbf{H}_s$) depending on their current survival capabilities. With this in mind, SHO divides the members of $\mathbf{H}_F$ as either dominant or subordinate members as follows:

$$\mathbf{H}_d^k = \left\{\mathbf{h}_i^k \in \mathbf{H}_F^k | \mathrm{SV}_{\mathbf{h}_i^k} \geq \mathrm{SV}_{\mathbf{H}_\mu^k}\right\} \tag{3.11}$$

$$\mathbf{H}_s^k = \left\{\mathbf{h}_i^k \in \mathbf{H}_F^k | \mathrm{SV}_{\mathbf{h}_i^k} < \mathrm{SV}_{\mathbf{H}_\mu^k}\right\} \tag{3.12}$$

where $\mathrm{SV}_{\mathbf{H}_\mu}$ represents the mean survival value of the herd's aggregation $\mathbf{H}$ as defined as follows:



$\mathbf{h}_1 = \mathbf{h}_L$
$\mathrm{SV}_{\mathbf{h}_1} = 1.0$
$\mathbf{h}_2$
$\mathrm{SV}_{\mathbf{h}_2} = 0.65$
$\mathbf{h}_6$
$\mathrm{SV}_{\mathbf{h}_6} = 0.53$
$\mathbf{h}_7$
$\mathrm{SV}_{\mathbf{h}_7} = 0.4$
$\mathbf{H}_D$
$\mathbf{h}_3$
$\mathrm{SV}_{\mathbf{h}_3} = 0.8$
$\mathbf{h}_5$
$\mathrm{SV}_{\mathbf{h}_5} = 0.71$
$\mathbf{h}_4$
$\mathrm{SV}_{\mathbf{h}_4} = 0.77$
rand(0,1) = 0.7
$\mathbf{H}_F$

Fig. 3.3 Designation of herd followers ($\mathbf{H}_F$) and herd deserting members ($\mathbf{H}_D$) for a given value rand(0, 1)

$$\mathrm{SV}_{\mathbf{H}_{\mu}^{k}} = \frac{\sum_{i=1}^{N_h} \mathrm{SV}_{\mathbf{h}_i^k}}{N_h} \tag{3.13}$$

In other words, individuals $\mathbf{h}_i$ belonging to $\mathbf{H}_F$ are classified within $\mathbf{H}_d$ (dominant members) if their survival value $\mathrm{SV}_{\mathbf{h}_i}$ is equal to or greater than the mean survival value $\mathrm{SV}_{\mathbf{H}_\mu}$ of the entire herd aggregation, otherwise, they grouped within $\mathbf{H}_s$ (subordinated members). As stated in [11], in most cases, dominant members (individuals with higher survival aptitudes) are more frequently able to secure safer position within the herd aggregation while subordinate members (individuals with lower survival aptitudes) are usually forced to assume higher risk positions. This fact is specially considered in SHO to model several different movement rules, which depend on the survival value of each member currently following the herd aggregation.

3.3.3.4 Relative Safer and Riskier Positions

In most cases, the predation risk of individuals within a selfish herd increases among individuals in the herd's periphery (where there are fewer animals to use as protection) and decreases toward the center of such aggregation. With this in mind, the SHO algorithm considers the existence of a relatively safer central position within the herd of prey. In this approach such location is given by the herd's population center of mass, as defined as follows:

$$\mathbf{h}_M^k = \frac{\sum_{i=1}^{N_h} \mathrm{SV}_{\mathbf{h}_i^k} \cdot \mathbf{h}_i^k}{\sum_{j=1}^{N_h} \mathrm{SV}_{\mathbf{h}_j^k}} \tag{3.14}$$

While the previous is useful to illustrate the location of a potentially safe location within a selfish herd, such assumption does not consider the presence of predators as the main source of danger to any individual within such aggregation. In a similar manner to Eq. (3.14), it is possible to define a position of relatively higher risk by considering both, the current locations and survival aptitudes of the attacking predators as follows:

$$\mathbf{p}_M^k = \frac{\sum_{i=1}^{N_p} \mathrm{SV}_{\mathbf{p}_i^k} \cdot \mathbf{p}_i^k}{\sum_{j=1}^{N_p} \mathrm{SV}_{\mathbf{p}_j^k}} \tag{3.15}$$

Furthermore, since both, $\mathbf{h}_M$ and $\mathbf{p}_M$ represent potential solutions within the solution space of a given optimization problem, a corresponding survival value $\mathrm{SV}_{\mathbf{h}_{cm}}$ and $\mathrm{SV}_{\mathbf{p}_{cm}}$ may be assigned to each of such positions by applying Eq. (3.4) (see Fig. 3.4).

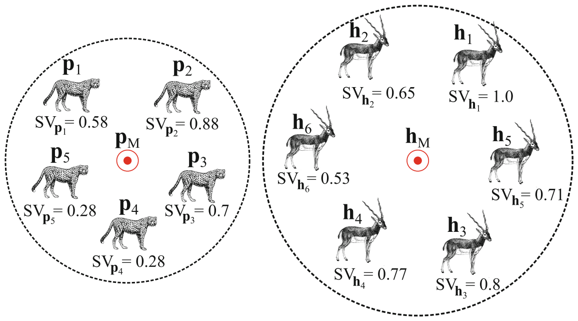


Fig. 3.4 Relatively safer and riskier positions represented by the herd's center of mass $\mathbf{h}_M$ and the predator's center of mass $\mathbf{p}_M$ respectively

3.3.4 Herd Movement Operators

In order to model the movement of each individual within a selfish herd, SHO considers two different sets of evolutionary operators: 1. a set of herd's leader movement operators, and 2. a set of herd's following and desertion movement operators. Such movement operators consider important characteristics shared by all members of a selfish herd, such as individual survival values and the distance to other members of the aggregation, in order to accurately model the movement patterns manifested by such prey aggregations.

3.3.4.1 Selfish Attraction and Repulsion

When endangered by the presence of one of more predators, individuals within a selfish herd pursue to improve their chances of surviving a predator attack by moving in a way which could allow them to put other conspecifics between them and the attacking predators. This behavior could be effectively modeled as an attraction toward other individuals among the herd aggregation. Analogous to this, SHO assumes that each individual within the herd's population **H** (as illustrated in Sect. 3.3.3.1) is able to manifest a certain degree of attraction toward other members within such aggregation. This attraction depends on both, the relative distance toward a given individual and the current survival value possessed by said individual. By considering this, we may define an attraction factor experienced by a given herd member $\mathbf{h}_i$ toward any different member $\mathbf{h}_j$ as follows:

$$\psi_{\mathbf{h}_i,\mathbf{h}_j} = \mathrm{SV}_{\mathbf{h}_j} \cdot e^{-\|\mathbf{h}_i-\mathbf{h}_j\|^2} \tag{3.16}$$

where $\mathrm{SV}_{\mathbf{h}_j}$ denotes for the survival value related to the herd member $\mathbf{h}_j$, whereas $\|\mathbf{h}_i - \mathbf{h}_j\|$ stands for the Euclidian distance between the prey individuals $\mathbf{h}_i$ and $\mathbf{h}_j$.

The factor $\psi_{\mathbf{h}_i,\mathbf{h}_j}$ in Eq. (3.16) is known as selfish attraction. While it is possible to compute such value for virtually any pair of prey individuals, four specific relationships are of particular importance within the proposed approach:

1. The selfish attraction $\varphi_{\mathbf{h}_i,\mathbf{h}_L}$, which represents the attraction experimented by $\mathbf{h}_i$ toward the current herd's leader $\mathbf{h}_L$ (as illustrated in Sect. 3.3.3.1). This value represents the influence exerted by the herd's leader which is responsible of guiding the movements of all other members following the herd aggregation. Such attraction value is defined as follows:

$$\psi_{\mathbf{h}_i,\mathbf{h}_L} = \mathrm{SV}_{\mathbf{h}_L} \cdot e^{-\|\mathbf{h}_i - \mathbf{h}_L\|^2} \tag{3.17}$$

2. The selfish attraction $\varphi_{\mathbf{h}_i,\mathbf{h}_{c_i}}$ representing the attraction factor experimented by $\mathbf{h}_i$ toward its nearest best neighbor $\mathbf{h}_{c_i}$ (as illustrated in Sect. 3.3.3.2). Such attraction value is given as:

$$\psi_{\mathbf{h}_i,\mathbf{h}_{c_i}} = \mathrm{SV}_{\mathbf{h}_{c_i}} \cdot e^{-\|\mathbf{h}_i - \mathbf{h}_{c_i}\|^2} \tag{3.18}$$

3. The selfish attraction $\varphi_{\mathbf{h}_i,\mathbf{h}_{cm}}$ denoting the attraction experienced by $\mathbf{h}_i$ toward the herd's center of mass $\mathbf{h}_M$ (as illustrated in Sect. 3.3.3.4). Such attraction is expressed as:

$$\psi_{\mathbf{h}_i,\mathbf{h}_M} = \mathrm{SV}_{\mathbf{h}_M} \cdot e^{-\|\mathbf{h}_i - \mathbf{h}_M\|^2} \tag{3.19}$$

Furthermore, while the previous allows to represent the attraction experimented by $\mathbf{h}_i$ toward other herd members, it is also useful to define an attraction with respect to the safest position currently known by the whole aggregation as follows:

$$\psi_{\mathbf{h}_i,\mathbf{x}_{\text{best}}} = e^{-\|\mathbf{h}_i - \mathbf{x}_{\text{best}}\|^2} \tag{3.20}$$

where $\mathbf{x}_{\text{best}}$ stands for the best position found so far by during SHO's evolutionary process, which satisfies that:

$$f(\mathbf{x}_{\text{best}}) = f_{\text{best}} \tag{3.21}$$

with $f(\mathbf{x}_{\text{best}})$ denoting the fitness value corresponding to the evaluation of the objective function $f(\cdot)$ with regard to $\mathbf{x}_{\text{best}}$ and where f_{best} stands for the best fitness value found so far, as given by Eq. (3.5).

Furthermore, while Eq. (3.16) represents an attraction factor toward other individuals within the selfish herd, it is important to remember that predators represent the main source of danger, and as such the movement patterns of individuals within such aggregation are also subject of being influenced by their presence. By considering this, SHO also assumes that individuals within the selfish herd are also able to experience some degree of repulsion toward the pack of attacking predators. To model such behavior, a repulsion factor may be defined as follows:

$$\varphi_{\mathbf{h}_i,\mathbf{p}_M} = -\mathrm{SV}_{\mathbf{p}_M} \cdot e^{-\|\mathbf{h}_i - \mathbf{p}_M\|^2} \tag{3.22}$$

where $\mathrm{SV}_{\mathbf{p}_M}$ denotes the survival value related to the predators' center of mass $\mathbf{p}_M$ as given by Eq. (3.15).

In SHO, the value $\varphi_{\mathbf{h}_i,\mathbf{p}_M}$ is known as selfish repulsion. Moreover, while it is possible to define a repulsion value between any given pair of prey and predator individuals, SHO only considers the repulsion experimented toward the predators' mass $\mathbf{p}_M$ under the assumption that any prey individual will always try to get as far as possible from all attacking predators.

3.3.4.2 Herd's Leader Movement Operators

Predation risk is the main motivation behind the individual decision-making behavior manifested by the members of a selfish herd. In this sense, while the leader of the herd is known to hold the safest position within the herd aggregation (and thus the highest chances of surviving a predator attack) this does not necessarily means that such individual is completely safe from the predators. With that being said, the leader of the selfish herd is able to manifest several different types of leadership behaviors depending on its current survival value. In SHO, the herd's leader position for the next iteration is updated as follows:

$$\mathbf{h}_L^{k+1} = \begin{cases} \mathbf{h}_L^k + \mathbf{c}^k & \text{if}\,\mathrm{SV}_{\mathbf{h}_L^k} = 1 \\ \mathbf{h}_L^k + \mathbf{s}^k & \text{if}\,\mathrm{SV}_{\mathbf{h}_L^k} < 1 \end{cases} \tag{3.23}$$

where k denoting the current iteration number.

The movement rule chosen by the herd's leader assuming that $\mathrm{SV}_{\mathbf{h}_L}^k = 1$ is called seemingly cooperative leadership. Such movement is performed under the assumption that the herd's leader $\mathbf{h}_L^k$ is located on either the current best location know so far by the selfish herd or an equally good position, thus granting it the highest possible survival value. In this sense, it can be shown from Eq. (3.4) that:

$$\text{if } f\left(\mathbf{h}_L^k\right) = f_{\text{best}} \rightarrow \mathrm{SV}_{\mathbf{h}_L}^k = 1 \tag{3.24}$$

It is important to recall that the movement of all other members within a selfish herd is strongly influenced by the decisions taken by the herd's leader. In this sense, a seemingly cooperative leadership aims to guide the movement of all other herd members in a way that could be potentially beneficial to the whole aggregation. In general, this is achieved by moving away of the pack of attacking predators (see Fig. 3.5). With that being said, the movement vector $\mathbf{c}^k$ could be expressed as:

$$\mathbf{c}^k = 2 \cdot \alpha \cdot \varphi_{\mathbf{h}_L,\mathbf{p}_M}^k \cdot \left(\mathbf{p}_M^k - \mathbf{h}_L^k\right) \tag{3.25}$$

where $\varphi_{\mathbf{h}_L,\mathbf{p}_M}^k$, as illustrated by Eq. (3.22), denotes the selfish repulsion experimented by the current herd's leader $\mathbf{h}_L^k$ toward the predators' center of mass $\mathbf{p}_M^k$ as given by Eq. (3.15), whereas α stand for random number within the interval [0, 1].

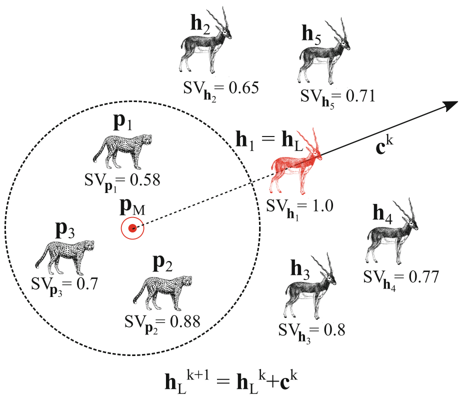


Fig. 3.5 Seemingly cooperative movement rule. Under such circumstance the herd's leader $\mathbf{h}_L$ (represented by $\mathbf{h}_1$) "altruistically" guides the herd aggregation away from the pack of attacking predators (represented by the predator's mass $\mathbf{p}_M$)

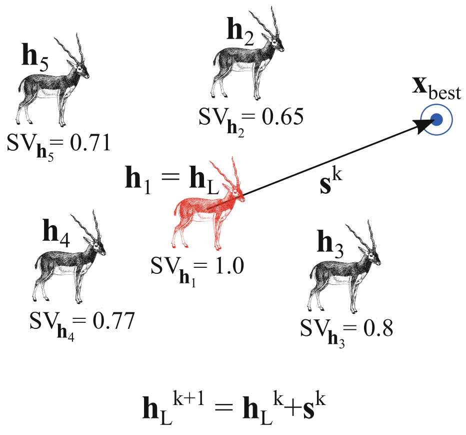


Fig. 3.6 Openly selfish movement rule. In such a case, the herd's leader $\mathbf{h}_L$ (shown as $\mathbf{h}_1$ in this case) aims to improve its own survival value by moving toward the currently known safest location (represented as $\mathbf{x}_{\text{best}}$)

On the other hand, if $\text{SV}_{\mathbf{h}_L^k} < 1$, the herd's leader chooses the movement rule known as openly selfish leadership. In the biological metaphor of the selfish herd behavior, each individual belonging to a selfish herd (including the herd's leader) will try at all cost to reduce their predation risk by moving to relatively safer positions. In an effort to improve its current survival value, the leader of the herd will opt to move toward the safest position currently known by the herd's aggregation (see Fig. 3.6). By considering this, the movement vector $\mathbf{s}^k$ may be defined as:

$$\mathbf{s}^k = 2 \cdot \alpha \cdot \psi^k_{\mathbf{h}_L,\mathbf{x}_{\text{best}}} \cdot \left(\mathbf{x}^k_{\text{best}} - \mathbf{h}^k_L\right) \tag{3.26}$$

where $\psi^k_{\mathbf{h}_L,\mathbf{x}_{\text{best}}}$, as in Eq. (3.20), denotes the selfish attraction experienced by the herd's leader $\mathbf{h}^k_L$ toward the global best position $\mathbf{x}^k_{\text{best}}$ found so far during SHO's evolutionary process, whereas α stand for random number from within the interval [0, 1].

From what was previously illustrated, the herd's leader selection criteria for a given movement rule may be summarized as follows: for any given iteration, if the leader of the herd $\mathbf{h}_L$ is the best individual among the members of the entire herd, the seemingly cooperative leadership movement rule is chosen; otherwise the openly selfish leadership movement rule is applied.

3.3.4.3 Herd's Following and Desertion Movement Operators

As illustrated in Sect. 3.3.3, depending on the current survival values of each individual within the selfish herd, SHO classifies the members of such aggregation in two distinctive groups: a group of herd followers ($\mathbf{H}_F$), and a group of herd deserters ($\mathbf{H}_D$). In SHO, the movement patterns manifested by the selfish herd are entirely dependent on the role assumed by each of its members. By considering this, at each iteration k, SHO computes the position update for each herd member as follows:

$$\mathbf{h}_i^{k+1} = \begin{cases} \mathbf{h}_i^k + \mathbf{f}_i^k & \text{if } \mathbf{h}_i^k \in \mathbf{H}_F^k \\ \mathbf{h}_i^k + \mathbf{d}_i^k & \text{if } \mathbf{h}_i^k \in \mathbf{H}_D^k \end{cases} \tag{3.27}$$

where $\mathbf{H}_F^k$ and $\mathbf{H}_D^k$ denote the sets of herd following and herd deserting members, as illustrated by Eqs. (3.9) and (3.10), respectively.

The movement rule chosen by the herd member $\mathbf{h}_i^k$, assuming that it belongs to the group of herd following members $\left(\mathbf{h}_i^k \in \mathbf{H}_F^k\right)$, is called herd following rule. This movement rule assumes that the herd member $\mathbf{h}_i^k$ opts to follow other members within the aggregation in an attempt to improve its own survival value, and as such, $\mathbf{f}_i^k$ denotes a movement vector computed with regard to the positions and survival aptitudes of other members within the herd. Furthermore, as previously stated, members within $\mathbf{H}_F^k$ may be further identified as either dominant members ($\mathbf{H}_d$) or subordinate members ($\mathbf{H}_s$), depending on how higher or lower their survival values are with respect to the mean survival value of the entire herd population. With this in mind, in SHO, the herd's following movement $\mathbf{f}_i^k$ performed by each herd member within $\mathbf{H}_F^k$ is calculated depending on whether it is a dominant or a subordinate member, as illustrated as follows:

$$\mathbf{f}_i^k = \begin{cases} 2 \cdot \left(\beta \cdot \psi_{\mathbf{h}_i,\mathbf{h}_L}^k \cdot \left(\mathbf{h}_L^k - \mathbf{h}_i^k\right) + \gamma \cdot \psi_{\mathbf{h}_i,\mathbf{h}_{c_i}}^k \cdot \left(\mathbf{h}_{c_i}^k - \mathbf{h}_i^k\right)\right) & \text{if } \mathbf{h}_i^k \in \mathbf{H}_d^k \\ 2 \cdot \delta \cdot \psi_{\mathbf{h}_i,\mathbf{h}_M}^k \cdot \left(\mathbf{h}_M^k - \mathbf{h}_i^k\right) & \text{if } \mathbf{h}_i^k \in \mathbf{H}_s^k \end{cases} \tag{3.28}$$

where $\mathbf{H}_d^k$ and $\mathbf{H}_s^k$ denote the sets of dominant and subordinate herd members , as given by Eqs. (3.11) and (3.12), respectively, whereas the values β, γ, and δ each represent a random number drawn from the interval [0, 1].

The movement rule performed by the herd's dominant members $\left(\mathbf{H}_d^k\right)$ is known as the nearest neighbor movement rule, which represent the situation in which the herd member $\mathbf{h}_i^k$ considers the positions of both, its nearest best neighbor and the current herd's leader when deciding where to move [9]. With that being said, $\mathbf{h}_L^k$ and

$\mathbf{h}_{c_i}^k$ stands for the position of the current herd's leader and the nearest best neighbor to $\mathbf{h}_i^k$, respectively, both as defined by Eqs. (3.7) and (3.8). Furthermore, $\psi_{\mathbf{h}_i,\mathbf{h}_L}^k$ and $\psi_{\mathbf{h}_i,\mathbf{h}_{c_i}}^k$, as given by Eqs. (3.17) and (3.18), denote the selfish attractions experimented by the herd member $\mathbf{h}_i^k$ toward $\mathbf{h}_L^{it}$ and $\mathbf{h}_{C_i}^{it}$ respectively (see Fig. 3.7).

On the other hand, the movement rule chosen by the herd's subordinate members $\left(\mathbf{H}_s^k\right)$ is known as crowded horizon movement. In this situation, each herd member considers both the location and survival values of all other members within the aggregation to guide its movement [17]. In SHO, such movement is performed by considering the herd's center of mass $\mathbf{h}_M^k$, as defined by Eq. (3.14), and its corresponding selfish attraction $\psi_{\mathbf{h}_i,\mathbf{h}_M}^k$, as given by according to Eq. (3.19) (see Fig. 3.8).

Finally, if the herd member $\mathbf{h}_i^k$ is instead identified as a herd deserting member $\left(\mathbf{h}_i^k \in \mathbf{H}_D^k\right)$, said individual performs movement rule known in SHO as herd desertion. In this case, individuals grouped as herd deserting members are assumed

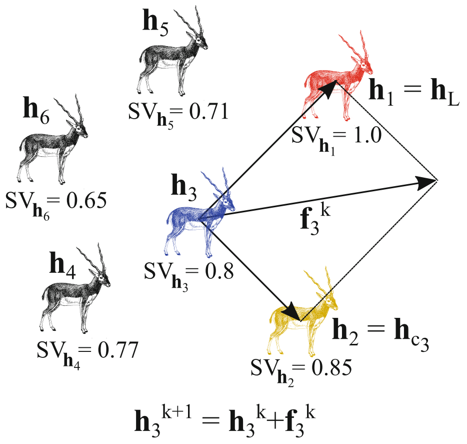

$\mathbf{h}_5$
$SV_{\mathbf{h}_5} = 0.71$
$\mathbf{h}_1 = \mathbf{h}_L$
$SV_{\mathbf{h}_1} = 1.0$
$\mathbf{h}_6$
$SV_{\mathbf{h}_6} = 0.65$
$\mathbf{h}_3$
$\mathbf{f}_3^k$
$SV_{\mathbf{h}_3} = 0.8$
$\mathbf{h}_4$
$SV_{\mathbf{h}_4} = 0.77$
$\mathbf{h}_2 = \mathbf{h}_{c_3}$
$SV_{\mathbf{h}_2} = 0.85$
$\mathbf{h}_3^{k+1} = \mathbf{h}_3^k + \mathbf{f}_3^k$

Fig. 3.7 Nearest neighbor movement rule for a dominant herd member (illustrated as $\mathbf{h}_3$). In this case, the dominant member considers the positions of both, its nearest best neighbor $\mathbf{h}_{c3}$ (shown as $\mathbf{h}_2$) and the current herd's leader $\mathbf{h}_L$ (represented by $\mathbf{h}_1$)

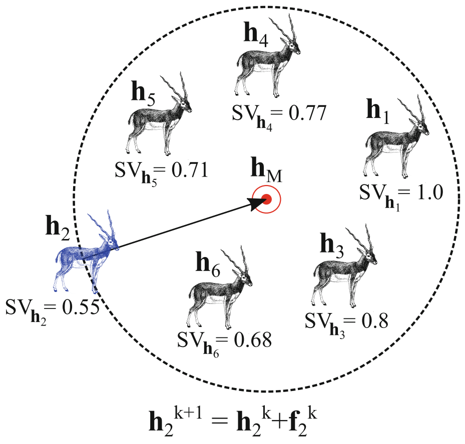

$\mathbf{h}_4$
$\mathbf{h}_5$
$SV_{\mathbf{h}_4} = 0.77$
$\mathbf{h}_1$
$SV_{\mathbf{h}_5} = 0.71$
$\mathbf{h}_M$
$SV_{\mathbf{h}_1} = 1.0$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_6$
$SV_{\mathbf{h}_2} = 0.55$
$SV_{\mathbf{h}_3} = 0.8$
$SV_{\mathbf{h}_6} = 0.68$
$\mathbf{h}_2^{k+1} = \mathbf{h}_2^k + \mathbf{f}_2^k$

Fig. 3.8 Crowded horizon movement rule for a subordinated herd member (represented by $\mathbf{h}_2$). In this situation, the subordinated member moves toward the location of the herd's center of mass (shown as $\mathbf{h}_M$)

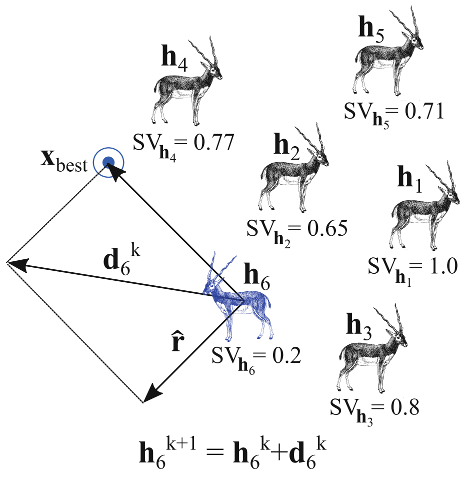


Fig. 3.9 Herd desertion movement rule. In such situation a given herd deserting member (shown as $\mathbf{h}_6$) move independently of all other members within the herd aggregation

to move independently to all other members of the herd [11], and as such $\mathbf{d}_i^k$ denotes a movement performed without regard to any other individuals within the aggregation (see Fig. 3.9). By considering this, the herd desertion movement $\mathbf{d}_i^k$ may then be given as following:

$$\mathbf{d}_i^k = 2 \cdot \left(\beta \cdot \psi_{\mathbf{h}_i,\mathbf{x}_{\text{best}}}^k \cdot \left(\mathbf{x}_{\text{best}}^k - \mathbf{h}_i^k \right) + \gamma \cdot \left(1 - \text{SV}_{\mathbf{h}_i^k} \right) \cdot \hat{\mathbf{r}} \right) \tag{3.29}$$

where $\psi_{\mathbf{h}_i,\mathbf{x}_{\text{best}}}^k$, as in Eq. (3.20), denotes the selfish attraction experienced by the herd member $\mathbf{h}_i^k$ toward the current global best position $\left(\mathbf{x}_{\text{best}}^k\right)$, whereas β and γ stand for random numbers drawn from the interval [0, 1]. Furthermore $\hat{\mathbf{r}}$ denotes unit vector pointing to a random direction within the given n-dimensional solution space.

3.3.5 Predators Movement Operators

Herd members within an aggregation typically have lower predation risk in comparison to solitary individuals. This is because the effects of delusion and the confusion caused by the movement of many individuals influence the predator's decision to aim its attack toward a particular individual [18]. However, these benefits are not homogenous among all members of a herd aggregation. As previously stated, the predation risk of any given member of a selfish herd is directly related to its current position within such aggregation [9]. Intuitively, attacking predators take advantage of these apparent vulnerabilities when choosing a prey for pursuing. In addition, the relative position occupied by such predators with respect to the members of the herd is also an influential factor when deciding which herd member is going to be attacked.

By considering these facts, SHO models the movement of each individual within the pack of predators by considering both, the survival aptitudes of individuals within attacked herd and the distance which separate such individuals from the attacking predators.

3.3.5.1 Pursuit Probabilities

In order to model the movement of individuals within the group of predators $\mathbf{P}$ (as defined in Sect. 3.3.3.1), it is first assumed that each member $\mathbf{h}_j$ within the herd $\mathbf{H}$ has a certain probability of being pursued by an attacking predator $\mathbf{p}_i$. In SHO, such pursuit probability is given as:

$$\mathcal{P}_{\mathbf{p}_i,\mathbf{h}_j} = \frac{\omega_{\mathbf{p}_i,\mathbf{h}_j}}{\sum_{m=1}^{N_h} \omega_{\mathbf{p}_i,\mathbf{h}_m}} \tag{3.30}$$

The value $\omega_{\mathbf{p}_i,\mathbf{h}_j}$ in Eq. (3.30) is referred as the prey attractiveness between $\mathbf{p}_i$ and $\mathbf{h}_j$. Such value considers both, the survival aptitudes possessed $\mathbf{h}_j$ and the distance separating such individual from the attacking predator $\mathbf{p}_i$, as illustrated as follows:

$$\omega_{\mathbf{p}_i,\mathbf{h}_j} = \left(1 - \mathrm{SV}_{\mathbf{h}_j}\right) \cdot e^{-\left\|\mathbf{p}_i - \mathbf{h}_j\right\|^2} \tag{3.31}$$

where $\mathrm{SV}_{\mathbf{h}_j}$ denotes for the survival value related to $\mathbf{h}_j$ whereas $\left\|\mathbf{p}_i - \mathbf{h}_j\right\|$ stands for the Euclidian distance between $\mathbf{p}_i$ and $\mathbf{h}_j$. Intuitively, prey attractiveness $\omega_{\mathbf{p}_i,\mathbf{h}_j}$ yield to higher values toward members $\mathbf{h}_j$ possessing lower survival values $\mathrm{SV}_{\mathbf{h}_j}$, whereas a higher survival value $\mathrm{SV}_{\mathbf{h}_j}$ implies a lower attractiveness value. This is analogous to a predator's preference to attack apparently weaker prey over those who are seemingly stronger. Similarly, it can also be observed that the value of $\omega_{\mathbf{p}_i,\mathbf{h}_j}$ increases as the distance $\left\|\mathbf{p}_i - \mathbf{h}_j\right\|$ between $\mathbf{p}_i$ and $\mathbf{h}_j$ decreases, while such value decreases as the distance gap between both individuals increases. Once again, this refers to a predator's preference to attack nearby preys rather than those occupying distant positions.

3.3.5.2 Predators Position Update

The pursuit probability $\mathcal{P}_{\mathbf{p}_i,\mathbf{h}_j}$ represents the probability for a given predator $\mathbf{p}_i$ to pursuit a certain herd member $\mathbf{h}_j$ given its perceived survival aptitudes and the distance separating them. With this in mind, at each iteration k, SHO models the movement of the each predator $\mathbf{p}_i$ within the pack of attacking predators $\mathbf{P}$ by considering the position of a particular herd member, as illustrated as follows:

$$\mathbf{p}_i^{k+1} = \mathbf{p}_i^k + 2 \cdot \rho \cdot \left(\mathbf{h}_r^k - \mathbf{p}_i^k\right) \tag{3.32}$$

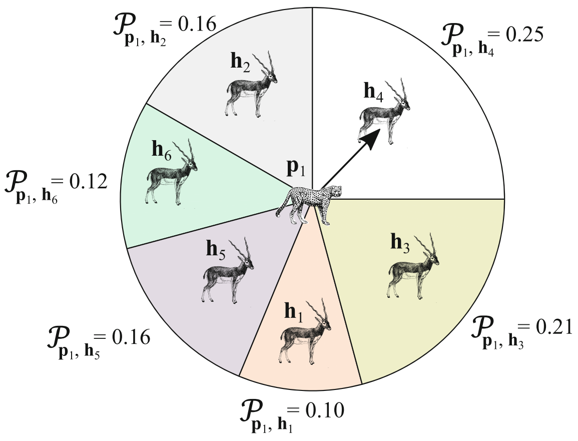

$\mathcal{P}_{\mathbf{p}_1,\mathbf{h}_2} = 0.16$
$\mathcal{P}_{\mathbf{p}_1,\mathbf{h}_4} = 0.25$
$\mathbf{h}_2$
$\mathbf{h}_4$
$\mathbf{h}_6$
$\mathbf{p}_1$
$\mathcal{P}_{\mathbf{p}_1,\mathbf{h}_6} = 0.12$
$\mathbf{h}_3$
$\mathbf{h}_5$
$\mathbf{h}_1$
$\mathcal{P}_{\mathbf{p}_1,\mathbf{h}_3} = 0.21$
$\mathcal{P}_{\mathbf{p}_1,\mathbf{h}_5} = 0.16$
$\mathcal{P}_{\mathbf{p}_1,\mathbf{h}_1} = 0.10$

Fig. 3.10 By applying the roulette selection method with regard to the pursuit probabilities $\mathcal{P}_{\mathbf{p}_i,\mathbf{h}_j}$, related to $\mathbf{p}_i$ (illustrated as $\mathbf{p}_1$) and each individual $\mathbf{h}_j$ within the herd of prey (shown as $\mathbf{h}_1$–$\mathbf{h}_6$), a single member $\mathbf{h}_j$ is chosen to be pursued by $\mathbf{p}_i$

where ρ denotes a random number between from the interval [0, 1]. Furthermore, $\mathbf{h}_r^k = \mathbf{h}_j^k \in \mathbf{H}^k$ (with $r \in \{1, 2, \ldots, N_h\}$) denotes a herd member randomly chosen from among the members of the whole herd aggregation $\left(\mathbf{H}^k\right)$ by applying the roulette selection method [19] with regard to their individual pursuit probabilities $\mathcal{P}_{\mathbf{p}_i,\mathbf{h}_j}$, as given by Eq. (3.31) (see Fig. 3.10).

3.3.6 Predation Phase

In nature, the biological interaction between groups of prey and predators, in which the former individuals are hunted by the latter, is known as predation. Such interactions often result in the death of prey and its eventual consumption by the predators. Typical hunting behavior suggest that there is a finite distance range over which a predator can launch a successful attack against a pursued prey. In the case of the selfish herd behavior described by [9], such finite range is described by the so called domain of danger which represents the area around a given prey in which, if a predator is present, there is a high change that such individual is attacked and possibly killed. Also, it is suggested that, if multiple domains of danger are invaded by attacking predator at the same time, there is usually a higher preference for killing the nearest prey; however other factors, such as the apparent survival aptitudes of each endangered prey, may also be an influential factor in the predator's final decision to attack.

In SHO, a computational procedure, called predation phase, is implemented to model such prey-predator interactions. In the predation phase, it is assumed that, after both the members of a selfish herd (the prey) and the pack of predators have performed a movement (according to the operators described in Sects. 3.3.4 and 3.3.5), there is a chance for several herd members to be killed by the attacking predators, which further implies the exclusion of their respective solutions from within the given decision space.

3.3.6.1 Domain of Danger

As previously illustrated, the domain of danger represents the area around a particular prey, in which, if a predator is present, there is a high chance for such prey to be attacked and killed. For the predation phase, SHO defines a finite domain of danger, represented as a circular area of finite radius around each prey as follows:

$$R = \frac{\sum_{j=1}^{n} \left| x_j^{\text{low}} - x_j^{\text{high}} \right|}{2 \cdot n} \tag{3.33}$$

where x_j^{low} and x_j^{high} stand for initial lower and upper bounds respectively, whereas n denotes the number of dimensions (for simplicity, SHO assumes that the radius of each domain of danger is the same for all prey).

3.3.6.2 Threatened Prey

At the start of the predation phase, it is assumed that no prey has been hunted (killed) by the attacking predators. To represent this, SHO first initializes an empty set **K**. That is:

$$\mathbf{K} = \{\emptyset\} \tag{3.34}$$

During the predation phase, such set **K** is used to group all herd members $\left(\mathbf{h}_j \in \mathbf{H}\right)$ that are killed by the attacking predators $(\mathbf{p}_i \in \mathbf{P})$.

In SHO, a predator $\mathbf{p}_i$ is assumed to be able to hunt a certain herd member $\mathbf{h}_j$ if two specific conditions are met: 1. the member $\mathbf{h}_j$ has a lower survival value than $\mathbf{p}_i$, and 2. the distance between $\mathbf{p}_i$ and $\mathbf{h}_j$ is equal to or lower than the domain of danger radius R (as given by Eq. (3.33)), which implies that $\mathbf{p}_i$ has invaded $\mathbf{h}_j$'s domain of danger. Furthermore, it is also assumed that more than one herd member could be threatened by particular predator, assuming the previous two conditions have been met. With that being said, for each predator $\mathbf{p}_i$ we may represent a set of threatened prey as follows:

$$\mathbf{T}_{\mathbf{p}_i} = \left\{ \mathbf{h}_j \in \mathbf{H} | \text{SV}_{\mathbf{h}_j} < \text{SV}_{\mathbf{p}_i}, \left\| \mathbf{p}_i - \mathbf{h}_j \right\| \leq R, \mathbf{h}_j \notin \mathbf{K} \right\} \tag{3.35}$$

where $\text{SV}_{\mathbf{p}_i}$ and $\text{SV}_{\mathbf{p}_j}$ denote the survival values of $\mathbf{p}_i$ and $\mathbf{h}_j$, respectively, whereas $\mathbf{p}_i - \mathbf{h}_j$ denotes the Euclidian distance between the individuals $\mathbf{p}_i$ and $\mathbf{h}_j$. Furthermore, note from Eq. (3.36) that only members that are not currently grouped within **K** (which groups all herd members hunted during the predation phase) are able targeted by $\mathbf{p}_i$ as candidates to be hunted.

3.3.6.3 Probability of Being Hunted

Once a set threatened prey $\mathbf{T}_{\mathbf{p}_i}$ (as given by Eq. (3.35)) has been identified for a particular predator $\mathbf{p}_i$, one of such threatened individuals must be chosen, and then, killed. In SHO, such decision is taken based each threatened herd member probabilities of being hunted, as given as follows:

$$\mathcal{H}_{\mathbf{p}_i,\mathbf{h}_j} = \frac{\omega_{\mathbf{p}_i,\mathbf{h}_j}}{\sum_{(\mathbf{h}_m \in \mathbf{T}_{\mathbf{p}_i})} \omega_{\mathbf{p}_i,\mathbf{h}_m}}, \mathbf{h}_\mathrm{j} \in \mathbf{T}_{\mathbf{p}_i} \tag{3.36}$$

where $\omega_{\mathbf{p}_i,\mathbf{h}_j}$ denotes the prey attractiveness between $\mathbf{p}_i$ and $\mathbf{h}_j$, as given by Eq. (3.31).

Finally, by applying the roulette selection method [19] with regard to the probabilities $\mathcal{H}_{\mathbf{p}_i,\mathbf{h}_j}$ of each threatened prey, one of such individuals is chosen, and then, considered to be killed by the attacking predator $\mathbf{p}_i$ (see Fig. 3.11). Furthermore, any member $\mathbf{h}_\mathrm{j} \in \mathbf{T}_{\mathbf{p}_i}$ selected by applying this method is grouped within the set of killed herd members $\mathbf{K}$. It should be noted that if $\mathbf{T}_{\mathbf{p}_i} = \{\emptyset\}$ for any given predator $\mathbf{p}_i$ (that is, there is no prey that could be threatened by $\mathbf{p}_i$), then no prey is hunted by $\mathbf{p}_i$ and as such, no changes are made to $\mathbf{K}$ for that case. With the previous being said, at the end of the predation phase, the previously empty set $\mathbf{K}$ may group all herd members (if any) hunted by each of the attacking predators, such that:

$$\begin{gathered} \mathbf{K} = \left\{\mathbf{k}_i = \left(\mathbf{h}_j \in \mathbf{H}\right)\right\} \\ \text{for } i = 1, \ldots, N_k,\ j \in \{1, 2, \ldots, N_h\} \end{gathered} \tag{3.37}$$

where N_k denote the total number of herd members $\mathbf{h}_j$ killed during the predation phase.

Furthermore, any solutions corresponding to a killed herd member $\mathbf{h}_j$ during the predation phase is further considered to be excluded from within the given decision space. However, as it will be illustrated in the next section, such excluded solutions will be replaced by some new solutions, generated by applying a special mating-like operator.

3.3.7 Restoration Phase

In nature, the size of prey populations change dynamically through time as a result of predation, but in general, in well balanced biological systems, such change tends to be periodical. This means that even if a population of prey decreases as a result of the predation, the natural balance of the ecosystem will eventually allow the restoration of such population [20].

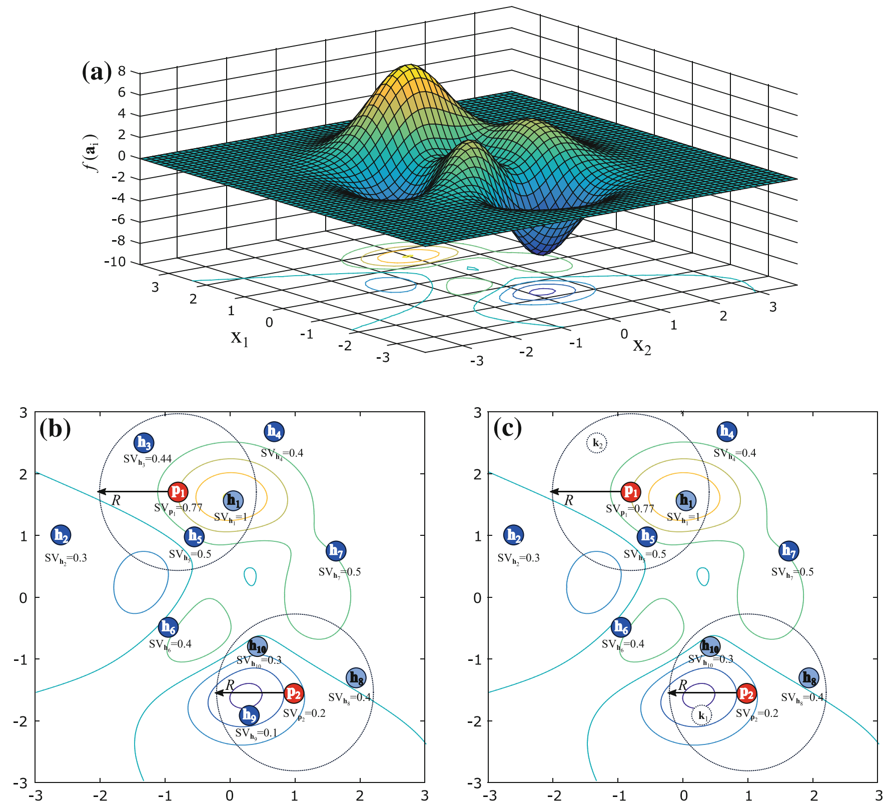


Fig. 3.11 Example of the predation phase procedure. **a** Optimization problem, **b** position's configuration for both prey (herd members) and predators, and **c** outcome of the predation phase where the herd members $\mathbf{h}_3$ and $\mathbf{h}_9$ are assumed to be killed by the attacking predators $\mathbf{p}_1$ and $\mathbf{p}_2$, respectively

Analogous to this phenomenon, SHO implements a computational procedure which allows the replacement of herd members killed during the predation phase (see Sect. 3.3.6). Such procedure, called restoration phase, employs a special mating-like operator to generate new solutions based on the survival aptitudes of the remaining herd members, and then such solutions are used to replace all solutions excluded as a result of such predation phase.

3.3.7.1 Mating Probabilities

SHO's mating operation considers the positions and survival values of all herd members that weren't hunted during the predation phase. As such, we may first define a set of mating candidates as follows:

$$\mathbf{M} = \{\mathbf{h}_j \notin \mathbf{K}\} \tag{3.38}$$

where **K** denotes the set of herd members killed during the predation phase, as given by Eq. (3.37).

For the mating operation applied in SHO, each member $\mathbf{h}_j$ within the set of mating candidates **M** is considered to have a certain probability $\mathcal{M}_{\mathbf{h}_j}$ of being considered for the generation of new solution. Such probability depends on the survival aptitude of each mating candidate, as illustrated as follows:

$$\mathcal{M}_{\mathbf{h}_j} = \frac{\mathrm{SV}_{\mathbf{h}_j}}{\sum_{(\mathbf{h}_m \in \mathbf{M})} \mathrm{SV}_{\mathbf{h}_m}}, \mathbf{h}_j \in \mathbf{M} \tag{3.39}$$

From the previous, it is clear that mating candidates possessing higher survival values $\mathrm{SV}_{\mathbf{h}_j}$ will a have a higher chance to influence the generation of new solutions, whereas individuals with lower survival values are less likely to be considered for such a process.

3.3.7.2 Mating Operation

In order to generate a new candidate solution, we first consider a set of n randomly chosen individuals $\{\mathbf{h}_{r_1}, \mathbf{h}_{r_2}, \ldots, \mathbf{h}_{r_n}\}$ (with $\mathbf{h}_{r_i} = \mathbf{h}_j \in \mathbf{M}$), selected by applying the roulette selection method [19] with regard to the probabilities $\mathcal{M}_{\mathbf{h}_j}$ of each member within the set of mating candidates **M**, as given by Eq. (3.39).

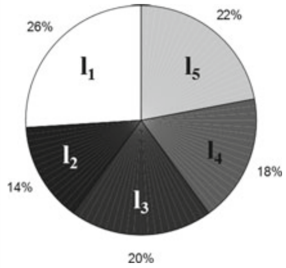
$$\mathbf{h}_{\text{new}} = \text{mix}\left(\left[h_{r_1,1}, h_{r_2,2}, \ldots, h_{r_n,n}\right]\right) \tag{3.40}$$

where $h_{r_i,l}$ (with $l = 1, 2, \ldots, n$) correspond to the l-esim position element of the random candidate $\mathbf{h}_{r_i}$ and where n stands for the dimensionality of the solution space. Furthermore, the function mix(·) is applied to change the 'l' indexing of each element $h_{r_i,l}$, such that each of said elements is indexed on a different entry (dimension). An example of SHO's mating operation for generating a new candidate solution is illustrated in Table 3.1.

3.3.7.3 Replacement of Excluded Herd Members

As previously stated, the mating operation in SHO is used to replace herd members (solutions) excluded as a result of the predation phase (as described in Sect. 3.3.6). With that being said, for each of member $\mathbf{h}_j \in \mathbf{K}$ (with **K** denoting the set of killed herd members, as given by Eq. (3.37)), we generate a new candidate solution $\mathbf{h}_{\text{new}}$ by applying the mating operation described by Eq. (3.40), and then such new solution is assigned to $\mathbf{h}_j$, such that:

Table 3.1 Example of a mating operation used to generate a new herd member $\mathbf{h}_{\text{new}}$

Mating candidates $(\mathbf{h}_j \in \mathbf{M})$	Positions $[h_{j,1}, h_{j,2}, h_{j,3}]$	$\text{SV}_{\mathbf{h}_j}$	$\mathcal{M}_{\mathbf{h}_j}$	$\mathbf{h}_{r_i}$	Roulette
$\mathbf{h}_1$	(0.3, 0.5, −0.8)	1	0.26	$\mathbf{h}_{r_1}$	
$\mathbf{h}_2$	(0.6, −0.4, 0.7)	0.55	0.14		
$\mathbf{h}_3$	(0.2, −0.5, 0.3)	0.76	0.20	$\mathbf{h}_{r_2}$	
$\mathbf{h}_4$	(0.1, 0.5, −0.7)	0.68	0.18		
$\mathbf{h}_5$	(−0.4, 0.3, 0.7)	0.83	0.22	$\mathbf{h}_{r_3}$	
Mixed elements		**New position**			
$[h_{5,3}, h_{1,1}, h_{3,1}]$		$\mathbf{h}_{\text{new}} = [0.7, 0.3, -0.5]$			

$$\mathbf{h}_j = \mathbf{h}_{\text{new}}, \mathbf{h}_j \in \mathbf{K} \tag{3.41}$$

By applying this procedure, all previously excluded herd members $\mathbf{h}_j \in \mathbf{K}$ are replaced with new solutions, formed by considering the positions and survival aptitudes of the members which survived the predation phase.

3.4 Summary of the SHO Algorithm

The Selfish Herd Optimizer (SHO) is an evolutionary algorithm developed to solve global optimization problems. The proposed method draws inspiration on the interesting selfish herd behavior observed in a wide variety of species subjected to a prey-predator relationship. Such method employs to kinds of search agents: a herd of prey and a pack predators. The movement of each of this agents within the solution space is performed by applying a set of unique evolutionary operators inspired in several movement rules observed in most selfish herds. Furthermore, the computational procedure known as predation phase emulates the situation in which prey belonging to a selfish herd are hunted down and killed by attacking predators. Such mechanism is employed to allow the exclusion of either bad or redundant solutions, found during SHO's the evolutionary process. Finally, the restoration phase allows the restitution of solutions previously excluded during the predation phase by generating a set of new candidate solution. Such procedure is reminiscent to the natural balance mechanism found in many biological systems and allows the diversification of the set of solutions found by SHO.

In general, the SHO algorithm's computational procedure may be summarized by the following steps:

Step 1 Initialize the animal population **A**.
Step 2 Split **A** in two groups: a group of prey **H** and a group of predators **P**.
Step 3 Calculate the survival values for each individual within **H** and **P**.
Step 4 Apply the herd movement operators to each individual of **H**.
Step 5 Apply the predators movement operators to each individual of **P**.
Step 6 Re-calculate the survival values for each individual within **H** and **P**.
Step 7 Perform predation phase.
Step 8 Perform restoration phase.
Step 9 If stop criterion is met, the process is finished; otherwise, return to Step 4.

A more detailed explanation of the computational procedure employed by the SHO method is given in by the following pseudocode (Algorithm 1):

Algorithm 1. SHO algorithm computational procedure

```
Step 1: Considering $N$ as the total number of $n$-dimensional animals, initialize the animal population **A**
1.1. Randomly initialize each animal individual's position
for ($i = 1{:}N$)
    for ($j = 1{:}n$)
        $a_{i,j}^{0} = x_j^{\text{low}} + \text{rand}(0,1) \cdot \left(x_j^{\text{high}} - x_j^{\text{low}}\right)$
    end for
end for
Step 2: Divide **A** in two groups: a group of herd members **H** and a group of predators **P**.
2.1. Define the number of herd members ($N_h$) and predators ($N_p$) within **A.**
$N_h = \text{floor}\left(N \cdot \text{rand}(0.7, 0.9)\right)$
$N_p = N - N_h$
2.2. Assign $N_h$ individuals from **A** as herd members.
for ($i = 1{:}N_h$)
    $\mathbf{h}_i = \mathbf{a}_i$
end for
2.3. Assign $N_p$ individuals from **A** as predators.
for ($j = 1{:}N_p$)
    $\mathbf{p}_j = \mathbf{a}_{N_h+j}$
end for
Step 3: Calculate the survival values of each member within **H** and **P**.
3.1. Calculate survival values for member in **H.**
for ($i = 1{:}N_h$)
    $\text{SV}_{\mathbf{h}_i} = \frac{f(\mathbf{h}_i) - f_{\text{worst}}}{f_{\text{best}} - f_{\text{worst}}}$
end for
3.2. Calculate survival values for each member in **P.**
for ($j = 1{:}N_p$)
    $\text{SV}_{\mathbf{p}_j} = \frac{f(\mathbf{p}_j) - f_{\text{worst}}}{f_{\text{best}} - f_{\text{worst}}}$
end for
Step 4: Move all members in **H** by applying the herd movement operators.
4.1. Selfish herd movement operators.
for ($i = 1{:}N_h$)
        if $\text{SV}_{\mathbf{h}_i^k} = \max_{j \in \{1,2,\ldots,N_h\}}(\text{SV}_{\mathbf{h}_j})$
        4.1.1. $\mathbf{h}_i^k$ is the leader of the herd ($\mathbf{h}_i^k = \mathbf{h}_L^k$). Apply herd's leader movement operators.
            if ($\text{SV}_{\mathbf{h}_L^k} = 1$)
            4.1.1(A). Apply seemingly cooperative leadership movement operators.
                $\mathbf{c}^k = 2 \cdot \alpha \cdot \varphi_{\mathbf{h}_L,\mathbf{p}_M}^k \cdot (\mathbf{p}_M^k - \mathbf{h}_M^k)$
                $\mathbf{h}_L^{k+1} = \mathbf{h}_L^k + \mathbf{c}^k$
            else
            4.1.1(B). Apply openly selfish leadership movement operators.
                $\mathbf{s}^k = 2 \cdot \alpha \cdot \psi_{\mathbf{h}_L,\mathbf{x}_{\text{best}}}^k \cdot \left(\mathbf{x}_{\text{best}}^k - \mathbf{h}_L^k\right)$
                $\mathbf{h}_L^{k+1} = \mathbf{h}_L^k + \mathbf{s}^k$
            end if
        else
        4.1.2. Apply herd's following and desertion movement operators.
```

```
                if $\mathrm{SV}_{\mathbf{h}_i^k} \geq \mathrm{rand}(0,1)$
                    4.1.2(A). $\mathbf{h}_i^k$ is a herd following member ($\mathbf{h}_i^k \in \mathbf{H}_F^k$). Apply herd following movement operators.
                        if $\mathrm{SV}_{\mathbf{h}_i^k} \geq \mathrm{SV}_{\mathbf{h}_\mu^k}$
                            $\mathbf{f}_i^k = 2 \cdot \left(\beta \cdot \psi_{\mathbf{h}_i,\mathbf{h}_L}^k \cdot \left(\mathbf{h}_L^k - \mathbf{h}_i^k\right) + \gamma \cdot \psi_{\mathbf{h}_i,\boldsymbol{h}_{c_i}}^k \cdot \left(\mathbf{h}_{c_i}^k - \mathbf{h}_i^k\right)\right)$
                        else
                            $\mathbf{f}_i^k = 2 \cdot \delta \cdot \psi_{\mathbf{h}_i,\mathbf{h}_M}^k \cdot \left(\mathbf{h}_M^k - \mathbf{h}_i^k\right)$
                        end if
                        $\mathbf{h}_i^{k+1} = \mathbf{h}_i^k + \mathbf{f}_i^k$
                else
                    4.1.2(B). $\mathbf{h}_i^k$ is a herd deserting member ($\mathbf{h}_i^t \in \mathbf{H}_D^k$). Apply herd desertion movement operators.
                        $\mathbf{d}_i^k = 2 \cdot \left(\beta \cdot \psi_{\mathbf{h}_i,\mathbf{x}_{\text{best}}}^k \cdot (\mathbf{x}_{\text{best}}^k - \mathbf{h}_i^k) + \gamma \cdot (1 - SV_{\mathbf{h}_i^k}) \cdot \hat{\mathbf{r}}\right)$
                        $\mathbf{h}_i^{k+1} = \mathbf{h}_i^k + \mathbf{d}_i^k$
                end if
        end if
end for
Step 5: Move all members in P by applying the predator movement operators.
5.1. Predators movement operators.
for ($i = 1{:}\,N_p$)
        5.1.1. For each member $\mathbf{p}_i$, compute the pursuit probabilities $\mathcal{P}_{\mathbf{p}_i,\mathbf{h}_j}$
        for ($j = 1{:}\,N_h$)
            $\mathcal{P}_{\mathbf{p}_i,\mathbf{h}_j} = \frac{\omega_{\mathbf{p}_i,\mathbf{h}_j}}{\sum_{m=1}^{N_h} \omega_{\mathbf{p}_i,\mathbf{h}_m}}$
        end for
        5.1.2. Choose a random member $\mathbf{h}_r^k$ from H by applying the roulette selection method according to their pursuit probabilities $\mathcal{P}_{\mathbf{p}_i,\mathbf{h}_j}$ and update the position of $\mathbf{p}_i$.
        $\mathbf{p}_i^{k+1} = \mathbf{p}_i^k + 2 \cdot \rho \cdot (\mathbf{h}_r^k - \mathbf{p}_i^k)$
end for
Step 6: Re-calculate the survival values of each member within H and P.
6.1. Re-calculate survival values for each member in H.
for ($i = 1{:}\,N_h$)
        $\mathrm{SV}_{\mathbf{h}_i^{k+1}} = \frac{f\left(\mathbf{h}_i^{k+1}\right) - f_{\text{worst}}}{f_{\text{best}} - f_{\text{worst}}}$
end for
6.2. Re-calculate survival values for each member in P.
for ($i = 1{:}\,N_p$)
        $\mathrm{SV}_{\mathbf{p}_i^{k+1}} = \frac{f\left(\mathbf{p}_i^{k+1}\right) - f_{\text{worst}}}{f_{\text{best}} - f_{\text{worst}}}$
end for
Step 7: Perform predation phase.
7.1. Calculate the domain of danger's radius $R$.
$R = \frac{\sum_{j=1}^{n} \left|x_j^{\text{low}} - x_j^{\text{high}}\right|}{2 \cdot n}$
7.2. Initialize the set of killed prey individuals K as an empty set.
$\mathbf{K} = \{\emptyset\}$
7.3. For each predator $\mathbf{p}_i \in \mathbf{P}$ choose a valid prey individual $\mathbf{h}_j \in \mathbf{H}$ to be hunted.
```

```
for ($i = 1: N_p$)
        7.3.1. Define a set of targeted prey $\mathbf{T}_{\mathbf{p}_i}$ for $\mathbf{p}_i$.
        $\mathbf{T}_{\mathbf{p}_i} = \left\{\mathbf{h}_j \in \mathbf{H} \middle| \mathrm{SV}_{\mathbf{h}_j} < \mathrm{SV}_{\mathbf{p}_i},\ \left\|\mathbf{p}_i - \mathbf{h}_j\right\| \le R,\ \mathbf{h}_j \notin \mathbf{K}\right\}$
end for
if $\mathbf{T}_{\mathbf{p}_i}$ is not empty $(\mathbf{T}_{\mathbf{p}_i} \neq \{\emptyset\})$
        7.3.2. Compute probabilities of being hunted for each member in $\mathbf{T}_{\mathbf{p}_i}$
        for $(j = 1: |\mathbf{T}_{\mathbf{p}_i}|)$
                $\mathcal{H}_{\mathbf{p}_i,\mathbf{h}_j} = \frac{\omega_{\mathbf{p}_i,\mathbf{h}_j}}{\sum_{(\mathbf{h}_m \in \mathbf{T}_{\mathbf{p}_i})} \omega_{\mathbf{p}_i,\mathbf{h}_m}},\quad \mathbf{h}_j \in \mathbf{T}_{\mathbf{p}_i}$
        end for
        7.3.3. Choose a random member $\mathbf{h}_j \in \mathbf{T}_{\mathbf{p}_i}$ by applying the roulette selection method with regard
        to their respective probabilities $\mathcal{H}_{\mathbf{p}_i,\mathbf{h}_j}$ and group it within the set $\mathbf{K}$.
        $\mathbf{K} = \{\mathbf{K}, \mathbf{h}_j\}$
end if
Step 8: Perform restoration phase.
8.1. Define a set of mating candidates $\mathbf{M}$.
$\mathbf{M} = \{\mathbf{h}_j \notin \mathbf{K}\}$
8.2. For each member within $\mathbf{M}$, calculate its corresponding mating probability $\mathcal{M}_{\mathbf{h}_j}$.
for each $\mathbf{h}_j \in \mathbf{M}$
        $\mathcal{M}_{\mathbf{h}_j} = \frac{\mathrm{SV}_{\mathbf{h}_j}}{\sum_{(\mathbf{h}_m \in \mathbf{M})} \mathrm{SV}_{\mathbf{h}_m}},\quad \mathbf{h}_j \in \mathbf{M}$
end for
8.3. Replace each $\mathbf{h}_j \in \mathbf{K}$ with a new solution
for each $\mathbf{h}_j \in \mathbf{k}$
        8.3.1. Generate a new solution $\mathbf{h}_{\mathrm{new}}$ by applying SHO's mating operation.
        $\mathbf{h}_{\mathrm{new}} = \mathrm{mix}([h_{r_1,1}, h_{r_2,2}, \ldots, h_{r_n,n}])$
        8.3.2. Assign $\mathbf{h}_{\mathrm{new}}$ to $\mathbf{h}_j$.
        $\mathbf{h}_j = \mathbf{h}_{\mathrm{new}}$
end for
Step 9: If stop criterion is met, the process is finished; otherwise, go back to Step 4.
```

3.5 Discussion About the SHO Algorithm

Within the framework of many Evolutionary Algorithms (EA), the dilemma concerning to the balance between exploration and exploitation of solutions has remained as an important topic for many years. In this sense, it is known that emphasizing to much on exploration increases the capacity of an EA to find new potential solutions; however, this usually yields to a degradation on the precision of such EA's evolutionary process. On the other hand, giving more importance to exploitation allows the refinement of currently existing solutions but, adversely, there is a tendency to drive the process toward local optima. With that being said, the ability of an EA to find a global optimum depends on its capacity to properly balance the exploration of the solutions and the exploitation of found-so-far elements. Furthermore, many EAs suffer from some common flaws, such as premature convergence and an inherent difficulty to overcome local optima [5, 6]. These issues usually arise from the operators used to update the position of search agents. In the case of PSO, for example, search agents are usually attracted towards the position of the current best individual which inherently causes the entire population to concentrate around the best particle seen-so-far, and thus, favoring premature convergence [7]. Also, many EAs, such as PSO, FA and CS, only model search agents with equal properties, and thus, restricting them to perform virtually the same behavior. Under these circumstances, these methods waste the possibility to add new and selective operators which could potentially improve some important traits, such as population diversity and search capabilities.

Different to other EA, the SHO algorithm models each individual of the entire population by first considering its role as either a prey belonging to a selfish herd or a hungry predator. Furthermore, individuals within such selfish herd manifest unique individual behaviors which depends not only on the aptitudes of better individuals, but also on their own aptitude with regard to a given optimization problem. This allows SHO to incorporate computational mechanisms that allow the avoidance of critical flaws commonly found in other EAs, such as premature convergence and inadequate balance between exploration and exploitation of solutions. From an optimization point of view, the use of selfish herd behaviors as a metaphor provide some interesting concepts to EAs: First of all, the division of the entire population into different search-agent categories and the implementation of selective and specialized operators allows SHO to improve the balance between exploration and exploitation without altering the total population size. Furthermore, the selfish herd behavior introduces an interesting computational scheme with three distinctive traits: 1. individuals are separately processed according to their classification as either prey or predators, with prey further manifesting a unique internal social structure; 2. although operators employed to modify the position of each individual differs depending of its type (as either a prey or a predator), they all use global information (such as the positions and aptitudes of other individuals); 3. the predation and restoration mechanisms allows the exclusion of potentially bad or redundant solutions, while at the same time enables the diversification of the whole set of solutions.

3.6 Comparative Experiments and Results

The SHO algorithm has been applied in the optimization of 15 benchmark functions collected from [21–23], and whose results have also been compared against those produced by Particle Swarm Optimization (PSO) [1], Artificial Bee Colony (ABC) [2], Firefly Algorithm (FA) [3], Differential Evolution (DE) [21], Genetic Algorithms (GA) [24], Crow Search Algorithm (CSA) [25], Dragonfly Algorithm (DA) [26], Moth-flame Optimization Algorithm (MOA) [27], and Sine Cosine Algorithm (SCA) [28]. For all comparisons, the population size is set to $N = 50$ individuals, while the maximum iteration number is set to $T = 1000$. Such stop criterion has been selected to keep consistency with other similar works currently reported on the literature [29, 30]. A detailed description of each implemented test functions is given in Table 3.2.

The parameter settings for each of the compared methods is as follows:

Table 3.2 Test functions used for our experiments

Name	Function	S	n	Minimum
Ackley	$f_1(\mathbf{x}) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + \exp$	$[-32.8, 32.8]^n$	30	$f_1(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots, 0)$
Sphere	$f_2(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	30	$f_2(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots, 0)$
Sum of Squares	$f_3(\mathbf{x}) = \sum_{i=1}^{n} i x_i^2$	$[-10, 10]^n$	30	$f_3(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots, 0)$
Powell	$f_4(\mathbf{x}) = \sum_{i=1}^{n/4}[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i} - x_{4i})^4]$	$[-4, 5]^n$	30	$f_4(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots 0)$
Levy	$f_5(\mathbf{x}) = \cos^2(\pi w_1) + \sum_{i=1}^{n-1}(w_i - 1)^2[1 + 10\sin \pi w_i + 1] + (w_n - 1)^2\left[1 + \sin^2 2\pi w_n\right]$ $w_i = 1 + \left(\frac{x_i+1}{4}\right)$	$[-10, 10]^n$	30	$f_5(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (1, \ldots, 1)$
Rosenbrock	$f_6(\mathbf{x}) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	$[-5, 10]^n$	30	$f_6(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (1, \ldots, 1)$
Schwefel 2	$f_7(\mathbf{x}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_i\right)^2$	$[-100, 100]^n$	30	$f_7(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots, 0)$

(continued)

Table 3.2 (continued)

Name	Function	S	n	Minimum
Schwefel 26	$f_8(\mathbf{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]^n$	30	$f_8(\mathbf{x}^*) = 0;$ $\mathbf{x}^* = (420.968, \ldots, 420.968)$
Trid	$f_9(\mathbf{x}) = \sum_{i=1}^{n} (x_i - 1)^2 - \sum_{i=2}^{n} x_i (x_{i-1})$	$\left[-n^2, n^2\right]^n$	30	$f_9(\mathbf{x}^*) = -4500$
Dixon and Price	$f_{10}(\mathbf{x}) = (x_i - 1)^2 + \sum_{i=1}^{n} i\left(2x_i^2 - x_{i-1}\right)^2$	$[-10, 10]^n$	30	$f_{10}(x_i) = 0$ $x_i = 2^{-\frac{2^i-2}{2^i}}$ for $i = 1, \ldots, n$
Rotated Hyper-Ellipsoid	$f_{11}(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{i} x_j^2$	$[-65.5, 65.5]^n$	30	$f_{11}(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots, 0)$
Zakharov	$f_{12}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^2 + \left(\sum_{n=1}^{n} 0.5ix_i\right)^4$	$[-5, 10]^n$	30	$f_{12}(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots, 0)$
Quartic	$f_{13}(\mathbf{x}) = \sum_{i=1}^{n} \left[(ix_i)^4 + rand\right]$	$[-1.28, 1.28]^n$	30	$f_{13}(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0.5, \ldots, 0.5)$
Salomon	$f_{14}(\mathbf{x}) = -\cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2 + 1}$	$[-100, 100]^n$	30	$f_{14}(\mathbf{x}^*) = 0;\ \mathbf{x}^* = (0, \ldots, 0)$
Qing	$f_{15}(\mathbf{x}) = \sum_{i=1}^{n} \left(x_i^2 - i\right)^2$	$[-500, 500]^n$	30	$f_{15}(\mathbf{x}^*) = 0;$ $\mathbf{x}^* = \left(\pm\sqrt{1}, \pm\sqrt{2}, \ldots, \pm\sqrt{n}\right)$

In the table, **S** indicates the subset of $\mathbb{R}^n$ which comprises the function's search space and n indicates the function's dimension. Also, the value $f_i(\mathbf{x}^*)$ indicates the optimum value of each function, while $\mathbf{x}^*$ indicates the optimum position

1. PSO: The algorithm's learning factors are set to $c_1 = 2$ and $c_2 = 2$; also, the inertia weight w is set to decreases linearly from 0.9 to 0.2 as the process evolves.
2. ABC: The algorithm was implemented by setting the parameter limit $=$ numOfFoodSources*dims, where numOfFoodSources $= N$ (population size) and dims $= n$ (dimensionality of the solution space).
3. FA: The parameters setup for the randomness factor and the light absorption coefficient are set to $\alpha = 0.2$ and $\gamma = 1.0$, respectively.
4. DE: The algorithm's differential weight is set to $\mathrm{F} = 1$ while the crossover probability is set to $\mathrm{CR} = 0.2$.
5. GA: The crossover and mutation probabilities are both set to $c_p = 0.8$ and $m_p = 0.2$ respectively.
6. CSA: The awareness probability and flight length are set to $\mathrm{AP} = 0.1$ and $\mathrm{fl} = 2$, respectively.
7. DA: The parameters are set as: $\mathrm{w} = 0.9$ (inertia weight), $\mathrm{s} = 0.1$ (separation weight), $\mathrm{a} = 0.1$ (alignment weight), $\mathrm{c} = 0.7$ (cohesion weight), $\mathrm{f} = 1.0$ (food factor) and $\mathrm{e} = 1.0$ (enemy factor).
8. MOA: The constant value used to model the moths' logarithmic spiral movement is set to $\mathrm{b} = 1$.
9. SCA: The constant value used to generate the random value r_1 is set to $\mathrm{a} = 2$.
10. SHO: The proposed method is tested by considering a herd population proportion randomly chosen from between a 70 and 90%, with the remaining individuals assigned as predators.

The previously illustrated sets of parameters were determined through exhaustive experimentation and as such represent the best possible configurations for each of the compared methods [16].

The experimental setup aims to compare SHO's performance against those of PSO, ABC, FA, DE and GA. The reported results consider the following performance indexes: the Average Best-so-far (AB) solution, the Median Best-so-far (MB) and the Standard Deviation (SD) of the best-so-far solution. The averaged results corresponding to 30 individual runs are reported in Table 3.3, where the best outcome for each function is boldfaced. According to this table, for most cases, SHO's performance over the considered test functions is superior to those of the other compared methods. Such large difference in performance is intuitively related to a better trade-off between exploration and exploitation (Fig. 3.12).

Also, the non-parametric statistical significance proof known as the Wilcoxon's rank sum test for independent samples [31, 32] was conducted over the best fitness values found by each of the compared method on 30 independent test runs (30 samples per set). Table 3.4 reports the p-values produced by the Wilcoxon's test for the pair-wise comparison over two independent fitness samples (SHO vs. PSO, ABC, FA, GA, CSA, DA, MOA and SCA), by considering a 5% significance level. As a null hypothesis, it is assumed that there is a significant difference between mean values of two algorithms. On the other hand, the alternative hypothesis (rejection of the null hypothesis) considers that the difference between the mean values of both

Table 3.3 Minimization results for the benchmark functions listed in Table 3.2

		SHO	PSO	ABC	FA	DE	GA	CSA
$f_1(\boldsymbol{x})$	AB	**3.30×10^{-13}**	1.40×10^{-04}	2.90×10^{-01}	5.30×10^{00}	5.66×10^{-06}	1.16×10^{00}	3.25×10^{00}
	MB	**7.50×10^{-10}**	1.70×10^{01}	1.20×10^{00}	6.80×10^{00}	5.41×10^{-06}	1.34×10^{00}	3.35×10^{00}
	SD	**2.80×10^{-09}**	6.80×10^{00}	3.20×10^{-01}	7.10×10^{-01}	1.62×10^{-06}	8.01×10^{-01}	6.31×10^{-01}
$f_2(\boldsymbol{x})$	AB	**1.00×10^{-13}**	7.50×10^{-11}	1.70×10^{-05}	1.70×10^{-01}	1.27×10^{-13}	6.12×10^{-04}	4.07×10^{-05}
	MB	2.50×10^{-13}	1.70×10^{00}	1.40×10^{-04}	5.00×10^{-01}	**1.06×10^{-13}**	2.71×10^{-04}	3.53×10^{-05}
	SD	**1.40×10^{-14}**	6.70×10^{00}	7.30×10^{-05}	2.70×10^{-01}	5.70×10^{-14}	8.89×10^{-04}	2.37×10^{-05}
$f_3(\boldsymbol{x})$	AB	**7.20×10^{-12}**	3.30×10^{-08}	2.20×10^{-03}	4.40×10^{00}	7.50×10^{-12}	7.30×10^{-02}	3.15×10^{-01}
	MB	**1.90×10^{-12}**	6.10×10^{02}	9.00×10^{-03}	1.90×10^{01}	6.58×10^{-12}	2.18×10^{-02}	2.64×10^{-01}
	SD	8.60×10^{-12}	5.00×10^{02}	6.10×10^{-03}	8.50×10^{00}	**3.36×10^{-12}**	1.21×10^{-01}	2.27×10^{-01}
$f_4(\boldsymbol{x})$	AB	**1.20×10^{-03}**	1.70×10^{02}	5.60×10^{-01}	3.00×10^{00}	8.05×10^{00}	7.30×10^{-02}	6.25×10^{-03}
	MB	**2.90×10^{-03}**	1.70×10^{03}	3.40×10^{00}	1.50×10^{01}	7.19×10^{00}	2.18×10^{-02}	6.20×10^{-03}
	SD	**1.10×10^{-03}**	1.80×10^{03}	1.90×10^{00}	7.30×10^{00}	4.83×10^{00}	1.21×10^{-01}	3.01×10^{-03}
$f_5(\boldsymbol{x})$	AB	**1.10×10^{-13}**	9.40×10^{-07}	3.60×10^{-04}	7.40×10^{-01}	1.91×10^{-12}	4.19×10^{-01}	3.36×10^{-01}
	MB	**3.40×10^{-13}**	5.60×10^{00}	1.00×10^{-03}	1.70×10^{00}	1.76×10^{-12}	4.54×10^{-01}	3.20×10^{-01}
	SD	**2.90×10^{-13}**	6.20×10^{00}	5.90×10^{-04}	6.40×10^{-01}	9.60×10^{-13}	4.24×10^{-01}	1.53×10^{-01}
$f_6(\boldsymbol{x})$	AB	**1.80×10^{00}**	7.30×10^{01}	2.20×10^{01}	5.70×10^{02}	2.99×10^{02}	7.05×10^{01}	3.10×10^{01}
	MB	**4.30×10^{01}**	2.00×10^{05}	6.80×10^{01}	2.40×10^{03}	2.65×10^{02}	7.18×10^{01}	2.89×10^{01}
	SD	3.60×10^{01}	1.00×10^{05}	**3.10×10^{01}**	1.60×10^{03}	1.19×10^{02}	4.46×10^{01}	1.17×10^{01}
$f_7(\boldsymbol{x})$	AB	**1.60×10^{-09}**	1.00×10^{04}	2.30×10^{00}	9.70×10^{04}	9.52×10^{-09}	2.37×10^{00}	2.14×10^{02}
	MB	**7.40×10^{-09}**	1.50×10^{06}	1.40×10^{01}	2.60×10^{05}	8.93×10^{-09}	9.74×10^{-01}	2.07×10^{02}
	SD	**4.90×10^{-09}**	1.20×10^{06}	8.60×10^{00}	9.80×10^{04}	5.30×10^{-09}	3.97×10^{00}	9.20×10^{01}
$f_8(\boldsymbol{x})$	AB	**3.80×10^{-04}**	2.50×10^{03}	6.60×10^{02}	9.20×10^{03}	3.80×10^{02}	1.20×10^{04}	1.25×10^{04}
	MB	6.60×10^{02}	4.80×10^{03}	1.10×10^{03}	1.00×10^{04}	**9.31×10^{00}**	1.20×10^{04}	1.25×10^{04}
	SD	4.40×10^{02}	8.10×10^{02}	2.30×10^{02}	3.40×10^{02}	6.52×10^{02}	2.90×10^{01}	2.56×10^{00}

(continued)

Table 3.3 (continued)

		SHO	PSO	ABC	FA	DE	GA	CSA
$f_9(\boldsymbol{x})$	AB	**-4.50×10^{03}**	1.10×10^{05}	5.30×10^{03}	4.30×10^{04}	4.65×10^{05}	-1.53×10^{03}	-1.39×10^{03}
	MB	**-2.60×10^{03}**	9.00×10^{05}	1.60×10^{04}	7.80×10^{04}	4.60×10^{05}	-1.53×10^{03}	-1.34×10^{03}
	SD	1.10×10^{03}	5.10×10^{05}	6.40×10^{03}	2.80×10^{04}	2.13×10^{05}	**2.21×10^{01}**	3.89×10^{02}
$f_{10}(\boldsymbol{x})$	AB	**6.70×10^{-01}**	6.70×10^{-01}	2.60×10^{00}	1.00×10^{01}	6.73×10^{00}	4.58×10^{00}	7.30×10^{01}
	MB	**6.80×10^{-01}**	1.40×10^{04}	6.80×10^{00}	4.00×10^{01}	6.67×10^{00}	4.47×10^{00}	3.03×10^{01}
	SD	**6.50×10^{-02}**	3.30×10^{04}	2.20×10^{00}	4.20×10^{01}	1.64×10^{-01}	2.83×10^{00}	9.83×10^{01}
$f_{11}(\boldsymbol{x})$	AB	**2.40×10^{-10}**	9.00×10^{-07}	1.30×10^{-01}	3.00×10^{03}	2.94×10^{-10}	4.50×10^{-02}	7.36×10^{02}
	MB	**1.00×10^{-10}**	3.70×10^{04}	3.10×10^{-01}	6.70×10^{03}	2.70×10^{-10}	1.94×10^{-02}	4.74×10^{02}
	SD	**6.90×10^{-11}**	3.10×10^{04}	1.80×10^{-01}	2.10×10^{03}	1.30×10^{-10}	7.57×10^{-02}	6.33×10^{02}
$f_{12}(\boldsymbol{x})$	AB	**1.20×10^{01}**	1.50×10^{02}	2.30×10^{02}	7.30×10^{01}	1.63×10^{03}	2.80×10^{02}	3.89×10^{00}
	MB	**2.10×10^{01}**	4.80×10^{02}	2.90×10^{02}	4.10×10^{03}	1.63×10^{03}	3.17×10^{02}	3.49×10^{00}
	SD	**5.70×10^{00}**	1.60×10^{02}	2.90×10^{01}	2.20×10^{04}	2.75×10^{02}	1.63×10^{02}	1.97×10^{00}
$f_{13}(x)$	AB	**8.30×10^{00}**	9.90×10^{00}	1.10×10^{01}	1.40×10^{01}	1.05×10^{02}	1.58×10^{01}	1.16×10^{01}
	MB	**9.30×10^{00}**	1.40×10^{01}	1.20×10^{01}	2.00×10^{01}	1.05×10^{02}	1.54×10^{01}	1.14×10^{01}
	SD	**5.40×10^{-01}**	4.50×10^{00}	6.20×10^{-01}	3.50×10^{00}	3.61×10^{00}	2.18×10^{00}	8.52×10^{-01}
$f_{14}(\boldsymbol{x})$	AB	**1.00×10^{-01}**	5.00×10^{-01}	3.10×10^{00}	2.30×10^{00}	4.11×10^{00}	5.67×10^{-01}	1.66×10^{00}
	MB	**1.20×10^{-01}**	1.70×10^{00}	4.00×10^{00}	3.60×10^{00}	4.01×10^{00}	6.00×10^{-01}	1.65×10^{00}
	SD	**4.10×10^{-02}**	2.90×10^{00}	4.80×10^{-01}	6.10×10^{-01}	3.01×10^{-01}	8.02×10^{-02}	3.19×10^{-01}
$f_{15}(\boldsymbol{x})$	AB	**2.40×10^{-04}**	7.60×10^{-04}	1.87×10^{01}	7.60×10^{06}	8.97×10^{02}	9.20×10^{-02}	1.61×10^{01}
	MB	**6.10×10^{-04}**	1.40×10^{-01}	5.94×10^{01}	6.20×10^{07}	9.03×10^{02}	3.89×10^{-02}	1.14×10^{01}
	SD	**5.60×10^{-04}**	3.00×10^{-01}	2.08×10^{01}	3.60×10^{07}	2.81×10^{02}	1.78×10^{-01}	1.28×10^{01}

(continued)

Table 3.3 (continued)

DF	MOA	SCA
6.36×10^{00}	1.88×10^{01}	1.12×10^{01}
5.67×10^{00}	1.91×10^{01}	1.84×10^{01}
1.83×10^{00}	1.20×10^{00}	9.61×10^{00}
9.96×10^{-01}	1.48×10^{01}	3.81×10^{-01}
8.96×10^{-01}	3.90×10^{-01}	6.42×10^{-02}
7.58×10^{-01}	1.77×10^{01}	9.01×10^{-01}
6.12×10^{01}	2.37×10^{03}	1.74×10^{01}
3.79×10^{01}	2.01×10^{03}	3.98×10^{00}
5.46×10^{01}	2.25×10^{03}	3.63×10^{01}
1.02×10^{00}	2.13×10^{01}	3.56×10^{-01}
7.22×10^{-01}	1.72×10^{01}	8.76×10^{-02}
1.05×10^{00}	1.90×10^{01}	4.96×10^{-01}
9.42×10^{-01}	2.25×10^{01}	4.60×10^{00}
8.46×10^{-01}	2.08×10^{01}	4.45×10^{00}
4.67×10^{-01}	7.49×10^{00}	6.81×10^{-01}
1.08×10^{03}	1.60×10^{05}	8.69×10^{03}
1.04×10^{03}	8.24×10^{02}	5.12×10^{03}
9.04×10^{02}	3.57×10^{05}	1.34×10^{04}
2.25×10^{03}	8.85×10^{04}	3.95×10^{01}
2.09×10^{03}	8.61×10^{04}	1.17×10^{01}
1.47×10^{03}	5.03×10^{04}	6.11×10^{01}
1.25×10^{04}	2.08×10^{04}	2.09×10^{04}
1.25×10^{04}	2.08×10^{04}	2.09×10^{04}
$\mathbf{0.00 \times 10^{00}}$	3.70×10^{-12}	4.13×10^{00}

(continued)

Table 3.3 (continued)

DF	MOA	SCA
5.23×10^{02}	5.08×10^{03}	4.60×10^{02}
1.69×10^{02}	2.82×10^{03}	2.44×10^{02}
1.26×10^{03}	8.32×10^{03}	6.19×10^{02}
2.49×10^{06}	1.67×10^{09}	3.70×10^{07}
4.72×10^{05}	7.92×10^{08}	1.97×10^{07}
5.42×10^{06}	2.74×10^{09}	4.84×10^{07}
2.02×10^{05}	8.02×10^{06}	3.67×10^{04}
1.62×10^{05}	7.25×10^{06}	5.52×10^{03}
1.91×10^{05}	5.09×10^{06}	6.45×10^{04}
1.33×10^{03}	1.31×10^{04}	2.56×10^{03}
9.72×10^{02}	1.25×10^{04}	2.43×10^{03}
1.19×10^{03}	3.35×10^{03}	7.76×10^{02}
3.30×10^{03}	4.84×10^{06}	1.38×10^{05}
2.06×10^{03}	2.43×10^{06}	2.81×10^{04}
4.75×10^{03}	6.11×10^{06}	2.05×10^{05}
1.86×10^{01}	7.08×10^{01}	8.20×10^{00}
1.90×10^{01}	8.83×10^{01}	6.76×10^{00}
8.04×10^{00}	4.02×10^{01}	6.48×10^{00}
4.22×10^{02}	1.08×10^{03}	1.92×10^{04}
3.76×10^{02}	2.18×10^{01}	1.97×10^{04}
2.76×10^{02}	1.68×10^{03}	1.69×10^{03}

Results were averaged from 30 individual runs, each by considering population size $N = 50$ and maximum number of iterations $T = 1000$

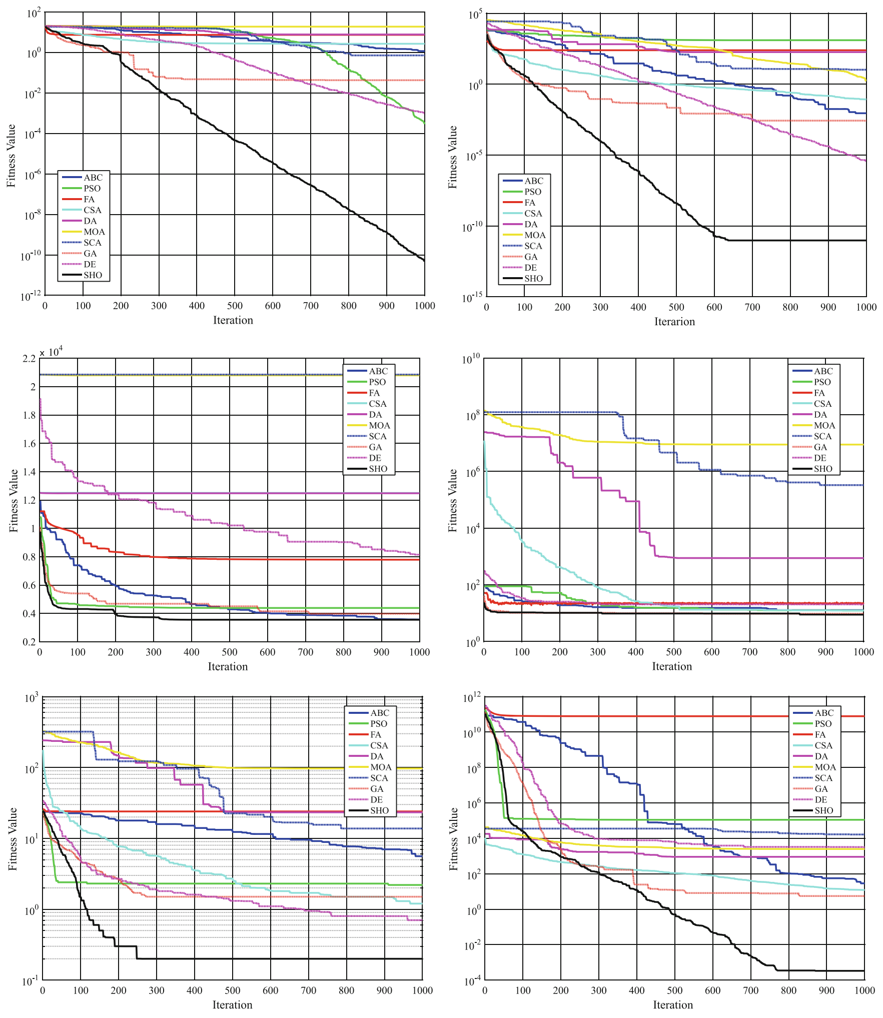


Fig. 3.12 Evolution curves for SHO, PSO, ABC, FA, DE, GA, CSA, DA, MOA and SCA considering as examples the functions **a** f_1, **b** f_3, **c** f_8, **d** f_{13}, **e** f_{14} and **f** f_{15} from the experimental set (see Table 3.4)

approaches is insignificant. As shown by all of the p-values on said table, there is enough evidence to reject the null hypothesis (this is that all values are less than 0.05, and as such satisfy the 5% significance level criteria). Such evidence indicates that the proposed method's results are statistically significant and that they had not occurred by coincidence (i.e. due to common noise contained in the process).

Table 3.4 Wilcoxon's test comparison for SHO versus PSO, ABC, FA, GA, CSA, DA, MOA and SCA

Function	SHO versus PSO	SHO versus ABC	SHO versus FA	SHO versus DE	SHO versus GA	SHO versus CSA	SHO versus DA	SHO versus MOA	SHO versus SCA
$f_1(\boldsymbol{x})$	5.22×10^{-12}	3.01×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_2(\boldsymbol{x})$	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	5.86×10^{-06}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_3(\boldsymbol{x})$	2.89×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	6.52×10^{-09}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_4(\boldsymbol{x})$	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	5.86×10^{-06}	3.02×10^{-11}	3.02×10^{-11}	3.82×10^{-09}
$f_5(\boldsymbol{x})$	3.00×10^{-11}	3.01×10^{-11}	3.01×10^{-11}	1.09×10^{-10}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_6(\boldsymbol{x})$	6.68×10^{-11}	4.28×10^{-02}	3.02×10^{-11}	3.02×10^{-11}	2.32×10^{-02}	8.42×10^{-01}	6.72×10^{-10}	3.02×10^{-11}	7.38×10^{-11}
$f_7(\boldsymbol{x})$	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	4.08×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_8(\boldsymbol{x})$	2.96×10^{-11}	7.71×10^{-06}	2.99×10^{-11}	2.50×10^{-03}	3.01×10^{-11}	3.01×10^{-11}	1.21×10^{-12}	1.21×10^{-12}	3.01×10^{-11}
$f_9(\boldsymbol{x})$	3.01×10^{-11}	3.01×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	1.29×10^{-06}	4.61×10^{-10}	2.00×10^{-05}	4.20×10^{-10}
$f_{10}(\boldsymbol{x})$	1.01×10^{-09}	4.11×10^{-12}	4.11×10^{-12}	4.11×10^{-12}	4.17×10^{-08}	4.11×10^{-12}	4.58×10^{-12}	4.11×10^{-12}	4.11×10^{-12}

(continued)

Table 3.4 (continued)

Function	SHO versus PSO	SHO versus ABC	SHO versus FA	SHO versus DE	SHO versus GA	SHO versus CSA	SHO versus DA	SHO versus MOA	SHO versus SCA
$f_{11}(\boldsymbol{x})$	2.97×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	1.29×10^{-09}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_{12}(\boldsymbol{x})$	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	1.11×10^{-06}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_{13}(x)$	9.92×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}
$f_{14}(\boldsymbol{x})$	2.70×10^{-11}	2.71×10^{-11}	2.71×10^{-11}	2.71×10^{-11}	2.67×10^{-11}	2.71×10^{-11}	5.07×10^{-10}	2.71×10^{-11}	2.71×10^{-11}
$f_{15}(\boldsymbol{x})$	8.15×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	4.08×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}	3.02×10^{-11}

The table shows the resulting p-values for each pair-wise comparison

3.7 Conclusions

In this chapter, a novel swarm optimization algorithm called Selfish Herd Optimizer (SHO), developed to solve global optimization problems, was presented. SHO is based on the widely observed selfish herd behavior, manifested as a result of the predation risk inherently related to most prey-predator interactions. As such, the proposed method considers two types of search agents: the members of a selfish herd (prey) and the individuals which hunt for the individuals in such aggregation (predators). Depending on their designation (and its internal social structure in the case of the members of the herd), each individual moves around the solution space of a given optimization problem by considering a set of distinctive evolutionary operators, which mimic the different kind of behaviors manifested in said predatory interactions. In contrast to most existing swarm optimization algorithms, the SHO allows not only to emulate such interesting selfish behaviors, but also to incorporate computational mechanisms devised to avoid critical flaws commonly found on other similar methods, such as premature convergence and the inappropriate balance between the exploration and exploitation of solutions. The performance of SHO has been analyzed through a series experiments involving a set of 15 different benchmark optimization functions commonly cited on the literature. Furthermore, the performance of SHO was also compared against popular methods, such as Particle Swarm Optimization (PSO) Artificial Bee Colony (ABC, Firefly Algorithm (FA), Differential Evolution (DE), Genetic Algorithms (GA), Crow Search Algorithm (CSA), Dragonfly Algorithm (DA), Moth-flame Optimization Algorithm (MOA) and Sine Cosine Algorithm (SCA). The experimental data obtained though such exhaustive experimentation demonstrates that SHO has a high performance in terms of solution quality. Such remarkable performance is associated with two different traits: 1. the use of operators which allow a better distribution within the search space, 2. the division of the population into different individual types which enable the proposed method to adopt different exploration and exploitation rates during the evolutionary process, and 3. The integration of unique predation based operators which allows the diversification of solutions.

References

1. Kennedy, J., Eberhart, R.: Particle swarm optimization. IEEE Int. Conf. Neural Netw. **4**, 1942–1948 (1995)
2. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Appl. Soft Comput. J. **8**(1), 687–697 (2008)
3. Yang, X.: Firefly algorithm, Lévy flights and global optimization (2010)
4. Rajabioun, R.: Cuckoo optimization algorithm. Appl. Soft Comput. J. **11**(8), 5508–5518 (2011)
5. Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B., Tian, Q.: Self-adaptive learning based particle swarm optimization. Inf. Sci. (Ny) **181**(20), 4515–4538 (2011)
6. Xiang, W., An, M.: Computers & operations research an efficient and robust artificial bee colony algorithm for numerical optimization. Comput. Oper. Res. **40**(5), 1256–1265 (2013)

7. Wang, H., Sun, H., Li, C., Rahnamayan, S., Pan, J.: Diversity enhanced particle swarm optimization with neighborhood search. Inf. Sci. (Ny) **223**, 119–135 (2013)
8. Banharnsakun, A., Achalakul, T., Sirinaovakul, B.: The best-so-far selection in artificial bee colony algorithm. Appl. Soft Comput. J. **11**(2), 2888–2901 (2011)
9. Hamilton, W.D.: Geometry for the selfish herd. J. Theor. Biol. **31**(2), 295–311 (1971)
10. Morrell, L.J., Ruxton, G.D., James, R.: Spatial positioning in the selfish herd. Behav. Ecol. 16–22 (2010)
11. Eshel, I., Sansone, E., Shaked, A.: On the evolution of group-escape strategies of selfish prey. Theor. Popul. Biol. **80**(2), 150–157 (2011)
12. Viscido, S.V., Wethey, D.S.: Quantitative analysis of fiddler crab flock movement: evidence for 'selfish herd' behaviour. Anim. Behav. **63**(4), 735–741 (2002)
13. Orpwood, J.E., Magurran, A.E., Armstrong, J.D., Griffiths, S.W.: Minnows and the selfish herd: effects of predation risk on shoaling behaviour are dependent on habitat complexity. Anim. Behav. **76**(1), 143–152 (2008)
14. Alcock, J.: Animal Behavior: An Evolutionary Approach. Sinauer Associates Inc., Sunderland, MA (2001)
15. Mcclure, M., Despland, E.: Collective foraging patterns of field colonies of Malacosoma disstria caterpillars. Entomol. Soc. Canada2 **142**(5), 473–480 (2010)
16. Fausto, F., Cuevas, E., Valdivia, A., González, A.: A global optimization algorithm inspired in the behavior of selfish herds. BioSystems **160**, 39–55 (2017)
17. Viscido, S.V., Miller, M., Wethey, D.S.: The dilemma of the selfish herd: the search for a realistic movement rule. J. Theor. Biol. **217**(2), 183–194 (2002)
18. Reluga, T.C., Viscido, S.: Simulated evolution of selfish herd behavior. J. Theor. Biol. **234**(2), 213–225 (2005)
19. Thomas, B.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, Inc. (1996)
20. Voltera, V.: Variations and fluctuations of the number of individuals in animal species libing together. In: Chapman, R.N. (ed.) Animal Ecology. McGraw-Hill (1931)
21. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
22. Yang, X.: Nature-Inspired Metaheuristic Algorithms, 2nd edn. Luniver Press, Beckington, UK (2008)
23. Karaboga, D., Akay, B.: A comparative study of artificial bee colony algorithm. Appl. Math. Comput. **214**(1), 108–132 (2009)
24. Mitchell, M.: An Introduction to Genetic Algorithms. The MIT Press, Cambridge, MA (1996)
25. Askarzadeh, A.: A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput. Struct. **169**, 1–12 (2016)
26. Mirjalili, S.: Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput. Appl. **27**(4), 1053–1073 (2015)
27. Mirjalili, S.: Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. Knowl.-Based Syst. **89**, 228–249 (2015)
28. Mirjalili, S.: SCA: a sine cosine algorithm for solving optimization problems. Knowl.-Based Syst. **96**, 120–133 (2016)
29. Ji, Y., Zhang, K., Qu, S.: A deterministic global optimization algorithm. Appl. Math. Comput. **185**(1), 382–387 (2007)
30. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: a gravitational search algorithm. Inf. Sci. (Ny) **179**(13), 2232–2248 (2009)
31. Wilcoxon, F.: Individual comparisons by ranking methods Frank Wilcoxon. Biometrics Bull. **1**(6), 80–83 (2006)
32. García, S., Molina, D., Lozano, M.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour : a case study on the CEC'2005 special session on real parameter optimization. 617–644 (2009)

Chapter 4
The Swarm Method of the Social-Spider

Abstract Swarm intelligence is a computer science field which emulates the cooperative behavior of natural systems such as insects or animals. Many methods resulting from these models have been suggested to solve several complex optimization problems. In this chapter, a metaheuristic approach known as the Social Spider Optimization (SSO) is analyzed for solving optimization problems. The SSO method considers the simulation of the collective operation of social-spiders. In SSO, candidate solutions represent a set of spiders which interacts among them based on the natural laws of the colony. The algorithm examines two different kinds of search agents (spiders): males and females. According to the gender, each element is conducted by a set of different operations which imitate different behaviors that are commonly observed in the colony.

4.1 Introduction

The collective intelligent behavior of insect or animal groups in nature such as flocks of birds, colonies of ants, schools of fish, swarms of bees and termites have attracted the attention of researchers. The aggregative conduct of insects or animals is known as swarm behavior. Entomologists have studied this collective phenomenon to model biological groups in nature while engineers have applied these models as a framework for solving complex real-world problems. This branch of artificial intelligence which deals with the collective behavior of elements through complex interaction of individuals with no supervision is frequently addressed as swarm intelligence. Bonabeau defined swarm intelligence as “any attempt to design algorithms or distributed problem solving devices inspired by the collective behavior of the social insect colonies and other animal societies” [1]. Swarm intelligence has some advantages such as scalability, fault tolerance, adaptation, speed, modularity, autonomy and parallelism [2].

The key components of swarm intelligence are self-organization and labor division. In a self-organizing system, each of the covered units responds to local stimuli individually and may act together to accomplish a global task, via a labor separation

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_4

which avoids a centralized supervision. The entire system can thus efficiently adapt to internal and external changes.

Several metaheuristic algorithms have been developed by a combination of deterministic rules and randomness, mimicking the behavior of insect or animal groups in nature. Such methods include the social behavior of bird flocking and fish schooling such as the Particle Swarm Optimization (PSO) algorithm [3], the cooperative behavior of bee colonies such as the Artificial Bee Colony (ABC) technique [4], the social foraging behavior of bacteria such as the Bacterial Foraging Optimization Algorithm (BFOA) [5], the simulation of the herding behavior of krill individuals such as the Krill Herd (KH) method [6], the mating behavior of firefly insects such as the Firefly (FF) method [7] and the emulation of the lifestyle of cuckoo birds such as the Cuckoo Optimization Algorithm (COA) [8].

In particular, insect colonies and animal groups provide a rich set of metaphors for designing metaheuristic optimization algorithms. Such cooperative entities are complex systems that are composed by individuals with different cooperative-tasks where each member tends to reproduce specialized behaviors depending on its gender [9]. However, most of metaheuristic algorithms model individuals as unisex entities that perform virtually the same behavior. Under such circumstances, algorithms waste the possibility of adding new and selective operators as a result of considering individuals with different characteristics such as sex, task-responsibility, etc. These operators could incorporate computational mechanisms to improve several important algorithm characteristics including population diversity and searching capacities.

Although PSO and ABC are the most popular metaheuristic algorithms for solving complex optimization problems, they present serious flaws such as premature convergence and difficulty to overcome local minima [10, 11]. The cause for such problems is associated to the operators that modify individual positions. In such algorithms, during their evolution, the position of each agent for the next iteration is updated yielding an attraction towards the position of the best particle seen so-far (in case of PSO) or towards other randomly chosen individuals (in case of ABC). As the algorithm evolves, those behaviors cause that the entire population concentrates around the best particle or diverges without control. It does favors the premature convergence or damage the exploration-exploitation balance [12, 13].

The interesting and exotic collective behavior of social insects have fascinated and attracted researchers for many years. The collaborative swarming behavior observed in these groups provides survival advantages, where insect aggregations of relatively simple and “unintelligent” individuals can accomplish very complex tasks using only limited local information and simple rules of behavior [14]. Social-spiders are a representative example of social insects [15]. A social-spider is a spider species whose members maintain a set of complex cooperative behaviors [16]. Whereas most spiders are solitary and even aggressive toward other members of their own species, social-spiders show a tendency to live in groups, forming long-lasting aggregations often referred to as colonies [17]. In a social-spider colony, each member, depending on its gender, executes a variety of tasks such as predation, mating, web design, and social interaction [17, 18]. The web it is an important part of the colony because it is not only used as a common environment for all members, but also as a communication channel among them [19]. Therefore, important information (such as trapped prays or mating possibilities) is transmitted by small vibrations through the web. Such information, considered as a local knowledge, is employed by each member to conduct its own cooperative behavior, influencing simultaneously the social regulation of the colony [20].

In this chapter, a metaheuristic algorithm, called the Social Spider Optimization (SSO) is analyzed for solving optimization tasks. The SSO algorithm is based on the simulation of the cooperative behavior of social-spiders. In this algorithm, individuals emulate a group of spiders which interact to each other based on the biological laws of the cooperative colony. The algorithm considers two different search agents (spiders): males and females. Depending on gender, each individual is conducted by a set of different swarm operators which mimic different cooperative behaviors that are typical in a colony. Different to most of existent metaheuristic algorithms, in the approach, each individual is modeled considering two genders. Such fact allows not only to emulate in a better realistic way the cooperative behavior of the colony, but also to incorporate computational mechanisms to avoid critical flaws commonly present in the popular PSO and ABC algorithms, such as the premature convergence and the incorrect exploration-exploitation balance. In order to illustrate the proficiency and robustness of the approach, it is compared to other well-known swarm methods. The comparison examines several standard benchmark functions which are commonly considered in the literature. The results show a high performance of the method for searching a global optimum in several benchmark functions.

This chapter is organized as follows. In Sect. 4.2, we introduce basic biological aspects of the algorithm. In Sect. 4.3, the novel SSO algorithm and its characteristics are both described. Section 4.4 presents the experimental results and the comparative study. Finally, in Sect. 4.5, conclusions are drawn.

4.2 Biological Concepts

Social insect societies are complex cooperative systems that self-organize within a set of constraints. Cooperative groups are better at manipulating and exploiting their environment, defending resources and brood, and allowing task specialization among group members [21, 22]. A social insect colony functions as an integrated unit that not only possesses the ability to operate at a distributed manner, but also to undertake enormous construction of global projects [23]. It is important to acknowledge that global order in social insects can arise as a result of internal interactions among members.

A few species of spiders have been documented exhibiting a degree of social behavior [15]. The behavior of spiders can be generalized into two basic forms: solitary spiders and social spiders [17]. This classification is made based on the level of cooperative behavior that they exhibit [18]. In one side, solitary spiders create and maintain their own web while live in scarce contact to other individuals of the same species. In contrast, social spiders form colonies that remain together over a communal web with close spatial relationship to other group members [19].

A social spider colony is composed of two fundamental components: its members and the communal web. Members are divided into two different categories: males and females. An interesting characteristic of social-spiders is the highly female-biased population. Some studies suggest that the number of male spiders barely reaches the 30% of the total colony members [17, 24]. In the colony, each member, depending on its gender, cooperate in different activities such as building and maintaining the communal web, prey capturing, mating and social contact [20]. Interactions among members are either direct or indirect [25]. Direct interactions imply body contact or the exchange of fluids such as mating. For indirect interactions, the communal web is used as a "medium of communication" which conveys important information that is available to each colony member [19]. This information encoded as small vibrations is a critical aspect for the collective coordination among members [20]. Vibrations are employed by the colony members to decode several messages such as the size of the trapped preys, characteristics of the neighboring members, etc. The intensity of such vibrations depend on the weight and distance of the spiders that have produced them.

In spite of the complexity, all the cooperative global patterns in the colony level are generated as a result of internal interactions among colony members [26]. Such internal interactions involve a set of simple behavioral rules followed by each spider in the colony. Behavioral rules are divided into two different classes: social interaction (cooperative behavior) and mating [27].

As a social insect, spiders perform cooperative interaction with other colony members. The way in which this behavior takes place depends on the spider gender. Female spiders which show a major tendency to socialize present an attraction or dislike over others, irrespectively of gender [17]. For a particular female spider, such attraction or dislike is commonly developed over other spiders according to their vibrations which are emitted over the communal web and represent strong colony members [20]. Since the vibrations depend on the weight and distance of the members which provoke them, stronger vibrations are produced either by big spiders or neighboring members [19]. The bigger a spider is, the better it is considered as a colony member. The final decision of attraction or dislike over a determined member is taken according to an internal state which is influenced by several factors such as reproduction cycle, curiosity and other random phenomena [20].

Different to female spiders, the behavior of male members is reproductive-oriented [28]. Male spiders recognize themselves as a subgroup of alpha males which dominate the colony resources. Therefore, the male population is divided into two classes: dominant and non-dominant male spiders [28]. Dominant male spiders have better fitness characteristics (normally size) in comparison to non-dominant. In a typical behavior, dominant males are attracted to the closest female spider in the communal web. In contrast, non-dominant male spiders tend to concentrate upon the center of the male population as a strategy to take advantage of the resources wasted by dominant males [29].

Mating is an important operation that no only assures the colony survival, but also allows the information exchange among members. Mating in a social-spider colony is performed by dominant males and female members [30]. Under such circumstances, when a dominant male spider locates one or more female members within a specific range, it mates with all the females in order to produce offspring [31].

4.3 The SSO Algorithm

In this chapter, the operational principles from the social-spider colony have been used as guidelines for developing a new metaheuristic optimization algorithm. The SSO assumes that entire search space is a communal web, where all the social-spiders interact to each other. In the approach, each solution within the search space represents a spider position in the communal web. Every spider receives a weight according to the fitness value of the solution that is symbolized by the social-spider. The algorithm models two different search agents (spiders): males and females. Depending on gender, each individual is conducted by a set of different evolutionary operators which mimic different cooperative behaviors that are commonly assumed within the colony.

An interesting characteristic of social-spiders is the highly female-biased populations. In order to emulate this fact, the algorithm starts by defining the number of female and male spiders that will be characterized as individuals in the search space. The number of females N_f is randomly selected within the range of 65–90% of the entire population N. Therefore, N_f is calculated by the following equation:

$$N_f = \text{floor}[(0.9 - \text{rand} \cdot 0.25) \cdot N] \tag{4.1}$$

where rand is a random number between [0, 1] whereas floor(·) maps a real number to an integer number. The number of male spiders N_m is computed as the complement between N and N_f. It is calculated as follows:

$$N_m = N - N_f \tag{4.2}$$

Therefore, the complete population **S,** composed by N elements, is divided in two sub-groups **F** and **M**. The Group **F** assembles the set of female individuals $\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{N_f}\}$ whereas **M** groups the male members ($\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{N_m}\}$), where $\mathbf{S} = \mathbf{F} \cup \mathbf{M}$ ($\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$), such that $\mathbf{S} = \left\{\mathbf{s}_1 = \mathbf{f}_1, \mathbf{s}_2 = \mathbf{f}_2, \ldots, \mathbf{s}_{N_f} = \mathbf{f}_{N_f}, \mathbf{s}_{N_f+1} = \mathbf{m}_1, \mathbf{s}_{N_f+2} = \mathbf{m}_2, \ldots, \mathbf{s}_N = \mathbf{m}_{N_m}\right\}$.

4.3.1 Fitness Assignation

In the biological metaphor, the spider size is the characteristic that evaluates the individual capacity to perform better over its assigned tasks. In the approach, every individual (spider) receives a weight w_i which represents the solution quality that corresponds to the spider i (irrespective of gender) of the population $\mathbf{S}$. In order to calculate the weight of every spider the next equation is used:

$$w_i = \frac{J(\mathbf{s}_i) - worst_{\mathbf{S}}}{best_{\mathbf{S}} - worst_{\mathbf{S}}} \tag{4.3}$$

where $J(\mathbf{s}_i)$ is the fitness value obtained by the evaluation of the spider position $\mathbf{s}_i$ with regard to the objective function $J(\cdot)$. The values $worst_{\mathbf{S}}$ and $best_{\mathbf{S}}$ are defined as follows (considering a maximization problem):

$$best_{\mathbf{S}} = \max_{k \in \{1,2,\ldots,N\}} (J(\mathbf{s}_k)) \text{ and } worst_{\mathbf{S}} = \min_{k \in \{1,2,\ldots,N\}} (J(\mathbf{s}_k)) \tag{4.4}$$

4.3.2 Modeling of the Vibrations Through the Communal Web

The communal web is used as a mechanism to transmit information among the colony members. This information is encoded as small vibrations that are critical for the collective coordination of all individuals in the population. The vibrations depend on the weight and distance of the spider which has generated them. Since the distance is relative to the individual that provokes the vibrations and the member who detects them, members located near to the individual that provokes the vibrations, perceive stronger vibrations in comparison with members located in distant positions. In order to reproduce this process, the vibrations perceived by the individual i as a result of the information transmitted by the member j are modeled according to the following equation:

$$Vib_{i,j} = w_j \cdot e^{-d_{i,j}^2} \tag{4.5}$$

where the $d_{i,j}$ is the Euclidian distance between the spiders i and j, such that $d_{i,j} = \left\| \mathbf{s}_i - \mathbf{s}_j \right\|$.

Although it is virtually possible to compute perceived-vibrations by considering any pair of individuals, three special relationships are considered within the SSO approach:

1. Vibrations $Vibc_i$ are perceived by the individual i ($\mathbf{s}_i$) as a result of the information transmitted by the member c ($\mathbf{s}_c$) who is an individual that has two important

characteristics: it is the nearest member to i and possesses a higher weight in comparison to i ($w_c > w_i$).

$$Vibc_i = w_c \cdot e^{-d_{i,c}^2} \tag{4.6}$$

2. The vibrations $Vibb_i$ perceived by the individual i as a result of the information transmitted by the member b ($\mathbf{s}_b$), with b being the individual holding the best weight (best fitness value) of the entire population $\mathbf{S}$, such that $w_b = \max_{k \in \{1,2,\ldots,N\}} (w_k)$.

$$Vibb_i = w_b \cdot e^{-d_{i,b}^2} \tag{4.7}$$

3. The vibrations $Vibf_i$ perceived by the individual i ($\mathbf{s}_i$) as a result of the information transmitted by the member f ($\mathbf{s}_f$), with f being the nearest female individual to i.

$$Vibf_i = w_f \cdot e^{-d_{i,f}^2} \tag{4.7}$$

Figure 4.1 shows the configuration of each special relationship: (a) $Vibc_i$, (b) $Vibb_i$ and (c) $Vibf_i$.

4.3.3 Initializing the Population

Like other swarm algorithms, the SSO is an iterative process whose first step is to randomly initialize the entire population (female and male). The algorithm begins by initializing the set $\mathbf{S}$ of N spider positions. Each spider position, $\mathbf{f}_i$ or $\mathbf{m}_i$, is a n-dimensional vector containing the parameter values to be optimized. Such values are randomly and uniformly distributed between the pre-specified lower initial parameter bound p_j^{low} and the upper initial parameter bound p_j^{high}, just as it described by the following expressions:

$$\begin{array}{ll} f_{i,j}^0 = p_j^{low} + \text{rand}(0,1) \cdot (p_j^{high} - p_j^{low}) & m_{k,j}^0 = p_j^{low} + \text{rand}(0,1) \cdot (p_j^{high} - p_j^{low}) \\ i = 1, 2, \ldots, N_f; \quad j = 1, 2, \ldots, n & k = 1, 2, \ldots, N_m; \quad j = 1, 2, \ldots, n \end{array} \tag{4.8}$$

where j, i and k are the parameter and individual indexes respectively whereas zero signals the initial population. The function rand (0, 1) generates a random number between 0 and 1. Hence, $f_{i,j}$ is the j-th parameter of the i-th female spider position.

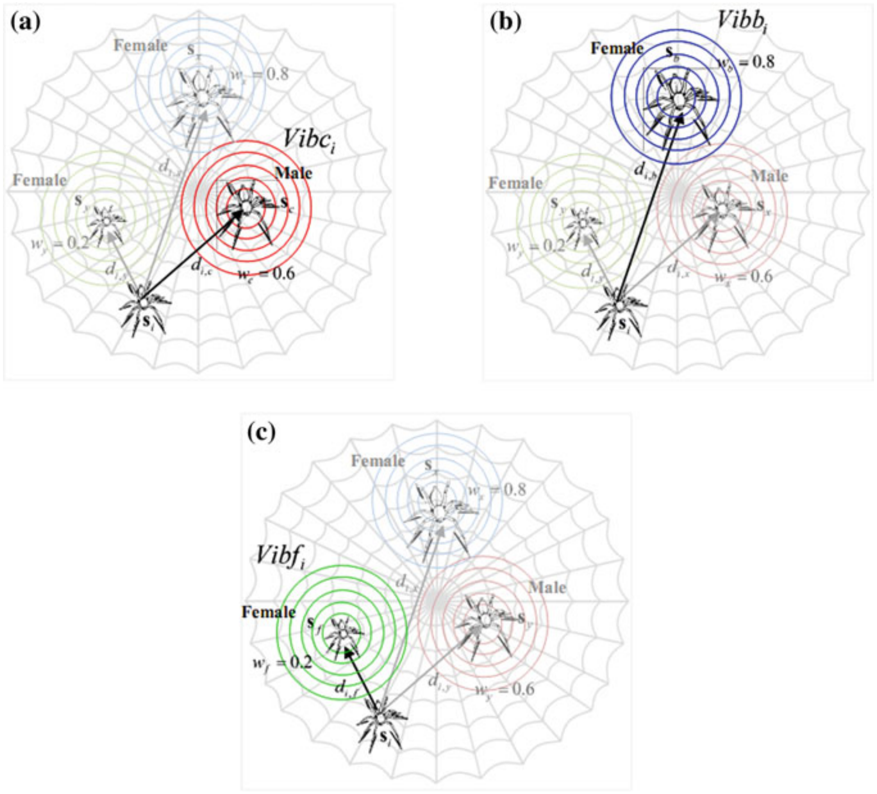


Fig. 4.1 Configuration of each special relation: **a** $Vibc_i$, **b** $Vibb_i$ and **c** $Vibf_i$

4.3.4 Cooperative Operators

Female cooperative operator

Social-spiders perform cooperative interaction over other colony members. The way in which this behavior takes place depends on the spider gender. Female spiders present an attraction or dislike over others irrespective of gender. For a particular female spider, such attraction or dislike is commonly developed over other spiders according to their vibrations which are emitted over the communal web. Since vibrations depend on the weight and distance of the members which have originated them, strong vibrations are produced either by big spiders or other neighboring members lying nearby the individual which is perceiving them. The final decision of attraction or dislike over a determined member is taken considering an internal state which is influenced by several factors such as reproduction cycle, curiosity and other random phenomena.

In order to emulate the cooperative behavior of the female spider, a new operator is defined. The operator considers the position change of the female spider i at each iteration. Such position change, which can be of attraction or repulsion, is computed as a combination of three different elements. The first one involves the change in regard to the nearest member to i that holds a higher weight and produces the vibration $Vibc_i$. The second one considers the change regarding the best individual of the entire population $\mathbf{S}$ who produces the vibration $Vibb_i$. Finally, the third one incorporates a random movement.

Since the final movement of attraction or repulsion depends on several random phenomena, the selection is modeled as a stochastic decision. For this operation, a uniform random number r_m is generated within the range [0, 1]. If r_m is smaller than a threshold PF, an attraction movement is generated; otherwise, a repulsion movement is produced. Therefore, such operator can be modeled as follows:

$$\mathbf{f}_i^{k+1} = \begin{cases} \mathbf{f}_i^k + \alpha \cdot Vibc_i \cdot (\mathbf{s}_c - \mathbf{f}_i^k) + \beta \cdot Vibb_i \cdot (\mathbf{s}_b - \mathbf{f}_i^k) & \\ \qquad + \delta \cdot \left(\text{rand} - \frac{1}{2}\right) & \text{with probability } PF \\ \mathbf{f}_i^k - \alpha \cdot Vibc_i \cdot (\mathbf{s}_c - \mathbf{f}_i^k) - \beta \cdot Vibb_i \cdot (\mathbf{s}_b - \mathbf{f}_i^k) & \\ \qquad + \delta \cdot \left(\text{rand} - \frac{1}{2}\right) & \text{with probability } 1 - PF \end{cases} \tag{4.9}$$

where α, β, δ and rand are random numbers between [0, 1] whereas k represents the iteration number. The individual $\mathbf{s}_c$ and $\mathbf{s}_b$ represent the nearest member to i that holds a higher weight and the best individual of the entire population $\mathbf{S}$, respectively.

Under this operation, each particle presents a movement which combines the past position that holds the attraction or repulsion vector over the local best element $\mathbf{s}_c$ and the global best individual $\mathbf{s}_b$ seen so-far. This particular type of interaction avoids the quick concentration of particles at only one point and encourages each particle to search around the local candidate region within its neighborhood ($\mathbf{s}_c$), rather than interacting to a particle ($\mathbf{s}_b$) in a distant region of the domain. The use of this scheme has two advantages. First, it prevents the particles from moving towards the global best position, making the algorithm less susceptible to premature convergence. Second, it encourages particles to explore their own neighborhood thoroughly before converging towards the global best position. Therefore, it provides the algorithm with global search ability and enhances the exploitative behavior of the approach.

Male cooperative operator

According to the biological behavior of the social-spider, male population is divided into two classes: dominant and non-dominant male spiders. Dominant male spiders have better fitness characteristics (usually regarding the size) in comparison to non-dominant. Dominant males are attracted to the closest female spider in the communal web. In contrast, non-dominant male spiders tend to concentrate in the center of the male population as a strategy to take advantage of resources that are wasted by dominant males.

For emulating such cooperative behavior, the male members are divided into two different groups (dominant members **D** and non-dominant members **ND**) according to their position with regard to the median member. Male members, with a weight value above the median value within the male population, are considered the dominant individuals **D**. On the other hand, those under the median value are labeled as non-dominant **ND** males. In order to implement such computation, the male population **M** $\left(\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{N_m}\}\right)$ is arranged according to their weight value in decreasing order. Thus, the individual whose weight w_{N_f+m} is located in the middle is considered the median male member. Since indexes of the male population **M** in regard to the entire population **S** are increased by the number of female members N_f, the median weight is indexed by $N_f + m$. According to this, change of positions for the male spider can be modeled as follows:

$$\mathbf{m}_i^{k+1} = \begin{cases} \mathbf{m}_i^k + \alpha \cdot Vibf_i \cdot (\mathbf{s}_f - \mathbf{m}_i^k) + \delta \cdot \left(\text{rand} - \frac{1}{2}\right) & \text{if } w_{N_f+i} > w_{N_f+m} \\ \mathbf{m}_i^k + \alpha \cdot \left(\frac{\sum_{h=1}^{N_m} \mathbf{m}_h^k \cdot w_{N_f+h}}{\sum_{h=1}^{N_m} w_{N_f+h}} - \mathbf{m}_i^k \right) & \text{if } w_{N_f+i} \le w_{N_f+m} \end{cases}, \tag{4.10}$$

where the individual $\mathbf{s}_f$ represents the nearest female individual to the male member i whereas $\left(\sum_{h=1}^{N_m} \mathbf{m}_h^k \cdot w_{N_f+h} / \sum_{h=1}^{N_m} w_{N_f+h}\right)$ correspond to the weighted mean of the male population **M**.

By using this operator, two different behaviors are produced. First, the set **D** of particles is attracted to others in order to provoke mating. Such behavior allows incorporating diversity into the population. Second, the set **ND** of particles is attracted to the weighted mean of the male population **M.** This fact is used to partially control the search process according to the average performance of a sub-group of the population. Such mechanism acts as a filter which avoids that very good individuals or extremely bad individuals influence the search process.

4.3.5 Mating Operator

Mating in a social-spider colony is performed by dominant males and the female members. Under such circumstances, when a dominant male $\mathbf{m}_g$ spider ($g \in \mathbf{D}$) locates a set $\mathbf{E}^g$ of female members within a specific range r (range of mating), it mates, forming a new brood $\mathbf{s}_{new}$ which is generated considering all the elements of the set $\mathbf{T}^g$ that, in turn, has been generated by the union $\mathbf{E}^g \cup \mathbf{m}_g$. It is important to emphasize that if the set $\mathbf{E}^g$ is empty, the mating operation is canceled. The range r is defined as a radius which depends on the size of the search space. Such radius r is computed according to the following model:

$$r = \frac{\sum_{j=1}^{n} (p_j^{high} - p_j^{low})}{2 \cdot n} \tag{4.11}$$

In the mating process, the weight of each involved spider (elements of $\mathbf{T}^g$) defines the probability of influence for each individual into the new brood. The spiders holding a heavier weight are more likely to influence the new product, while elements with lighter weight have a lower probability. The influence probability Ps_i of each member is assigned by the roulette method, which is defined as follows:

$$Ps_i = \frac{w_i}{\sum_{j \in \mathbf{T}^k} w_j}, \tag{4.12}$$

where $i \in \mathbf{T}^g$.

Once the new spider is formed, it is compared to the new spider candidate $\mathbf{s}_{new}$ holding the worst spider $\mathbf{s}_{wo}$ of the colony, according to their weight values (where $w_{wo} = \min_{l \in \{1,2,\ldots,N\}}(w_l)$). If the new spider is better than the worst spider, the worst spider is replaced by the new one. Otherwise, the new spider is discarded and the population does not suffer changes. In case of replacement, the new spider assumes the gender and index from the replaced spider. Such fact assures that the entire population $\mathbf{S}$ maintains the original rate between female and male members.

In order to demonstrate the mating operation, Fig. 4.2a illustrates a simple optimization problem. As an example, it is assumed a population $\mathbf{S}$ of eight different 2-dimensional members ($N = 8$), five females ($N_f = 5$) and three males ($N_m = 3$). Figure 4.2b shows the initial configuration of the proposed example with three different female members $\mathbf{f}_2(\mathbf{s}_2)$, $\mathbf{f}_3(\mathbf{s}_3)$ and $\mathbf{f}_4(\mathbf{s}_4)$ constituting the set $\mathbf{E}^2$ which is located inside of the influence range r of a dominant male $\mathbf{m}_2$ ($\mathbf{s}_7$). Then, the new candidate spider $\mathbf{s}_{new}$ is generated from the elements $\mathbf{f}_2$, $\mathbf{f}_3$, $\mathbf{f}_4$ and $\mathbf{m}_2$ which constitute the set $\mathbf{T}^2$. Therefore, the value of the first decision variable $s_{new,1}$ for the new spider is chosen by means of the roulette mechanism considering the values already existing from the set $\{f_{2,1}, f_{3,1}, f_{4,1}, m_{2,1}\}$. The value of the second decision variable $s_{new,2}$ is also chosen in the same manner. Table 4.1 shows the data for constructing the new spider through the roulette method. Once the new spider $\mathbf{s}_{new}$ is formed, its weight w_{new} is calculated. As $\mathbf{s}_{new}$ is better than the worst member $\mathbf{f}_1$ that is present in the population $\mathbf{S}$, $\mathbf{f}_1$ is replaced by $\mathbf{s}_{new}$. Therefore, $\mathbf{s}_{new}$ assumes the same gender and index from $\mathbf{f}_1$. Figure 4.2c shows the configuration of $\mathbf{S}$ after the mating process.

Under this operation, new generated particles locally exploit the search space inside the mating range in order to find better individuals.

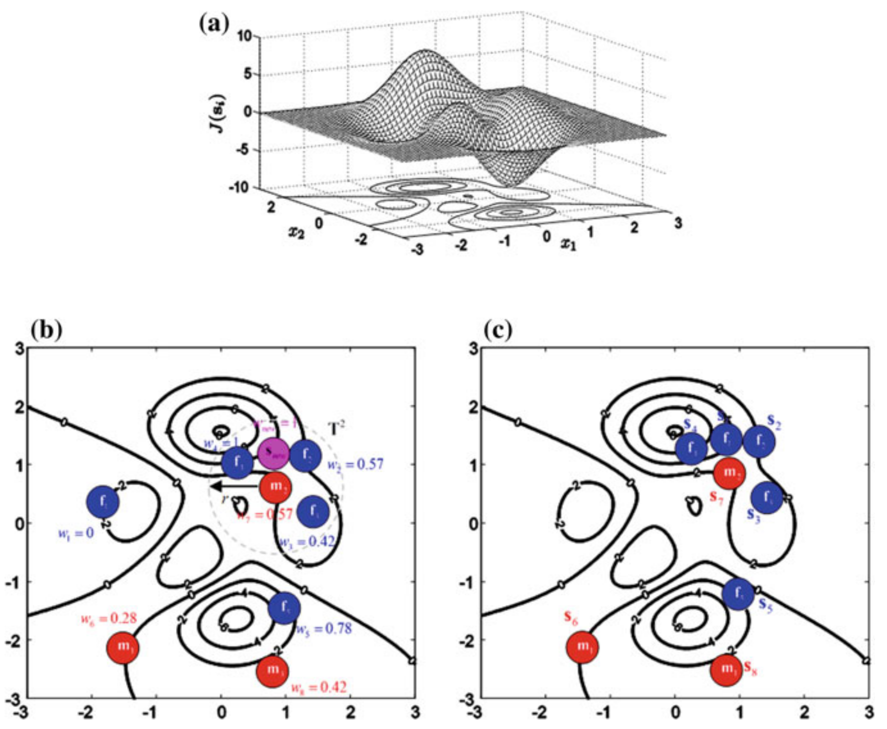
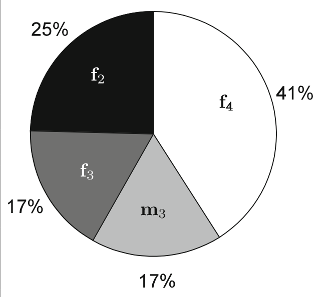


Fig. 4.2 Example of the mating operation: **a** optimization problem, **b** initial configuration before mating and **c** configuration after the mating operation

Table 4.1 Data for constructing the new spider $\mathbf{s}_{new}$ through the roulette method

Spider		Position	w_i	Ps_i	Roulette
$\mathbf{s}_1$	$\mathbf{f}_1$	(−1.9, 0.3)	0.00	–	
$\mathbf{s}_2$	$\mathbf{f}_2$	(1.4, 1.1)	0.57	0.22	
$\mathbf{s}_3$	$\mathbf{f}_3$	(1.5, 0.2)	0.42	0.16	
$\mathbf{s}_4$	$\mathbf{f}_4$	(0.4, 1.0)	1.00	0.39	
$\mathbf{s}_5$	$\mathbf{f}_5$	(1.0, −1.5)	0.78	–	
$\mathbf{s}_6$	$\mathbf{m}_1$	(−1.3, −1.9)	0.28	–	
$\mathbf{s}_7$	$\mathbf{m}_2$	(0.9, 0.7)	0.57	0.22	
$\mathbf{s}_8$	$\mathbf{m}_3$	(0.8, −2.6)	0.42	–	
$\mathbf{s}_{new}$		(0.9, 1.1)	1.00	–	

4.3.6 Computational Procedure

The computational procedure for the algorithm can be summarized as follows:

Step 1: Considering N as the total number of n-dimensional colony members, define the number of male N_m and females N_f spiders in the entire population **S**.

$$N_f = \text{floor}\left[(0.9 - \text{rand} \cdot 0.25) \cdot N\right] \text{ and } N_m = N - N_f,$$

where rand is a random number between [0,1] whereas $\text{floor}(\cdot)$ maps a real number to an integer number.

Step 2: Initialize randomly the female ($\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ \ , \mathbf{f}_{N_f}\}$) and male ($\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \ \ , \mathbf{m}_{N_m}\}$) members (where $\mathbf{S} = \left\{\mathbf{s}_1 = \mathbf{f}_1, \mathbf{s}_2 = \mathbf{f}_2, \ \ , \mathbf{s}_{N_f} = \mathbf{f}_{N_f}, \mathbf{s}_{N_f+1} = \mathbf{m}_1, \mathbf{s}_{N_f+2} = \mathbf{m}_2, \ \ , \mathbf{s}_N = \mathbf{m}_{N_m}\right\}$ and calculate the radius of mating.

$$r = \frac{\sum_{j=1}^{n}(p_j^{high} - p_j^{low})}{2 \cdot n}$$

```
for ($i = 1$ ; $i < N_f + 1$; $i$++)
    for ($j = 1$; $j <$ n + 1; $j$++)
        $f_{i,j}^0 = p_j^{low} + \text{rand}(0,1) \cdot (p_j^{high} - p_j^{low})$
    end for
end for
for ($k = 1$; $k < N_m + 1$; $k$++)
    for ($j = 1$; $j <$ n + 1; $j$++)
        $m_{k,j}^0 = p_j^{low} + \text{rand} \cdot (p_j^{high} - p_j^{low})$
    end for
end for
```

Step 3: Calculate the weight of every spider of **S** (Sect. 4.1.1).

```
for ($i = 1$, $i < N + 1$; $i$++)
    $w_i = \frac{J(\mathbf{s}_i) - worst_{\mathbf{S}}}{best_{\mathbf{S}} - worst_{\mathbf{S}}}$
    where $best_{\mathbf{S}} = \max_{k \in \{1,2,\ldots,N\}}(J(\mathbf{s}_k))$ and $worst_{\mathbf{S}} = \min_{k \in \{1,2,\ldots,N\}}(J(\mathbf{s}_k))$
end for
```

Step 4: Move female spiders according to the female cooperative operator (Sect. 4.1.4).

```
for ($i = 1$; $i < N_f + 1$; $i$++)
    Calculate $Vibc_i$ and $Vibb_i$ (Sect. 4.1.2)
    If ($r_m < PF$); where $r_m \in \text{rand}(0,1)$
        $\mathbf{f}_i^{k+1} = \mathbf{f}_i^k + \alpha \cdot Vibc_i \cdot (\mathbf{s}_c - \mathbf{f}_i^k) + \beta \cdot Vibb_i \cdot (\mathbf{s}_b - \mathbf{f}_i^k) + \delta \cdot (\text{rand} - \frac{1}{2})$
    else if
        $\mathbf{f}_i^{k+1} = \mathbf{f}_i^k - \alpha \cdot Vibc_i \cdot (\mathbf{s}_c - \mathbf{f}_i^k) - \beta \cdot Vibb_i \cdot (\mathbf{s}_b - \mathbf{f}_i^k) + \delta \cdot (\text{rand} - \frac{1}{2})$
    end if
end for
```

```
Step 5: Move the male spiders according to the male cooperative operator (Sect. 3.1.4).
        Find the median male individual $\left(w_{N_f+m}\right)$ from $\mathbf{M}$.
        for ($i = 1$; $i < N_m + 1$; $i$++)
            Calculate $Vibf_i$ (Sect. 4.1.2)
            If $\left(w_{N_f+i} > w_{N_f+m}\right)$
                $\mathbf{m}_i^{k+1} = \mathbf{m}_i^k + \alpha \cdot Vibf_i \cdot (\mathbf{s}_f - \mathbf{m}_i^k) + \delta \cdot \left(\text{rand} - \frac{1}{2}\right)$
            Else if
                $\mathbf{m}_i^{k+1} = \mathbf{m}_i^k + \alpha \cdot \left( \frac{\sum_{h=1}^{N_m} \mathbf{m}_h^k \cdot w_{N_f+h}}{\sum_{h=1}^{N_m} w_{N_f+h}} - \mathbf{m}_i^k \right)$
            end if
        end for
Step 6: Perform the mating operation (Sect. 4.1.5).
        for ($i = 1$; $i < N_m + 1$; $i$++)
            If $\left(\mathbf{m}_i \in \mathbf{D}\right)$
                Find $\mathbf{E}^i$
                If ( $\mathbf{E}^i$ is not empty)
                    Form $\mathbf{s}_{new}$ using the roulette method
                    If $\left(w_{new} > w_{wo}\right)$
                        $\mathbf{s}_{wo} = \mathbf{s}_{new}$
                    end if
                end if
            end if
        end for
Step 7: If the stop criteria is met, the process is finished; otherwise, go back to Step 3
```

4.3.7 Discussion About the SSO Algorithm

Evolutionary algorithms (EA) have been widely employed for solving complex optimization problems. These methods are found to be more powerful than conventional methods based on formal logics or mathematical programming [32]. In an EA algorithm, search agents have to decide whether to explore unknown search positions or to exploit already tested positions in order to improve their solution quality. Pure exploration degrades the precision of the evolutionary process but increases its capacity to find new potential solutions. On the other hand, pure exploitation allows refining existent solutions but adversely drives the process to local optimal solutions. Therefore, the ability of an EA to find a global optimal solutions depends on its capacity to find a good balance between the exploitation of found-so-far elements and the exploration of the search space [33]. So far, the exploration–exploitation dilemma has been an unsolved issue within the framework of evolutionary algorithms.

EA defines individuals with the same property, performing virtually the same behavior. Under these circumstances, algorithms waste the possibility to add new and selective operators as a result of considering individuals with different characteristics. These operators could incorporate computational mechanisms to improve several important algorithm characteristics such as population diversity or searching capacities.

On the other hand, PSO and ABC are the most popular metaheuristic algorithms for solving complex optimization problems. However, they present serious flaws such as premature convergence and difficulty to overcome local minima [10, 11]. Such problems arise from operators that modify individual positions. In such algorithms, the position of each agent in the next iteration is updated yielding an attraction towards the position of the best particle seen so-far (in case of PSO) or any other randomly chosen individual (in case of ABC). Such behaviors produce that the entire population concentrates around the best particle or diverges without control as the algorithm evolves, either favoring the premature convergence or damaging the exploration-exploitation balance [12, 13].

Different to other EA, at SSO each individual is modeled considering the gender. Such fact allows incorporating computational mechanisms to avoid critical flaws such as premature convergence and incorrect exploration-exploitation balance commonly present in both, the PSO and the ABC algorithm. From an optimization point of view, the use of the social-spider behavior as a metaphor introduces interesting concepts in EA: the fact of dividing the entire population into different search-agent categories and the employment of specialized operators that are applied selectively to each of them. By using this framework, it is possible to improve the balance between exploitation and exploration, yet preserving the same population, i.e. individuals who have achieved efficient exploration (female spiders) and individuals that verify extensive exploitation (male spiders). Furthermore, the social-spider behavior mechanism introduces an interesting computational scheme with three important particularities: first, individuals are separately processed according to their characteristics. Second, operators share the same communication mechanism allowing the employment of important information of the evolutionary process to modify the influence of each operator. Third, although operators modify the position of only an individual type, they use global information (positions of all individual types) in order to perform such modification. Figure 4.3 presents a schematic representation of the algorithm-data-flow. According to Fig. 4.3, the female cooperative and male cooperative operators process only female or male individuals, respectively. However, the mating operator modifies both individual types.

4.4 Experimental Results

A comprehensive set of 19 functions, which have been collected from Refs. [34–40], has been used to test the performance of the SSO approach. Table 4.4 in the Appendix presents the benchmark functions used in our experimental study. In the table, n

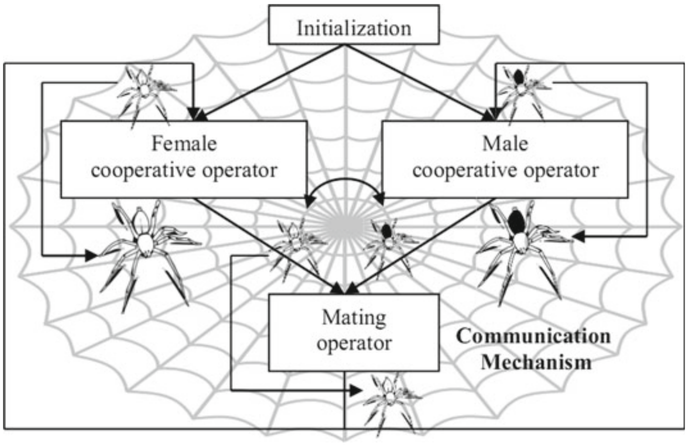


Fig. 4.3 Schematic representation of the SSO algorithm-data-flow

indicates the function dimension, $f(\mathbf{x}^*)$ the optimum value of the function, $\mathbf{x}^*$ the optimum position and S the search space (subset of R^n). A detailed description of each function is given in the Appendix.

4.4.1 Performance Comparison to Other Metaheuristic Algorithms

We have applied the SSO algorithm to 19 functions whose results have been compared to those produced by the Particle Swarm Optimization (PSO) method [3] and the Artificial Bee Colony (ABC) algorithm [4]. These are considered as the most popular metaheuristic algorithms for many optimization applications. In all comparisons, the population has been set to 50 individuals. The maximum iteration number for all functions has been set to 1000. Such stop criterion has been selected to maintain compatibility to similar works reported in the literature [41, 42].

The parameter setting for each algorithm in the comparison is described as follows:

1. PSO: The parameters are set to $c_1 = 2$ and $c_2 = 2$; besides, the weight factor decreases linearly from 0.9 to 0.2 [3].
2. ABC: The algorithm has been implemented using the guidelines provided by its own reference [4], using the parameter $limit = 100$.
3. SSO: Once it has been determined experimentally, the parameter PF has been set to 0.7. It is kept for all experiments in this section.

The experiment compares the SSO to other algorithms such as PSO and ABC. The results for 30 runs are reported in Table 4.2 considering the following performance indexes: the Average Best-so-far (AB) solution, the Median Best-so-far (MB) and the Standard Deviation (SD) of best-so-far solution. The best outcome for each function is boldfaced. According to this table, SSO delivers better results than PSO and ABC for all functions. In particular, the test remarks the largest difference in performance which is directly related to a better trade-off between exploration and exploitation. Figure 4.4 presents the evolution curves for PSO, ABC and the SSO algorithm considering as examples the functions f_1, f_3, f_5, f_{10}, f_{15} and f_{19} from the experimental set. Among them, the rate of convergence of SSO is the fastest, which finds the best solution in less of 400 iterations on average while the other three algorithms need much more iterations. A non-parametric statistical significance proof known as the Wilcoxon's rank sum test for independent samples [43, 44] has been conducted over the "average best-so-far" (AB) data of Table 4.2, with an 5% significance level. Table 4.3 reports the p-values produced by Wilcoxon's test for the pair-wise comparison of the "average best so-far" of two groups. Such groups are constituted by SSO versus PSO and SSO versus ABC. As a null hypothesis, it is assumed that there is no significant difference between mean values of the two algorithms. The alternative hypothesis considers a significant difference between the "average best-so-far" values of both approaches. All p-values reported in Table 4.3 are less than 0.05 (5% significance level) which is a strong evidence against the null hypothesis. Therefore, such evidence indicates that SSO results are statistically significant and it has not occurred by coincidence (i.e. due to common noise contained in the process).

Table 4.2 Minimization results of benchmark functions of Table 4.4 with $n = 30$. Maximum number of iterations = 1000

		SSO	ABC	PSO
$f_1(x)$	AB	**1.96E − 03**	2.90E − 03	1.00E + 03
	MB	2.81E − 03	1.50E − 03	**2.08E − 09**
	SD	**9.96E − 04**	1.44E − 03	3.05E + 03
$f_2(x)$	AB	**1.37E − 02**	1.35E − 01	5.17E + 01
	MB	**1.34E − 02**	1.05E − 01	5.00E + 01
	SD	**3.11E − 03**	8.01E − 02	2.02E + 01
$f_3(x)$	AB	**4.27E − 02**	1.13E + 00	8.63E + 04
	MB	**3.49E − 02**	6.11E − 01	8.00E + 04
	SD	**3.11E − 02**	1.57E + 00	5.56E + 04
$f_4(x)$	AB	**5.40E − 02**	5.82E + 01	1.47E + 01
	MB	**5.43E − 02**	5.92E + 01	1.51E + 01
	SD	**1.01E − 02**	7.02E + 00	3.13E + 00
$f_5(x)$	AB	**1.14E + 02**	1.38E + 02	3.34E + 04
	MB	**5.86E + 01**	1.32E + 02	4.03E + 02
	SD	**3.90E + 01**	1.55E + 02	4.38E + 04

(continued)

Table 4.2 (continued)

		SSO	ABC	PSO
$f_6(x)$	AB	**2.68E − 03**	4.06E − 03	1.00E + 03
	MB	2.68E − 03	3.74E − 03	**1.66E − 09**
	SD	**6.05E − 04**	2.98E − 03	3.06E + 03
$f_7(x)$	AB	**1.20E + 01**	1.21E + 01	1.50E + 01
	MB	**1.20E + 01**	1.23E + 01	1.37E + 01
	SD	**5.76E − 01**	9.00E − 01	4.75E + 00
$f_8(x)$	AB	**2.14E + 00**	3.60E + 00	3.12E + 04
	MB	**3.64E + 00**	8.04E − 01	2.08E + 02
	SD	**1.26E + 00**	3.54E + 00	5.74E + 04
$f_9(x)$	AB	**6.92E − 05**	1.44E − 04	2.47E + 00
	MB	**6.80E − 05**	8.09E − 05	9.09E − 01
	SD	**4.02E − 05**	1.69E − 04	3.27E + 00
$f_{10}(x)$	AB	**4.44E − 04**	1.10E − 01	6.93E + 02
	MB	**4.05E − 04**	4.97E − 02	5.50E + 02
	SD	**2.90E − 04**	1.98E − 01	6.48E + 02
$f_{11}(x)$	AB	**6.81E + 01**	3.12E + 02	4.11E + 02
	MB	**6.12E + 01**	3.13E + 02	4.31E + 02
	SD	**3.00E + 01**	4.31E + 01	1.56E + 02
$f_{12}(x)$	AB	**5.39E − 05**	1.18E − 04	4.27E + 07
	MB	**5.40E − 05**	1.05E − 04	1.04E − 01
	SD	**1.84E − 05**	8.88E − 05	9.70E + 07
$f_{13}(x)$	AB	**1.76E − 03**	1.87E − 03	5.74E − 01
	MB	**1.12E − 03**	1.69E − 03	1.08E − 05
	SD	**6.75E − 04**	1.47E − 03	2.36E + 00
$f_{14}(x)$	AB	**−9.36E + 02**	−9.69E + 02	−9.63E + 02
	MB	**−9.36E + 02**	−9.60E + 02	−9.92E + 02
	SD	**1.61E + 01**	6.55E + 01	6.66E + 01
$f_{15}(x)$	AB	**8.59E + 00**	2.64E + 01	1.35E + 02
	MB	**8.78E + 00**	2.24E + 01	1.36E + 02
	SD	**1.11E + 00**	1.06E + 01	3.73E + 01
$f_{16}(x)$	AB	**1.36E − 02**	6.53E − 01	1.14E + 01
	MB	**1.39E − 02**	6.39E − 01	1.43E + 01
	SD	**2.36E − 03**	3.09E − 01	8.86E + 00
$f_{17}(x)$	AB	**3.29E − 03**	5.22E − 02	1.20E + 01
	MB	**3.21E − 03**	4.60E − 02	1.35E − 02
	SD	**5.49E − 04**	3.42E − 02	3.12E + 01

(continued)

Table 4.2 (continued)

		SSO	ABC	PSO
$f_{18}(x)$	AB	**1.87E + 00**	2.13E + 00	1.26E + 03
	MB	**1.61E + 00**	2.14E + 00	5.67E + 02
	SD	**1.20E + 00**	1.22E + 00	1.12E + 03
$f_{19}(x)$	AB	**2.74E − 01**	4.14E + 00	1.53E + 00
	MB	**3.00E − 01**	4.10E + 00	5.50E − 01
	SD	**5.17E − 02**	4.69E − 01	2.94E + 00

Bold data represent the best results

4.5 Conclusions

In this chapter, a novel metaheuristic algorithm called the Social Spider Optimization (SSO) has been proposed for solving optimization tasks. The SSO algorithm is based on the simulation of the cooperative behavior of social-spiders whose individuals emulate a group of spiders which interact to each other based on the biological laws of a cooperative colony. The algorithm considers two different search agents (spiders): male and female. Depending on gender, each individual is conducted by a set of different swarm operators which mimic different cooperative behaviors within the colony.

In contrast to most of existent metaheuristic algorithms, the SSO approach models each individual considering two genders. Such fact allows not only to emulate the cooperative behavior of the colony in a realistic way, but also to incorporate computational mechanisms to avoid critical flaws commonly delivered by the popular PSO and ABC algorithms, such as the premature convergence and the incorrect exploration-exploitation balance.

SSO has been experimentally tested considering a suite of 19 benchmark functions. The performance of SSO has been also compared to the following metaheuristic algorithms: the Particle Swarm Optimization method (PSO) [16], and the Artificial Bee Colony (ABC) algorithm [38]. Results have confirmed a acceptable performance of the SSO method in terms of the solution quality of the solution for all tested benchmark functions.

The SSO's remarkable performance is associated with two different reasons: (i) their operators allow a better particle distribution in the search space, increasing the algorithm's ability to find the global optima; and (ii) the division of the population into different individual types, provides the use of different rates between exploration and exploitation during the evolution process.

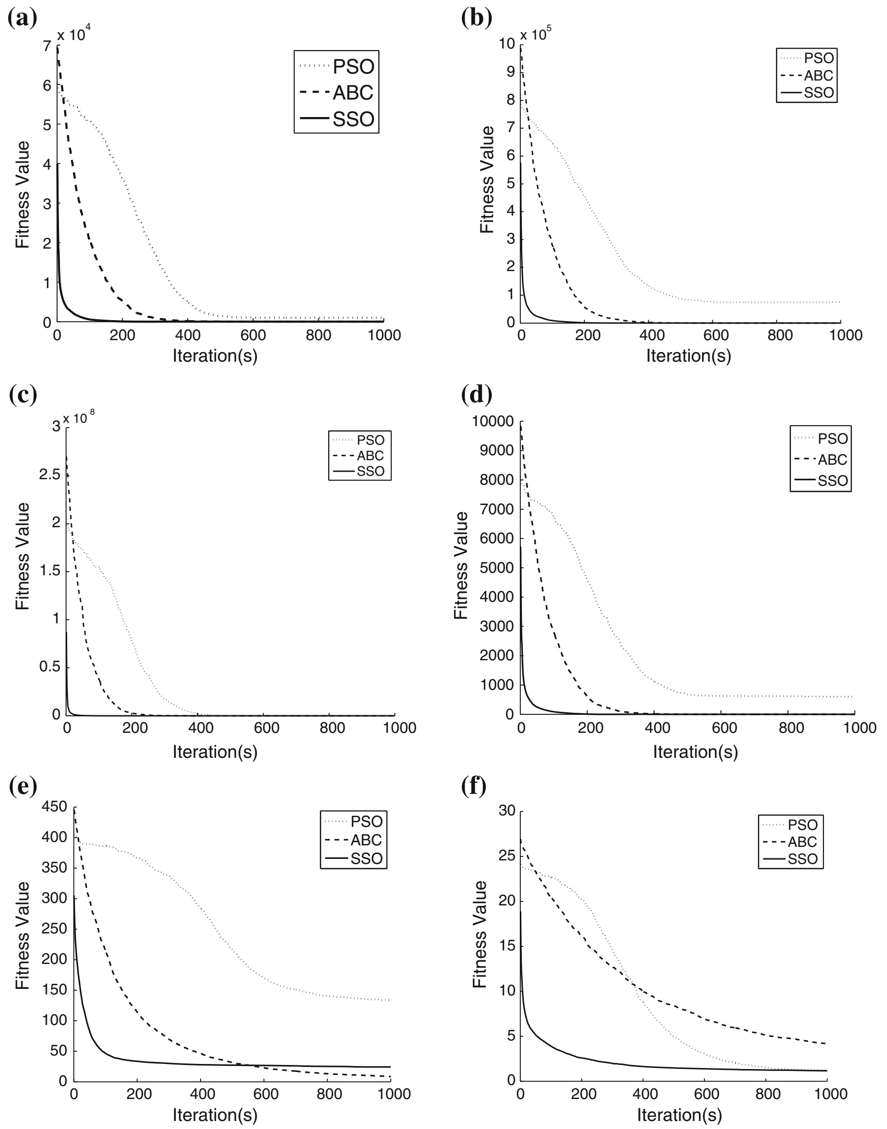


Fig. 4.4 Evolution curves for PSO, ABC and the SSO algorithm considering as examples the functions **a** f_1, **b** f_3, **c** f_5, **d** f_{10}, **e** f_{15} and **f** f_{19} from the experimental set

Table 4.3 *p*-values produced by Wilcoxon's test comparing SSO versus ABC and SSO versus PSO, over the "average best-so-far" (AB) values from Table 4.2

Function	SSO versus ABC	SSO versus PSO
$f_1(x)$	0.041	1.8E − 05
$f_2(x)$	0.048	0.059
$f_3(x)$	5.4E − 04	6.2E − 07
$f_4(x)$	1.4E − 07	4.7E − 05
$f_5(x)$	0.045	7.1E − 07
$f_6(x)$	2.3E − 04	5.5E − 08
$f_7(x)$	0.048	0.011
$f_8(x)$	0.017	0.043
$f_9(x)$	8.1E − 04	2.5E − 08
$f_{10}(x)$	4.6E − 06	1.7E − 09
$f_{11}(x)$	9.2E − 05	7.8E − 06
$f_{12}(x)$	0.022	1.1E − 10
$f_{13}(x)$	0.048	2.6E − 05
$f_{14}(x)$	0.044	0.049
$f_{15}(x)$	4.5E − 05	7.9E − 08
$f_{16}(x)$	2.8E − 05	4.1E − 06
$f_{17}(x)$	7.1E − 04	6.2E − 10
$f_{18}(x)$	0.013	8.3E − 10
$f_{19}(x)$	4.9E − 05	5.1E − 08

Appendix: List of Benchmark Functions

Table 4.4

Table 4.4 Test functions used in the experimental study

Name	Function	S	Dim	Minimum
Sphere	$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Schwefel 2.22	$f_2(\mathbf{x}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-10, 10]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Schwefel 1.2	$f_3(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-100, 100]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
F4	$f_4(\mathbf{x}) = 418.9829n + \sum_{i=1}^{n} \left(-x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)\right)$	$[-100, 100]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Rosenbrock	$f_5(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$[-30, 30]^n$	$n = 30$	$\mathbf{x}^* = (1, \ldots, 1)$; $f(\mathbf{x}^*) = 0$
Step	$f_6(\mathbf{x}) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Quartic	$f_7(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + random(0, 1)$	$[-1.28, 1.28]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Dixon and price	$f_8(\mathbf{x}) = (x_1 - 1)^2 + \sum_{i=1}^{n} i\left(2x_i^2 - x_{i-1}\right)^2$	$[-10, 10]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$

(continued)

Table 4.4 (continued)

Name	Function	S	Dim	Minimum
Levy	$f_9(\mathbf{x}) = 0.1\left\{\begin{array}{l}\sin^2(3\pi x_1)\\ +\sum_{i=1}^{n}(x_i-1)^2\left[1+\sin^2(3\pi x_i+1)\right]\\ +(x_n-1)^2\left[1+\sin^2(2\pi x_n)\right]\end{array}\right\}$ $+\sum_{i=1}^{n} u(x_i, 5, 100, 4);$ $u(x_i, a, k, m) = \begin{cases} k(x_i-a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i-a)^m & x_i < -a \end{cases}$	$[-10, 10]^n$	$n = 30$	$\mathbf{x}^* = (1, \ldots, 1);$ $f(\mathbf{x}^*) = 0$
Sum of squares	$f_{10}(\mathbf{x}) = \sum_{i=1}^{n} i x_i^2$	$[-10, 10]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0);$ $f(\mathbf{x}^*) = 0$
Zakharov	$f_{11}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^4$	$[-5, 10]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0);$ $f(\mathbf{x}^*) = 0$

(continued)

Table 4.4 (continued)

Name	Function	S	Dim	Minimum
Penalized	$f_{12}(\mathbf{x}) = \frac{\pi}{n}\left\{ \begin{array}{l} 10\sin(\pi y_1) + \\ \sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right] + (y_n-1)^2 \end{array} \right\}$ $+\sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{(x_i+1)}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i-a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i-a)^m & x_i < a \end{cases}$	$[-50, 50]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Penalized 2	$f_{13}(\mathbf{x}) = 0.1\left\{ \begin{array}{l} \sin^2(3\pi x_1) \\ +\sum_{i=1}^{n} \begin{array}{l} (x_i-1)^2\left[1+\sin^2(3\pi x_i+1)\right] \\ +(x_n-1)^2\left[1+\sin^2(2\pi x_n)\right] \end{array} \end{array} \right\}$ $+\sum_{i=1}^{n} u(x_i, 5, 100, 4)$ where $u(x_i, a, k, m)$ is the same as Penalized function.	$[-50, 50]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Schwefel	$f_{14}(\mathbf{x}) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]^n$	$n = 30$	$\mathbf{x}^* = (420, \ldots, 420)$; $f(\mathbf{x}^*) = -418.9829 \times n$
Rastrigin	$f_{15}(\mathbf{x}) = \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12, 5.12]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$

(continued)

Table 4.4 (continued)

Name	Function	S	Dim	Minimum
Ackley	$f_{16}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp$	$[-32, 32]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Griewank	$f_{17}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Powell	$f_{18}(\mathbf{x}) = \sum_{i=1}^{n/k}(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4$	$[-4, 5]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$
Salomon	$f_{19}(\mathbf{x}) = -\cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2} + 1$	$[-100, 100]^n$	$n = 30$	$\mathbf{x}^* = (0, \ldots, 0)$; $f(\mathbf{x}^*) = 0$

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence: from natural to artificial systems. Oxford University Press Inc., New York, NY, USA (1999)
2. Kassabalidis, I., El-Sharkawi, M.A., Marks, R.J., Arabshahi, P., Gray, A.A.: Swarm intelligence for routing in communication networks. In: IEEE Global Telecommunications Conference, GLOBECOM'01, vol. 6, pp. 3613–3617 (2001)
3. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
4. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report-TR06. Engineering Faculty, Computer Engineering Department, Erciyes University (2005)
5. Passino, K.M.: Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst. Mag. **22**(3), 52–67 (2002)
6. Hossein, A., Hossein-Alavi, A.: Krill herd: a new bio-inspired optimization algorithm. Commun. Nonlinear Sci. Numer. Simulat. **17**, 4831–4845 (2012)
7. Yang, X.S.: Engineering optimization: an introduction with metaheuristic applications. Wiley, London (2010)
8. Rajabioun, R.: Cuckoo optimization algorithm. Appl. Soft Comput. **11**, 5508–5518 (2011)
9. Bonabeau, E.: Social insect colonies as complex adaptive systems. Ecosystems **1**, 437–443 (1998)
10. Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B., Tian, Q.: Self-adaptive learning based particle swarm optimization. Inf. Sci. **181**(20), 4515–4538 (2011)
11. Wan-Li, X., Mei-Qing, A.: An efficient and robust artificial bee colony algorithm for numerical optimization. Comput. Oper. Res. **40**, 1256–1265 (2013)
12. Wang, H., Sun, H., Li, C., Rahnamayan, S., Jeng-shyang, P.: Diversity enhanced particle swarm optimization with neighborhood. Inf. Sci. **223**, 119–135 (2013)
13. Banharnsakun, A., Achalakul, T., Sirinaovakul, B.: The best-so-far selection in artificial bee colony algorithm. Appl. Soft Comput. **11**, 2888–2901 (2011)
14. Gordon, D.: The organization of work in social insect colonies. Complexity **8**(1), 43–46 (2003)
15. Lubin, T.B.: The evolution of sociality in spiders. In: Brockmann, H.J. (ed.) Advances in the Study of Behavior, vol. 37, pp. 83–145 (2007)
16. Uetz, G.W.: Colonial web-building spiders: balancing the costs and benefits of group-living. In: Choe, E.J., Crespi, B. (eds.) The Evolution of Social Behavior in Insects and Arachnids, pp. 458–475. Cambridge University Press, Cambridge, England
17. Aviles, L.: Sex-ratio bias and possible group selection in the social spider Anelosimus eximius. Am. Nat. **128**(1), 1–12 (1986)
18. Burgess, J.W.: Social spacing strategies in spiders. In: Rovner, P.N. (ed.) Spider Communication: Mechanisms and Ecological Significance, pp. 317–351. Princeton University Press, Princeton, New Jersey (1982)
19. Maxence, S.: Social organization of the colonial spider Leucauge sp. in the Neotropics: vertical stratification within colonies. J. Arachnol. **38**, 446–451 (2010)
20. Eric, C., Yip, K.S.: Cooperative capture of large prey solves scaling challenge faced by spider societies. Proc. Natl. Acad. Sci. U. S. A. **105**(33), 11818–11822 (2008)
21. Oster, G., Wilson, E.: Caste and ecology in the social insects. N. J. Princeton University Press, Princeton (1978)
22. Hölldobler, B., Wilson, E.O.: Journey to the ants: a story of scientific exploration (1994). ISBN 0-674-48525-4
23. Hölldobler, B., Wilson, E.O.: The ants. Harvard University Press (1990). ISBN 0-674-04075-9
24. Avilés, L.: Causes and consequences of cooperation and permanent-sociality in spiders. In: Choe, B.C. (ed.) The Evolution of Social Behavior in Insects and Arachnids, pp. 476–498. Cambridge University Press, Cambridge, Massachusetts (1997)
25. Rayor, E.C.: Do social spiders cooperate in predator defense and foraging without a web? Behav. Ecol. Sociobiol. **65**(10), 1935–1945 (2011)

26. Gove, R., Hayworth, M., Chhetri, M., Rueppell, O.: Division of labour and social insect colony performance in relation to task and mating number under two alternative response threshold models. Insect. Soc. **56**(3), 19–331 (2009)
27. Ann, L., Rypstra, R.S.: Prey size, prey perishability and group foraging in a social spider. Oecologia **86**(1), 25–30 (1991)
28. Pasquet, A.: Cooperation and prey capture efficiency in a social spider, Anelosimus eximius (Araneae, Theridiidae). Ethology **90**, 121–133 (1991)
29. Ulbrich, K., Henschel, J.: Intraspecific competition in a social spider. Ecol. Model. **115**(2–3), 243–251 (1999)
30. Jones, T., Riechert, S.: Patterns of reproductive success associated with social structure and microclimate in a spider system. Anim. Behav. **76**(6), 2011–2019 (2008)
31. Damian, O., Andrade, M., Kasumovic, M.: Dynamic population structure and the evolution of spider mating systems. Adv. Insect. Physiol. **41**, 65–114 (2011)
32. Yang, X.-S.: Nature-inspired metaheuristic algorithms. Luniver Press, Beckington (2008)
33. Chen, D.B., Zhao, C.X.: Particle swarm optimization with adaptive population size and its application. Appl. Soft Comput. **9**(1), 39–48 (2009)
34. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1995)
35. Yang, E., Barton, N.H., Arslan, T., Erdogan, A.T.: A novel shifting balance theory-based approach to optimization of an energy-constrained modulation scheme for wireless sensor networks. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, Hong Kong, China, pp. 2749–2756, 1–6 June 2008, IEEE (2008)
36. Duan, X., Wang, G.G., Kang, X., Niu, Q., Naterer, G., Peng, Q.: Performance study of mode-pursuing sampling method. In: Engineering Optimization, vol. 41, no. 1 (2009)
37. Vesterstrom, J., Thomsen, R.: A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: Proceedings of the 2004 Congress on Evolutionary Computation (CEC 2004), vol. 2, pp. 1980–1987, 19–23 June 2004
38. Mezura-Montes, E., Velázquez-Reyes, J., Coello Coello, C.A.: A comparative study of differential evolution variants for global optimization. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO '06), ACM, New York, NY, USA, pp. 485–492 (2006)
39. Karaboga, D., Akay, B.: A comparative study of artificial bee colony algorithm. Appl. Math. Comput. **214**(1), 108–132 (2009). ISSN 0096-3003
40. Krishnanand, K.R., Nayak, S.K., Panigrahi, B.K., Rout, P.K.: Comparative study of five bio-inspired evolutionary optimization techniques. In: 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), pp. 1231–1236 (2009)
41. Ying, J., Ke-Cun, Z., Shao-Jian, Q.: A deterministic global optimization algorithm. Appl. Math. Comput. **185**(1), 382–387 (2007)
42. Rashedia, E., Nezamabadi-pour, H., Saryazdi, S.: Filter modeling using gravitational search algorithm. Eng. Appl. Artif. Intell. **24**(1), 117–122 (2011)
43. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics **1**, 80–83 (1945)
44. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special session on real parameter optimization. J. Heurist. (2008). https://doi.org/10.1007/s10732-008-9080-4

Chapter 5
The Locust Swarm Optimization Algorithm

Abstract In recent years swarm intelligence emulate the behavior of insects or animal. In this chapter, an optimization algorithm called Locust Search (LS) is presented. The LS is inspired of the behavior of the locust swarms. In the algorithm consider two different behaviors: solitary and social. This tow types of behavior interact with each other in ways to allow find solution to a complex optimization problem. In order to illustrate the efficiency and robustness the LS was compared with other well-known optimization algorithms. The algorithm was proved with several benchmark functions.

5.1 Introduction

The collective intelligent behavior of insect or animal groups in nature such as flocks of birds, colonies of ants, schools of fish, swarms of bees and termites have attracted the attention of researchers. The aggregative conduct of insects or animals is known as swarm behavior. Even though the single members of swarms are non-sophisticated individuals, they are able to achieve complex tasks in cooperation. The collective swarm behavior emerges from relatively simple actions or interactions among the members. Entomologists have studied this collective phenomenon to model biological swarms while engineers have applied these models as a framework for solving complex real-world problems. The discipline of artificial intelligence which is concerned with the design of intelligent multi-agent algorithms by taking inspiration from the collective behavior of social insects or animals is known as swarm intelligence [1]. Swarm algorithms have several advantages such as scalability, fault tolerance, adaptation, speed, modularity, autonomy and parallelism [2].

Several swarm algorithms have been developed by a combination of deterministic rules and randomness, mimicking the behavior of insect or animal groups in nature. Such methods include the social behavior of bird flocking and fish schooling such as the Particle Swarm Optimization (PSO) algorithm [3], the cooperative behavior of bee colonies such as the Artificial Bee Colony (ABC) technique [4], the social foraging behavior of bacteria such as the Bacterial Foraging Optimization Algorithm (BFOA) [5], the simulation of the herding behavior of krill individuals such as the

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_5

Krill Herd (KH) method [6], the mating behavior of firefly insects such as the Firefly (FF) method [7] the emulation of the lifestyle of cuckoo birds such as the Cuckoo Search (CS) [8], the social-spider behavior such as the Social Spider Optimization (SSO) [9], the simulation of the animal behavior in a group such as the Collective Animal Behavior [10] and the emulation of the differential evolution in species such as the Differential Evolution (DE) [11].

In particular, insect swarms and animal groups provide a rich set of metaphors for designing swarm optimization algorithms. Such methods are complex systems composed by individuals that tend to reproduce specialized behaviors [12]. However, most of swarm algorithms and other evolutionary algorithms tend to exclusively concentrate the individuals in the current best positions. Under such circumstances, these algorithms seriously limit their search capacities.

Although PSO and DE are the most popular algorithms for solving complex optimization problems, they present serious flaws such as premature convergence and difficulty to overcome local minima [13, 14]. The cause for such problems is associated to the operators that modify individual positions. In such algorithms, during their evolution, the position of each agent for the next iteration is updated yielding an attraction towards the position of the best particle seen so-far (in case of PSO) or towards other promising individuals (in case of DE). As the algorithm evolves, these behaviors cause that the entire population rapidly concentrates around the best particles, favoring the premature convergence and damaging the appropriate exploration of the search space [15, 16].

The interesting and exotic collective behavior of insects have fascinated and attracted researchers for many years. The intelligent behavior observed in these groups provides survival advantages, where insect aggregations of relatively simple and "unintelligent" individuals can accomplish very complex tasks using only limited local information and simple rules of behavior [17]. Locusts (*Schistocerca gregaria*) are a representative example of such collaborative insects [18]. Locust is a kind of grasshopper that can change reversibly between a solitary and a social phase, which differ considerably in behavior [19]. The two phases show many differences including both overall levels of activity and the degree to which locusts are attracted or repulsed among them [20]. In the solitary phase, locusts avoid contact each other (locust concentrations). As consequence, they distribute throughout the space, exploring sufficiently the plantation [20]. On other hand, in the social phase, locusts frantically concentrate around the elements that have already found good food sources [21]. Under such a behavior, locust attempt to efficiently find better nutrients by devastating promising areas within the plantation.

In this chapter, a swarm algorithm, called the Locust Search (LS) is applied for solving optimization tasks. The LS algorithm is based on the simulation of the behavior presented in swarms of locusts. In the proposed algorithm, individuals emulate a group of locusts which interact to each other based on the biological laws of the cooperative swarm. The algorithm considers two different behaviors: solitary and social. Depending on the behavior, each individual is conducted by a set of evolutionary operators which mimic the different cooperative behaviors that are typically found in the swarm. Different to most of existent swarm algorithms, in the proposed approach, the modeled behavior explicitly avoids the concentration of individuals in the current best positions. Such fact allows not only to emulate in a better realistic way the cooperative behavior of the locust colony, but also to incorporate a computational mechanism to avoid critical flaws commonly present in the popular PSO and DE algorithms, such as the premature convergence and the incorrect exploration-exploitation balance. In order to illustrate the proficiency and robustness of the proposed approach, it is compared to other well-known evolutionary methods. The comparison examines several standard benchmark functions which are commonly considered in the literature. The results show a high performance of the proposed method for searching a global optimum in several benchmark functions.

5.2 Biological Fundamentals and Mathematical Models

Social insect societies are complex cooperative systems that self-organize within a set of constraints. Cooperative groups are better at manipulating and exploiting their environment, defending resources and brood, and allowing task specialization among group members [22, 23]. A social insect colony functions as an integrated unit that not only possesses the ability to operate at a distributed manner, but also to undertake enormous construction of global projects [24]. It is important to acknowledge that global order in insects can arise as a result of internal interactions among members.

Locusts are a kind of grasshoppers that exhibit two opposite behavioral phases, solitary and social (gregarious). Individuals in the solitary phase avoid contact each other (locust concentrations). As consequence, they distribute throughout the space, exploring sufficiently the plantation [20]. In contrast, locusts in the gregarious phase form several concentrations. These concentrations may contain up to 10^{10} members, cover cross-sectional areas of up to 10 km^2, and travel up to 10 km per day for a period of days or weeks as they feed causing devastating crop loss [25]. The mechanism for the switch from the solitary phase to the gregarious phase is complex, and has been a subject of significant biological inquiry. A set of factors recently has been implicated, including geometry of the vegetation landscape and the olfactory stimulus [26].

Only few works [20, 21] that mathematically model the locust behavior have been published. In such approaches, it is developed two different minimal models with the goal of reproducing the macroscopic structure and motion of a group of locusts. Since the method proposed in [20] models the behavior of each locust in the group, it is used to develop the algorithm proposed in this paper.

5.2.1 Solitary Phase

In this section, it is described the way in which the position of each locust is modified as a consequence of its behavior under the solitary phase. Considering that $\mathbf{x}_i^k$ represents the current position of the ith locust in a group of N different elements, the new position $\mathbf{x}_i^{k+1}$ is calculated by using the following model:

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \Delta\mathbf{x}_i, \tag{5.1}$$

where $\Delta\mathbf{x}_i$ corresponds to the change of position experimented by $\mathbf{x}_i^k$ as a consequence of its social interaction with all the other elements in the group.

Two locusts in the solitary phase exert forces on each other according to basic biological principles of attraction and repulsion (see, e.g., [20]). Repulsion operates very strongly over a short length scale in order to avoid concentrations. Attraction is weaker, and operates over a longer length scale, providing the social force necessary for maintaining the cohesion in the group. Therefore, it is modeled the strength of these social forces using the function:

$$s(r) = F \cdot e^{-r/L} - e^{-r} \tag{5.2}$$

Here, r is a distance, F describes the strength of attraction, and L is the typical attractive length scale. We have scaled the time and space coordinates so that the repulsive strength and length scale are unity. We assume that $F < 1$ and $L > 1$ so that repulsion is stronger and shorter-scale, and attraction in weaker and longer-scale. This is typical for social organisms [21]. The social force exerted by locust j on locust i is:

$$\mathbf{s}_{ij} = s(r_{ij}) \cdot \mathbf{d}_{ij}, \tag{5.3}$$

where $r_{ij} = \left|\mathbf{x}_j - \mathbf{x}_i\right|$ is the distance between the two locusts and $\mathbf{d}_{ij} = (\mathbf{x}_j - \mathbf{x}_i)/r_{ij}$ is the unit vector pointing from $\mathbf{x}_i$ to $\mathbf{x}_j$. The total social force on each locust can be modeled as the superposition of all of the pairwise interactions:

$$\mathbf{S}_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{s}_{ij}, \tag{5.4}$$

The change of position $\Delta\mathbf{x}_i$ is modeled as the total social force experimented by $\mathbf{x}_i^k$ as the superposition of all of the pairwise interactions. Therefore, $\Delta\mathbf{x}_i$ is defined as follows:

$$\Delta\mathbf{x}_i = \mathbf{S}_i, \tag{5.5}$$

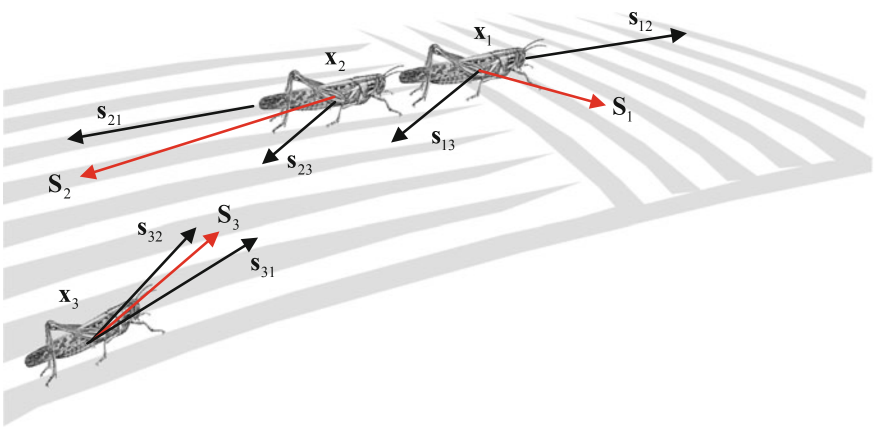


Fig. 5.1 Behavioral model under the solitary phase

In order to illustrate the behavioral model under the solitary phase, Fig. 5.1 presents an example. It is assumed a population of three different members ($N = 3$) which adopt a determined configuration in the current iteration k. As a consequence of the social forces, each element suffers an attraction or repulsion to other elements depending on the distance among them. Such forces are represented by $\mathbf{s}_{12}$, $\mathbf{s}_{13}$, $\mathbf{s}_{21}$, $\mathbf{s}_{23}$, $\mathbf{s}_{31}$, $\mathbf{s}_{32}$. Since $\mathbf{x}_1$ and $\mathbf{x}_2$ are too close, the social forces $\mathbf{s}_{12}$ and $\mathbf{s}_{13}$ present a repulsive nature. On the other hand, as the distances $|\mathbf{x}_1 - \mathbf{x}_3|$ and $|\mathbf{x}_2 - \mathbf{x}_3|$ are quite long, the social forces $\mathbf{s}_{13}, \mathbf{s}_{23}, \mathbf{s}_{31}$ and $\mathbf{s}_{32}$ between $\mathbf{x}_1 \leftrightarrow \mathbf{x}_3$ and $\mathbf{x}_2 \leftrightarrow \mathbf{x}_3$ are from the attractive nature. Therefore, the change of position $\Delta\mathbf{x}_1$ is computed as the resultant between $\mathbf{s}_{12}$ and $\mathbf{s}_{13}(\Delta\mathbf{x}_1 = \mathbf{s}_{12} + \mathbf{s}_{13})$. The values $\Delta\mathbf{x}_2$ and $\Delta\mathbf{x}_3$ of the locusts $\mathbf{x}_1$ and $\mathbf{x}_2$ are also calculated accordingly.

In addition to the presented model [20], some studies [27–29] suggest that the social force $\mathbf{s}_{ij}$ is also affected by the dominance of the involved individuals $\mathbf{x}_i$ and $\mathbf{x}_j$ in the pairwise process. Dominance is a property that relatively qualifies the capacity of an individual to survive, in relation to other elements in a group. Dominance in locust is determined for several characteristics such as size, chemical emissions, location with regard to food sources, etc. Under such circumstances, the social force is magnified or weakened depending on the most dominant individual involved in the repulsion-attraction process.

5.2.2 *Social Phase*

In this phase, locusts frantically concentrate around the elements that have already found good food sources. Under such a behavior, locust attempt to efficiently find better nutrients by devastating promising areas within the plantation.

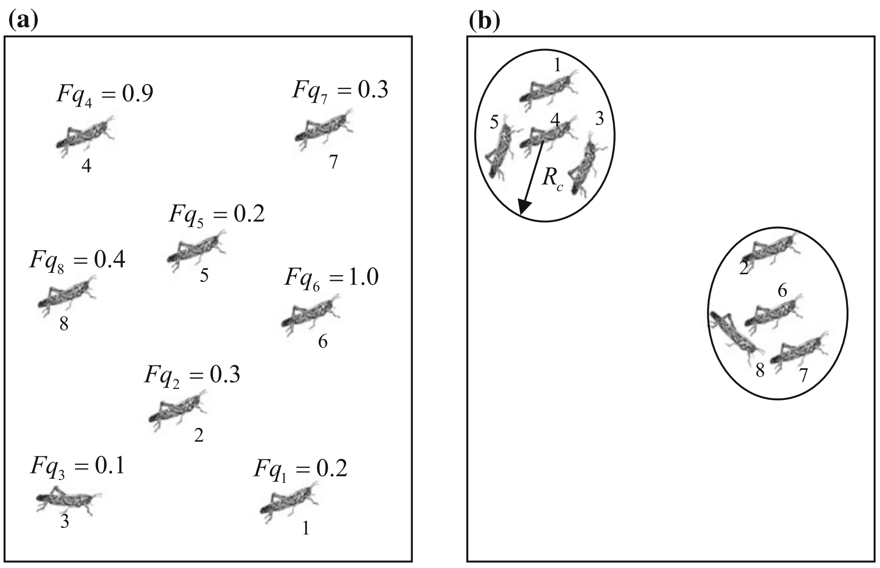


Fig. 5.2 Behavioral model under the social phase. **a** Initial configuration and food quality indexes, **b** final configuration after the operation of the social phase

In order to simulate the social phase, to each locust $\mathbf{x}_i$ of the group, it is associated a food quality index Fq_i. This index reflex the quality of the food source where $\mathbf{x}_i$ is located.

Under this behavioral model, it is first ranked the N elements of the group according to their food quality indexes. Afterward, the b elements with the best food quality indexes are selected ($b \ll N$). Considering a concentration radius R_c created around each selected element, a set of c new locusts is randomly generated inside R_c. As a result, most of the locusts will be concentrated around the best b elements. Figure 5.2 shows a simple example of behavioral model under the social phase. In the example, it is assumed a configuration of eight locust ($N = 8$), as it is illustrated in Fig. 5.2a. In the Figure, it is also presented the food quality index for each locust. A food quality index near to one indicates a better food source. Therefore, considering $b = 2$, the final configuration after the social phase, it is presented in Fig. 5.2b.

5.3 The Locust Search (LS) Algorithm

In this paper, the behavioral principles from a swarm of locusts have been used as guidelines for developing a new swarm optimization algorithm. The LS assumes that entire search space is a plantation, where all the locusts interact to each other. In the proposed approach, each solution within the search space represents a locust position in the plantation. Every locust receives a food quality index according to the fitness value of the solution that is symbolized by the locust. The algorithm implements two different behaviors: solitary and social. Depending on the behavior, each individual is conducted by a set of evolutionary operators which mimic the different cooperative behaviors that are typically found in the swarm.

From the implementation point of view, in the LS operation, a population $\mathbf{L}^k(\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\})$ of N locusts (individuals) is evolved from the initial point ($k = 0$) to a total *gen* number iterations ($k = gen$). Each locust $\mathbf{l}_i^k(i \in [1, \ldots, N]))$ represents an n-dimensional vector $\left\{l_{i,1}^k, l_{i,2}^k, \ldots, l_{i,n}^k\right\}$ where each dimension corresponds to a decision variable of the optimization problem to be solved. The set of decision variables constitutes the feasible search space $\mathbf{S} = \left\{\mathbf{l}_i^k \in \mathbb{R}^n \middle| lb_d \leq l_{i,d}^k \leq ub_d\right\}$, where lb_d and ub_d corresponds to the lower and upper bounds for the dimension d, respectively. The food quality index associated to each locust $\mathbf{l}_i^k$ (candidate solution) is evaluated by using an objective function $f(\mathbf{l}_i^k)$ whose final result represents the fitness value of $\mathbf{l}_i^k$. In LS, each iteration of the evolution process consists of two operators: (A) solitary and (B) social. Beginning by the solitary stage, the set of locusts is operated in order to sufficiently explore the search space. Then, during the social operation, existent solutions are refined within a determined neighborhood (exploitation).

5.3.1 Solitary Operation (A)

One of the most interesting features of the proposed method is the use of the solitary operator to modify the current locust positions. Under this approach, locusts are displaced as a consequence of the social forces produced by the positional relations among the elements of the swarm. Therefore, near individuals tend to repel with each other, avoiding the concentration of elements in regions. On the other hand, distant individuals tend to attract with each other, maintaining the cohesion of the swarm. Different to the original model [20], in the proposed operator, social forces are also magnified or weakened depending on the best fitness value (the most dominant) of the individuals involved in the repulsion-attraction process.

In the solitary operation, a new position $\mathbf{p}_i(i \in [1, \ldots, N])$ is produced by perturbing the current locust position $\mathbf{l}_i^k$ with a change of position $\Delta\mathbf{l}_i(\mathbf{p}_i = \mathbf{l}_i^k + \Delta\mathbf{l}_i)$. The change of position $\Delta\mathbf{l}_i$ is the result of the social interactions experimented by $\mathbf{l}_i^k$ as a consequence of its repulsion-attraction behavioral model. Such social interactions are pairwise computed among $\mathbf{l}_i^k$ and the other N-1 individuals in the swarm. In the original model, social forces are calculated by using Eq. (5.3). However, in the proposed method, it is modified to include the best fitness value (the most dominant) of the individuals involved in the repulsion-attraction process. Therefore, the social force exerted between $\mathbf{l}_j^k$ and $\mathbf{l}_i^k$ is calculated by using the following new model:

$$\mathbf{s}_{ij}^m = \rho(\mathbf{l}_i^k, \mathbf{l}_j^k) \cdot s(r_{ij}) \cdot \mathbf{d}_{ij} + rand(1, -1), \tag{5.6}$$

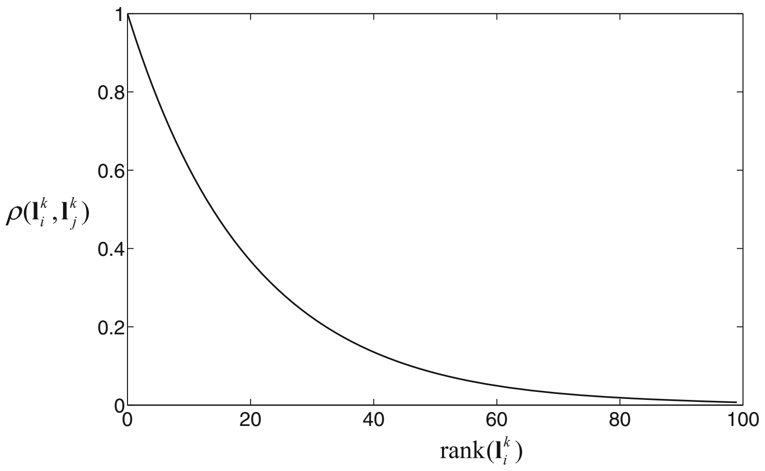


Fig. 5.3 Behavior of $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ considering 100 individuals

where $s(r_{ij})$ is the social force strength defined in Eq. (5.2) and $\mathbf{d}_{ij} = (\mathbf{l}_j^k - \mathbf{l}_i^k)/r_{ij}$ is the unit vector pointing from $\mathbf{l}_i^k$ to $\mathbf{l}_j^k$. Besides, *rand*$(1, -1)$ is a number randomly generated between 1 and -1.

$\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ is the dominance function that calculates the dominance value of the most dominant individual from $\mathbf{l}_j^k$ and $\mathbf{l}_i^k$. In order to operate $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$, all the individuals from $\mathbf{L}^k(\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\})$ are ranked according to their fitness values. The ranks are assigned so that the best individual receives the rank 0 (zero) whereas the worst individual obtains the rank N-1. Therefore, the function $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ is defined as follows:

$$\rho(\mathbf{l}_i^k, \mathbf{l}_j^k) = \begin{cases} e^{-(5\cdot\mathrm{rank}(\mathbf{l}_i^k)/N)} & \text{if } \mathrm{rank}(\mathbf{l}_i^k) < \mathrm{rank}(\mathbf{l}_j^k) \\ e^{-(5\cdot\mathrm{rank}(\mathbf{l}_j^k)/N)} & \text{if } \mathrm{rank}(\mathbf{l}_i^k) > \mathrm{rank}(\mathbf{l}_j^k) \end{cases}, \tag{5.7}$$

where the function rank(α) delivers the rank of the α-individual. According to Eq. 5.7, $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ gives as a result a value within the interval (1, 0). The maximum value of one is reached by $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ when one of the individuals $\mathbf{l}_j^k$ and $\mathbf{l}_i^k$ is the best element of the population $\mathbf{L}^k$ in terms of its fitness value. On the other hand, a value close to zero, it is obtained when both individuals $\mathbf{l}_j^k$ and $\mathbf{l}_i^k$ possess quite bad fitness values. Figure 5.3 shows the behavior of $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ considering 100 individuals. In the Figure, it is assumed that $\mathbf{l}_i^k$ represents one of the 99 individuals with ranks between 0 and 98 whereas $\mathbf{l}_j^k$ is fixed to the element with the worst fitness value (rank 99).

Under the incorporation of $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ in Eq. (5.6), social forces are magnified or weakened depending on the best fitness value (the most dominant) of the individuals involved in the repulsion-attraction process.

Finally, the total social force on each individual $\mathbf{l}_i^k$ is modeled as the superposition of all of the pairwise interactions exerted over it:

$$\mathbf{S}_i^m = \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{s}_{ij}^m, \tag{5.8}$$

Therefore, the change of position $\Delta \mathbf{l}_i$ is considered as the total social force experimented by $\mathbf{l}_i^k$ as the superposition of all of the pairwise interactions. Therefore, $\Delta \mathbf{l}_i$ is defined as follows:

$$\Delta \mathbf{l}_i = \mathbf{S}_i^m, \tag{5.9}$$

After calculating the new positions $\mathbf{P}$ $(\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\})$ of the population $\mathbf{L}^k(\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\})$, the final positions $\mathbf{F}$ $(\{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N\})$ must be calculated. The idea is to admit only the changes that guarantee an improvement in the search strategy. If the fitness value of $\mathbf{p}_i\,(f(\mathbf{p}_i))$ is better than $\mathbf{l}_i^k\,(f(\mathbf{l}_i^k))$, then $\mathbf{p}_i$ is accepted as the final solution. Otherwise, $\mathbf{l}_i^k$ is retained. This procedure can be resumed by the following statement (considering a minimization problem):

$$\mathbf{f}_i = \begin{cases} \mathbf{p}_i & \text{if } f(\mathbf{p}_i) < f(\mathbf{l}_i^k) \\ \mathbf{l}_i^k & \text{otherwise} \end{cases} \tag{5.10}$$

In order to illustrate the performance of the solitary operator, Fig. 5.4 presents a simple example where the solitary operator is iteratively applied. It is assumed a population of 50 different members ($N = 50$) which adopt a concentrated configuration as initial condition (Fig. 5.4a). As a consequence of the social forces, the set of element tends to distribute through the search space. Examples of different distributions are shown in Fig. 5.4b, c and d after applying 25, 50 and 100 different solitary operations, respectively.

5.3.2 *Social Operation (B)*

The social procedure represents the exploitation phase of the LS algorithm. Exploitation is the process of refining existent individuals within a small neighborhood in order to improve their solution quality.

The social procedure is a selective operation which is applied only to a subset $\mathbf{E}$ of the final positions $\mathbf{F}$ (where $\mathbf{E} \subseteq \mathbf{F}$). In the operation first is necessary to sort $\mathbf{F}$ according to their fitness values and store the sorted elements in a temporal population $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N\}$. The elements in $\mathbf{B}$ are sorted so that the best individual receives the position $\mathbf{b}_1$ whereas the worst individual obtains the location $\mathbf{b}_N$. Therefore, the subset $\mathbf{E}$ is integrated by only the first g locations of $\mathbf{B}$ (promising solutions).

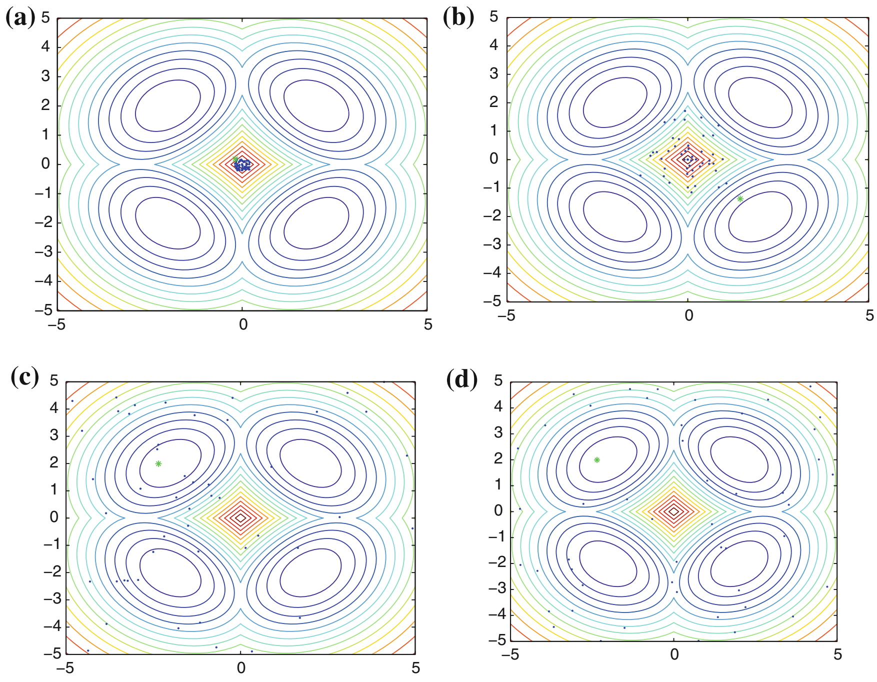


Fig. 5.4 Examples of different distributions. **a** initial condition, **b** distribution after applying 25 operations, **c** 50 and **d** 100

Under this operation, a subspace C_j is created around each selected particle $\mathbf{f}_j \in \mathbf{E}$. The size of C_j depends on the distance e_d which is defined as follows:

$$e_d = \frac{\sum_{q=1}^{n} (ub_q - lb_q)}{n} \cdot \beta \tag{5.11}$$

where ub_q and lb_q are the upper and lower bounds in the q-th dimension, n is the number of dimensions of the optimization problem, whereas $\beta \in [0, 1]$ is a tuning factor. Therefore, the limits of C_j are modeled as follows:

$$\begin{aligned} uss_j^q &= b_{j,q} + e_d \\ lss_j^q &= b_{j,q} - e_d \end{aligned} \tag{5.12}$$

where uss_j^q and lss_j^q are the upper and lower bounds of the qth dimension for the subspace C_j, respectively.

Considering the subspace C_j around each element $\mathbf{f}_j \in \mathbf{E}$, a set of h new particles $\left(\mathbf{M}_j^h = \{\mathbf{m}_j^1, \mathbf{m}_j^2, \ldots, \mathbf{m}_j^h\}\right)$ are randomly generated inside the bounds defined by Eq. (5.12). Once the h samples are generated, the individual $\mathbf{l}_j^{k+1}$ of the next population

$\mathbf{L}^{k+1}$ must be created. In order to calculate $\mathbf{l}_j^{k+1}$, the best particle $\mathbf{m}_j^{best}$, in terms of fitness value from the h samples (where $\mathbf{m}_j^{best} \in [\mathbf{m}_j^1, \mathbf{m}_j^2, \ldots, \mathbf{m}_j^h]$), is compared to $\mathbf{f}_j$. If $\mathbf{m}_j^{best}$ is better than $\mathbf{f}_j$ according to their fitness values, $\mathbf{l}_j^{k+1}$ is updated with $\mathbf{m}_j^{best}$, otherwise $\mathbf{f}_j$ is selected. The elements of $\mathbf{F}$ that have not been processed by the procedure ($\mathbf{f}_w \notin \mathbf{E}$) transfer their corresponding values to $\mathbf{L}^{k+1}$ with no change.

The social operation is used to exploit only prominent solutions. According to the propose method, inside each subspace C_j, h random samples are selected. Since the number of selected samples in each subspace is very small (typically $h < 4$), the use of this operator reduces substantially the number of fitness function evaluations.

In order to demonstrate the social operation, a numerical example has been set by applying the proposed process to a simple function. Such function considers the interval of $-3 \leq d_1, d_2 \leq 3$ whereas the function possesses one global maxima of value 8.1 at (0, 1.6). Notice that d_1 and d_2 correspond to the axis coordinates (commonly x and y). For this example, it is assumed a final position population $\mathbf{F}$ of six 2-dimensional members ($N = 6$). Figure 5.5 shows the initial configuration of the proposed example, the black points represents the half of the particles with the best fitness values (the first three element of **B,** $g = 3$) whereas the grey points ($\mathbf{f}_2, \mathbf{f}_4, \mathbf{f}_6 \notin \mathbf{E}$) corresponds to the remaining individuals. From Fig. 5.5, it can be seen that the social procedure is applied to all black particles ($\mathbf{f}_5 = \mathbf{b}_1, \mathbf{f}_3 = \mathbf{b}_2$ and $\mathbf{f}_1 = \mathbf{b}_3, \mathbf{f}_5, \mathbf{f}_3, \mathbf{f}_1 \in \mathbf{E}$) yielding two new random particles ($h = 2$), characterized by the white points $\mathbf{m}_1^1$, $\mathbf{m}_1^2$, $\mathbf{m}_3^1$, $\mathbf{m}_3^2$, $\mathbf{m}_5^1$ and $\mathbf{m}_5^2$ for each black point inside of their corresponding subspaces C_1, C_3 and C_5. Considering the particle $\mathbf{f}_3$ in Fig. 5.5, the particle $\mathbf{m}_3^2$ corresponds to the best particle ($\mathbf{m}_3^{best}$) from the two randomly generated particles (according to their fitness values) within C_3. Therefore, the particle $\mathbf{m}_3^{best}$ will substitute $\mathbf{f}_3$ in the individual $\mathbf{l}_3^{k+1}$ for the next generation, since it holds a better fitness value than $\mathbf{f}_3\big(f(\mathbf{f}_3) < f(\mathbf{m}_3^{best})\big)$.

5.3.3 *Complete LS Algorithm*

LS is a simple algorithm with only five adjustable parameters: the strength of attraction F, the attractive length L, number of promising solutions g, the population size N and the number of generations *gen*. The operation of LS is divided in three parts: Initialization, solitary operation and the social process. In the initialization ($k = 0$), the first population $\mathbf{L}^0(\{\mathbf{l}_1^0, \mathbf{l}_2^0, \ldots, \mathbf{l}_N^0\})$ is produced. The values $\left\{l_{i,1}^0, l_{i,2}^0, \ldots, l_{i,n}^0\right\}$ of each individual $\mathbf{l}_i^k$ and each dimension d are randomly and uniformly distributed between the pre-specified lower initial parameter bound lb_d and the upper initial parameter bound ub_d.

$$\begin{aligned} l_{i,j}^0 &= lb_d + \text{rand} \cdot (ub_d - lb_d); \\ i &= 1, 2, \ldots, N; \quad d = 1, 2, \ldots, n. \end{aligned} \tag{5.13}$$

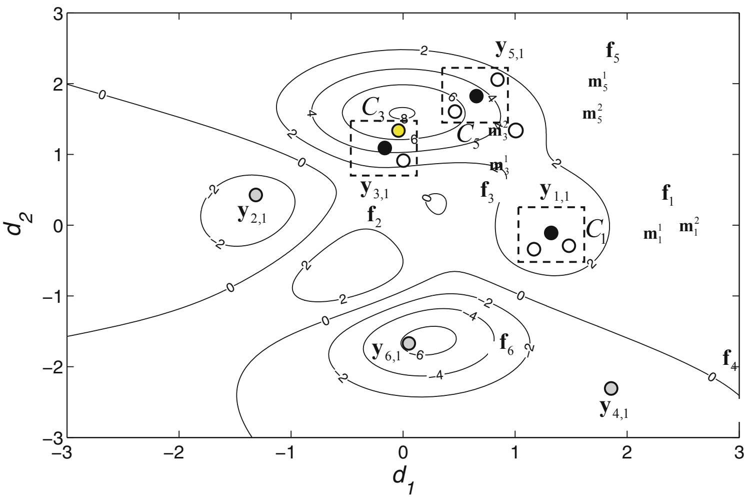


Fig. 5.5 Operation of the social procedure

In the evolution process, the solitary (A) and social (B) operations are iteratively applied until the number of iterations $k = gen$ has been reached. The complete LS procedure is illustrated in the Algorithm 1.

Algorithm 1. Locust Search (LS) algorithm

```
1:  Input: $F, L, g, N$ and $gen$
2:  Initialize $\mathbf{L}^0$ $(k = 0)$
3:  until $(k = gen)$
5:    $\mathbf{F} \leftarrow$ SolitaryOperation($\mathbf{L}^k$)            Solitary operator (3.1)
6:    $\mathbf{L}^{k+1} \leftarrow$ SocialOperation($\mathbf{L}^k$, $\mathbf{F}$)   Social operator (3.2)
8:    $k=k+1$
7:  end until
```

5.3.4 *Discussion About the LS Algorithm*

Evolutionary algorithms (EA) have been widely employed for solving complex optimization problems. These methods are found to be more powerful than conventional methods based on formal logics or mathematical programming [30]. In an EA algorithm, search agents have to decide whether to explore unknown search positions or to exploit already tested positions in order to improve their solution quality. Pure exploration degrades the precision of the evolutionary process but increases its capacity to find new potentially solutions. On the other hand, pure exploitation allows refining existent solutions but adversely drives the process to local optimal solutions. Therefore, the ability of an EA to find a global optimal solution depends on its capacity to find a good balance between the exploitation of found-so-far elements and the exploration of the search space [31]. So far, the exploration-exploitation dilemma has been an unsolved issue within the framework of evolutionary algorithms.

Most of swarm algorithms and other evolutionary algorithms tend to exclusively concentrate the individuals in the current best positions. Under such circumstances, these algorithms seriously limit their exploration-exploitation capacities.

Different to most of existent evolutionary algorithms, in the proposed approach, the modeled behavior explicitly avoids the concentration of individuals in the current best positions. Such fact allows not only to emulate in a better realistic way the cooperative behavior of the locust colony, but also to incorporate a computational mechanism to avoid critical flaws commonly present in the popular PSO and DE algorithms, such as the premature convergence and the incorrect exploration-exploitation balance.

5.4 Experimental Results

A comprehensive set of 13 functions, collected from Refs. [32–37], has been used to test the performance of the proposed approach. Tables 5.5 and 5.6 in the Appendix present the benchmark functions used in our experimental study. Such functions are classified into two different categories: Unimodal test functions (Table 5.5) and multimodal test functions (Table 5.6). In these tables, n is the dimension of function, f_{opt} is the minimum value of the function, and $\mathbf{S}$ is a subset of R^n. The optimum location ($\mathbf{x}_{opt}$) for functions in Tables 5.5 and 5.6, are in $[0]^n$, except for f_5, f_{12}, f_{13} with $\mathbf{x}_{opt}$ in $[1]^n$ and f_8 in $[420.96]^n$. A detailed description of optimum locations is given in Table 5.6 of the Appendix.

5.4.1 Performance Comparison

We have applied the LS algorithm to 13 functions whose results have been compared to those produced by the Particle Swarm Optimization (PSO) method [3] and the Differential Evolution (DE) algorithm [11]. These are considered as the most popular algorithms for many optimization applications. In all comparisons, the population has been set to 40 ($N = 40$) individuals. The maximum iteration number for all functions has been set to 1000. Such stop criterion has been selected to maintain compatibility to similar works reported in the literature [34, 35].

The parameter settings for each of the algorithms in the comparison are described as follows:

1. PSO: In the algorithm, $c_1 = c_2 = 2$ while the inertia factor (ω) is decreasing linearly from 0.9 to 0.2.
2. DE: The DE/Rand/1 scheme is employed. The parameter settings follow the instructions in [11]. The crossover probability is $CR = 0.9$ and the weighting factor is $F = 0.8$.
3. In LS, F and L are set to 0.6 and L, respectively. Besides, g is fixed to 20 ($N/2$) whereas gen and N are configured to 1000 and 40, respectively. Once these parameters have been determined experimentally, they are kept for all experiments in this section.

Uni-Modal Test Functions
Functions f_1 to f_7 are unimodal functions. The results for unimodal functions, over 30 runs, are reported in Table 5.1 considering the following performance indexes: the average best-so-far solution (ABS), the median of the best solution in the last iteration (MBS) and the standard deviation (SD). According to this table, LS provides better results than PSO and DE for all functions. In particular this test yields the largest difference in performance which is directly related to a better trade-off between exploration and exploitation produced by LS operators.

A non-parametric statistical significance proof known as the Wilcoxon's rank sum test for independent samples [38, 39] has been conducted with an 5% significance level, over the "average best-so-far" data of Table 5.1. Table 5.2 reports the p-values produced by Wilcoxon's test for the pair-wise comparison of the "average best so-far" of two groups. Such groups are formed by LS versus PSO and LS versus DE. As a null hypothesis, it is assumed that there is no significant difference between mean values of the two algorithms. The alternative hypothesis considers a significant difference between the "average best-so-far" values of both approaches. All p-values reported in the table are less than 0.05 (5% significance level) which is a strong evidence against the null hypothesis, indicating that the LS results are statistically significant and that it has not occurred by coincidence (i.e. due to the normal noise contained in the process).

Multimodal Test Functions
Multimodal functions have many local minima, being the most difficult to optimize. For multimodal functions, the final results are more important since they reflect the algorithm's ability to escape from poor local optima and locate a near-global optimum. We have done experiments on f_8 to f_{13} where the number of local minima increases exponentially as the dimension of the function increases. The dimension of these functions is set to 30. The results are averaged over 30 runs, reporting the performance indexes in Table 5.3 as follows: the average best-so-far solution (ABS), the median of the best solution in the last iteration (MBS) and the standard deviation (SD). Likewise, p-values of the Wilcoxon signed-rank test of 30 independent runs are listed in Table 5.4.

For f_9, f_{10}, f_{11} and f_{12}, LS yields a much better solution than the others. However, for functions f_8 and f_{13}, LS produces similar results to DE. The Wilcoxon rank test results, presented in Table 5.4, show that LS performed better than PSO and DE

Table 5.1 Minimization result of benchmark functions in Table 5.5 with $n = 30$

		PSO	DE	LS
f_1	ABS	1.66×10^{-1}	6.27×10^{-3}	4.55×10^{-4}
	MBS	0.23	5.85×10^{-3}	2.02×10^{-4}
	SD	3.79×10^{-1}	1.68×10^{-1}	6.98×10^{-4}
f_2	ABS	4.83×10^{-1}	2.02×10^{-1}	5.41×10^{-3}
	MBS	0.53	1.96×10^{-1}	5.15×10^{-3}
	SD	1.59×10^{-1}	0.66	1.45×10^{-2}
f_3	ABS	2.75	5.72×10^{-1}	1.61×10^{-3}
	MBS	3.16	6.38×10^{-1}	1.81×10^{-3}
	SD	1.01	0.15	1.32×10^{-3}
f_4	ABS	1.84	0.11	1.05×10^{-2}
	MBS	1.79	0.10	1.15×10^{-2}
	SD	0.87	0.05	6.63×10^{-3}
f_5	ABS	3.07	2.39	4.11×10^{-2}
	MBS	3.03	2.32	3.65×10^{-2}
	SD	0.42	0.36	2.74×10^{-3}
f_6	ABS	6.36	6.51	5.88×10^{-2}
	MBS	6.19	6.60	5.17×10^{-2}
	SD	0.74	0.87	1.67×10^{-2}
f_7	ABS	6.14	0.12	2.71×10^{-2}
	MBS	2.76	0.14	1.10×10^{-2}
	SD	0.73	0.02	1.18×10^{-2}

Maximum number of iterations = 1000

Table 5.2 p-values produced by Wilcoxon's test comparing LS versus PSO and DE over the "average best-so-far" values from Table 5.1

LS versus	PSO	DE
f_1	1.83×10^{-4}	1.73×10^{-2}
f_2	3.85×10^{-3}	1.83×10^{-4}
f_3	1.73×10^{-4}	6.23×10^{-3}
f_4	2.57×10^{-4}	5.21×10^{-3}
f_5	4.73×10^{-4}	1.83×10^{-3}
f_6	6.39×10^{-5}	2.15×10^{-3}
f_7	1.83×10^{-4}	2.21×10^{-3}

Table 5.3 Minimization result of benchmark functions in Table 5.5 with $n = 30$

		PSO	DE	LS
f_8	ABS	-6.7×10^3	-1.26×10^4	-1.26×10^4
	MBS	-5.4×10^3	-1.24×10^4	-1.23×10^4
	SD	6.3×10^2	3.7×10^2	1.1×10^2
f_9	ABS	14.8	4.01×10^{-1}	2.49×10^{-3}
	MBS	13.7	2.33×10^{-1}	3.45×10^{-3}
	SD	1.39	5.1×10^{-2}	4.8×10^{-4}
f_{10}	ABS	14.7	4.66×10^{-2}	2.15×10^{-3}
	MBS	18.3	4.69×10^{-2}	1.33×10^{-3}
	SD	1.44	1.27×10^{-2}	3.18×10^{-4}
f_{11}	ABS	12.01	1.15	1.47×10^{-4}
	MBS	12.32	0.93	3.75×10^{-4}
	SD	3.12	0.06	1.48×10^{-5}
f_{12}	ABS	6.87×10^{-1}	3.74×10^{-1}	5.58×10^{-3}
	MBS	4.66×10^{-1}	3.45×10^{-1}	5.10×10^{-3}
	SD	7.07×10^{-1}	1.55×10^{-1}	4.18×10^{-4}
f_{13}	ABS	1.87×10^{-1}	1.81×10^{-2}	1.78×10^{-2}
	MBS	1.30×10^{-1}	1.91×10^{-2}	1.75×10^{-2}
	SD	5.74×10^{-1}	1.66×10^{-2}	1.64×10^{-3}

Maximum number of iterations = 1000

Table 5.4 *p*-values produced by Wilcoxon's test comparing LS versus PSO and DE over the "average best-so-far" values from Table 5.3

LS versus	PSO	DE
f_8	1.83×10^{-4}	0.061
f_9	1.17×10^{-4}	2.41×10^{-4}
f_{10}	1.43×10^{-4}	3.12×10^{-3}
f_{11}	6.25×10^{-4}	1.14×10^{-3}
f_{12}	2.34×10^{-5}	7.15×10^{-4}
f_{13}	4.73×10^{-4}	0.071

considering the four problems $f_9 - f_{12}$, whereas, from a statistical viewpoint, there is not difference in results between LS and DE for f_8 and f_{13}.

5.5 Conclusions

In this paper, a novel swarm algorithm, called the Locust Search (LS) has been proposed for solving optimization tasks. The LS algorithm is based on the simulation of the behavior presented in swarms of locusts. In the proposed algorithm, individuals emulate a group of locusts which interact to each other based on the biological laws of the cooperative swarm. The algorithm considers two different behaviors: solitary and social. Depending on the behavior, each individual is conducted by a set of evolutionary operators which mimic the different cooperative behaviors that are typically found in the swarm.

Different to most of existent evolutionary algorithms, in the proposed approach, the modeled behavior explicitly avoids the concentration of individuals in the current best positions. Such fact allows not only to emulate in a better realistic way the cooperative behavior of the locust colony, but also to incorporate a computational mechanism to avoid critical flaws commonly present in the popular PSO and DE algorithms, such as the premature convergence and the incorrect exploration-exploitation balance.

LS has been experimentally tested considering a suite of 13 benchmark functions. The performance of LS has been also compared to the following algorithms: the Particle Swarm Optimization method (PSO) [3], and the Differential Evolution (DE) algorithm [11]. Results have confirmed an acceptable performance of the proposed method in terms of the solution quality for all tested benchmark functions.

The LS remarkable performance is associated with two different reasons: (i) the solitary operator allows a better particle distribution in the search space, increasing the algorithm's ability to find the global optima; and (ii) the use of the social operation, provides of a simple exploitation operator that intensifies the capacity of finding better solutions during the evolution process.

Appendix: List of Benchmark Functions

In Table 5.5, n is the dimension of function, f_{opt} is the minimum value of the function, and $\mathbf{S}$ is a subset of R^n. The optimum location ($\mathbf{x}_{opt}$) for functions in Table 5.5 is in $[0]^n$, except for f_5 with $\mathbf{x}_{opt}$ in $[1]^n$.

The optimum location ($\mathbf{x}_{opt}$) for functions in Table 5.6, are in $[0]^n$, except for f_8 in $[420.96]^n$ and f_{12}–f_{13} in $[1]^n$.

Table 5.5 Unimodal test functions

Test function	**S**	f_{opt}
$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	0
$f_2(\mathbf{x}) = \sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	$[-10, 10]^n$	0
$f_3(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	$[-100, 100]^n$	0
$f_4(\mathbf{x}) = \max_i\{\lvert x_i\rvert,\ 1 \le i \le n\}$	$[-100, 100]^n$	0
$f_5(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$[-30, 30]^n$	0
$f_6(\mathbf{x}) = \sum_{i=1}^{n} (x_i + 0.5)^2$	$[-100, 100]^n$	0
$f_7(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + rand(0, 1)$	$[-1.28, 1.28]^n$	0

Table 5.6 Multimodal test functions

Test function	**S**	f_{opt}
$f_8(\mathbf{x}) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]^n$	$-418.98{*}n$
$f_9(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12, 5.12]^n$	0
$f_{10}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20$	$[-32, 32]^n$	0
$f_{11}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^n$	0
$f_{12}(\mathbf{x}) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{x_i + 1}{4} \quad u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	$[-50, 50]^n$	0
$f_{13}(\mathbf{x}) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	$[-50, 50]^n$	0

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press Inc., New York (1999)
2. Kassabalidis, I., El-Sharkawi, M.A., Marks, R.J. II, Arabshahi, P., Gray, A.A.: Swarm intelligence for routing in communication networks. In: Global Telecommunications Conference, GLOBECOM '01, 6, IEEE, pp. 3613–3617 (2001)
3. Kennedy, J., & Eberhart, R.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (Dec. 1995)
4. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report-TR06. Engineering Faculty, Computer Engineering Department, Erciyes University (2005)
5. Passino, K.M.: Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst. Mag. **22**(3), 52–67 (2002)
6. Hossein, A., Hossein-Alavi, A.: Krill herd: a new bio-inspired optimization algorithm. Commun. Nonlinear Sci. Numer. Simul. **17**, 4831–4845 (2012)
7. Yang, X.S.: Engineering optimization: an introduction with metaheuristic applications. Wiley, New York (2010)
8. Yang, X.S., Deb, S.: Proceedings of World Congress on Nature & Biologically Inspired Computing. IEEE Publications, India, pp. 210–214 (2009)
9. Cuevas, E., Cienfuegos, M., Zaldívar, D., Pérez-Cisneros, M.: A swarm optimization algorithm inspired in the behavior of the social-spider. Expert Syst. Appl. **40**(16), 6374–6384 (2013)
10. Cuevas, E., González, M., Zaldivar, D., Pérez-Cisneros, M., García, G.: An algorithm for global optimization inspired by collective animal behaviour. *Discrete Dynamics in Nature and Society* 2012, art. no. 638275
11. Storn, R., Price, K.: Differential evolution—a simple and efficient adaptive scheme for global optimisation over continuous spaces. Technical Report TR-95–012, ICSI, Berkeley, CA (1995)
12. Bonabeau, E.: Social insect colonies as complex adaptive systems. Ecosystems **1**, 437–443 (1998)
13. Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B., Tian, Q.: Self-adaptive learning based particle swarm optimization. Inf. Sci. **181**(20), 4515–4538 (2011)
14. Tvrdík, Josef: Adaptation in differential evolution: a numerical comparison. Appl. Soft Comput. **9**(3), 1149–1155 (2009)
15. Wang, H., Sun, H., Li, C., Rahnamayan, S., Jeng-shyang, P.: Diversity enhanced particle swarm optimization with neighborhood. Inf. Sci. **223**, 119–135 (2013)
16. Gong, Wenyin, Fialho, Álvaro, Cai, Zhihua, Li, Hui: Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. Inf. Sci. **181**(24), 5364–5386 (2011)
17. Gordon, D.: The organization of work in social insect colonies. Complexity **8**(1), 43–46 (2003)
18. Kizaki, Shinya, Katori, Makoto: A Stochastic lattice model for locust outbreak. Phys. A **266**, 339–342 (1999)
19. Rogers, Stephen M., Cullen, Darron A., Anstey, Michael L., Burrows, Malcolm, Dodgson, Tim, Matheson, Tom, Ott, Swidbert R., Stettin, Katja, Sword, Gregory A., Despland, Emma, Simpson, Stephen J.: Rapid behavioural gregarization in the desert locust, *Schistocerca gregaria* entails synchronous changes in both activity and attraction to conspecifics. J. Insect Physiol. **65**, 9–26 (2014)
20. Topaz, C.M., Bernoff, A.J., Logan, S., Toolson, W.: A model for rolling swarms of locusts. Eur. Phys. J. Special Topics **157**, 93–109 (2008)
21. Topaz, C.M., D'Orsogna, M.R., Edelstein-Keshet, L., Bernoff, A.J.: Locust dynamics: behavioral phase change and swarming. *PLOS Computational Biology* **8**(8), 1–11
22. Oster, G., Wilson, E.: Caste and Ecology in the Social Insects. N.J. Princeton University Press, Princeton (1978)
23. Hölldobler, B., Wilson, E.O.: Journey to the Ants: A Story of Scientific Exploration (1994). ISBN 0-674-48525-4
24. Hölldobler, B., Wilson, E.O.: The Ants. Harvard University Press (1990). ISBN 0-674-04075-9

25. Tanaka, Seiji, Nishide, Yudai: Behavioral phase shift in nymphs of the desert locust, *Schistocerca gregaria*: Special attention to attraction/avoidance behaviors and the role of serotonin. J. Insect Physiol. **59**, 101–112 (2013)
26. Gaten, Edward, Huston, Stephen J., Dowse, Harold B., Matheson, Tom: Solitary and gregarious locusts differ in circadian rhythmicity of a visual output neuron. J. Biol. Rhythms **27**(3), 196–205 (2012)
27. Benaragama, Indika, Gray, John R.: Responses of a pair of flying locusts to lateral looming visual stimuli. J. Comp. Physiol. A. **200**(8), 723–738 (2014)
28. Michael G. Sergeev, Distribution patterns of grasshoppers and their kin in the boreal zone, vol. 2011, Article ID 324130, 9 pages (2011)
29. Ely, S.O., Njagi, P.G.N., Bashir, M.O., El-Amin, S.E.-T., Diel, A.H.: Behavioral activity patterns in adult solitarious desert locust, *Schistocerca gregaria* (Forskål). Psyche Volume 2011, Article ID 459315, 9 (2011)
30. Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, Beckington (2008)
31. Cuevas, E., Echavarría, A., Ramírez-Ortegón, M.A.: An optimization algorithm inspired by the States of Matter that improves the balance between exploration and exploitation. Appl. Intell. **40**(2), 256–272 (2014)
32. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J. Global Optim. **31**(4), 635–672 (2005)
33. Chelouah, R., Siarry, P.: A continuous genetic algorithm designed for the global optimization of multimodal functions. J. Heurist. **6**(2), 191–213 (2000)
34. Herrera, F., Lozano, M., Sánchez, A.M.: A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. Int. J. Intell. Syst. **18**(3), 309–338 (2003)
35. Laguna, M., Martí, R.: Experimental testing of advanced scatter search designs for global optimization of multimodal functions. J. Global Optim. **33**(2), 235–255 (2005)
36. Lozano, M., Herrera, F., Krasnogor, N., Molina, D.: Real-coded memetic algorithms with crossover hill-climbing. Evol. Comput. **12**(3), 273–302 (2004)
37. Moré, J.J., Garbow, B.S., Hillstrom, K.E.: Testing unconstrained optimization software. ACM Trans. Math. Softw. **7**(1), 17–41 (1981)
38. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics **1**, 80–83 (1945)
39. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special session on real parameter optimization. J. Heurist. (2008). https://doi.org/10.1007/s10732-008-9080-4

Chapter 6
A Swarm Algorithm Inspired by the Collective Animal Behavior

Abstract In this chapter, a swarm algorithm for global optimization called the Collective Animal Behavior (CAB) is introduced. The algorithm is based on animal groups, such as schools of fish, flocks of birds, swarms of locusts, and herds of wildebeest, that exhibit a variety of behaviors including swarming about a food source, milling around a central location or migrating over large distances in aligned groups. These collective behaviors are often advantageous to groups, allowing them to increase their harvesting efficiency, to follow better migration routes, to improve their aerodynamic and to avoid predation. In the presented algorithm in this chapter, the searcher agents emulate a group of animals which interact to each other based on the biological laws of collective motion. The optimization method presented in this chapter has been compared to other well-known optimization algorithms. The results, experiments and practical examples confirm the high performance of the presented method to find a global optimum of several benchmark functions.

6.1 Introduction

Global Optimization (GO) has yielded remarkable applications to many areas of science, engineering, economics and others through mathematical modelling [1]. In general, the goal is to find a global optimum of an objective function which has been defined over a given search space. Global optimization algorithms are usually broadly divided into deterministic and metaheuristic [2]. Since deterministic methods only provide a theoretical guarantee of locating a local minimum of the objective function, they often face great difficulties in solving global optimization problems [3]. On the other hand, metaheuristic or swarm methods are usually faster in locating a global optimum [4]. They virtuously adapt to black-box formulations and extremely ill-behaved functions, whereas deterministic methods usually require some theoretical assumptions about the problem formulation and its analytical properties (such as Lipschitz continuity) [5].

Several metaheuristic algorithms have been developed by a combination of rules and randomness mimicking several phenomena. Such phenomena include evolutionary processes e.g. the evolutionary algorithm proposed by Fogel et al. [6], De Jong

© Springer Nature Switzerland AG 2020
E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_6

[7], and Koza [8], the genetic algorithm (GA) proposed by Holland [9] and Goldberg [10] and the artificial immune systems proposed by De Castro et al. [11]. On the other hand, physical processes consider the simulated annealing proposed by Kirkpatrick et al. [12], the electromagnetism-like algorithm proposed by İlker et al. [13], the gravitational search algorithm proposed by Rashedi et al. [14] and the musical process of searching for a perfect state of harmony, which has been proposed by Geem et al. [15], Lee and Geem [16] and Geem [17].

Many studies have been inspired by animal behavior phenomena for developing optimization techniques. For instance, the Particle swarm optimization (PSO) algorithm which models the social behavior of bird flocking or fish schooling [18]. PSO consists of a swarm of particles which move towards best positions, seen so far, within a searchable space of possible solutions. Another behavior-inspired approach is the Ant Colony Optimization (ACO) algorithm proposed by Dorigo et al. [19], which simulates the behavior of real ant colonies. Main features of the ACO algorithm are the distributed computation, the positive feedback and the constructive greedy search. Recently, a new swarm approach which is based on the animal behavior while hunting has been proposed in [20]. Such algorithm considers hunters as search positions and preys as potential solutions.

Just recently, the concept of individual-organization [21, 22] has been widely referenced to understand collective behavior of animals. The central principle of individual-organization is that simple repeating interactions between individuals can produce complex behavioral patterns at group level [21, 23, 24]. Such inspiration comes from behavioral patterns previously seen in several animal groups. Examples include ant pheromone trail networks, aggregation of cockroaches and the migration of fish schools, all of which can be accurately described in terms of individuals following simple sets of rules [25]. Some examples of these rules [24, 26] are keeping the current position (or location) for best individuals, local attraction or repulsion, random movements and competition for the space within of a determined distance.

On the other hand, new studies [27–29] have also shown the existence of collective memory in animal groups. The presence of such memory establishes that the previous history of the group structure influences the collective behavior exhibited in future stages. According to such principle, it is possible to model complex collective behaviors by using simple individual rules and configuring a general memory.

In this chapter, a new optimization algorithm inspired by the collective animal behavior is presented. In this algorithm, the searcher agents emulate a group of animals that interact to each other based on simple behavioral rules which are modeled as mathematical operators. Such operations are applied to each agent considering that the complete group has a memory storing their own best positions seen so far, by using a competition principle. The presented approach has been compared to other well-known optimization methods. The results confirm a high performance of the presented method for solving various benchmark functions.

This chapter is organized as follows. In Sect. 6.2, we introduce the basic biologic aspects of the algorithm. In Sect. 6.3, the novel CAB algorithm and its characteristics are both described. Section 6.4 presents the experimental results and the comparative study. Finally, in Sect. 6.5, the conclusions are discussed.

6.2 Biological Fundamentals

The remarkable collective behavior of organisms such as swarming ants, schooling fish and flocking birds has long captivated the attention of naturalists and scientists. Despite a long history of scientific research, the relationship between individuals and group-level properties has just recently begun to be deciphered [30].

Grouping individuals often have to make rapid decisions about where to move or what behavior to perform in uncertain and dangerous environments. However, each individual typically has only a relatively local sensing ability [31]. Groups are, therefore, often composed of individuals that differ with respect to their informational status and individuals are usually not aware of the informational state of others [32], such as whether they are knowledgeable about a pertinent resource or about a threat.

Animal groups are based on a hierarchic structure [33] which considers different individuals according to a fitness principle called Dominance [34] which is the domain of some individuals within a group that occurs when competition for resources leads to confrontation. Several studies [35, 36] have found that such animal behavior lead to more stable groups with better cohesion properties among individuals.

Recent studies have begun to elucidate how repeated interactions among grouping animals scale to collective behavior. They have remarkably revealed that collective decision-making mechanisms across a wide range of animal group types, from insects to birds (and even among humans in certain circumstances) seem to share similar functional characteristics [21, 25, 37]. Furthermore, at a certain level of description, collective decision-making by organisms shares essential common features such as a general memory. Although some differences may arise, there are good reasons to increase communication between researchers working in collective animal behavior and those involved in cognitive science [24].

Despite the variety of behaviors and motions of animal groups, it is possible that many of the different collective behavioral patterns are generated by simple rules followed by individual group members. Some authors have developed different models, one of them, known as the self-propelled particle (SPP) model, attempts to capture the collective behavior of animal groups in terms of interactions between group members which follow a diffusion process [38–41].

On other hand, following a biological approach, Couzin et al. [24, 25] have proposed a model in which individual animals follow simple rules of thumb: (1) keep the current position (or location) for best individuals; (2) move from or to nearby neighbors (local attraction or repulsion); (3) move randomly and (4) compete for the space within of a determined distance. Each individual thus admits three different movements: attraction, repulsion or random and holds two kinds of states: preserve the position or compete for a determined position. In the model, the movement, which is executed by each individual, is decided randomly (according to an internal motivation). On the other hand, the states follow a fixed criteria set.

The dynamical spatial structure of an animal group can be explained in terms of its history [36]. Despite such a fact, the majority of studies have failed in considering the existence of memory in behavioral models. However, recent research [27, 42] have also shown the existence of collective memory in animal groups. The presence of such memory establishes that the previous history of the group structure influences the collective behavior which is exhibited in future stages. Such memory can contain the location of special group members (the dominant individuals) or the averaged movements produced by the group.

According to these new developments, it is possible to model complex collective behaviors by using simple individual rules and setting a general memory. In this work, the behavioral model of animal groups inspires the definition of novel swarm operators which outline the CAB algorithm. A memory is incorporated to store best animal positions (best solutions) considering a competition-dominance mechanism.

6.3 Collective Animal Behavior Algorithm (CAB)

The CAB algorithm assumes the existence of a set of operations that resembles the interaction rules that model the collective animal behavior. In the approach, each solution within the search space represents an animal position. The "fitness value" refers to the animal dominance with respect to the group. The complete process mimics the collective animal behavior.

The approach in this chapter implements a memory for storing best solutions (animal positions) mimicking the aforementioned biologic process. Such memory is divided into two different elements, one for maintaining the best locations at each generation ($\mathbf{M}_g$) and the other for storing the best historical positions during the complete evolutionary process ($\mathbf{M}_h$).

6.3.1 Description of the CAB Algorithm

Following other swarm approaches, the CAB algorithm is an iterative process that starts by initializing the population randomly (generated random solutions or animal positions). Then, the following four operations are applied until a termination criterion is met (i.e. the iteration number NI):

1. Keep the position of the best individuals.
2. Move from or to nearby neighbors (local attraction and repulsion).
3. Move randomly.
4. Compete for the space within a determined distance (update the memory).

6.3.1.1 Initializing the Population

The algorithm begins by initializing a set A of N_p animal positions $\left(\mathbf{A} = \left\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{N_p}\right\}\right)$ Each animal position $\mathbf{a}_l$ is a D-dimensional vector containing parameter values to be optimized. Such values are randomly and uniformly distributed between the pre-specified lower initial parameter bound a_j^{low} and the upper initial parameter bound a_j^{high}.

$$a_{i,j} = a_j^{low} + \text{rand}\,(0, 1) \cdot \left(a_j^{high} - a_j^{low}\right);$$
$$j = 1, 2, \ldots, D;\; i = 1, 2, \ldots, N_p \tag{6.1}$$

with j and i being the parameter and individual indexes respectively. Hence, $a_{i,j}$ is the j-th parameter of the i-th individual.

All the initial positions A are sorted according to the fitness function (dominance) to form a new individual set $\mathbf{X} = \left\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\right\}$, so that we can choose the best B positions and store them in the memory $\mathbf{M}_g$ and $\mathbf{M}_h$. The fact that both memories share the same information is only allowed at this initial stage.

6.3.1.2 Keep the Position of the Best Individuals

Analogous to the biological metaphor, this behavioral rule, typical from animal groups, is implemented as a swarm operation in our approach. In this operation, the first B elements $(\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_B\})$, of the new animal position set A, are generated. Such positions are computed by the values contained inside the historical memory $\mathbf{M}_h$, considering a slight random perturbation around them. This operation can be modeled as follows:

$$\mathbf{a}_l = \mathbf{m}_h^l + \mathbf{v} \tag{6.2}$$

where $l \in \{1, 2, \ldots, B\}$ while $\mathbf{m}_h^l$ represents the l-element of the historical memory $\mathbf{M}_h$. $\mathbf{v}$ is a random vector with a small enough length.

6.3.1.3 Move from or to Nearby Neighbors

From the biological inspiration, animals experiment a random local attraction or repulsion according to an internal motivation. Therefore, we have implemented new swarm operators that mimic such biological pattern. For this operation, a uniform random number r_m is generated within the range [0, 1]. If r_m is less than a threshold H, a determined individual position is moved (attracted or repelled) considering the nearest best historical position within the group (i.e. the nearest position in $\mathbf{M}_h$); otherwise, it goes to the nearest best location within the group for the current generation (i.e. the nearest position in $\mathbf{M}_g$). Therefore such operation can be modeled as follows:

$$\mathbf{a}_i = \begin{cases} \mathbf{x}_i \pm r \cdot (\mathbf{m}_h^{nearest} - \mathbf{x}_i) \text{ with probability } H \\ \mathbf{x}_i \pm r \cdot (\mathbf{m}_g^{nearest} - \mathbf{x}_i) \text{ with probability } (1 - H) \end{cases} \tag{6.3}$$

where $i \in \{B+1, B+2, \ldots, N_p\}$, $\mathbf{m}_h^{nearest}$ and $\mathbf{m}_g^{nearest}$ represent the nearest elements of $\mathbf{M}_h$ and $\mathbf{M}_g$ to $\mathbf{x}_i$, while r is a random number.

6.3.1.4 Move Randomly

Following the biological model, under some probability P, one animal randomly changes its position. Such behavioral rule is implemented considering the next expression:

$$\mathbf{a}_i = \begin{cases} \mathbf{r} \text{ with probability } P \\ \mathbf{x}_i \text{ with probability } (1 - P) \end{cases} \tag{6.4}$$

being $i \in \{B+1, B+2, \ldots, N_p\}$ and $\mathbf{r}$ a random vector defined in the search space. This operator is similar to re-initialize the particle in a random position, as it is done by Eq. (6.1).

6.3.1.5 Compete for the Space Within of a Determined Distance (Update the Memory)

Once the operations to keep the position of the best individuals, such as moving from or to nearby neighbors and moving randomly, have all been applied to the all N_p animal positions, generating N_p new positions, it is necessary to update the memory $\mathbf{M}_h$.

In order to update de memory $\mathbf{M}_h$, the concept of dominance is used. Animals that interact within a group maintain a minimum distance among them. Such distance ρ depends on how aggressive the animal behaves [34, 42]. Hence, when two animals confront each other inside such distance, the most dominant individual prevails meanwhile other withdraw. Figure 6.1 depicts the process.

In the presented algorithm, the historical memory $\mathbf{M}_h$ is updated considering the following procedure:

1. The elements of $\mathbf{M}_h$ and $\mathbf{M}_g$ are merged into $\mathbf{M}_U$ $(\mathbf{M}_U = \mathbf{M}_h \cup \mathbf{M}_g)$.
2. Each element $\mathbf{m}_U^i$ of the memory $\mathbf{M}_U$ is compared pair-wise to remaining memory elements $(\{\mathbf{m}_U^1, \mathbf{m}_U^2, \ldots, \mathbf{m}_U^{2B-1}\})$. If the distance between both elements is less than ρ, the element getting a better performance in the fitness function prevails meanwhile the other is removed.

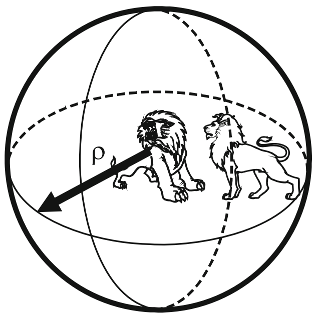


Fig. 6.1 Dominance concept as it is presented when two animals confront each other inside of a ρ distance

3. From the resulting elements of $\mathbf{M}_U$ (from step 2), it is selected the B best value to build the new $\mathbf{M}_h$.

Unsuitable values of ρ yield a lower convergence rate, a longer computational time, a larger function evaluation number, the convergence to a local maximum or to an unreliable solution. The ρ value is computed considering the following equation:

$$\rho = \frac{\prod_{j=1}^{D} (a_j^{high} - a_j^{low})}{10 \cdot D} \tag{6.5}$$

where a_j^{low} and a_j^{high} represent the pre-specified lower and upper bound of the j-parameter respectively, in an D-dimensional space.

6.3.1.6 Computational Procedure

The computational procedure for the presented algorithm can be summarized as follows:

Step 1 Set the parameters N_p, B, H, P and NI.
Step 2 Generate randomly the position set $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{N_p}\}$ using Eq. (6.1).
Step 3 Sort A according to the objective function (dominance) to build $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\}$.
Step 4 Choose the first B positions of $\mathbf{X}$ and store them into the memory $\mathbf{M}_g$.
Step 5 Update $\mathbf{M}_h$ according to Sect. 6.3.1.5 (during the first iteration: $\mathbf{M}_h = \mathbf{M}_g$).
Step 6 Generate the first B positions of the new solution set $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_B\}$. Such positions correspond to the elements of $\mathbf{M}_h$ making a slight random perturbation around them.
$\mathbf{a}_l = \mathbf{m}_h^l + \mathbf{v}$; being v a random vector of a small enough length.
Step 7 Generate the rest of the A elements using the attraction, repulsion and random movements.

```
for $i=B+1$: $N_p$
    if ( $r_1 < P$) then
    attraction and repulsion movement
        { if ( $r_2 < H$) then
            $\mathbf{a}_i = \mathbf{x}_i \pm r \cdot (\mathbf{m}_h^{nearest} - \mathbf{x}_i)$
          else if
            $\mathbf{a}_i = \mathbf{x}_i \pm r \cdot (\mathbf{m}_g^{nearest} - \mathbf{x}_i)$
        }
    else if
    random movement
        {
            $\mathbf{a}_i = \mathbf{r}$
        }
end for   where $r_1, r_2, r \in$ rand(0,1)
```

Step 8 If *NI* is completed, the process is finished; otherwise go back to step 3. The best value in $\mathbf{M}_h$ represents the global solution for the optimization problem.

6.4 Experimental Results

A comprehensive set of 23 functions, collected from Refs. [43–53], has been used to test the performance of the presented approach. Tables 6.1, 6.2, 6.3 present the benchmark functions used in the experimental study. Such functions are classified into three different categories: Unimodal test functions (Table 6.1), multimodal test functions (Table 6.2) and multimodal test functions with fix dimensions (Table 6.2). In these tables, n is the dimension of function, f_{opt} is the minimum value of the function, and S is a subset of R^n. The optimum location ($\mathbf{x}_{opt}$) for functions in Tables 6.1 and 6.2, are in $[0]^n$, except for f_5, f_{12} and f_{13} with $\mathbf{x}_{opt}$ in $[1]^n$ and f_8 in $[420.96]^n$. A detailed description of optimum locations is given in Table 6.4.

6.4.1 Effect of the CAB Parameters

To study the impact of parameters P and H (described on Sects. 6.3.1.3 and 6.3.1.4) over the performance of CAB, different values have been tested on 5 typical functions. The maximum number of iterations is set to 1000. N_p and B are fixed to 50 and 10 respectively. The mean best function values μ and the standard deviations (σ^2) of CAB, averaged over 30 runs, for the different values of P and H are listed in Tables 6.5, 6.6 respectively. The results suggest that a proper combination of different parameter

Table 6.1 Unimodal test functions

Test function	S	f_{opt}
$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	0
$f_2(\mathbf{x}) = \sum_{i=1}^{n} \|x_i\| + \prod_{i=1}^{n} \|x_i\|$	$[-10, 10]^n$	0
$f_3(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-100, 100]^n$	0
$f_4(\mathbf{x}) = \max_i \{\|x_i\|, 1 \leq i \leq n\}$	$[-100, 100]^n$	0
$f_5(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$[-30, 30]^n$	0
$f_6(\mathbf{x}) = \sum_{i=1}^{n} (x_i + 0.5)^2$	$[-100, 100]^n$	0
$f_7(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + rand(0, 1)$	$[-1.28, 1.28]^n$	0

values can improve the performance of CAB and the quality of solutions. Table 6.5 shows the results of an experiment which consist in fixing $H = 0.8$ and varying P from 0.5 to 0.9. On a second test, the experimental setup is swapped i.e. $P = 0.8$ and H varies from 0.5 to 0.9. The best results in the experiments are highlighted in Tables 6.5 and 6.6.

6.4.2 Performance Comparison

The CAB has been applied over the 23 functions comparing the results with those produced by Real Genetic Algorithm (RGA) [54], the PSO [18], the Gravitational Search Algorithm (GSA) [55] and the Differential Evolution method (DE) [56]. In all cases, population size is set to 50. The maximum iteration number is 1000 for functions in Tables 6.1 and 6.2 and 500 for functions in Table 6.3. Such stop criteria have been chosen as to keep compatibility to similar works in [14] and [57].

The parameter settings for each of the algorithms in the comparison are described as follows:

1. RGA: According to [54], the approach uses arithmetic crossover, Gaussian mutation and roulette wheel selection. The crossover and mutation probabilities have been set to 0.3 and 0.1 respectively.
2. PSO: In the algorithm, $c_1 = c_2 = 2$ while the inertia factor (ω) is decreasing linearly from 0.9 to 0.2.

Table 6.2 Multimodal test functions

Test function	S	f_{opt}
$f_8(\mathbf{x}) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]^n$	$-418.98*n$
$f_9(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12, 5.12]^n$	0
$f_{10}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20$	$[-32, 32]^n$	0
$f_{11}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^n$	0
$f_{12}(\mathbf{x}) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$; $y_i = 1 + \frac{x_i+1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	$[-50, 50]^n$	0
$f_{13}(\mathbf{x}) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	$[-50, 50]^n$	0

3. In GSA, G_0 is set to 100 and α is set to 20; T is the total number of iterations (set to 1000 for functions $f_1 - f_{13}$ and to 500 for functions $f_{14} - f_{23}$). Besides, K_0 is set to 50 (total number of agents) and is decreased linearly to 1. Such values have been found as the best configuration set according to [55].
4. DE: The DE/Rand/1 scheme is employed. The parameter settings follow the instructions in [56]. The crossover probability is $CR = 0.9$ and the weighting factor is $F = 0.8$.

Unimodal test functions

Functions f_1 to f_7 are unimodal functions. The results for unimodal functions, over 30 runs, are reported in Table 6.7 considering the following performance indexes: the average best-so-far solution, the average mean fitness function and the median of the best solution in the last iteration. The best result for each function is boldfaced. According to this table, CAB provides better results than RGA, PSO, GSA and DE for all functions. In particular this test yields the largest difference in performance which

Table 6.3 Multimodal test functions with fix dimensions

Test function	S	f_{opt}
$f_{14}(\mathbf{x}) = \left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}\right)^{-1}$ $a_{ij} = \begin{pmatrix} -32, -16, 0, 16, 32, -32, \ldots, 0, 16, 32 \\ -32, -32, -32, -32, 16, \ldots, 32, 32, 32 \end{pmatrix}$	$[-65.5, 65.5]^2$	1
$f_{15}(\mathbf{x}) = \sum_{i=1}^{11} \left[a_i - \frac{x_i(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4}\right]^2$ $\mathbf{a} = [0.1957, 0.1947, 0.1735, 0.1600, 0.0844,$ $0.0627, 0.0456, 0.0342, 0.0342, 0.0235, 0.0246]$ $\mathbf{b} = [0.25, 0.5, 1, 2, 4, 6, 8, 10, 12, 14, 16]$	$[-5, 5]^4$	0.00030
$f_{16}(x_1, x_2) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	$[-5, 5]^2$	−1.0316
$f_{17}(x_1.x_2) =$ $\left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	$x_1 \in$ $[-5, 10]$ $x_2 \in [0, 15]$	0.398
$f_{18}(x_1, x_2) = \Big[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2$ $-14x_2 + 6x_1x_2 + 3x_2^2\Big] \times \Big[30 + (2x_1 - 3x_2)^2$ $\times(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\Big]$	$[-5, 5]^2$	3
$f_{19}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$ $\mathbf{a} = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 30 \end{bmatrix}$ $\mathbf{c} = [1, 1.2, 3, 3.2]$ $\mathbf{P} = \begin{bmatrix} 0.3689 & 0.117 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.0381 & 0.5743 & 0.8828 \end{bmatrix}$	$[0, 1]^3$	−3.86

(continued)

Table 6.3 (continued)

Test function	S	f_{opt}
$f_{20}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$ $\mathbf{a} = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 17 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}$ $\mathbf{c} = [1, 1.2, 3, 3.2]$ $\mathbf{P} = \begin{bmatrix} 0.131 & 0.169 & 0.556 & 0.012 & 0.828 & 0.588 \\ 0.232 & 0.413 & 0.830 & 0.373 & 0.100 & 0.999 \\ 0.234 & 0.141 & 0.352 & 0.288 & 0.304 & 0.665 \\ 0.404 & 0.882 & 0.873 & 0.574 & 0.109 & 0.038 \end{bmatrix}$	$[0, 1]^6$	-3.32
$f_{21}(\mathbf{x}) = -\sum_{i=1}^{5} \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i\right]^{-1}$ $\mathbf{a} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix}$ $\mathbf{c} = [0.1, 0.2, 0.2, 0.4, 0.4, 0.6, 0.3, 0.7, 0.5, 0.5]$	$[0, 10]^4$	-10.1532
$f_{22}(\mathbf{x}) = -\sum_{i=1}^{7} \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i\right]^{-1}$ a and c, equal to f_{21}	$[0, 10]^4$	-10.4028
$f_{23}(\mathbf{x}) = -\sum_{i=1}^{7} \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i\right]^{-1}$ a and c, equal to f_{21}.	$[0, 10]^4$	-10.5363

Table 6.4 Optimum locations of Table 6.3

Test function	$\mathbf{x}_{opt}$	Test function	$\mathbf{x}_{opt}$
f_{14}	(−32, 32)	f_{19}	(0.114, 0.556, 0.852)
f_{15}	(0.1928, 0.1908, 0.1231, 0.1358)	f_{20}	(0.201, 0.15, 0.477, 0.275, 0.311, 0.657)
f_{16}	(0.089, −0.71), (−0.0089, 0.712)	f_{21}	5 local minima in a_{ij}, $i = 1, \ldots, 5$
f_{17}	(−3.14, 12.27), (3.14, 2.275), (9.42, 2.42)	f_{22}	7 local minima in a_{ij}, $i = 1, \ldots, 7$
f_{18}	(0, −1)	f_{23}	10 local minima in a_{ij}, $i = 1, \ldots, 10$

is directly related to a better trade-off between exploration and exploitation produced by CAB operators. Moreover, the good convergence rate of CAB can be observed from Fig. 6.2 According to this figure, CAB tends to find the global optimum faster than other algorithms and yet offering the highest convergence rate.

A non-parametric statistical significance proof known as the Wilcoxon's rank sum test for independent samples [58, 59] has been conducted with 5% significance level, over the "average best-so-far" data of Table 6.7. Table 6.8 reports the p-values produced by Wilcoxon's test for the pair-wise comparison of the "average best so-far" of four groups. Such groups are formed by CAB versus RGA, CAB versus PSO, CAB versus GSA and CAB versus DE. As a null hypothesis, it is assumed that there is no significant difference between mean values of the two algorithms. The alternative hypothesis considers a significant difference between the "average best-so-far" values of both approaches. All p-values reported in the table are less than 0.05 (5% significance level) which is a strong evidence against the null hypothesis, indicating that the CAB results are statistically significant and that it has not occurred by coincidence (i.e. due to the normal noise contained in the process).

Multimodal test functions

Multimodal functions have many local minima, being the most difficult to optimize. For multimodal functions, the final results are more important since they reflect the algorithm's ability to escape from poor local optima and locate a near-global optimum. Experiments have been done on f_8 to f_{13} where the number of local minima increases exponentially as the dimension of the function increases. The dimension of these functions is set to 30. The results are averaged over 30 runs, reporting the performance indexes in Table 6.9 as follows: the average best-so-far solution, the average mean fitness function and the median of the best solution in the last iteration (the best result for each function is highlighted) Likewise, p-values of the Wilcoxon signed-rank test of 30 independent runs are listed in Table 6.10. For f_9, f_{10}, f_{11} and f_{12}, CAB yields a much better solution than the others. However, for functions f_8 and f_{13}, CAB produces similar results to RGA and GSA respectively. The Wilcoxon rank test results, presented in Table 6.10, show that CAB performed better than

Table 6.5 Results of CAB with variant values of parameter P over 5 typical functions, with $H = 0.8$

Function	n	$P = 0.5$, $\mu(\sigma^2)$	$P = 0.6$, $\mu(\sigma^2)$	$P = 0.7$, $\mu(\sigma^2)$	$P = 0.8$, $\mu(\sigma^2)$	$P = 0.9$, $\mu(\sigma^2)$
f_1	30	2.63×10^{-11} (2.13×10^{-12})	1.98×10^{-17} (6.51×10^{-18})	1.28×10^{-23} (3.54×10^{-24})	$\mathbf{2.33 \times 10^{-29}}$ $\mathbf{(4.41 \times 10^{-30})}$	4.53×10^{-23} (5.12×10^{-24})
f_3	30	5.71×10^{-13} (1.11×10^{-14})	7.78×10^{-19} (1.52×10^{-20})	4.47×10^{-27} (3.6×10^{-28})	$\mathbf{7.62 \times 10^{-31}}$ $\mathbf{(4.23 \times 10^{-32})}$	3.42×10^{-26} (3.54×10^{-27})
f_5	30	5.68×10^{-11} (2.21×10^{-12})	1.54×10^{-17} (1.68×10^{-18})	5.11×10^{-22} (4.42×10^{-23})	$\mathbf{9.02 \times 10^{-28}}$ $\mathbf{(6.77 \times 10^{-29})}$	4.77×10^{-20} (1.94×10^{-21})
f_{10}	30	3.50×10^{-5} (3.22×10^{-6})	2.88×10^{-9} (3.28×10^{-10})	2.22×10^{-12} (4.21×10^{-13})	$\mathbf{8.88 \times 10^{-16}}$ $\mathbf{(3.49 \times 10^{-17})}$	1.68×10^{-11} (5.31×10^{-12})
f_{11}	30	1.57×10^{-2} (1.25×10^{-3})	1.14×10^{-6} (3.71×10^{-7})	2.81×10^{-8} (5.21×10^{-9})	$\mathbf{4.21 \times 10^{-10}}$ $\mathbf{(4.87 \times 10^{-11})}$	4.58×10^{-4} (6.92×10^{-5})

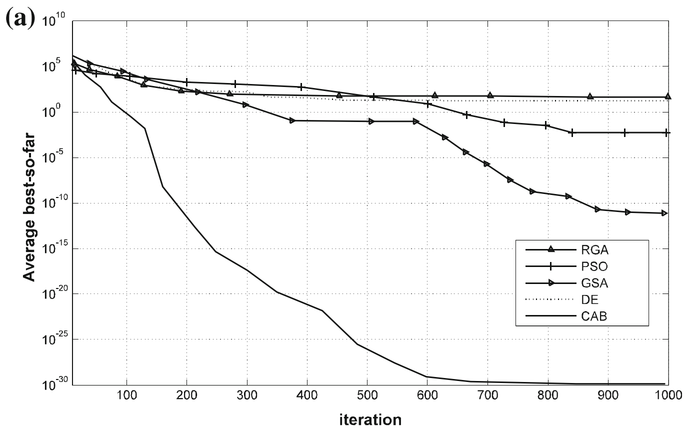
Table 6.6 Results of CAB with variant values of parameter H over 5 typical functions, with $P = 0.8$

Function	n	$P = 0.5$, $\mu(\sigma^2)$	$P = 0.6$, $\mu(\sigma^2)$	$P = 0.7$, $\mu(\sigma^2)$	$P = 0.8$, $\mu(\sigma^2)$	$P = 0.9$, $\mu(\sigma^2)$
f_1	30	2.23×10^{-10} (8.92×10^{-11})	3.35×10^{-18} (3.21×10^{-19})	3.85×10^{-22} (6.78×10^{-23})	2.33×10^{-29} (4.41×10^{-30})	4.72×10^{-21} (6.29×10^{-22})
f_3	30	5.71×10^{-10} (5.12×10^{-11})	3.24×10^{-18} (1.32×10^{-19})	6.29×10^{-27} (8.26×10^{-23})	7.62×10^{-31} (4.23×10^{-32})	5.41×10^{-22} (5.28×10^{-23})
f_5	30	8.80×10^{-9} (5.55×10^{-10})	6.72×10^{-21} (1.11×10^{-22})	1.69×10^{-23} (1.34×10^{-24})	9.02×10^{-28} (6.77×10^{-29})	7.39×10^{-21} (4.41×10^{-22})
f_{10}	30	2.88×10^{-4} (3.11×10^{-5})	3.22×10^{-10} (2.18×10^{-12})	1.23×10^{-14} (4.65×10^{-15})	8.88×10^{-16} (3.49×10^{-17})	5.92×10^{-7} (3.17×10^{-9})
f_{11}	30	1.81×10^{-4} (2.16×10^{-5})	2.89×10^{-6} (6.43×10^{-7})	2.36×10^{-7} (3.75×10^{-4})	4.21×10^{-10} (4.87×10^{-11})	3.02×10^{-4} (4.37×10^{-6})

Table 6.7 Minimization result of benchmark functions in Table 6.1 with $n = 30$

		RGA	PSO	GSA	DE	CAB
f_1	Average best so-far	23.13	1.8×10^{-3}	7.3×10^{-11}	11.21	$\mathbf{2.3 \times 10^{-29}}$
	Median best so-far	21.87	1.2×10^{-3}	7.1×10^{-11}	13.21	$\mathbf{1.1 \times 10^{-20}}$
	Average mean fitness	23.45	1.2×10^{-2}	2.1×10^{-10}	11.78	$\mathbf{1.2 \times 10^{-10}}$
f_2	Average best so-far	1.07	2.0	4.03×10^{-5}	0.95	$\mathbf{5.28 \times 10^{-20}}$
	Median best so-far	1.13	1.9×10^{-3}	4.07×10^{-5}	1.05	$\mathbf{2.88 \times 10^{-11}}$
	Average mean fitness	1.07	2.0	6.9×10^{-5}	0.90	$\mathbf{1.43 \times 10^{-9}}$
f_3	Average best so-far	5.6×10^{3}	4.1×10^{3}	0.16×10^{3}	0.12	$\mathbf{7.62 \times 10^{-31}}$
	Median best so-far	5.6×10^{3}	2.2×10^{3}	0.15×10^{3}	0.09	$\mathbf{1.28 \times 10^{-19}}$
	Average mean fitness	5.6×10^{3}	2.9×10^{3}	0.16×10^{3}	0.11	$\mathbf{3.51 \times 10^{-12}}$
f_4	Average best so-far	11.78	8.1	3.7×10^{-6}	0.012	$\mathbf{2.17 \times 10^{-17}}$
	Median best so-far	11.94	7.4	3.7×10^{-6}	0.058	$\mathbf{5.65 \times 10^{-12}}$
	Average mean fitness	11.78	23.6	8.5×10^{-6}	0.013	$\mathbf{4.96 \times 10^{-10}}$
f_5	Average best so-far	1.1×10^{3}	3.6×10^{4}	25.16	0.25	$\mathbf{9.025 \times 10^{-28}}$
	Median best so-far	1.0×10^{3}	1.7×10^{3}	25.18	0.31	$\mathbf{3.10 \times 10^{-18}}$
	Average mean fitness	1.1×10^{3}	3.7×10^{4}	25.16	0.24	$\mathbf{6.04 \times 10^{-14}}$
f_6	Average best so-far	24.01	1.0×10^{-3}	8.3×10^{-11}	1.25×10^{-3}	$\mathbf{4.47 \times 10^{-29}}$
	Median best so-far	24.55	6.6×10^{-3}	7.7×10^{-11}	3.33×10^{-3}	$\mathbf{4.26 \times 10^{-21}}$
	Average mean fitness	24.52	0.02	2.6×10^{-10}	1.27×10^{-3}	$\mathbf{1.03 \times 10^{-12}}$
f_7	Average best so-far	0.06	0.04	0.018	6.87×10^{-3}	$\mathbf{3.45 \times 10^{-5}}$
	Median best so-far	0.06	0.04	0.015	4.72×10^{-3}	$\mathbf{7.39 \times 10^{-4}}$
	Average mean fitness	0.56	1.04	0.533	1.28×10^{-2}	$\mathbf{8.75 \times 10^{-4}}$

Maximum number of iterations = 1000



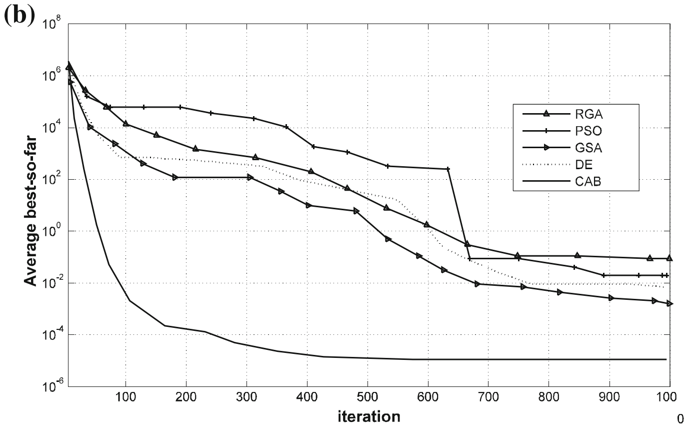


Fig. 6.2 Performance comparison of RGA, PSO, GSA, DE and CAB for minimization of **a** f_1 and **b** f_7 considering $n = 30$

RGA, PSO, GSA and DE considering the four problems $f_9 - f_{12}$, whereas, from a statistical viewpoint, there is not difference in results between CAB and RGA for f_8, and between CAB and GSA for f_{13}. The progress of the "average best-so-far" solution over 30 runs for functions f_{10} and f_{12} are shown in Fig. 6.3.

Multimodal test functions with fix dimensions

Table 6.11 shows the comparison between CAB, RGA, PSO, GSA and DE on multimodal benchmark functions with fix dimensions of Table 6.3. The results show that RGA, PSO and GSA have similar solutions and performances are nearly the same as it can be seen in Fig. 6.4. The results, presented in Table 6.11, show how swarm algorithms maintain a similar average performance when they faced low-dimensional functions [57].

Comparison to continuous optimization methods

Finally, CAB has been further compared to continuous optimization methods by

Table 6.8 p-values produced by Wilcoxon's test comparing CAB versus RGA, PSO, GSA and DE over the "average best-so-far" values from Table 6.8

CAB	RGA	PSO	GSA	DE
f_1	1.21×10^{-6}	3.94×10^{-5}	7.39×10^{-4}	1.04×10^{-6}
f_2	2.53×10^{-6}	5.62×10^{-5}	4.92×10^{-4}	2.21×10^{-6}
f_3	8.34×10^{-8}	6.42×10^{-8}	7.11×10^{-7}	1.02×10^{-4}
f_4	3.81×10^{-8}	1.91×10^{-8}	7.39×10^{-4}	1.27×10^{-6}
f_5	4.58×10^{-8}	9.77×10^{-9}	4.75×10^{-7}	0.23×10^{-4}
f_6	8.11×10^{-8}	1.98×10^{-6}	5.92×10^{-4}	2.88×10^{-5}
f_7	5.12×10^{-7}	4.77×10^{-7}	8.93×10^{-6}	1.01×10^{-4}

considering the presented benchmark functions. Since the BFSG algorithm [60] is one of the most effective continuous methods for solving unconstrained optimization problems, it has been considered as a basis for the algorithms in the comparison. All experiments have been tested in MatLAB© over the same Dell Optiplex GX260 computer with a Pentium-4 2.66G-HZ processor, running Windows XP operating system over 1 Gb of memory.

Local optimization

In the first experiment, the performance of algorithms BFGS and CAB over unimodal functions is compared. In unimodal functions, the global minimum matches the local minimum. Quasi-Newton methods, such as the BFGS, have a fast rate of local convergence despite it depends on the problem's dimension [61, 62]. Considering that not all unimodal functions of Table 6.1 fulfill the requirements imposed by the gradient based approaches (i.e. f_2 and f_4 are not differentiable meanwhile f_7 is non-smooth), we have chosen the Rosenbrock function (f_5) as a benchmark.

In the test, both algorithms (BFGS and CAB) are employed to minimize f_5, considering different dimensions. For the BFGS implementation, it is considered $B_0 = I$ as initial matrix. Likewise, parameters δ and δ are set to 0.1 and 0.9 respectively. Although several performance criteria may define a comparison index, most can be applied to only one method timely (such as the number of gradient evaluations). Therefore, this chapter considers the elapsed time and the iteration number (once the minimum has been reached) as performance indexes in the comparison. In the case of BFGS, the termination condition is assumed as $\|g_5(\mathbf{x})\| \leq 1 \times 10^{-6}$, with $g_5(\mathbf{x})$ being the gradient of $f_5(\mathbf{x})$. On the other hand, the stopping criterion of CAB considers when no more changes to the best element in memory $\mathbf{M}_h$ are registered. Table 6.12 presents the results of both algorithms considering several dimensions ($n \in \{2, 10, 30, 50, 70, 100, 120\}$) of f_5. In order to assure consistency, such results

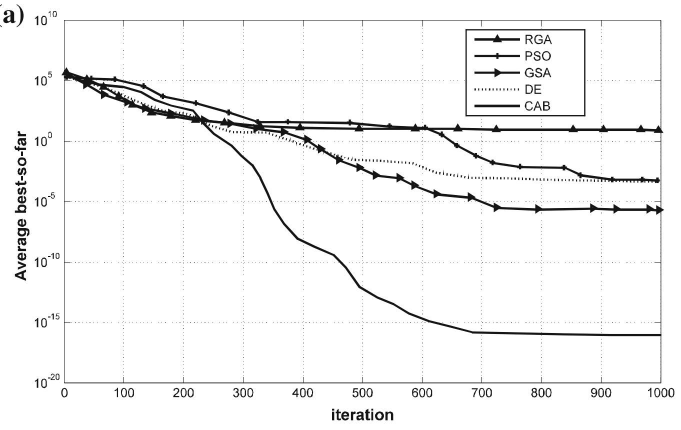
Table 6.9 Minimization result of benchmark functions in Table 6.2 with $n = 30$

		RGA	PSO	GSA	DE	CAB
f_8	Average best so-far	$\mathbf{-1.26 \times 10^4}$	-9.8×10^3	-2.8×10^3	-4.1×10^3	$\mathbf{-1.2 \times 10^4}$
	Median best so-far	$\mathbf{-1.26 \times 10^4}$	-9.8×10^3	-2.6×10^3	-4.1×10^3	$\mathbf{-1.2 \times 10^4}$
	Average mean fitness	$\mathbf{-1.26 \times 10^4}$	-9.8×10^3	-1.1×10^3	-4.1×10^3	$\mathbf{-1.2 \times 10^4}$
f_9	Average best so-far	5.90	55.1	15.32	30.12	$\mathbf{1.0 \times 10^{-3}}$
	Median best so-far	5.71	56.6	14.42	31.43	$\mathbf{7.6 \times 10^{-4}}$
	Average mean fitness	5.92	72.8	15.32	30.12	$\mathbf{1.0 \times 10^{-3}}$
f_{10}	Average best so-far	2.13	9.0×10^{-3}	6.9×10^{-6}	3.1×10^{-3}	$\mathbf{8.88 \times 10^{-16}}$
	Median best so-far	2.16	6.0×10^{-3}	6.9×10^{-6}	2.3×10^{-3}	$\mathbf{2.97 \times 10^{-11}}$
	Average mean fitness	2.15	0.02	1.1×10^{-5}	3.1×10^{-3}	$\mathbf{9.0 \times 10^{-10}}$
f_{11}	Average best so-far	1.16	0.01	0.29	1.0×10^{-3}	$\mathbf{1.14 \times 10^{-13}}$
	Median best so-far	1.14	0.0081	0.04	1.0×10^{-3}	$\mathbf{1.14 \times 10^{-13}}$
	Average mean fitness	1.16	0.055	0.29	1.0×10^{-3}	$\mathbf{1.14 \times 10^{-13}}$
f_{12}	Average best so-far	0.051	0.29	0.01	0.12	$\mathbf{2.32 \times 10^{-30}}$
	Median best so-far	0.039	0.11	4.2×10^{-13}	0.01	$\mathbf{5.22 \times 10^{-22}}$
	Average mean fitness	0.053	9.3×10^3	0.01	0.12	$\mathbf{4.63 \times 10^{-17}}$
f_{13}	Average best so-far	0.081	3.1×10^{-18}	$\mathbf{3.2 \times 10^{-32}}$	1.77×10^{-25}	$\mathbf{1.35 \times 10^{-32}}$
	Median best so-far	0.032	2.2×10^{23}	$\mathbf{2.3 \times 10^{-32}}$	1.77×10^{-25}	$\mathbf{2.20 \times 10^{-21}}$
	Average mean fitness	0.081	4.8×10^5	$\mathbf{3.2 \times 10^{-32}}$	1.77×10^{-25}	$\mathbf{3.53 \times 10^{-17}}$

Maximum number of iterations = 1000

Table 6.10 *p*-values produced by Wilcoxon's test comparing CAB versus RGA, PSO, GSA and DE over the "average best-so-far" values from Table 6.9

CAB	RGA	PSO	GSA	DE
f_8	0.89	8.38×10^{-4}	1.21×10^{-4}	4.61×10^{-4}
f_9	7.23×10^{-7}	1.92×10^{-9}	5.29×10^{-8}	9.97×10^{-8}
f_{10}	6.21×10^{-9}	4.21×10^{-5}	1.02×10^{-4}	3.34×10^{-4}
f_{11}	7.74×10^{-9}	3.68×10^{-7}	4.10×10^{-7}	8.12×10^{-5}
f_{12}	1.12×10^{-8}	8.80×10^{-9}	2.93×10^{-7}	4.02×10^{-8}
f_{13}	4.72×10^{-9}	3.92×10^{-5}	0.93	2.20×10^{-4}

(a)


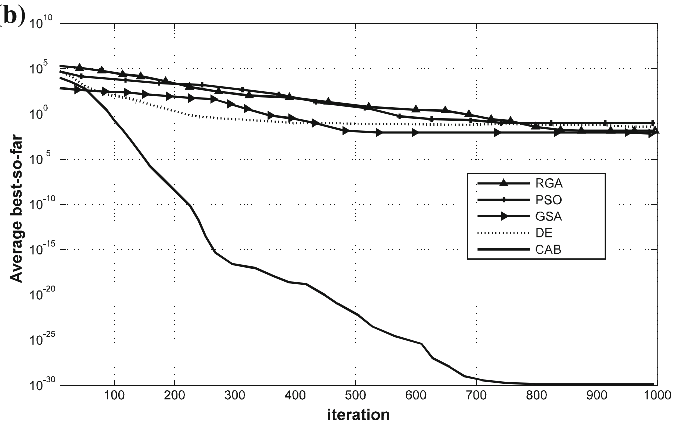
(b)


Fig. 6.3 Performance comparison of RGA, PSO, GSA, DE and CAB for minimization of **a** f_{10} and **b** f_{12} considering $n = 30$

Table 6.11 Minimization result of benchmark functions in Table 6.3 with $n = 30$

		RGA	PSO	GSA	DE	CAB
f_{14}	Average best so-far	0.998	0.998	3.70	0.998	0.998
	Median best so-far	0.998	0.998	2.07	0.998	0.998
	Average mean fitness	0.998	0.998	9.17	0.998	0.998
f_{15}	Average best so-far	4.0×10^{-3}	2.8×10^{-3}	8.2×10^{-3}	2.2×10^{-3}	1.1×10^{-3}
	Median best so-far	1.7×10^{-4}	7.1×10^{-4}	7.4×10^{-4}	5.3×10^{-4}	2.2×10^{-4}
	Average mean fitness	4.0×10^{-3}	215.60	9.0×10^{-3}	2.2×10^{-3}	1.1×10^{-3}
f_{16}	Average best so-far	−1.0313	−1.0316	−1.0316	−1.0316	−1.0316
	Median best so-far	−1.0315	−1.0316	−1.0316	−1.0316	−1.0316
	Average mean fitness	−1.0313	−1.0316	−1.0316	−1.0316	−1.0316
f_{17}	Average best so-far	0.3996	0.3979	0.3979	0.3979	0.3979
	Median best so-far	0.3980	0.3979	0.3979	0.3979	0.3979
	Average mean fitness	0.3996	2.4112	0.3979	0.3979	0.3979
f_{18}	Average best so-far	−3.8627	−3.8628	−3.8628	−3.8628	−3.8628
	Median best so-far	−3.8628	−3.8628	−3.8628	−3.8628	−3.8628
	Average mean fitness	−3.8627	−3.8628	−3.8628	−3.8628	−3.8628
f_{19}	Average best so-far	−3.3099	−3.3269	−3.7357	−3.3269	−3.8501
	Median best so-far	−3.3217	−3.2031	−3.8626	−3.3269	−3.8501
	Average mean fitness	−3.3098	−3.2369	−3.8020	−3.3269	−3.8501
f_{20}	Average best so-far	−3.3099	−3.2369	−2.0569	−3.2369	−3.2369
	Median best so-far	−3.3217	−3.2031	−1.9946	−3.2369	−3.2369
	Average mean fitness	−3.3098	−3.2369	−1.6014	−3.2369	−3.2369

(continued)

Table 6.11 (continued)

		RGA	PSO	GSA	DE	CAB
f_{21}	Average best so-far	−5.6605	−6.6290	−6.0748	−6.6290	−10.1532
	Median best so-far	−2.6824	−5.1008	−5.0552	−6.0748	−10.1532
	Average mean fitness	−5.6605	−5.7496	−6.0748	−6.6290	−10.1532
f_{22}	Average best so-far	−7.3421	−9.1118	−9.3339	−9.3339	−10.4028
	Median best so-far	−10.3932	−10.402	−10.402	−9.3339	−10.4028
	Average mean fitness	−7.3421	−9.9305	−9.3399	−9.3339	−10.4028
f_{23}	Average best so-far	−6.2541	−9.7634	−9.4548	−9.7634	−10.5363
	Median best so-far	−4.5054	−10.536	−10.536	−9.7636	−10.5363
	Average mean fitness	−6.2541	−8.7626	−9.4548	−9.7634	−10.5363

Maximum number of *iterations* = 500

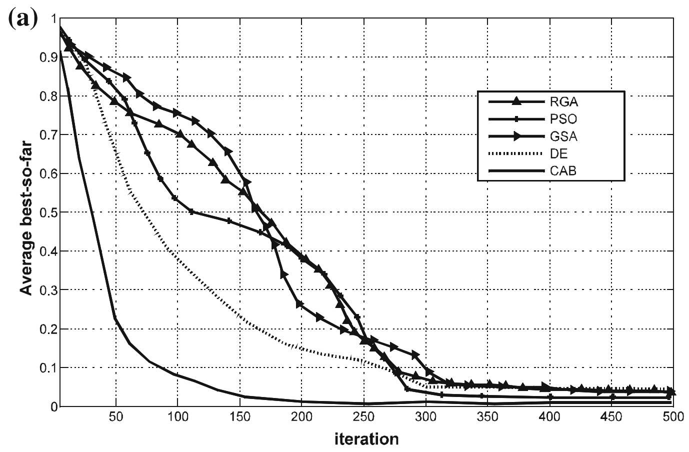
Table 6.12 Performance comparison between the BFGS and the CAB algorithm, considering different dimensions over the Rosenbrock function

f_5 n	AET		AIN	
	BFGS	CAB	BFGS	CAB
2	0.15	4.21	6	89
10	0.55	5.28	22	98
30	1.35	5.44	41	108
50	2.77	5.88	68	112
70	4.23	6.11	93	115
100	5.55	6.22	105	121
120	6.64	6.71	125	129

The averaged elapsed time (AET) is referred in seconds

represent the averaged elapsed time (AET) and the averaged iteration number (AIN) over 30 different executions. It is additionally considered that at each execution both methods are initialized to a random point (inside the search space).

From Table 6.12, we can observe that the BFGS algorithm produces shorter elapsed times and fewer iterations than the CAB method. However, from $n =$ 70, the CAB algorithm contend with similar results. The fact that the BFGS algorithm outperforms the CAB approach cannot be deemed as a negative feature considering the restrictions imposed to the functions by the BFGS method.



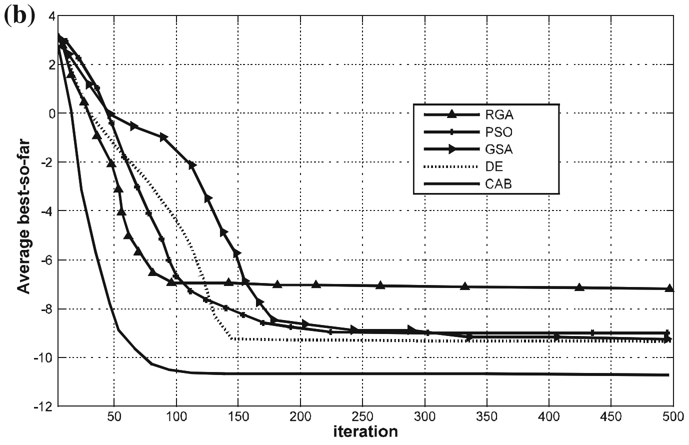


Fig. 6.4 Performance comparison of RGA, PSO, GSA, DE and CAB for minimization of **a** f_{15} and **b** f_{22}

Global optimization
Since the BFGS algorithm exploits only local information, it may easily get trap into local optima restricting its use for global optimization. Therefore several other methods for continuous optimization have been proposed. One of the most widely used techniques is the so called Multi-Start [63] (MS). In MS a point is randomly chosen from a feasible region as initial solution and subsequently a continuous optimization algorithm (local search) starts from it. Then, the process is repeated until a near global optimum is reached. The weakness of MS is that the same local minima may be found over and over again, wasting computational resources [64].

In order to compare the performance of the CAB approach to continuous optimization methods in the context of global optimization, the MS algorithm ADAPT [65] has been chosen. ADAPT uses an iterative BFGS algorithm as local search method.

Table 6.13 Performance comparison between the ADAPT and the CAB algorithm considering different multimodal functions

Function	n	ADAPT				CAB		
		ALS	AET	AIN	ABS	AET	AIN	ABS
f_9	30	3705	45.4	23,327	1.2×10^{-2}	10.2	633	1.0×10^{-3}
f_{10}	30	4054	1′05.7	38,341	6.21×10^{-12}	12.1	723	8.88×10^{-16}
f_{11}	30	32,452	2′12.1	102,321	4.51×10^{-10}	15.8	884	1.14×10^{-13}
f_{17}	2	1532	33.2	20,202	0.3976	7.3	332	0.3979
f_{18}	2	1233	31.6	18,845	−3.8611	6.6	295	−3.8628

The averaged elapsed time (AET) is referred in the format M's (Minute'second)

Thus, ADAPT possess two different stop criteria, one for the local procedure BFGS and other for the complete MS approach. For the comparison, the ADAPT algorithm has been implemented as suggested by [65].

In the second experiment, the performance of the ADAPT and the CAB algorithms is compared over several multimodal functions described in Tables 6.2 and 6.3. The study considers the following performance indexes: the elapsed time, the iteration number and the average best so-far solution. In case of the ADAPT algorithm, the iteration number is computed as the total iteration number produced by all the local search procedures as the MS method operates. The termination condition of the ADAPT local search algorithm (BFGS) is assumed when $\|g_k(\mathbf{x})\| \leq 1 \times 10^{-5}$, being $g_k(\mathbf{x})$ the gradient of $f_k(\mathbf{x})$. On the other hand, the stopping criterion for the CAB and the ADAPT algorithms, is considered when no more changes in the best element are registered, i.e. the best element in $\mathbf{M}_h$ for CAB or the best found-so-far element during the MS process. Table 6.13 presents results from both algorithms considering several multimodal functions. In order to assure consistency, results ponder the averaged elapsed time (AET), the averaged iteration number (AIN) and the average best so-far solution (ABS) over 30 different executions. In Table 6.13, the averaged number of local searches (ALS) executed by ADAPT during the optimization, is additionally considered.

Table 6.13 provides a summarized performance comparison between the ADAPT and the CAB algorithms. Despite both algorithms are able to acceptably locate the global minimum for both cases, there exist significant differences in the required time for reaching it. When comparing the averaged elapsed time (AET) and the averaged iteration number (AIN) in Table 6.13, CAB uses significantly less time and fewer iterations to reach the global minimum than the ADAPT algorithm.

6.5 Summary

In recent years, several swarm optimization methods have been developed including some that have been inspired by nature-related phenomena. This chapter presents a novel optimization algorithm that is called the Collective Animal Behavior Algorithm (CAB). In CAB, the searcher agents emulate a group of animals that interact to each other based on simple behavioral rules which are modeled as mathematical operators. Such operations are applied to each agent considering that the complete group has a memory storing their own best positions seen so far by using a competition principle.

CAB has been experimentally tested considering a challenging test suite consisting of 23 benchmark functions. The performance of CAB has been compared against the following swarm algorithms: the Real Genetic Algorithm (RGA) [54], the PSO [18], the Gravitational Search Algorithm (GSA) [55] and the Differential Evolution (DE) method [56]. The experiments have demonstrated that CAB generally outperforms other swarm algorithms for most of the benchmark functions regarding the solution quality. In this study the CAB algorithm has also been compared to algorithms based in continuous optimization methods such as the BFGS [60] and the ADAPT [65]. The results shown that although BFGS outperforms CAB for local optimization tasks, ADAPT faces great difficulties in solving global optimization problems. CAB's remarkable performance is due to two different reasons: (i) the defined operators allow a better exploration of the search space, increasing the capacity to find multiple optima; and (ii) the diversity of the solutions contained in the $\mathbf{M}_h$ memory, in terms of multimodal optimization, is maintained and even improved by implementing the competition principle (dominance concept).

References

1. Pardalos Panos, M., Romeijn Edwin, H., Tuy, Hoang: Recent developments and trends in global optimization. J. Comput. Appl. Math. **124**, 209–228 (2000)
2. Floudas, C., Akrotirianakis, I., Caratzoulas, S., Meyer, C., Kallrath, J.: Global optimization in the 21st century: advances and challenges. Comput. Chem. Eng. **29**(6), 1185–1202 (2005)
3. Ying, J., Ke-Cun, Z., Shao-Jian, Q.: A deterministic global optimization algorithm. Appl. Math. Comput. **185**(1), 382–387 (2007)
4. Georgieva, A., Jordanov, I.: Global optimization based on novel heuristics, low-discrepancy sequences and genetic algorithms. Eur. J. Oper. Res. **196**, 413–422 (2009)
5. Lera, D., Y, Sergeyev: Lipschitz and Hölder global optimization using space-filling curves. Appl. Nume. Math. **60**(1–2), 115–129 (2010)
6. Fogel, L.J., Owens, A.J., Walsh, M.J.: Artificial intelligence through simulated evolution. Wiley, Chichester, UK (1966)
7. De Jong, K.: Analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan, Ann Arbor, MI (1975)
8. Koza, J.R.: Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Report no. STAN-CS-90-1314. Stanford University, CA (1990)
9. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975)

10. Goldberg, D.E.: Genetic algorithms in search. In: Optimization and Machine Learning, Addison Wesley, Boston, MA (1989)
11. de Castro, L.N., Von Zuben F.J., Artificial immune systems: Part I—basic theory and applications. Technical report TR-DCA 01/99 (Dec 1999)
12. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
13. İlker, B., Birbil, S., Shu-Cherng, F.: An electromagnetism-like mechanism for global optimization. J. Glob. Optim. **25**, 263–282 (2003)
14. Rashedia, E., Nezamabadi-pour, H., Saryazdi, S.: Filter modeling using gravitational search algorithm. Eng. Appl. Artif. Intell. **24**(1), 117–122 (2011)
15. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: harmony search. Simulation **76**(2), 60–68 (2001)
16. Lee, K.S., Geem, Z.W.: A new meta-heuristic algorithm for continues engineering optimization: harmony search theory and practice. Comput. Methods Appl. Mech. Eng. **194**, 3902–3933 (2004)
17. Zong Woo Geem: Novel derivative of harmony search algorithm for discrete design variables. Appl. Math. Comput. **199**, 223–230 (2008)
18. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
19. Dorigo, M., Maniezzo, V., Colorni, A.: Positive feedback as a search strategy. Technical report no. 91-016. Politecnico di Milano (1991)
20. Oftadeh, R., Mahjoob, M.J., Shariatpanahi, M.: A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. Comput. Math Appl. **60**, 2087–2098 (2010)
21. Sumper, D.: The principles of collective animal behaviour. Philos. Trans. R. Soc. Lond. B Biol. Sci. **361**(1465), 5–22 (2006)
22. Petit, O., Bon, R.: Decision-making processes: The case of collective movements. Behav. Proc. **84**, 635–647 (2010)
23. Kolpas, A., Moehlis, J., Frewen, T., Kevrekidis, I.: Coarse analysis of collective motion with different communication mechanisms. Math. Biosci. **214**, 49–57 (2008)
24. Couzin, I.: Collective cognition in animal groups. Trends Cogn. Sci. **13**(1), 36–43 (2008)
25. Couzin, I.D., Krause, J.: Self-organization and collective behavior in vertebrates. Adv. Stud. Behav. **32**, 1–75 (2003)
26. Bode, N., Franks, D., Wood, A.: Making noise: emergent stochasticity in collective motion. J. Theor. Biol. **267**, 292–299 (2010)
27. Couzi, I., Krause, I., James, R., Ruxton, G., Franks, N.: Collective memory and spatial sorting in animal groups. J. Theor. Biol. **218**, 1–11 (2002)
28. Couzin, I.D.: Collective minds. Nature **445**, 715–728 (2007)
29. Bazazi, S., Buhl, J., Hale, J.J., Anstey, M.L., Sword, G.A., Simpson, S.J., Couzin, I.D.: Collective motion and cannibalism in locust migratory bands. Curr. Biol. **18**, 735–739 (2008)
30. Bode, N., Wood, A., Franks, D.: The impact of social networks on animal collective motion. Anim. Behav. **82**(1), 29–38 (2011)
31. Lemasson, B., Anderson, J., Goodwin, R.: Collective motion in animal groups from a neurobiological perspective: the adaptive benefits of dynamic sensory loads and selective attention. J. Theor. Biol. **261**(4), 501–510 (2009)
32. Bourjade, M., Thierry, B., Maumy, M., Petit, O.: Decision-making processes in the collective movements of Przewalski horses families Equus ferus Przewalskii: influences of the environment. Ethology **115**, 321–330 (2009)
33. Banga, A., Deshpande, S., Sumanab, A., Gadagkar, R.: Choosing an appropriate index to construct dominance hierarchies in animal societies: a comparison of three indices. Anim. Behav. **79**(3), 631–636 (2010)
34. Hsu, Y., Earley, R., Wolf, L.: Modulation of aggressive behaviour by fighting experience: mechanisms and contest outcomes. Biol. Rev. **81**(1), 33–74 (2006)

35. Broom, M., Koenig, A., Borries, C.: Variation in dominance hierarchies among group-living animals: modeling stability and the likelihood of coalitions. Behav. Ecol. **20**, 844–855 (2009)
36. Bayly, K.L., Evans, C.S., Taylor, A.: Measuring social structure: a comparison of eight dominance indices. Behav. Proc. **73**, 1–12 (2006)
37. Conradt, L., Roper, T.J.: Consensus decision-making in animals. Trends Ecol. Evol. **20**, 449–456 (2005)
38. Okubo, A.: Dynamical aspects of animal grouping. Adv. Biophys. **22**, 1–94 (1986)
39. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioural model. Comp. Graph. **21**, 25–33 (1987)
40. Gueron, S., Levin, S.A., Rubenstein, D.I.: The dynamics of mammalian herds: from individual to aggregations. J. Theor. Biol. **182**, 85–98 (1996)
41. Czirok, A., Vicsek, T.: Collective behavior of interacting self-propelled particles. Phys. A **281**, 17–29 (2000)
42. Ballerini, M.: Interaction ruling collective animal behavior depends on topological rather than metric distance: evidence from a field study. Proc. Natl. Acad. Sci. U.S.A. **105**, 1232–1237 (2008)
43. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J. Glob. Optim. **31**(4), 635–672 (2005)
44. Chelouah, R., Siarry, P.: A continuous genetic algorithm designed for the global optimization of multimodal functions. J. Heuristics **6**(2), 191–213 (2000)
45. Herrera, F., Lozano, M., Sánchez, A.M.: A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. Int. J. Intell. Syst. **18**(3), 309–338 (2003)
46. Laguna, M., Martí, R.: Experimental testing of advanced scatter search designs for global optimization of multimodal functions. J. Global Optim. **33**(2), 235–255 (2005)
47. Lozano, M., Herrera, F., Krasnogor, N., Molina, D.: Real-coded memetic algorithms with crossover hill-climbing. Evol. Comput. **12**(3), 273–302 (2004)
48. Moré, J.J., Garbow, B.S., Hillstrom, K.E.: Testing unconstrained optimization software. ACM Trans. Math. Softw. **7**(1), 17–41 (1981)
49. Ortiz-Boyer, D., Hervás-Martınez, C., García-Pedrajas, N.: CIXL2: a crossover operator for evolutionary algorithms based on population features. J. Artif. Intell. Res. **24**(1), 1–48 (2005)
50. Price, K., Storn, R.M., Lampinen, J.A.: Differential evolution: a practical approach to global optimization. Springer, New York (2005)
51. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-based differential evolution. IEEE Trans. Evol. Comput. **12**(1), 64–79 (2008)
52. Whitley, D., Rana, D., Dzubera, J., Mathias, E.: Evaluating evolutionary algorithms. Artif. Intell. **85**(1–2), 245–276 (1996)
53. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. IEEE Trans. Evol. Comput. **3**(2), 82–102 (1999)
54. Hamzaçebi, C.: Improving genetic algorithms' performance by local search for continuous function optimization. Appl. Math. Comput. **196**(1), 309–317 (2008)
55. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: a gravitational search algorithm. Inf. Sci. **179**, 2232–2248 (2009)
56. Storn, R., Price, K.: Differential evolution—a simple and efficient adaptive scheme for global optimisation over continuous spaces. Technical report TR-95–012. ICSI, Berkeley, CA (1995)
57. Shilane, D., Martikainen, J., Dudoit, S., Ovaska, S.: A general framework for statistical performance comparison of evolutionary computation algorithms. Inf. Sci. **178**, 2870–2879 (2008)
58. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics **1**, 80–83 (1945)
59. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special session on real parameter optimization. J. Heurist. (2008). https://doi.org/10.1007/s10732-008-9080-4
60. Al-Baali, M.: On the behavior of bombined extra-updating/self scaling BFGS method. J. Comput. Appl. Math. **134**, 269–281 (2001)

61. Powell, M.: How bad are the BFGS and DFP methods when the objective function is quadratic? Math. Program. **34**, 34–37 (1986)
62. Hansen, E., Walster, G.: Global Optimization Using Interval Analysis. CRC Press (2004)
63. Lasdona, L., Plummer, J.: Multistart algorithms for seeking feasibility. Comput. Oper. Res. **35**(5), 1379–1393 (2008)
64. Theos, F., Lagaris, I., Papageorgiou, D.: PANMIN: sequential and parallel global optimization procedures with a variety of options for the local search strategy. Comput. Phys. Commun. **159**, 63–69 (2004)
65. Voglis, C., Lagaris, I.: Towards "Ideal Multistart". A stochastic approach for locating the minima of a continuous function inside a bounded domain. Appl. Math. Comput. **213**, 216–229 (2009)

Chapter 7
Auto-calibration of Fractional Fuzzy Controllers by Using the Swarm Social-Spider Method

Abstract Fuzzy controllers (FCs) based on integer concepts have proved their interesting capacities in several engineering domains. The fact that dynamic processes can be more precisely modeled by using fractional systems has generated a great interest in considering the design of FCs under fractional principles. In the design of fractional FCs, the parameter adjustment operation is converted into a multidimensional optimization task where fractional orders, and controller parameters, are assumed as decision elements. In the design of fractional FCs, the complexity of the optimization problem produces multi-modal error surfaces which are significantly hard to solve. Several swarm algorithms have been successfully used to identify the optimal elements of fractional FCs. But, most of them present a big weakness since they usually get sub-optimal solutions as a result of their improper balance between exploitation and exploration in their search process. This chapter analyses the optimal parameter calibration of fractional FCs. To determine the best elements, the approach employs the Social Spider Optimization (SSO) algorithm, which is based on the simulation of the cooperative operation of social-spiders. In SSO, candidate solutions represent a group of spiders, which interact with each other by considering the biological concepts of the spider colony. Different to most of the swarm algorithms, the approach explicitly avoids the concentration of solutions in the promising positions, eliminating critical defects such as the premature convergence and the deficient balance of exploration-exploitation.

7.1 Introduction

A fractional order model is a system that is characterized by a fractional differential equation containing derivatives of non-integer order. In fractional calculus, the integration and the differentiation operators are generalized into a non-integer order element, where is a fractional number and a and t symbolize the operator limits [1, 2]. Several dynamic systems can be more accurately described and controlled by fractional models in comparison to integer order schemes. For this reason, in the last decade, the fractional order controllers [3–5] have attracted the interests of several research communities.

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_7

A fractional-order controller incorporates an integrator of order and a differentiator of order. The superior performance of such controllers with regard to conventional PIDs has been widely demonstrated in the literature [6].

On the other hand, fuzzy logic [7] provides an alternative method to design controllers through the use of heuristic information. Remarkably, such heuristic information may come from a human-operator who directly manipulates the process. In the fuzzy logic methodology, a human operator defines a set of rules on how to control a process, then incorporated it into a fuzzy controller that emulates the decision-making process of the operator [8].

Fractional fuzzy controllers (FFCs) are the results of the combination of conventional fuzzy controllers and fractional operators. Under such combination, FFCs exhibit better results than conventional FCs for an extensive variety of dynamical systems. This capacity is attributed to the additional flexibility offered by the inclusion of the fractional parameters.

Parameter calibration is an important step to implement applications with FFCs. This procedure is long and time consuming, since it is commonly conducted through trial and error. Therefore, the problem of parameter estimation in FFCs can be handled by evolutionary optimization methods. In general, they have demonstrated to deliver interesting results in terms of accuracy and robustness [5]. In these methods, an individual is represented by a candidate parameter set, which symbolizes the configuration of a determined fractional fuzzy controller. Just as the evolution process unfolds, a set of swarm operators is applied in order to produce better individuals. The quality of each candidate solution is evaluated through an objective function whose final result represents the performance of the parameter set in terms of the produced error. Some examples of such approaches being applied to the identification of fractional order systems have involved methods such as Genetic Algorithms (GA) [5], Particle Swarm Optimization (PSO) [9], Harmony Search (HS) [10], Gravitational Search Algorithm (GSA) [11] and Cuckoo Search (CS) [12]. Although such algorithms present interesting results, they have exhibited an important limitation: they frequently obtained sub-optimal solutions as a consequence of the limited balance between exploration and exploitation in their search strategies. Such limitation is associated to the evolutionary operators that have been employed to modify individual positions. In such algorithms, during their operation, the position of each individual for the next iteration is updated, producing an attraction towards the position of the best particle seen so far or towards other promising individuals. Therefore, as the algorithm evolves, such behaviors cause that the entire population concentrate rapidly around the best particles, favoring the premature convergence and damaging the appropriate exploration of the search space [13, 14].

The Social Spider Optimization (SSO) algorithm [15] is a recent swarm method that is inspired on the emulation of the collaborative behavior of social-spiders. In SSO, solutions imitate a set of spiders, which cooperate to each other based on the natural laws of the cooperative colony. Unlike the most popular swarm algorithms such as GA [16], PSO [17], HS [18], GSA [19] and CS [20], it explicitly evades the concentration of individuals in the best positions, avoiding critical flaws such as the premature convergence to sub-optimal solutions and the limited balance of exploration-exploitation. Such characteristics have motivated the use of SSO to solve an extensive variety of engineering applications such as energy theft detection [21], machine learning [22], electromagnetics [23], image processing [24] and integer programming problems [25].

This chapter presents a method for the optimal parameter calibration of fractional FCs based on the SSO algorithm. Under this approach, the calibration process is transformed into a multidimensional optimization problem where fractional orders, as well as controller parameters of the fuzzy system, are considered as a candidate solution to the calibration task. In the method, the SSO algorithm searches the entire parameter space while an objective function evaluates the performance of each parameter set. Conducted by the values of such objective function, the group of candidate solutions are evolved through the SSO algorithm so that the optimal solution can be found. Experimental evidence shows the effectiveness and robustness of the method for calibrating fractional FCs. A comparison with similar methods such as the Genetic Algorithms (GA), the Particle Swarm Optimization (PSO), the Harmony Search (HS), the Gravitational Search Algorithm (GSA) and the Cuckoo Search (CS) on different dynamical systems has been incorporated to demonstrate the performance of this approach. Conclusions of the experimental comparison are validated through statistical tests that properly support the discussion.

The chapter is organized as follows: Sect. 7.2 introduces the concepts of fractional order systems; Sect. 7.3 describes the fractional fuzzy controller used in the calibration; Sect. 7.4 presents the characteristics of SSO; Sect. 7.5 formulates the parameter calibration problem; Sect. 7.6 shows the experimental results while some final conclusions are discussed in Sect. 7.7.

7.2 Fractional-Order Models

Dynamical fractional-order systems are modeled by using differential equations, which involve non-integer integral and/or derivative operators [26, 27]. Since these operators produce irrational continuous transfer functions, or infinite dimensional discrete transfer functions, fractional models are normally studied through simulation tools instead of analytical methods [5, 28–33]. The remainder of this section provides a background of the fundamental aspects of the fractional calculus, and the discrete integer-order approximations of fractional order operators that are used in this paper.

7.2.1 Fractional Calculus

Fractional calculus is a generalization of integration and differentiation to the non-integer order fundamental operator. The differential-integral operator, denoted by

${}_aD_t^\alpha$, takes both the fractional derivative and the fractional integral into a single expression defined as:

$$
{}_aD_t^\alpha = \begin{cases} \frac{d^\alpha}{dt^\alpha}, & \alpha > 0, \\ 1, & \alpha = 0, \\ \int\limits_a^t (d\tau)^\alpha, & \alpha < 0. \end{cases} \tag{7.1}
$$

where a and t represent the operation bounds, whereas $\alpha \in \Re$. The commonly used definitions for fractional derivatives are the Grünwald-Letnikov, Riemann-Liouville [34] and Caputo [35]. According to the Grünwald-Letnikov approximation, the fractional-order derivative of order α is defined as follows:

$$
D_t^\alpha f(t) = \lim_{h \to 0} \frac{1}{h^\alpha} \sum_{j=0}^{\infty} (-1)^j \binom{\alpha}{j} f(t - jh) \tag{7.2}
$$

In the numerical calculation of fractional-order derivatives, the explicit numerical approximation of the α-th derivative at the points kh, $(k = 1, 2, \ldots)$ maintains the following form [36]:

$$
{}_{(k-M_m/h)}D_{t_k}^\alpha f(t) \approx h^{-\alpha} \sum_{j=0}^{k} (-1)^j \binom{\alpha}{j} f(t_k - j) \tag{7.3}
$$

where M_m is the memory length $t_k = kh$, h is the time step and $(-1)^j \binom{\alpha}{j}$ are the binomial coefficients. For their calculation, we can use the following expression:

$$
c_0^{(\alpha)} = 1, \quad c_j^{(\alpha)} = \left(1 - \frac{1+\alpha}{j}\right) c_{j-1}^{(\alpha)} \tag{7.4}
$$

Then, the general numerical solution of the fractional differential equation is defined as follows:

$$
y(t_k) = f(y(t_k), t_k)h^\alpha - \sum_{j=1}^{k} c_j^{(\alpha)} y(t_{k-j}) \tag{7.5}
$$

7.2.2 Approximation of Fractional Operators

Assuming zero initial conditions, the fractional operator is defined in the Laplace domain as $L({}_aD_t^\alpha f(t)) = s^\alpha F(s)$. Several approaches [37–39] have been proposed for producing discrete versions of continuous operators of type s^α. In this chapter, the Grünwald-Letnikov approximation has been used due to its interesting properties for generating discrete equivalences [40–43]. Under this method, the discretization considers the following model:

$$D^{\alpha}(z^{-1}) = \left(\frac{1-z^{-1}}{T_c}\right) = \sum_{k=0}^{\infty}\left(\frac{1}{T_c}\right)^{\alpha}(-1)\binom{\alpha}{k}z^{-k} = \sum_{k=0}^{\infty}h^{\alpha}(k)z^{-k}, \tag{7.6}$$

where $h^{\alpha}(k)$ is the impulse response sequence, whereas T_c represents the sampling frequency. It has been already demonstrated in the literature [36] that rational models converge faster than polynomial methods. Consequently, the Padé approximation approach has been employed to obtain a fractional model from the impulse response by using the definition provided in Eq. (7.7).

$$H(z^{-1}) = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + \ldots + a_n z^{-n}} = \sum_{k=0}^{\infty}h(k)z^{-k},\ m \le n \tag{7.7}$$

where m, n and the parameters a_i and b_i are calculated by adjusting the first $m + n + 1$ coefficients of $h^{\alpha}(k)$.

7.3 Fuzzy Controller

A fuzzy controller (FC) is a nonlinear system produced from empirical rules. Such empirical information may come from a human operator who directly manipulates the process. Each rule, just as the natural language, presents an IF-THEN format. The collection of all rules constitutes the rule base that emulates the decision-making process of the operator. An important characteristic of one FC is the partitioning of the control scheme into regions [44]. At each region, the control strategy can be simply modeled by using a rule that associates the region under which certain actions are performed. Despite proposing several configurations of FCs in the literature, the fuzzy fractional PD$^{\alpha}$+I has been selected since it presents interesting characteristics of robustness and stability [5]. In this structure, the integral error is incorporated to the output of the fuzzy fractional PD$^{\alpha}$ controller. Under this configuration, the integral action supports the elimination of the final steady state error.

The controller configuration is shown in Fig. 7.1. In the figure, *E*, *DE* and *IE* represents the error, the fractional derivative error and the integral error, respectively. It has four gains K_p, K_d, K_i and K_u to be calibrated, the first three gains correspond to the input and the last one to the output. The control action u is a nonlinear mapping function of *E*, *DE* and *IE* with the following model:

$$\begin{aligned} u(k) &= (f(E, CE) + IE)K_u \\ u(k) &= \left[f\left(K_p e,\, K_d D^{\alpha} e\right) + K_i I e\right]\cdot K_u, \end{aligned} \tag{7.8}$$

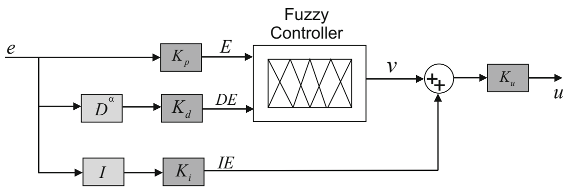


Fig. 7.1 Fuzzy PD$^\alpha$ + I controller

Table 7.1 Rule base of the controller to be calibrated

E/DE	NL	NM	NS	ZR	PS	PM	PL
NL	NL	NL	NL	NL	NM	NS	ZR
NM	NL	NL	NL	NM	NS	ZR	PS
NS	NL	NL	NM	NS	ZR	PS	PM
ZR	NL	NM	NS	ZR	PS	PM	PL
PS	NM	NS	ZR	PS	PM	PL	PL
PM	NS	ZR	PS	PM	PL	PL	PL
PL	ZR	PS	PM	PL	PL	PL	PL

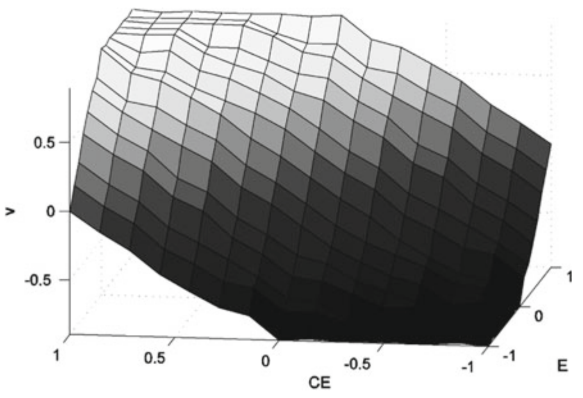


Fig. 7.2 Control surface

A fuzzy controller consists of three conceptual components: a rule base, which contains a selection of fuzzy rules; a database, which defines the membership functions used by the fuzzy rules; and a reasoning mechanism, which performs the inference procedure. There are two different fuzzy systems: the Mamdani [45] and the Takagi-Sugeno (TS) [46]. In order to maintain compatibility to similar works reported in the literature, the rule base and the membership functions are selected the same as [5, 9]. Under such conditions, Table 7.1 shows the rule base used by the fuzzy controller to be calibrated. In Table 7.1, **NL**, **NM**, **NS**, **ZR**, **PS**, **PM** and **PL** represent the linguistic variables "Negative Large", "Negative Medium", "Negative Small", "Zero", "Positive Small", "Positive Medium" and "Positive Large", respectively. Figure 7.2 shows the membership functions that model the premises and the consequences of each rule. Consequently, a determined rule from Table 7.1 can be constructed in the following form:

If $\boldsymbol{E}$ is **NL** and $\boldsymbol{DE}$ is **ZR** then $\boldsymbol{v}$ is **NL**

In this rule, the control strategy can be simply modeled as follows: if the error is "Negative Large" and the error derivate "Zero" then the output is "Negative Large". The acting of all rules produces the control strategy which is shown by the nonlinear surface in Fig. 7.2.

7.4 Social Spider Optimization (SSO)

The social spider optimization (SSO) algorithm [15] is a swarm computation method that emulates the cooperative behavior of spiders within a communal colony. SSO has been designed to find the global solution of a nonlinear optimization problem with box constraints in the form:

$$\begin{array}{l} \text{minimize } \; f(\mathbf{x}) \;\; \mathbf{x} = (x^1, \ldots, x^d) \in \mathbb{R}^d \\ \text{subject to } \mathbf{x} \in \mathbf{X} \end{array} \tag{7.9}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a nonlinear function whereas $\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^d \big| l_h \le x^h \le u_h, h = 1, \ldots, d\}$ is a bounded feasible space, constrained by the lower (l_h) and upper (u_h) limits.

SSO utilizes a population $\mathbf{S}$ of N candidate solutions to solve the problem formulated in Eq. (7.1). Each candidate solution represents a spider position whereas the general web symbolizes the search space $\mathbf{X}$. In SSO, the spider population $\mathbf{S}$ is classified into two categories: males ($\mathbf{M}$) and females ($\mathbf{F}$). In order to simulate a real spider colony, in SSO, the number N_f of females $\mathbf{F}$ is randomly selected within a range of 65–90% of the entire population $\mathbf{S}$, whereas the rest N_m is considered as male individuals $\left(N_m = N - N_f\right)$. Under such conditions, the Group $\mathbf{F}$ assembles the set of female individuals $\left(\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{N_f}\}\right)$ whereas $\mathbf{M}$ groups the male members $\left(\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{N_m}\}\right)$, where $\mathbf{S} = \mathbf{F} \cup \mathbf{M}$ ($\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$), such that $\mathbf{S} = \{\mathbf{s}_1 = \mathbf{f}_1, \mathbf{s}_2 = \mathbf{f}_2, \ldots, \mathbf{s}_{N_f} = \mathbf{f}_{N_f}, \mathbf{s}_{N_f+1} = \mathbf{m}_1, \mathbf{s}_{N_f+2} = \mathbf{m}_2, \ldots, \mathbf{s}_N = \mathbf{m}_{N_m}\}$.

In the approach, each spider i maintain a weight w_i according to its solution quality. Therefore, w_i is calculated as follows:

$$w_i = \frac{fitness_i - worst}{best - worst} \tag{7.10}$$

where $fitness_i$ represents the fitness value produced by the evaluation of the i-th spider's position, $i \in 1, \ldots, N$. *best* and *worst* symbolize the best fitness value and worst fitness value of the whole population $\mathbf{S}$, respectively.

In the optimization process, the main mechanism of SSO is the information exchange, which it is simulated trough vibrations produced in the communal web. The vibration that a spider i perceives from a spider j is modeled with the following expression:

$$V_{i,j} = w_j e^{d_{i,j}^2} \tag{7.11}$$

where w_j represents the weight of the spider j and $d_{i,j}^2$ the distance between both spiders. It is considered that each spider i is only able to perceive three types of vibrations $V_{i,c}$, $V_{i,b}$ and $V_{i,f}$.

$V_{i,c}$ is the vibration transmitted by the nearest individual c with a higher weight with regard to i $(w_c > w_i)$. $V_{i,b}$ represents the vibration emitted by best element of the entire population **S**. Finally, $V_{i,f}$ considers the vibration produced by the nearest female spider. This vibration type is only applicable if i is a male individual.

In the operation of SSO, a population of N spiders is processed from the initial stage ($k = 0$) to a determined number *gen* of iterations ($k = gen$). Each individual depending on its gender is conducted by a set of different swarm operators. Therefore, in case of the female members, a new position $\mathbf{f}_i^{k+1}$ is generated by modifying the current element location $\mathbf{f}_i^k$. The modification is randomly controlled by using a probability factor *PF*. Consequently, the movement is produced in relation to other spiders according their vibrations, which are transmitted trough the communal web:

$$\mathbf{f}_i^{k+1} = \begin{cases} \mathbf{f}_i^k + \alpha \cdot V_{i,c} \cdot (\mathbf{s}_c - \mathbf{f}_i^k) + \beta \cdot V_{i,b} \cdot (\mathbf{s}_b - \mathbf{f}_i^k) \\ \qquad + \delta \cdot (\text{rand} - \frac{1}{2}) & \text{with probability } PF \\ \mathbf{f}_i^k - \alpha \cdot V_{i,c} \cdot (\mathbf{s}_c - \mathbf{f}_i^k) - \beta \cdot V_{i,b} \cdot (\mathbf{s}_b - \mathbf{f}_i^k) \\ \qquad + \delta \cdot (\text{rand} - \frac{1}{2}) & \text{with probability } 1 - PF \end{cases} \tag{7.12}$$

here α, β, δ and *rand* represent random numbers between [0, 1] whereas k is the iteration number. The individuals $\mathbf{s}_c$ and $\mathbf{s}_b$ symbolize the nearest member to i that maintains a higher weight and the best element of the complete population **S**, respectively.

On the other hand, male spider members are classified into two types: non-dominant (**ND**) and dominant (**D**). The dominant group **D** is composed by the half of the male individuals whose fitness values are better with regard to the complete male set. Consequently, the non-dominant (**ND**) category collects the rest of the male elements. In the optimization process, male members are operated according to the following model:

$$\mathbf{m}_i^{k+1} = \begin{cases} \mathbf{m}_i^k + \alpha \cdot V_{i,f} \cdot (\mathbf{s}_f - \mathbf{m}_i^k) + \delta \cdot (\text{rand} - \frac{1}{2}) & \text{if} \mathbf{m}_i^k \in \mathbf{D} \\ \mathbf{m}_i^k + \alpha \cdot \left(\frac{\sum_{h \in \mathbf{ND}} \mathbf{m}_h^k \cdot w_h}{\sum_{h \in \mathbf{ND}} w_h} - \mathbf{m}_i^k \right) & \text{if} \mathbf{m}_i^k \in \mathbf{ND} \end{cases} \tag{7.13}$$

where $\mathbf{s}_f$ symbolizes the nearest female element to the male individual i.

The final operation in SSO is mating. It is performed between dominant males and the female individuals. Under this operation, a new individual $\mathbf{s}_{new}$ is produced by the combination of a dominant male $\mathbf{m}_g$ and other female members within a specific range r. The weight of each involved element defines the probability of influence of each spider into $\mathbf{s}_{new}$. The elements with heavier weights are more likely to influence the new individual $\mathbf{s}_{new}$. Once $\mathbf{s}_{new}$ is generated, it is compared with the worst element of the colony. If $\mathbf{s}_{new}$ is better than the worst spider, the worst spider is replaced by $\mathbf{s}_{new}$. Otherwise, $\mathbf{s}_{new}$ is discarded. Figure 7.3 illustrates the operations of the optimization process performed by the SSO algorithm. More details can be found in [15].

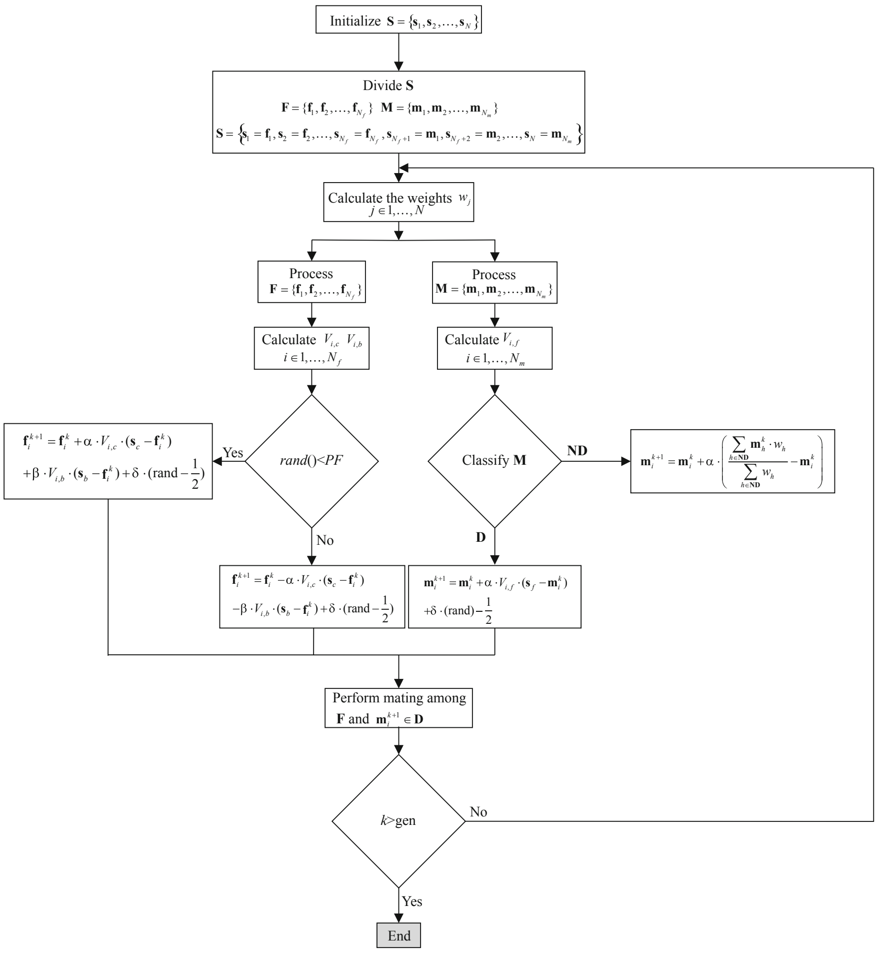

Initialize $\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$
Divide $\mathbf{S}$
$\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{N_f}\}$ $\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{N_m}\}$
$\mathbf{S} = \{\mathbf{s}_1 = \mathbf{f}_1, \mathbf{s}_2 = \mathbf{f}_2, \ldots, \mathbf{s}_{N_f} = \mathbf{f}_{N_f}, \mathbf{s}_{N_f+1} = \mathbf{m}_1, \mathbf{s}_{N_f+2} = \mathbf{m}_2, \ldots, \mathbf{s}_N = \mathbf{m}_{N_m}\}$
Calculate the weights w_j $j \in 1, \ldots, N$
Process $\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{N_f}\}$
Process $\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{N_m}\}$
Calculate $V_{i,c}$ $V_{i,b}$ $i \in 1, \ldots, N_f$
Calculate $V_{i,f}$ $i \in 1, \ldots, N_m$
$rand() < PF$
Yes
No
Classify $\mathbf{M}$
ND
D
$\mathbf{f}_i^{k+1} = \mathbf{f}_i^k + \alpha \cdot V_{i,c} \cdot (\mathbf{s}_c - \mathbf{f}_i^k) + \beta \cdot V_{i,b} \cdot (\mathbf{s}_b - \mathbf{f}_i^k) + \delta \cdot (\text{rand} - \frac{1}{2})$
$\mathbf{f}_i^{k+1} = \mathbf{f}_i^k - \alpha \cdot V_{i,c} \cdot (\mathbf{s}_c - \mathbf{f}_i^k) - \beta \cdot V_{i,b} \cdot (\mathbf{s}_b - \mathbf{f}_i^k) + \delta \cdot (\text{rand} - \frac{1}{2})$
$\mathbf{m}_i^{k+1} = \mathbf{m}_i^k + \alpha \cdot V_{i,f} \cdot (\mathbf{s}_f - \mathbf{m}_i^k) + \delta \cdot (\text{rand}) - \frac{1}{2}$
$\mathbf{m}_i^{k+1} = \mathbf{m}_i^k + \alpha \cdot \left(\frac{\sum_{h \in \mathbf{ND}} \mathbf{m}_h^k \cdot w_h}{\sum_{h \in \mathbf{ND}} w_h} - \mathbf{m}_i^k \right)$
Perform mating among $\mathbf{F}$ and $\mathbf{m}_i^{k+1} \in \mathbf{D}$
k>gen
No
Yes
End

Fig. 7.3 Operations of the optimization process performed by the SSO algorithm

7.5 Problem Formulation

In the design stage of fractional FCs, the parameter calibration process is transformed into a multidimensional optimization problem where fractional orders, as well as controller parameters of the fuzzy system, are both considered as decision variables. Under this approach, the complexity of the optimization problem tends to produce multimodal error surfaces whose cost functions are significantly difficult to minimize.

This chapter presents an algorithm for the optimal parameter calibration of fractional FCs. To determine the parameters, the estimation method uses the Social Spider Optimization (SSO) method. Different to the most of existent swarm algorithms, the method explicitly evades the concentration of individuals in the best positions, avoiding critical flaws such as the premature convergence to sub-optimal solutions and the limited balance of exploration-exploitation.

Therefore, the calibration process consists of finding the optimal controller parameters that present the best possible performance for the regulation of a dynamical system. Figure 7.4 illustrates the SSO scheme for the parameter calibration process.

Under such conditions, the fractional fuzzy controller parameters $(\alpha, K_p, K_d, K_i, K_u)$ represent the dimensions of each candidate solution (spider position) for the calibration problem. To evaluate the performance of the fractional fuzzy controller under each parameter configuration (candidate solution), the Integral Time Absolute Error (ITAE) [47] criterion has been considered. The ITAE index J measures the similarity between the closed-loop step response $y(t)$ produced by a determined parameter configuration $(\alpha, K_p, K_d, K_i, K_u)$ and the step function $r(t)$. Therefore, the quality of each candidate solution is evaluated according to the following model:

$$J(\alpha, K_p, K_d, K_i, K_u) = \int_0^\infty t|r(t) - y(t)| \tag{7.14}$$

Thereby, the problem of parameter calibration can be defined by the following optimization formulation:

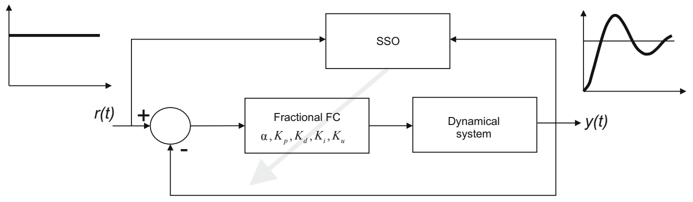


Fig. 7.4 SSO scheme for the parameter calibration process

$$\begin{aligned} &\text{minimize } J(\mathbf{x}) \; \mathbf{x} = (\alpha, K_p, K_d, K_i, K_u) \in \mathbb{R}^5 \\ &\text{subject to} \quad 0 \le \alpha \le 3 \\ &\qquad\qquad\quad 0 \le K_p \le 5 \\ &\qquad\qquad\quad 0 \le K_d \le 5 \\ &\qquad\qquad\quad 0 \le K_i \le 5 \\ &\qquad\qquad\quad 0 \le K_u \le 5 \end{aligned} \tag{7.15}$$

7.6 Numerical Simulations

This section presents the performance of the SSO scheme for the calibration of fractional FCs considering several dynamical systems. The algorithm is also evaluated in comparison to other similar approaches that are based on swarm algorithms. To test the performance of the SSO approach, the technique uses a representative set of three transfer functions that have been previously employed. Equations (7.4)–(7.6) present the transfer functions that are used in our simulations. Such functions involve three different system categories: High-order plants ($G_1(s)$), non-minimum systems ($G_2(s)$) and dynamical fractional systems ($G_3(s)$).

$$G_1(s) = \frac{1}{(s+1)(1+0.5s)(1+0.25s)(1+0.125s)} \tag{7.16}$$

$$G_2(s) = \frac{1-5s}{(s+1)^3} \tag{7.17}$$

$$G_3(s) = \frac{1}{(s^{1.5}+1)} \tag{7.18}$$

In the experiments, we have applied the SSO algorithm to calibrate the fractional parameters for each dynamical systems, and the results are compared to those produced by the Genetic Algorithms (GA) [5], Particle Swarm Optimization (PSO) [9], Harmony Search (HS) [10], Gravitational Search Algorithm (GSA) [11] and Cuckoo Search (CS) [12]. In the comparison, all methods have been set according to their own reported guidelines. Such configurations are described as follows:

1. PSO, parameters $c_1 = 2$, $c_2 = 2$ and weights factors have been set to $w_{\max} = 0.9$, and $w_{\min} = 0.4$ [17].
2. GA, the crossover probability is 0.55, the mutation probability is 0.10 and number of elite individuals is 2. Furthermore, the roulette wheel selection and the 1-point crossover are both applied.
3. GSA, from the model $G_t = G_O e^{-\alpha \frac{t}{T}}$, it is considered $\alpha = 10$, $G_O = 100$ and $T = 100$ or $T = 500$.
4. HS, its parameters are set as follows: the harmony memory consideration rate $HMCR = 0.7$, the pitch adjustment rate $PAR = 0.3$ and the Bandwidth rate $BW = 0.1$.

5. CS, its elements are configured such as the discovery rate $p_a = 0.25$ and the stability index $\beta = 3/2$.
6. SSO, the parameter PF has been set to 0.7 following an experimental definition.

The experimental results are divided into three sections. In the first Sect. 7.6.1, the performance of the SSO algorithm is evaluated with regard to high-order plants $(G_1(s))$. In the second Sect. 7.6.2, the results for non-minimum systems $(G_2(s))$ are provided and finally, in the third Sect. 7.6.3, the performance of the calibration scheme over fractional dynamical systems $(G_3(s))$ is discussed.

7.6.1 Results Over High-Order Plants $(G_1(s))$

In this experiment, the performance of the SSO calibration scheme is compared to GA, PSO, HS, GSA and CS, considering the regulation of high-order dynamical systems $(G_1(s))$. In the simulations, a temporal response from 0 to 10 s has been considered. In the comparison, all algorithms are operated with a population of 50 individuals (**N** = 50). To appropriately evaluate the convergence properties of all calibration methods, the maximum number of generations has been set to (A) 100 iterations and (B) 500 iterations. This stop criterion has been selected to maintain compatibility to similar works reported in the literature [5, 9, 28]. By selecting such number of iterations, the experiment aims to test the quality of the produced solutions when the operation of each calibration method is limited to a reduced number of iterations.

All the experimental results in this section consider the analysis of 35 independent executions of each algorithm. Table 7.2 presents the calibrated parameters obtained through each method. Such results consider the best controller parameters in terms of the produced ITAE values after 100 iterations. On the other hand, Table 7.3 shows the calibrated parameters considering 500 iterations.

According to Tables 7.2 and 7.3, the SSO scheme provides better performance than GA, PSO, HS, GSA and CS for both cases. Such differences are directly related to a better trade-off between exploration and exploitation of the SSO method. It is also evident that the SSO method produces similar results with 100 or 500 iterations.

Table 7.2 Calibrated parameters for $G_1(s)$ produced by each algorithm after 100 iterations

$G_1(s)$	K_p	K_d	K_i	K_u	α	ITAE
PSO	0.3034	0	0.4475	2.9988	0	5281.2115
GA	0.7581	0.3510	0.3038	4.3276	0.8000	926.1352
GSA	1.3387	2.7209	0.5482	1.0545	0.2311	4164.1935
HS	0.7867	0.8128	0.8271	3.6129	0.9319	3562.1834
CS	0.9700	0.3497	0.4054	3.0002	0.9516	916.5816
SSO	0.8100	0.3493	0.2392	4.8235	0.9897	492.2912

Table 7.3 Calibrated parameters for $G_1(s)$ produced by each algorithm after 500 iterations

$G_1(s)$	K_p	K_d	K_i	K_u	α	ITAE
PSO	0	0.5061	0.4681	4.0578	0.4629	2900.7502
GA	1.1860	0.3826	0.5204	1.5850	0.9497	974.0881
GSA	1.2000	0.6531	0.9607	2.5442	0.8745	1975.3254
HS	1.3112	0.9450	0.9262	1.2702	0.6075	2776.2160
CS	1.0093	0.4506	0.2611	4.6964	1.0002	464.5376
SSO	1.0386	0.4621	0.2751	4.4147	0.9998	473.7492

Therefore, it can be established that SSO maintains better convergence properties than GA, PSO, HS, GSA and CS in the process of parameter calibration.

Figure 7.5 exhibits the step responses produced by each parameter set, considering 100 and 500 iterations. The remarkable convergence rate of the SSO algorithm can be observed at Fig. 7.5. According to the graphs, the step responses produced by the controller parameters that have been defined through SSO, are practically the same irrespective of the number of iterations. This fact means that the SSO scheme is able to find an acceptable solution in less than 100 iterations.

7.6.2 Results Over Non-minimum Systems $(G_2(s))$

This section presents the comparison of the SSO calibration scheme with GA, PSO, HS, GSA and CS, considering the regulation of non-minimum systems $(G_2(s))$. Non-minimum systems are defined by transfer functions with one or more poles or zeros in the right half of the s-plane. As a consequence, the response of a non-minimum system to a step input exhibits an "undershoot", which indicates that the output of the dynamical system becomes negative first before changing direction to positive values.

The experimental simulation runs from 0 to 50 s. All algorithms are operated with a population of 50 individuals ($\mathbf{N} = 50$), matching with the experiments in Sect. 7.6.1. Table 7.4 presents the calibrated parameter of each method after 100 iterations, while Table 7.5 exhibits the results for 500 iterations. Both tables show that the SSO scheme delivers better results than GA, PSO, HS, GSA and CS in terms of the ITAE index. Figure 7.6 presents the step responses produced by each parameter set, considering 100 and 500 iterations. By analyzing the plot in Fig. 7.6, it is observed that the step response of the SSO scheme is less sensitive to the number of iterations than other techniques.

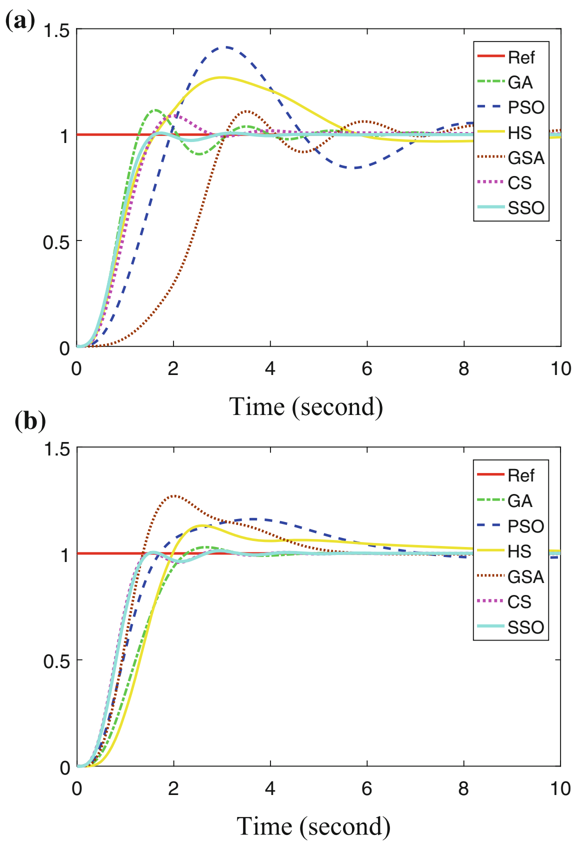


Fig. 7.5 Step responses after applying the calibrated parameters to the high order plant $G_1(s)$ with **a** 100 iterations, and with **b** 500 iterations

Table 7.4 Calibrated parameters for $G_2(s)$ produced by each algorithm after 100 iterations

$G_2(s)$	K_p	K_d	K_i	K_u	α	ITAE
PSO	0.4645	0	0.2378	0.4147	0.0643	50,289.0994
GA	0.6061	0.0326	0.3175	0.2909	0.2000	55,043.6316
GSA	0.9377	1.8339	0.5020	0.1427	1.0266	101,160.6241
HS	2.2838	3.7685	0.1328	0.5324	2.3214	126,996.7047
CS	0.9305	1.1329	1.1045	0.0674	0.0222	90,962.6199
SSO	0.4668	0.1165	0.2139	0.4642	0.5470	44,368.6620

Table 7.5 Calibrated parameters for $G_2(s)$ produced by each algorithm after 500 iterations

$G_2(s)$	K_p	K_d	K_i	K_u	α	ITAE
PSO	0.4606	0	0.2027	0.4866	0.0688	46,912.4985
GA	1.0449	1.1921	0.8839	0.0903	0.0822	81,550.0790
GSA	0.8862	1.3919	0.3746	0.2386	1.9259	63,186.5783
HS	1.0362	1.1105	0.5360	0.1389	0.9007	91,536.3894
CS	0.0386	0.0059	0.0243	4.0625	0.5516	43,565.1588
SSO	0.4537	0.1597	0.2004	0.5124	0.6422	41,772.3344

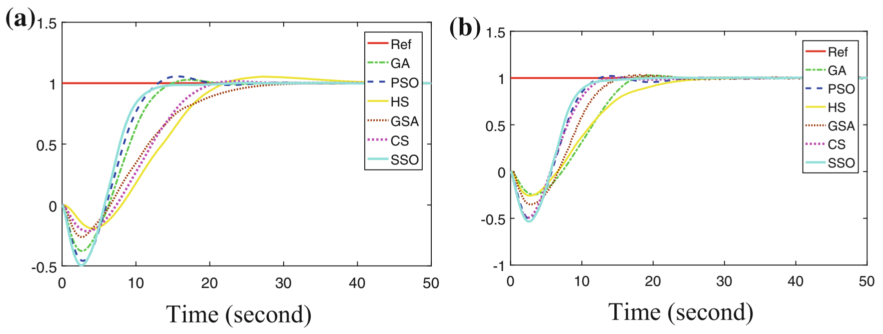


Fig. 7.6 Step responses after applying the calibrated parameters to the high order plant $G_2(s)$ with **a** 100 iterations, and with **b** 500 iterations

7.6.3 Results Over Fractional Systems ($G_3(s)$)

Unlike high-order plants and non-minimum systems, fractional dynamical systems produce multimodal error surfaces with different local optima. As a consequence, fractional fuzzy controllers that regulate their behavior are, in general, more difficult to calibrate [9]. Under such conditions, the experiment reflects the capacity of each calibration algorithm to locate the global optimum in presence of several local optima.

In this experiment, the performance of the SSO calibration scheme is compared to GA, PSO, HS, GSA and CS, considering the regulation of fractional dynamical systems ($G_3(s)$). In the simulations, a temporal response from 0 to 3 s is considered. In the test, all algorithms are operated with a population of 50 individuals ($\mathbf{N} = 50$).

The calibrated parameters are averaged over 30 executions obtaining the values reported in Tables 7.6 and 7.7. The results exhibit the configuration for each method with 100 and 500 iterations, respectively. It is evident that the SSO scheme presents better performance than PSO, HS and GSA independently on the number of iterations. However, the difference between GA and the SSO approach in terms of the ITAE index is relatively small for the case of 100 iterations. On the other hand, in the case of 500 iterations, the performance among the SSO approach, GA and CS are practically the same. Figure 7.7 presents the step response that is produced by each

Table 7.6 Calibrated parameters for $G_3(s)$ produced by each algorithm after 100 iterations

$G_3(s)$	K_p	K_d	K_i	K_u	α	ITAE
PSO	1.331	0	0.6937	5	5	311.4558
GA	1.3329	0.6341	0.6130	5	0.4932	97.7016
GSA	1.0823	0.6463	0.2924	4.0152	0.5802	346.6765
HS	0.7867	0.8128	0.8271	3.6129	0.9319	3562.1834
CS	1.2220	0.6590	0.6647	5	0.4232	105.0266
SSO	1.3173	0.6560	0.5932	4.9797	0.5091	98.5974

Table 7.7 Calibrated parameters for $G_3(s)$ produced by each algorithm after 500 iterations

$G_3(s)$	K_p	K_d	K_i	K_u	α	ITAE
PSO	1.0187	1.2010	0.7553	5	0	181.5380
GA	1.3320	0.5599	0.5991	5	0.5631	97.1981
GSA	1.3319	0.7502	0.6689	3.4968	0.5185	152.2198
HS	1.3112	0.9450	0.9262	1.2702	0.6075	2776.2160
CS	1.3325	0.5857	0.5987	4.9999	0.5450	97.0307
SSO	1.3087	0.5883	0.5808	4.9991	0.5642	97.1085

parameter set, considering 100 and 500 iterations. Similar to Sects. 7.6.1 and 7.6.2, it is demonstrated (from Fig. 7.7) that the SSO-calibrator obtains better solutions than GA, HS and PSO yet demanding a lower number of iterations.

Finally, in order to stress the importance of the fractional PD$^\alpha$ + I scheme, an experiment that evaluates the influence of the parameter α in the regulation of $G_3(s)$ is conducted. In the experiment, the fractional PD$^\alpha$ + I controller is operated as an integer fuzzy controller by setting $\alpha = 1$. Under such conditions, the rest of the parameters of PD$^\alpha$ + I (K_p, K_d, K_i, K_u) are calibrated through the SSO approach considering the regulation of the fractional system $G_3(s)$. Then, the values α vary from 0 to 1 while registering the performance of the regulation.

As a result of the optimization method, the following parameter values are obtained: $(K_p, K_d, K_i, K_u) \equiv (1.3087, 0.5883, 0.5808, 4.0012)$ with ITAE $= 418.8032$. After calibrating the integer fuzzy controller, the values of α are modified from 0 to 1, while the parameter set remains fixed to $(1.3087, 0.5883, 0.5808, 4.0012)$. Table 7.8 presents the results obtained from the experiment. Such values report the regulation quality of $G_3(s)$ in terms of the ITEA values. By analyzing Table 7.8, it is clear that the regulation quality strongly depends on the selection of the order for α. Particularly in this experiment, the best regulation performance is reached when the order of α is set to 0.6. Figure 7.8 presents the influence of α in terms of the regulation quality.

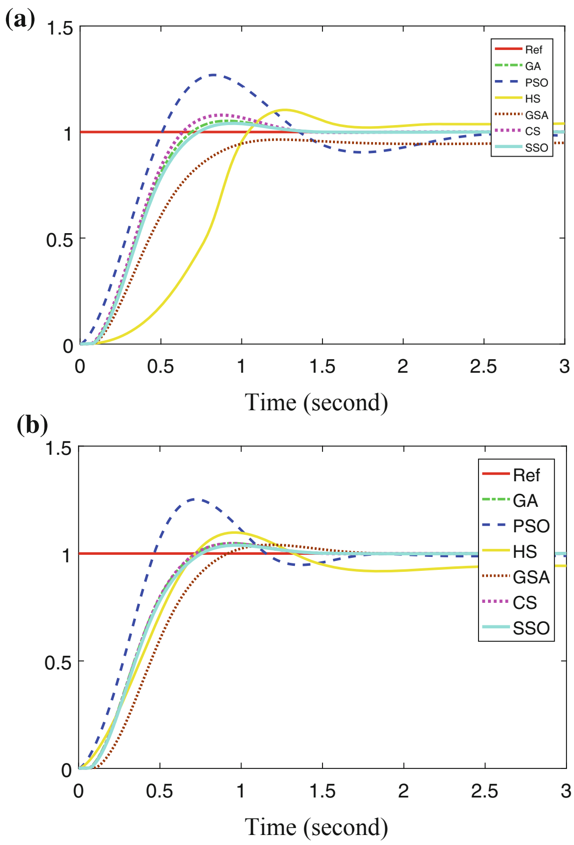


Fig. 7.7 Step responses after applying the calibrated parameters to the high order plant $G_3(s)$ with **a** 100 iterations, and with **b** 500 iterations

Table 7.8 Regulation quality of $G_3(s)$ in terms of the ITEA values

α	ITAE
0	216.839600
0.1	198.666000
0.2	180.783670
0.3	159.871905
0.4	134.127486
0.5	108.724153
0.6	**105.942730**
0.7	136.404140
0.8	183.213124
0.9	262.521840
1	418.803215

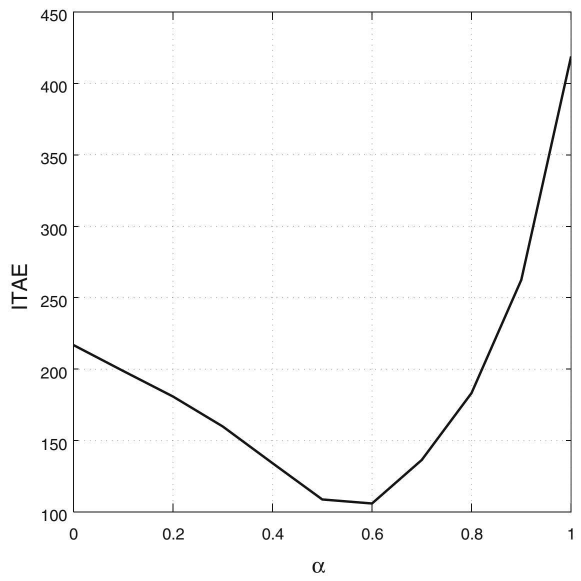


Fig. 7.8 Influence of α in the regulation quality of $G_3(s)$ in terms of the ITEA values

7.7 Conclusions

Due to its multiple applications, the calibration of fractional fuzzy controllers has attracted the interests of several research communities. In the calibration process, the parameter estimation is transformed into a multidimensional optimization problem whose fractional order and the corresponding controller parameters of the fuzzy system are considered as decision variables. Under this approach, the complexity of fractional-order chaotic systems tends to produce multimodal error surfaces for which their cost functions are significantly difficult to minimize. Several algorithms that are based on swarm intelligence principles have been successfully applied to calibrate the parameters of fuzzy systems. However, most of them still have an important limitation since they frequently obtain sub-optimal results as a consequence of an inappropriate balance between exploration and exploitation in their search strategies.

This chapter presents a method for the optimal parameter calibration of fractional FCs that is based on the SSO algorithm. The SSO algorithm is a novel swarm method that is inspired on the emulation of the collaborative behavior of social-spiders. Unlike most of the existing swarm algorithms, the method explicitly evades the concentration of individuals in best positions, avoiding critical flaws such as the premature convergence to sub-optimal solutions and the limited balance of exploration-exploitation.

In order to illustrate the proficiency and the robustness of this approach, SSO scheme has been experimentally evaluated considering three different system categories: high-order plants, non-minimum systems and dynamical fractional systems.

To assess the performance of the SSO algorithm, it has been compared to other similar swarm approaches such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Harmony Search (HS), Gravitational Search Algorithm (GSA) and Cuckoo Search (CS). The experiments have demonstrated that the SSO method outperforms other techniques in terms of solution quality and convergence.

References

1. Oldham, K., Spanier, J.: The Fractional Calculus: Theory and Application of Differentiation and Integration to Arbitrary Order. Academic Press, New York (1974)
2. Podlubny, I.: Fractional Differential Equations. Academic Press, San Diego (1999)
3. Das, S., Pan, I., Das, S., Gupta, A.: A novel fractional order fuzzy PID controller and its optimal time domain tuning based on integral performance indices. J. Eng. Appl. Artif. Intell. **25**, 430–442 (2012)
4. Delavari, H., Ghaderi, R., Ranjbar, A., Momani, S.: Fuzzy fractional order sliding mode controller for nonlinear systems. Commun. Nonlinear Sci. Numer. Simul. **15**(4), 963–978 (2010)
5. Jesus, I.S., Barbosa, R.S.: Genetic optimization of fuzzy fractional PD+I controllers. ISA Trans. **57**, 220–230 (2015)
6. Barbosa, R.S., Jesus, I.S.: A methodology for the design of fuzzy fractional PID controllers. In: ICINCO 2013—Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics, vol. 1, pp. 276–281 (2013)
7. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**, 338–353 (1965)
8. He, Y., Chen, H., He, Z., Zhou, L.: Multi-attribute decision making based on neutral averaging operators for intuitionistic fuzzy information. Appl. Soft Comput. **27**, 64–76 (2015)
9. Pana, I., Das, S.: Fractional order fuzzy control of hybrid power system with renewable generation using chaotic PSO. ISA Trans. **62**, 19–29 (2016)
10. Roy, G.G., Chakraborty, P., Das, S.: Designing fractional-order PIλDμ controller using differential harmony search algorithm. Int. J. Bio-Inspired Comput. **2**(5), 303–309 (2010)
11. Xu, Y., Zhou, J., Xue, X., Fu, W., Zhu, W., Li, C.: An adaptively fast fuzzy fractional order PID control for pumped storage hydro unit using improved gravitational search algorithm. Energy Convers. Manag. **111**, 67–78 (2016)
12. Sharma, R., Rana, K.P.S., Kumar, V.: Performance analysis of fractional order fuzzy PID controllers applied to a robotic manipulator. Expert Syst. Appl. **41**, 4274–4289 (2014)
13. Tan, K.C., Chiam, S.C., Mamun, A.A., Goh, C.K.: Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization. Eur. J. Oper. Res. **197**, 701–713 (2009)
14. Chen, G., Low, C.P., Yang, Z.: Preserving and exploiting genetic diversity in evolutionary programming algorithms. IEEE Trans. Evol. Comput. **13**(3), 661–673 (2009)
15. Cuevas, E., Cienfuegos, M., Zaldívar, D., Pérez-Cisneros, M.: A swarm optimization algorithm inspired in the behavior of the social-spider. Expert Syst. Appl. **40**(16), 6374–6384 (2013)
16. Goldberg, D.E.: Genetic Algorithm in Search Optimization and Machine Learning. Addison-Wesley (1989)
17. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, Dec 1995
18. Geem, Z.W., Kim, J., Loganathan, G.: Music-inspired optimization algorithm harmony search. Simulation **76**, 60–68 (2001)
19. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: a gravitational search algorithm. Inf. Sci. **179**, 2232–2248 (2009)
20. Yang, X.S., Deb, S.: Engineering optimization by cuckoo search. Int. J. Math. Model. Numer. Optim. **1**(4), 339–343 (2010)

21. Pereira, D.R., Pazoti, M.A., Pereira, L.A.M., Rodrigues, D., Ramos, C.O., Souza, A.N., Papa, J.P.: Social-spider optimization-based support vector machines applied for energy theft detection. Comput. Electr. Eng. **49**, 25–38 (2016)
22. Mirjalili, S.Z., Saremi, S., Mirjalili, S.M.: Designing evolutionary feedforward neural networks using social spider optimization algorithm. Neural Comput. Appl. **26**(8), 1919–1928 (2015)
23. Klein, C.E., Segundo, E.H.V., Mariani, V.C., Coelho, L.D.S.: Modified social-spider optimization algorithm applied to electromagnetic optimization. IEEE Trans. Magn. **52**(3), 2–10 (2016)
24. Ouadfel, S., Taleb-Ahmed, A.: Social spiders optimization and flower pollination algorithm for multilevel image thresholding: a performance study. Expert Syst. Appl. **55**, 566–584 (2016)
25. Tawhid, M.A., Ali, A.F.: A simplex social spider algorithm for solving integer programming and minimax problems. Memetic Comput. (In press)
26. Jesus, I., Machado, J.: Application of fractional calculus in the control of heat systems. J. Adv. Comput. Intell. Intell. Inform. **11**(9), 1086–1091 (2007)
27. Machado, J.: Analysis and design of fractional-order digital control systems. SAMS J. Syst. Anal. Modell. Simul. **27**, 107–122 (1997)
28. Liu, L., Pan, F., Xue, D.: Variable-order fuzzy fractional PID controller. ISA Trans. **55**, 227–233 (2015)
29. Shah, P., Agashe, S.: Review of fractional PID controller. Mechatronics **38**, 29–41 (2016)
30. El-Khazali, R.: Fractional-order PIλDμ controller design. Comput. Math. Appl. **66**, 639–646 (2013)
31. Owolabi, K.M.: Robust and adaptive techniques for numerical simulation of nonlinear partial differential equations of fractional order. Commun. Nonlinear Sci. Numer. Simul. **44**, 304–317 (2017)
32. Hélie, T.: Simulation of fractional-order low-pass filters. IEEE/ACM Trans. Audio Speech Lang. Process. **22**(11), 1636–1647 (2014)
33. Hwang, C., Leu, J.-F., Tsay, S.-Y.: A note on time-domain simulation of feedback fractional-order systems. IEEE Trans. Autom. Control **47**(4), 625–631 (2002)
34. Podlubny, I.: Fractional Differential Equations. Academic Press (1998)
35. Miller, K.S., Ross, B.: An Introduction to the Fractional Calculus and Fractional Differential Equations. Wiley (1993)
36. Dorcak, L.: Numerical models for the simulation of the fractional-order control systems. Numer. Model. Simul. Fractional-Order Control Syst. (1994)
37. Barbosa, R., Machado, J.A., Silva, M.: Time domain design of fractional differintegrators using least-squares. Signal Process. **86**(10), 2567–2581 (2006)
38. Chen, Y.Q., Vinagre, B., Podlubny, I.: Continued fraction expansion to discretize fractional order derivatives—an expository review. Nonlinear Dyn. **38**(1–4), 155–170 (2004)
39. Vinagre, B.M., Chen, Y., Petráš, I.: Two direct Tustin discretization methods for fractional-order differentiator/integrator. J. Frankl. Inst. **340**(5), 349–362 (2003)
40. Jacobs, B.A.: A new Grünwald-Letnikov derivative derived from a second-order scheme. Abstr. Appl. Anal. **2015**, Article ID 952057, 9 pages (2015). https://doi.org/10.1155/2015/952057
41. Scherer, R., Kalla, S.L., Tang, Y., Huang, J.: The Grünwald-Letnikov method for fractional differential equations. Comput. Math Appl. **62**, 902–917 (2011)
42. Liu, H., Li, S., Cao, J., Li, G., Alsaedi, A., Alsaadi, F.E.: Adaptive fuzzy prescribed performance controller design for a class of uncertain fractional-order nonlinear systems with external disturbances. Neurocomputing (In press)
43. Bigdeli, N.: The design of a non-minimal state space fractional-order predictive functional controller for fractional systems of arbitrary order. J. Process Control **29**, 45–56 (2015)
44. Cordón, O., Herrera, F.: A three-stage evolutionary process for learning descriptive and approximate fuzzy-logic-controller knowledge bases from examples. Int. J. Approximate Reasoning **17**(4), 369–407 (1997)

45. Mamdani, E., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man Mach. Stud. **7**, 1–13 (1975)
46. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE Trans. Syst. Man Cybern. SMC-15, 116–132 (1985)
47. Xu, J.-X., Li, C., Hang, C.C.: Tuning of fuzzy PI controllers based on gain/phase margin specifications and ITAE index. ISA Trans. **35**(1), 79–91 (1996)

Chapter 8
Locust Search Algorithm Applied to Multi-threshold Segmentation

Abstract In a computer vision one problem is image segmentation as an alternative the problem has been handled whit optimization algorithm. Most of the methods have two difficulties (1) sub-optimal results (2) the number of classes is previously known. In this chapter presented an algorithm that automatic selection of this classes for image segmentation whit a new objective function in gaussian mixture model. The optimization algorithm called Locust Search (LS), is based on behavior of locust swarms, presented a balance between exploration and exploitation. The method was tested over several images to validate the efficacy over other techniques.

8.1 Introduction

Image segmentation [1] consists in grouping image pixels based on some criteria such as intensity, color, texture, etc., and still represents a challenging problem within the field of image processing. Edge detection [2], region-based segmentation [3] and thresholding methods [4] are the most popular solutions for image segmentation problems.

Among such algorithms, thresholding is the simplest method. It works by considering threshold (points) values to adequately separate distinct pixels regions within the image being processed. In general, thresholding methods are divided into two types depending on the number of threshold values namely, bi-level and multilevel. In bi-level thresholding, only a threshold value is required to separate the two objects of an image (e.g. foreground and background). On the other hand, multilevel thresholding divides pixels into more than two homogeneous classes that require several threshold values.

The thresholding methods use a parametric or nonparametric approach [5]. In parametric approaches [6, 7], it is necessary to estimate the parameters of a probability density function that is capable of modelling each class. A nonparametric technique [8–11] employs a given criteria such as the between-class variance or the entropy and error rate, in order to determine optimal threshold values.

A common method to accomplish parametric thresholding is the modeling of the image histogram through a Gaussian mixture model [12] whose parameters define

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_8

a set of pixel classes (threshold points). Therefore, each pixel that belongs to a determined class is labeled according to its corresponding threshold points with several pixel groups gathering those pixels that share a homogeneous gray-scale level.

The problem of estimating the parameters of a Gaussian mixture that better model an image histogram has been commonly solved through the Expectation Maximization (EM) algorithm [13, 14] or Gradient-based methods such as Levenberg-Marquardt, LM [15]. Unfortunately, EM algorithms are very sensitive to the choice of the initial values [16], meanwhile Gradient-based methods are computationally expensive and may easily get stuck within local minima [17].

As an alternative to classical techniques, the problem of Gaussian mixture identification has also been handled through evolutionary methods. In general, they have demonstrated to deliver better results than those based on classical approaches in terms of accuracy and robustness [18]. Under these methods, an individual is represented by a candidate Gaussian mixture model. Just as the evolution process unfolds, a set of evolutionary operators are applied in order to produce better individuals. The quality of each candidate solution is evaluated through an objective function whose final result represents the similarity between the mixture model and the histogram. Some examples of these approaches involve optimization methods such as Artificial Bee Colony (ABC) [19], Artificial Immune Systems (AIS) [20], Differential Evolution (DE) [21], Electromagnetism optimization (EO) [22], Harmony Search (HS) [23] and Learning Automata (LA) [24]. Although these algorithms own interesting results, they present two important limitations. (1) They frequently obtain sub-optimal approximations as a consequence of a limited balance between exploration and exploitation in their search strategies. (2) They are based on the assumption that the number of Gaussians (classes) in the mixture is pre-known and fixed, otherwise they cannot work. The cause of the first limitation is associated to their evolutionary operators employed to modify the individual positions. In such algorithms, during their evolution, the position of each agent for the next iteration is updated yielding an attraction towards the position of the best particle seen so-far or towards other promising individuals. Therefore, as the algorithm evolves, these behaviors cause that the entire population rapidly concentrates around the best particles, favoring the premature convergence and damaging the appropriate exploration of the search space [25, 26]. The second limitation is produced as a consequence of the objective function that evaluates the similarity between the mixture model and the histogram. Under such an objective function, the number of Gaussians functions in the mixture is fixed. Since the number of threshold values (Gaussian functions) used for image segmentation varies depending on the image, the best threshold number and values are obtained by an exhaustive trial and error procedure.

On the other hand, bio-inspired algorithms represent a field of research that is concerned with the use of biology as a metaphor for producing optimization algorithms. Such approaches use our scientific understanding of biological systems as an inspiration that, at some level of abstraction, can be represented as optimization processes.

In the last decade, several optimization algorithms have been developed by a combination of deterministic rules and randomness, mimicking the behavior of natural phenomena. Such methods include the social behavior of bird flocking and fish schooling such as the Particle Swarm Optimization (PSO) algorithm [27] and the emulation of the differential evolution in species such as the Differential Evolution (DE) [28]. Although PSO and DE are the most popular algorithms for solving complex optimization problems, they present serious flaws such as premature convergence and difficulty to overcome local minima [29, 30]. The cause for such problems is associated to the operators that modify individual positions. In such algorithms, during the evolution, the position of each agent for the next iteration is updated yielding an attraction towards the position of the best particle seen so-far (in case of PSO) or towards other promising individuals (in case of DE). As the algorithm evolves, these behaviors cause that the entire population rapidly concentrates around the best particles, favoring the premature convergence and damaging the appropriate exploration of the search space [31, 32].

Recently, the collective intelligent behavior of insect or animal groups in nature has attracted the attention of researchers. The intelligent behavior observed in these groups provides survival advantages, where insect aggregations of relatively simple and "unintelligent" individuals can accomplish very complex tasks using only limited local information and simple rules of behavior [33]. Locusts (Schistocerca Gregaria) are a representative example of such collaborative insects [34]. Locust is a kind of grasshopper that can change reversibly between a solitary and a social phase, with clear behavioral differences among both phases [35]. The two phases show many differences regarding the overall level of activity and the degree to which locusts are attracted or repulsed among them [36]. In the solitary phase, locusts avoid contact to each other (locust concentrations). As consequence, they distribute throughout the space, exploring sufficiently over the plantation [36]. On other hand, in the social phase, locusts frantically concentrate around those elements that have already found good food-sources [37]. Under such a behavior, locust attempt to efficiently find better nutrients by devastating promising areas within the plantation.

This paper presents an algorithm for the automatic selection of pixel classes for image segmentation. The proposed method combines a novel evolutionary method with the definition of a new objective function that appropriately evaluates the segmentation quality with regard to the number of classes. The new evolutionary algorithm, called Locust Search (LS), is based on the behavior presented in swarms of locusts. In the proposed algorithm, individuals emulate a group of locusts which interact to each other based on the biological laws of the cooperative swarm. The algorithm considers two different behaviors: solitary and social. Depending on the behavior, each individual is conducted by a set of evolutionary operators which mimics different cooperative conducts that are typically found in the swarm. Different to most of existent evolutionary algorithms, the behavioral model in the proposed approach explicitly avoids the concentration of individuals in the current best positions. Such fact allows to avoid critical flaws such as the premature convergence to sub-optimal solutions and the incorrect exploration-exploitation balance. In order to automatically define the optimal number of pixel classes (Gaussian functions in the mixture), a new objective function has been also incorporated. The new objective function is divided in two parts. The first part evaluates the quality of each candidate solution in terms of its similarity with regard to the image histogram. The second part penalizes the overlapped area among Gaussian functions (classes). Under these circumstances, Gaussian functions that do not "positively" participate in the histogram approximation could be easily eliminated in the final Gaussian mixture model.

In order to illustrate the proficiency and robustness of the proposed approach, several numerical experiments have been conducted. Such experiments are divided into two sections. In the first part, the proposed LS method is compared to other well-known evolutionary techniques over a set of benchmark functions. In the second part, the performance of the proposed segmentation algorithm is compared to other segmentation methods which are also based on evolutionary principles. The results in both cases validate the efficiency of the proposed technique with regard to accuracy and robustness.

8.2 Gaussian Mixture Modelling

In this section, the modeling of image histograms through Gaussian mixture models is presented. Let consider an image holding L gray levels $[0, \ldots, L-1]$ whose distribution is defined by a histogram $h(g)$ represented by the following formulation:

$$\begin{gathered} h(g) = \frac{n_g}{Np}, \quad h(g) > 0, \\ Np = \sum_{g=0}^{L-1} n_g, \text{ and } \sum_{g=0}^{L-1} h(g) = 1, \end{gathered} \tag{8.1}$$

where n_g denotes the number of pixels with gray level g and Np the total number of pixels in the image. Under such circumstances, $h(g)$ can be modeled by using a mixture $p(x)$ of Gaussian functions of the form:

$$p(x) = \sum_{i=1}^{K} \frac{P_i}{\sqrt{2\pi}\sigma_i} \exp\left[\frac{-(x-\mu_i)^2}{2\sigma_i^2}\right] \tag{8.2}$$

where K symbolizes the number of Gaussian functions of the model whereas P_i is the apriori probability of function i. μ_i and σ_i represent the mean and standard deviation of the i-th Gaussian function, respectively. Furthermore, the constraint $\sum_{i=1}^{K} P_i = 1$ must be satisfied by the model. In order to evaluate the similarity between the image histogram and a candidate mixture model, the mean square error can be used as follows:

$$J = \frac{1}{n}\sum_{j=1}^{L}\left[p(x_j) - h(x_j)\right]^2 + \omega \cdot \left|\left(\sum_{i=1}^{K} P_i\right) - 1\right| \qquad (8.3)$$

where ω represents the penalty associated with the constrain $\sum_{i=1}^{K} P_i = 1$. Therefore, J is considered as the objective function which must be minimized in the estimation problem. In order to illustrate the histogram modeling through a Gaussian mixture, Fig. 8.3 presents an example, assuming three classes, i.e. $K = 3$. Considering Fig. 8.3a as the image histogram $h(x)$, the Gaussian mixture $p(x)$, that is shown in Fig. 8.3c, is produced by adding the Gaussian functions $p_1(x)$, $p_2(x)$ and $p_3(x)$ in the configuration presented in Fig. 8.3b. Once the model parameters that better model the image histogram have been determined, the final step is to define the threshold values $T_i(i \in [1, \ldots, K])$ which can be calculated by simple standard methods, just as it is presented in [19–21].

8.3 The Locust Search (LS) Algorithm

The Locust Search (LS) algorithm is a swarm optimization method inspired in several behaviors commonly found in swarms of locust [36]. In the LS method, the entire search space is assumed as a plantation, where all individuals within the swarm interact to each other. In the LS approach, each solution within the search space represents a locust position within the plantation. Also, each locust receives a food quality index based on the fitness value related to the solution that is represented by each of such individuals [20]. Furthermore, and unique to the LS method, individuals within the swarm's population are guided by a set of evolutionary operators based on two distinctive behaviors that are commonly observed in swarms of locust: (1) a solitary phase, and (2) a social phase.

8.3.1 LS Solitary Phase

One distinctive feature of the LS method is the integration of a unique behavior known as solitary phase. Under this behavioral model, each locust within the swarm is assumed to be displaced as a result of a social force, which is related to their positional relationships with respect to other members of the aggregation. Therefore, the net effect caused by such social force are: (1) an attraction toward distant individuals, or (2) a repulsion between nearer individuals.

In the LS approach, the concept of social force is applied to develop a solitary operator specifically devised to explicitly avoid the concentration of individual toward the best solutions found during the evolutionary process, which in turn allows a sufficient exploration of the entire search space [21]. In order to illustrate such operator, let $\mathbf{L}^k = \left\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\right\}$ denote a population (set of solutions) comprised by

N locusts, where $k = 1, 2, \ldots, gen$ denotes the current iteration number of whole the evolutionary process (with gen denoting the maximum number of iterations). At each iteration k, the solitary operation produces a new position $\mathbf{p}_i^k$ by perturbing the current locust position $\mathbf{l}_i^k$ with a change of position $\Delta\mathbf{l}_i^k$, such that:

$$\mathbf{p}_i^k = \mathbf{l}_i^k + \Delta\mathbf{l}_i^k \tag{8.4}$$

The position change $\Delta\mathbf{l}_i^k$ results from the social force experimented by the individual $\mathbf{l}_i^k$ with respect to the other $N-1$ individuals in the entire locust population. With that being said, the social force exerted between a given individual $\mathbf{l}_i^k$ and any other locust $\mathbf{l}_j^k$ within the swarm is calculated as follows:

$$\mathbf{s}_{ij}^k = \rho\left(\mathbf{l}_i^k, \mathbf{l}_j^k\right) \cdot s(r_{ij}) \cdot \mathbf{d}_{ij} + \text{rand}(1, -1) \tag{8.5}$$

where $r_{ij} = \left\| \mathbf{l}_i^k - \mathbf{l}_j^k \right\|$ denotes the Euclidian distance between individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$. Furthermore, $\mathbf{d}_{ij} = (\mathbf{l}_j^k - \mathbf{l}_i^k)/r_{ij}$ represent a unit vector which points from $\mathbf{l}_i^k$ to $\mathbf{l}_j^k$, while rand$(1, -1)$ stands for a randomly generated number from within the interval $[1, -1]$. Also, $s(r_{ij})$ represents the so called social relation between $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$, as defined as follows:

$$s(r_{ij}) = F \cdot e^{-r_{ij}/L} - e^{-r_{ij}} \tag{8.6}$$

where F and L denote, an attraction magnitude and an attractive length scale, respectively [36]. On the other hand, $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ stand for what is referred as the dominance value between $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$. In the LS approach, each individual from $\mathbf{L}^k$ $\left(\left\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\right\}\right)$ is ranked according to their fitness values, with the best individual (most dominant locust) receiving a rank of 0 (zero), whereas the worst individual (least dominant locust) gains the rank $N-1$. With that being said, the value of $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ is computed by considering the rank corresponding to the individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$, as defined as follows:

$$\rho(\mathbf{l}_i^k, \mathbf{l}_j^k) = \begin{cases} e^{-\left(\text{rank}(\mathbf{l}_i^k)/N\right)} & \text{if rank}(\mathbf{l}_i^k) < \text{rank}(\mathbf{l}_j^k) \\ e^{-\left(\text{rank}(\mathbf{l}_j^k)/N\right)} & \text{if rank}(\mathbf{l}_i^k) > \text{rank}(\mathbf{l}_j^k) \end{cases} \tag{8.7}$$

where rank$(\mathbf{l}_i^k)$ and rank$(\mathbf{l}_j^k)$ stand for the ranks of the individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$ respectively. Intuitively, the value $\rho(\mathbf{l}_i^k, \mathbf{l}_j^k)$ has the property to magnify or weaken the social force experimented between the individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$ depending on the fitness of the most dominant member among them.

Finally, the total social force experimented by individual $\mathbf{l}_i^k$ is computed as a superposition of all pairwise interactions exerted on it, as given as follows:

$$\mathbf{S}_i^k = \sum_{\substack{j=1 \\ j \neq 1}}^{N} \mathbf{s}_{ij}^k \tag{8.8}$$

Therefore, the change of position $\Delta \mathbf{l}_i^k$ is given by the total social force $\mathbf{S}_i^k$ experimented by $\mathbf{l}_i^k$, such that (Fig. 8.1):

$$\Delta \mathbf{l}_i^k = \mathbf{S}_i^k \tag{8.9}$$

Once a new set of positions $\mathbf{P}^k = \left\{\mathbf{p}_1^k, \mathbf{p}_2^k, \ldots, \mathbf{p}_N^k\right\}$, corresponding to the individuals within the population $\mathbf{L}^k\left(\left\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\right\}\right)$, has been computed, each individual's position $\mathbf{l}_i^k$ for the next iteration of the evolutionary process is updated as follows:

$$\mathbf{l}_i^{k+1} = \begin{cases} \mathbf{p}_i^k & \text{if } f(\mathbf{p}_i^k) > f(\mathbf{l}_i^k) \\ \mathbf{l}_i^k & \text{otherwise} \end{cases} \tag{8.10}$$

where $f(\mathbf{p}_i^k)$ and $f(\mathbf{l}_i^k)$ denotes the fitness evaluation function with respect to positions $\mathbf{p}_i^k$ and $\mathbf{l}_i^k$ respectively. In other words, the position of a given individual $\mathbf{l}_i^k$ for the iteration $k + 1$ is updated only if its respective position $\mathbf{p}_i^k$ promotes such individual to get a better fitness value than that on its current position; otherwise, its position remains unchanged. It is important to note that the previous is illustrated by considering a maximization optimization problem.

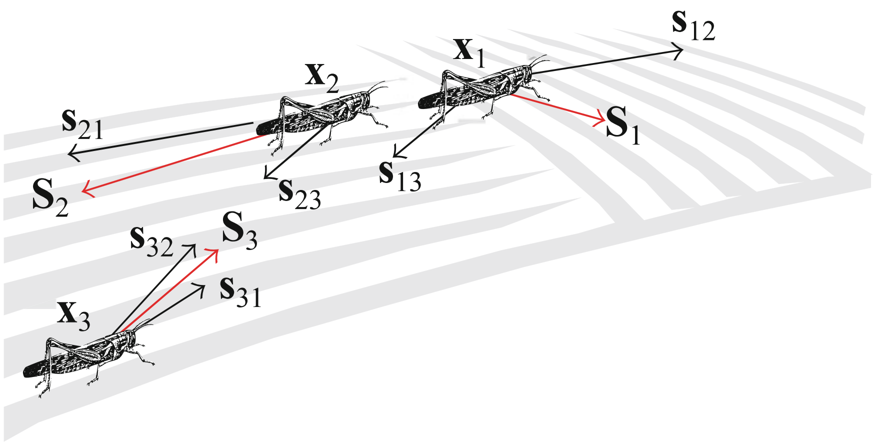


Fig. 8.1 LS solitary phase. Under this behavioral model, each locust's movement is computed respective to the total social force experimented by such individual

8.3.2 LS Social Phase

Different to the solitary phase illustrated in Sect. 8.3.1, in the LS approach, the so called social phase represents a selective operator used to refine a particular subset of individuals $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_q\}$ in order to improve their solution quality. Such subset $\mathbf{B}$ is formed by the q best individuals within the set of solutions $\mathbf{L}^{k+1} = \left\{\mathbf{l}_1^{k+1}, \mathbf{l}_2^{k+1}, \ldots, \mathbf{l}_N^{k+1}\right\}$, which correspond to the individuals' positions for the next iteration of the evolutionary process. Therefore, for each individual within the subset $\mathbf{B} \in \mathbf{L}^{k+1}$, a subspace C_j around each of such individuals is created. The limits of each of such subspaces depend on distance r, as defined as follows:

$$r = \frac{\sum_{d=1}^{n}\left(b_d^{high} - b_d^{low}\right)}{n} \cdot \beta \tag{8.11}$$

where b_d^{low} and b_d^{high} denote the lower and upper bounds in the d-th dimension, while n stand for total number of decision variables (dimensions). Furthermore, $\beta \in [0, 1]$ represents a scalar factor used to modulate the size of the subspace. Therefore, for each individual $\mathbf{l}_j^{k+1} = \left[l_{j,1}^{k+1}, l_{j,2}^{k+1}, \ldots, l_{j,n}^{k+1}\right]$ (where $\mathbf{l}_j^{k+1} \in \mathbf{B}$), the limits of each subspace C_j is given as follows:

$$\begin{aligned} C_{j,d}^{low} &= l_{j,d}^{k+1} - r \\ C_{j,d}^{high} &= l_{j,d}^{k+1} + r \end{aligned} \tag{8.12}$$

where $C_{j,d}^{low}$ and $C_{j,d}^{high}$ represent the upper and lower bounds of each subspace C_j at the d-th dimension, respectively. Finally, for each of subspace C_j, a new set of h new solutions $\mathbf{M}^j = \left\{\mathbf{m}_1^j, \mathbf{m}_2^j, \ldots, \mathbf{m}_h^j\right\}$ is generated within the bounds of each of such subspaces (see Fig. 8.4). If any of the fitness value of any of the solutions within $\mathbf{M}^j$ is better than that of their corresponding individual $\mathbf{l}_j^{k+1} \in \mathbf{B}$, then $\mathbf{l}_j^{k+1}$ is replaced with such better solution; otherwise, no changes are made to $\mathbf{l}_j^{k+1}$ (Fig. 8.2).

8.4 Segmentation Algorithm Based on LS

In the proposed method, the segmentation process is approached as an optimization problem. Computational optimization generally involves two distinct elements: (1) a search strategy to produce candidate solutions (individuals, particles, insects, locust, etc.) and (2) an objective function that evaluates the quality of each selected candidate solution. Several computational algorithms are available to perform the first element. The second element, where the objective function is designed, is unquestionably the most critical. In the objective function, it is expected to embody the full complexity of the performance, biases, and restrictions of the problem to be solved. In the

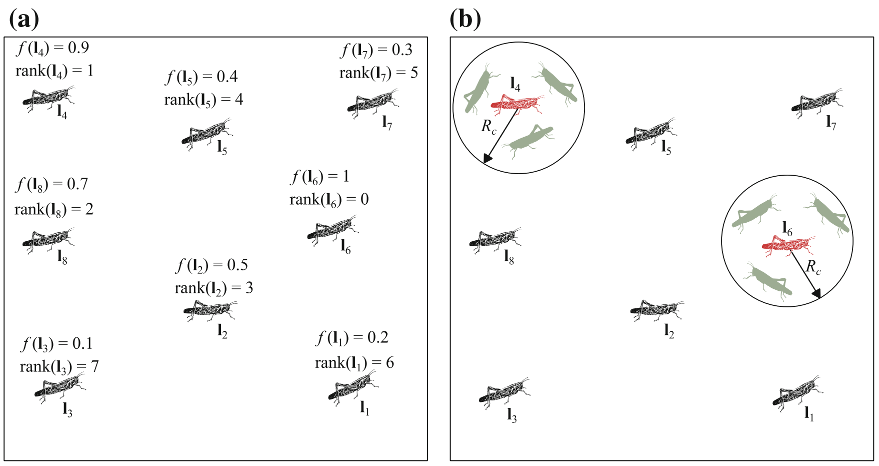


Fig. 8.2 LS social phase. **a** Initial locusts' configuration and rank according to their respective food quality indexes, and **b** social phase operator applied by considering $q = 2$ and $h = 3$

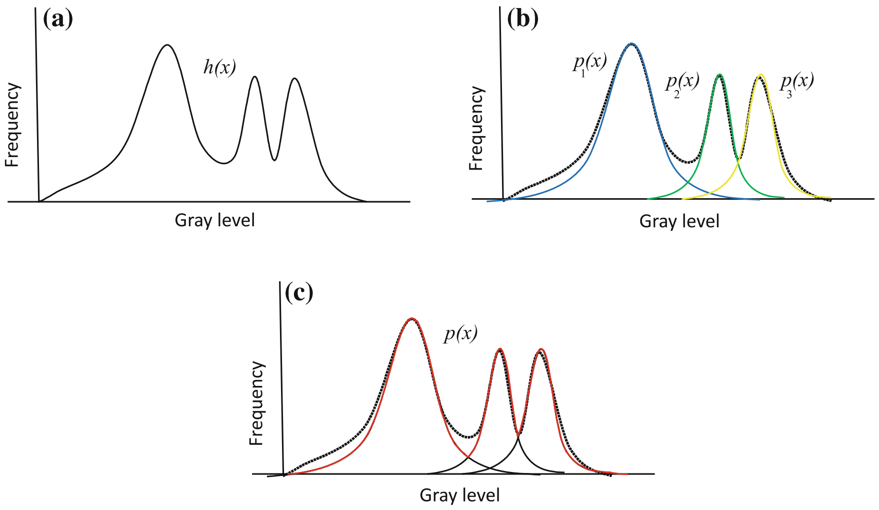


Fig. 8.3 Histogram approximation through a Gaussian mixture. **a** Original histogram, **b** configuration of the Gaussian components $p_1(x)$, $p_2(x)$ and $p_3(x)$, **c** final Gaussian mixture $p(x)$

segmentation problem, candidate solutions represent Gaussian mixtures. The objective function J (Eq. 8.3) is used as a fitness value to evaluate the similarity presented between the Gaussian mixture and the image histogram. Therefore, guided by the fitness values (J values), the set of encoded candidate solutions are evolved using the evolutionary operators until the best possible resemblance can be found (Fig. 8.3).

Over the last decade, several algorithms based on evolutionary and swarm principles [19–22] have been proposed to solve the problem of segmentation through a Gaussian mixture model. Although these algorithms own certain good performance indexes, they present two important limitations. (1) They frequently obtain sub-optimal approximations as a consequence of an inappropriate balance between exploration and exploitation in their search strategies. (2) They are based on the assumption that the number of Gaussians (classes) in the mixture is pre-known and fixed, otherwise they cannot work.

In order to eliminate such flaws, the proposed approach includes (A) a new search strategy and (B) the definition of a new objective function. For the search strategy, it is adopted the LS method. Under LS, the concentration of individuals in the current best positions is explicitly avoided. Such fact allows reducing critical problems such as the premature convergence to sub-optimal solutions and the incorrect exploration-exploitation balance.

8.4.1 New Objective Function J^{new}

Previous segmentation algorithms based on evolutionary and swarm methods use Eq. (8.3) as objective function. Under these circumstances, each candidate solution (Gaussian mixture) is only evaluated in terms of its approximation with the image histogram.

Since the proposed approach aims to automatically select the number of Gaussian functions K in the final mixture $p(x)$, the objective function must be modified. The new objective function J^{new} is defined as follows:

$$J^{new} = J + \lambda \cdot Q, \tag{8.13}$$

where λ is a scaling constant. The new objective function is divided in two parts. The first part J evaluates the quality of each candidate solution in terms of its similarity with regard to the image histogram (Eq. 8.3). The second part Q penalizes the overlapped area among Gaussian functions (classes), with Q being defined as follows:

$$Q = \sum_{i=1}^{K} \sum_{\substack{j=1 \\ j \neq i}}^{K} \sum_{l=1}^{L} \min\big(P_i \cdot p_i(l),\, P_j \cdot p_j(l)\big), \tag{8.14}$$

where K and L represents the number of classes and the gray levels, respectively. The parameters $p_i(l)$ and $p_j(l)$ symbolize the Gaussian functions i and j respectively, that are to be evaluated on the point l (gray level) whereas the elements P_i and P_j represent the apriori probabilities of the Gaussian functions i and j, respectively.

Under such circumstances, mixtures with Gaussian functions that do not "positively" participate in the histogram approximation are severely penalized.

Figure 8.4 illustrates the effect of the new objective function J^{new} in the evaluation of Gaussian mixtures (candidate solutions). From the image histogram (Fig. 8.4a), it is evident that two Gaussian functions are enough to accurately approximate the original histogram. However, if the Gaussian mixture is modeled by using a greater number of functions (for example four as it is shown in Fig. 8.4b), the original objective function J is unable to obtain a reasonable approximation. Under the new objective function J^{new}, it is penalized the overlapped area among Gaussian functions (classes). Such areas, in Fig. 8.4c, correspond to Q_{12}, Q_{23}, Q_{34}, where Q_{12} represents the penalization value produced between the Gaussian function $p_1(x)$ and $p_2(x)$. Therefore, due to the penalization, the Gaussian mixture shown in Fig. 8.4b and c provides a solution of low quality. On the other hand, the Gaussian mixture presented in Fig. 8.4d maintains a low penalty, thus, it represents a solution of high quality. From Fig. 8.4d, it is easy to see that functions $p_1(x)$ and $p_4(x)$ can be removed from the final mixture. This elimination could be performed by a simple comparison with a threshold value θ since $p_1(x)$ and $p_4(x)$ have a reduced amplitude ($p_1(x) \approx p_2(x) \approx 0$). Therefore, under J^{new}, it is possible to find the reduced Gaussian mixture model starting from a considerable number of functions.

Since the proposed segmentation method is conceived as an optimization problem, the overall operation can be reduced to solve the formulation of Eq. (8.15) by using the LS algorithm.

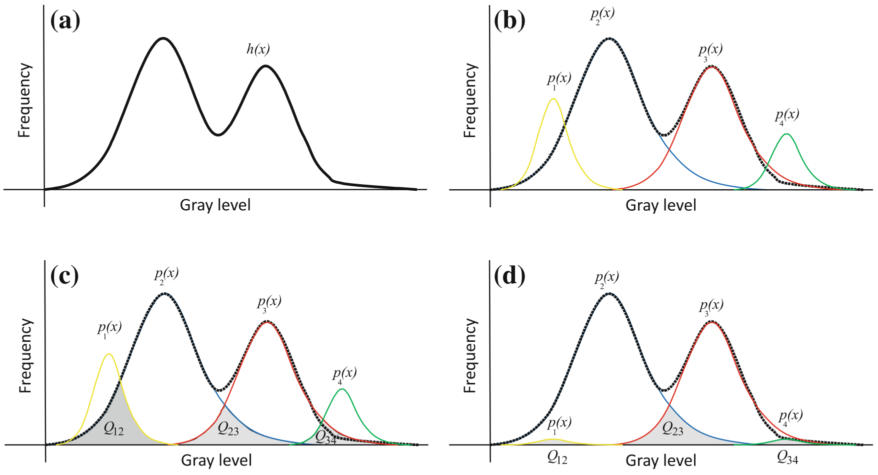


Fig. 8.4 Effect of the new objective function J^{new} in the evaluation of Gaussian mixtures (candidate solutions). **a** Original histogram, **b** Gaussian mixture considering four classes, **c** penalization areas and **d** Gaussian mixture of better quality solution

$$
\begin{array}{ll}
\text{minimize} & \begin{array}{c} J^{new}(\mathbf{x}) = J(\mathbf{x}) + \lambda \cdot Q(\mathbf{x}), \\ \mathbf{x} = (P_1, \mu_1, \sigma_1, \ldots, P_K, \mu_K, \sigma_K) \in \mathbf{R}^{3 \cdot K} \end{array} \\
\text{subject to} & \begin{array}{c} 0 \leq P_d \leq 1, \;\; d \in (1, \ldots, K) \\ 0 \leq \mu_d \leq 255 \\ 0 \leq \sigma_d \leq 25 \end{array}
\end{array}
\tag{8.15}
$$

where P_d, μ_d and σ_d represent the probability, mean and standard deviation of the class d. It is important to remark that the new objective function J^{new} allows the evaluation of a candidate solution in terms of the necessary number of Gaussian functions and its approximation quality. Under such circumstances, it can be used in combination with any other evolutionary method and not only with the proposed LS algorithm.

8.4.2 Complete Segmentation Algorithm

Once the new search strategy (LS) and objective function (J^{new}) have been defined, the proposed segmentation algorithm can be summarized by the Algorithm 1. The new algorithm combines operators defined by LS and operations for calculating the threshold values.

Algorithm 1. Segmentation LS algorithm

```
 1:  Input: $F$, $L$, $g$, $gen$, $N$, $K$, $\lambda$ and $\theta$.
 2:  Initialize $\mathbf{L}^0$ $(k = 0)$
 3:  until $(k = gen)$
 5:  $\mathbf{F} \leftarrow$ SolitaryOperation($\mathbf{L}^k$) Solitary operator (Sect. 8.3.1)
 6:  $\mathbf{L}^{k+1} \leftarrow$ SocialOperation($\mathbf{L}^k$, $\mathbf{F}$) Social operator (Sect. 8.3.2)
 8:  $k = k + 1$
 7:  end until
 8:  Obtain $\mathbf{l}_{best}^{gen}$
 9:  Reduce $\mathbf{l}_{best}^{gen}$
10:  Calculate the threshold values $T_i$ from the reduced model
11:  Use $T_i$ to segment the image
```

(Line 1) The algorithm sets the operative parameters F, L, g, gen, N, K, λ and θ. They rule the behavior of the segmentation algorithm. (Line 2) Afterwards, the population $\mathbf{L}^0$ is initialized considering N different random Gaussian mixtures of K functions. The idea is to generate an N-random Gaussian mixture subjected to the constraints formulated in Eq. (8.20). The parameter K must hold a high value in order to correctly segment all images (recall that the algorithm is able to reduce the Gaussian mixture to its minimum expression). (Line 3) Then, the Gaussian mixtures are evolved by using the LS operators and the new objective function J^{new}. This process is repeated during gen cycles. (Line 8) After this procedure, the best Gaussian mixture $\mathbf{l}_{best}^{gen}$ is selected according to its objective function J^{new}. (Line 9) Then, the Gaussian mixture $\mathbf{l}_{best}^{gen}$ is reduced by eliminating those functions whose amplitudes are lower than $\theta(p_i(x) \leq \theta)$. (Line 10) Then, it is calculated the threshold values T_i from the reduced model. (Line 11) Finally, the calculated T_i values are employed to segment the image. Figure 8.5 shows a flow chart of the complete segmentation procedure.

The proposed segmentation algorithm permits to automatically detect the number of segmentation partitions (classes). Furthermore, due to its remarkable search capacities, the LS method maintains a better accuracy than previous algorithms based

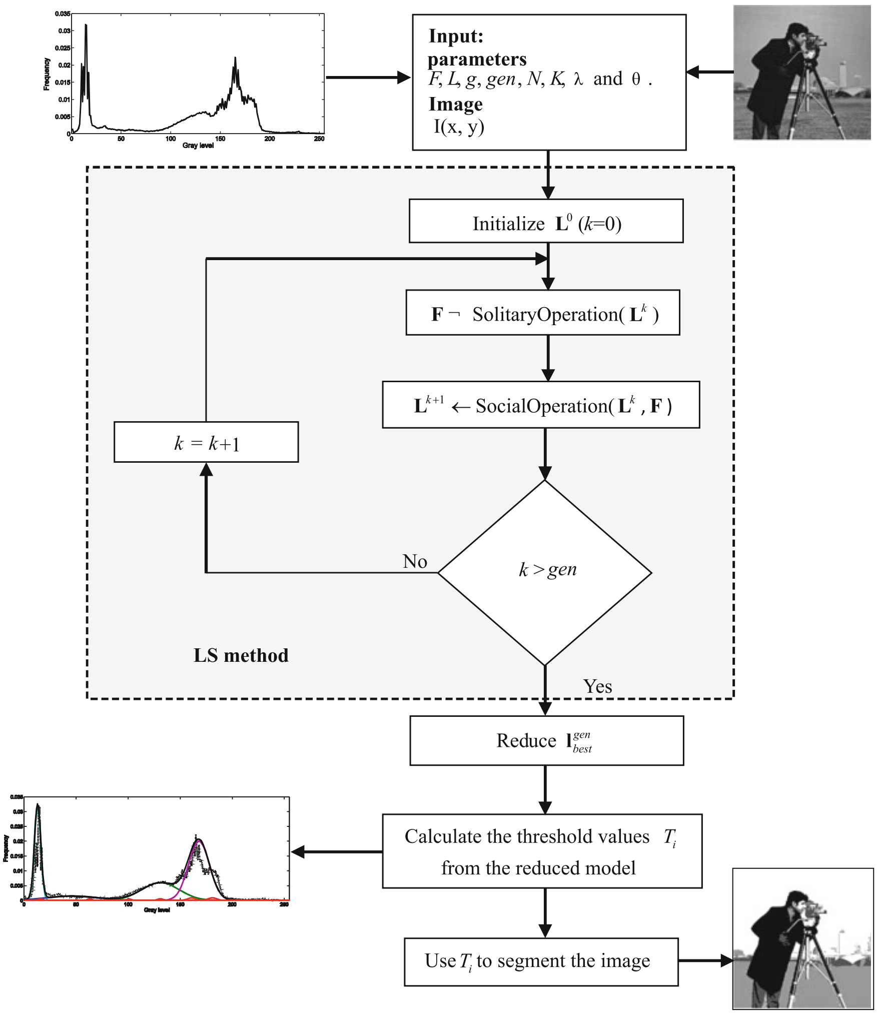

Input:
parameters
F, L, g, gen, N, K, λ and θ .
Image
I(x, y)
Initialize L⁰ (k=0)
F ← SolitaryOperation(Lᵏ)
Lᵏ⁺¹ ← SocialOperation(Lᵏ , F)
k = k+1
k > gen
No
Yes
LS method
Reduce l best gen
Calculate the threshold values Tᵢ from the reduced model
Use Tᵢ to segment the image
Frequency
Gray level

Fig. 8.5 Flow chart of the complete segmentation procedure

on evolutionary principles. However, the proposed method presents two disadvantages: first, it is related to its implementation which in general is more complex than most of the other segmentators based on evolutionary basics. The second refers to the segmentation procedure of the proposed approach which does not consider any spatial pixel characteristics. As a consequence, pixels that may belong to a determined region due to its position are labeled as a part of other region due to its gray-level intensity. Such a fact adversely affects the segmentation performance of the method.

8.5 Segmentation Results

This section analyses the performance of the proposed segmentation algorithm. The discussion is divided into three parts: the first one shows the performance of the proposed LS segmentation algorithm while the second presents a comparison between the proposed approach and others segmentation algorithms that are based on evolutionary and swam methods. The comparison mainly considers the capacities of each algorithm to accurately and robustly approximate the image histogram. Finally, the third part presents an objective evaluation of segmentation results produced by all algorithms that have been employed in the comparisons.

8.5.1 Performance of LS Algorithm in Image Segmentation

This section presents two experiments that analyze the performance of the proposed approach. Table 8.1 presents the algorithm's parameters that have been experimentally determined and kept for all the test images through all experiments.

<u>First image</u>
The first test considers the histogram shown by Fig. 8.6b while Fig. 8.6a presents the original image. After applying the proposed algorithm, just as it has been configured according to parameters in Table 8.1, a minimum model of four classes emerges after the start from Gaussian mixtures of 10 classes. Considering 30 independent executions, the averaged parameters of the resultant Gaussian mixture are presented in Table 8.2. One final Gaussian mixture (ten classes), which has been obtained by LS, is presented in Fig. 8.7. Furthermore, the approximation of the reduced Gaussian mixture is also visually compared with the original histogram in Fig. 8.6b. On the

Table 8.1 Parameter setup for the LS segmentation algorithm

F	L	g	gen	N	K	λ	θ
0.6	0.2	20	1000	40	10	0.01	0.0001

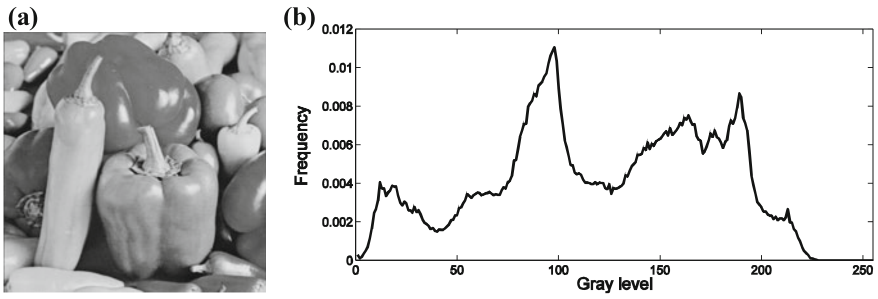


Fig. 8.6 **a** Original first image used on the first experiment, and **b** its histogram

Table 8.2 Results of the reduced Gaussian mixture for the first and the second image

Parameters	First image	Second image
$\overline{P_1}$	0.004	0.032
$\overline{\mu_1}$	18.1	12.1
$\overline{\sigma_1}$	8.2	2.9
$\overline{P_2}$	0.0035	0.0015
$\overline{\mu_2}$	69.9	45.1
$\overline{\sigma_2}$	18.4	24.9
$\overline{P_3}$	0.01	0.006
$\overline{\mu_3}$	94.9	130.1
$\overline{\sigma_3}$	8.9	17.9
$\overline{P_4}$	0.007	0.02
$\overline{\mu_4}$	163.1	167.2
$\overline{\sigma_4}$	29.9	10.1

other hand, Fig. 8.8 presents the segmented image after calculating the threshold points.

Second image

For the second experiment, the image in Fig. 8.9 is tested. The method aims to segment the image by using a reduced Gaussian mixture model obtained by the LS approach. After executing the algorithm according to parameters from Table 8.1, the resulting averaged parameters of the resultant Gaussian mixture are presented in Table 8.2. In order to assure consistency, the results are averaged considering 30 independent executions. Figure 8.10 shows the approximation quality that is obtained by the reduced Gaussian mixture model in (a) and the segmented image in (b).

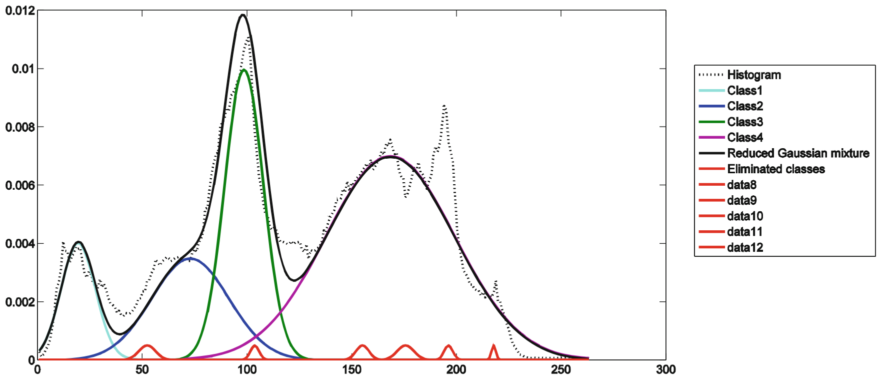


Fig. 8.7 Gaussian mixture obtained by LS for the first image

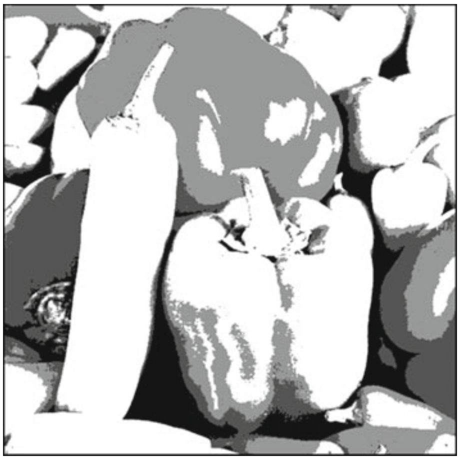


Fig. 8.8 Image segmented with the reduced Gaussian mixture

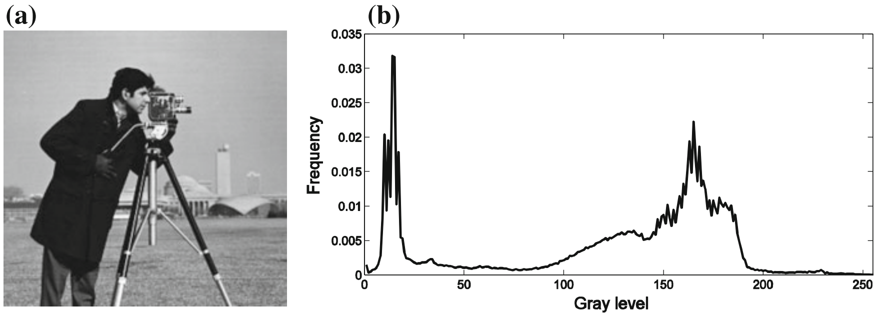
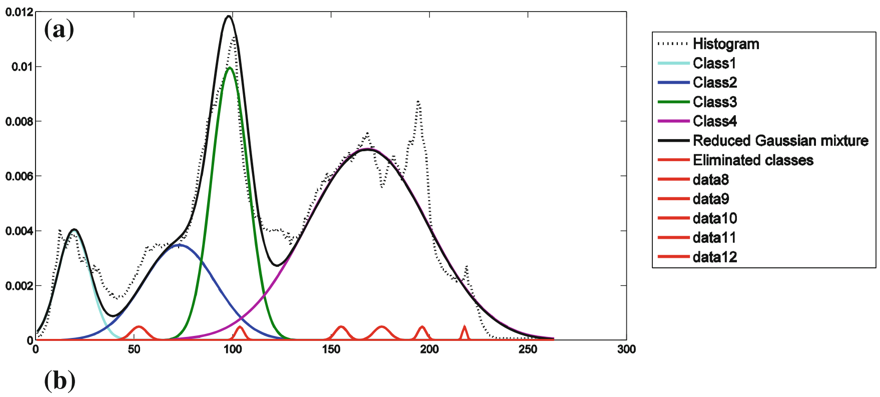


Fig. 8.9 **a** Original second image used on the first experiment, and **b** its histogram



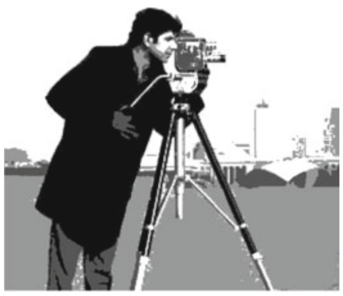


Fig. 8.10 **a** Segmentation result obtained by LS for the first image and **b** the segmented image

8.5.2 Histogram Approximation Comparisons

This section discusses the comparison between the proposed segmentation algorithm and other evolutionary-segmentation methods that have been proposed in the literature. The set of methods used in the experiments includes the J + ABC [19], J + AIS [20] and J + DE [21]. These algorithms consider the combination between the original objective function J (Eq. 8.3) and an evolutionary technique such as Artificial Bee Colony (ABC), the Artificial Immune Systems (AIS) and the Differential Evolution (DE) [21], respectively. Since the proposed segmentation approach considers the combination of the new objective function J^{new} (Eq. 8.13) and the LS algorithm, it will be referred as J^{new} + LS. The comparison focuses mainly on the capacities of each algorithm to accurately and robustly approximate the image histogram.

In the experiments, the populations have been set to 40 ($N = 40$) individuals. The maximum iteration number for all functions has been set to 1000. Such stop criterion has been considered to maintain compatibility to similar experiments that are reported in the literature [18]. The parameter setting for each of the segmentation algorithms in the comparison is described as follows:

1. J + J + ABC [19]: In the algorithm, its parameters are configured as follows: the abandonment limit = 100, $\alpha = 0.05$ and $limit = 30$.
2. J + AIS [20]: it presents the following parameters, $h = 120$, $N_c = 80$, $\rho = 10$, $P_r = 20$, $L = 22$ and $T_e = 0.01$.
3. J + DE [21]: The DE/Rand/1 scheme is employed. The parameter settings follow the instructions in [21]. The crossover probability is $CR = 0.9$ and the weighting factor is $F = 0.8$.
4. In J^{new} + LS, the method is set according to the values described in Table 8.5.

In order to conduct the experiments, a synthetic image is designed to be used as a reference in the comparisons. The main idea is to know in advance the exact number of classes (and their parameters) that are contained in the image so that the histogram can be considered as a ground truth. The synthetic image is divided into four sections. Each section corresponds to a different class which is produced by setting each gray level pixel Pv_i to a value that is determined by the following model:

$$Pv_i = e^{-\left(\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)}, \tag{8.16}$$

where i represents the section, whereas μ_i and σ_i are the mean and the dispersion of the gray level pixels, respectively. The comparative study employs the image of 512 × 512 that is shown in Fig. 8.11a and the algorithm's parameters that have been presented in Table 8.3. Figure 8.11b illustrates the resultant histogram.

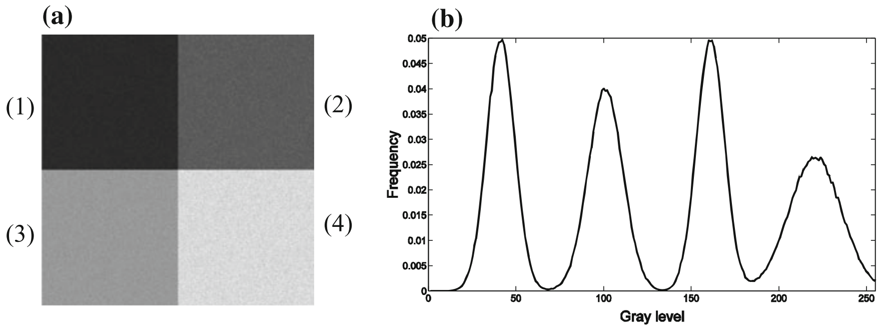


Fig. 8.11 **a** Synthetic image used in the comparison, and **b** its histogram

Table 8.3 Employed parameters for the design of the reference image

	P_i	μ_i	σ_i
(1)	0.05	40	8
(2)	0.04	100	10
(3)	0.05	160	8
(4)	0.027	220	15

In the comparison, the discussion focuses on the following issues: first of all, accuracy; second, convergence; and third, computational cost.

Convergence

This section analyzes the approximation convergence when the number of classes that are used by the algorithm during the evolution process is different to the actual number of classes in the image. Recall that the proposed approach automatically finds the reduced Gaussian mixture which better adapts to the image histogram.

In the experiment, the methods: J + ABC, J + AIS and J + DE are executed considering Gaussian mixtures composed of 8 functions. Under such circumstances, the number of classes to be detected is higher than the actual number of classes in the image. On the other hand, the proposed algorithm maintains the same configuration of Table 8.5. Therefore, it can detect and calculate up to ten classes ($K = 10$).

As a result, the techniques J + ABC, J + AIS and J + DE tend to overestimate the image histogram. This effect can be seen in Fig. 8.12a, where the resulting Gaussian functions are concentrated within actual classes. Such a behavior is a consequence of the evaluation that is considered by the original objective function J, which privileges only the approximation between the Gaussian mixture and the image histogram. This effect can be graphically illustrated by Fig. 8.13a that shows the pixel misclassification produced by the wrong segmentation of Fig. 8.9a. On the other hand, the proposed approach obtains a reduced Gaussian mixture model which allows the detection of each class from the actual histogram (see Fig. 8.12b). As a consequence, the segmentation is significantly improved by eliminating the pixel misclassification, as it is shown by Fig. 8.13b.

It is evident from Fig. 8.12 that the techniques, J + ABC, J + AIS and J + DE, all need an a priori knowledge of the number of classes that are contained in the actual histogram in order to obtain a satisfactory result. On the other hand, the proposed algorithm is able to find a reduced Gaussian mixture whose classes coincide with the actual number of classes that are contained in the image histogram.

Accuracy

In this section, the comparison among the algorithms in terms of accuracy is reported. Most of the reported comparisons [19–26] are concerned about comparing the parameters of the resultant Gaussian mixtures by using real images. Under such circumstances, it is difficult to consider a clear reference in order to define a meaningful error. Therefore, the image defined in Fig. 8.11 has been used in the experiments because its construction parameters are clearly defined in Table 8.3.

Since the parameter values from Table 8.3 act as ground truth, a simple averaged difference between them and the values that are computed by each algorithm could be used as comparison error. However, as each parameter maintains different active intervals, it is necessary to express the differences in terms of percentage. Therefore, if $\Delta\beta$ is the parametric difference and β the ground truth parameter, the percentage error $\Delta\beta\%$ can be defined as follows (Fig. 8.14):

$$\Delta\beta\% = \frac{\Delta\beta}{\beta} \cdot 100\% \qquad (8.17)$$

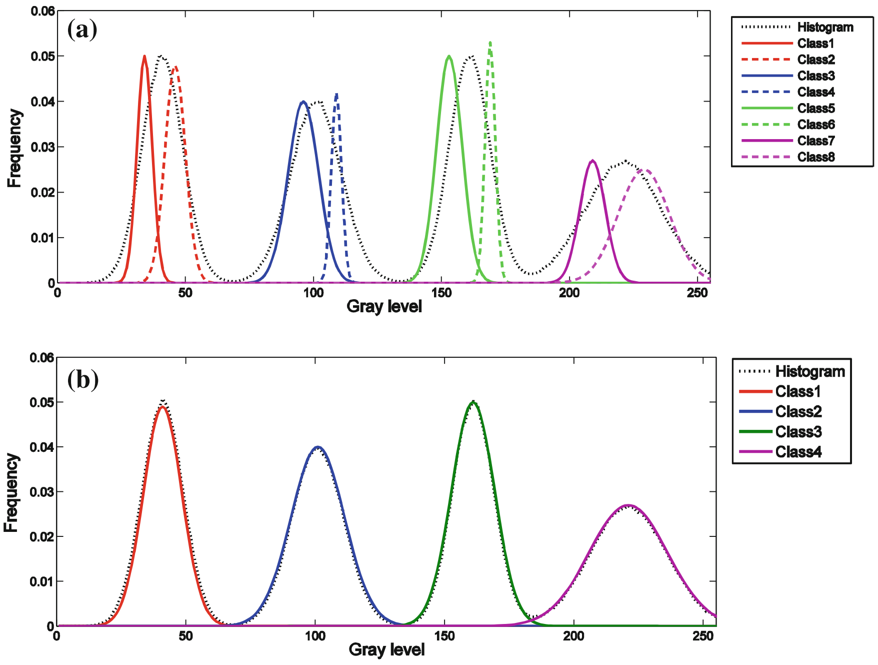


Fig. 8.12 Convergence results. **a** Convergence of the following methods: J + ABC, J + AIS and J + DE considering Gaussian mixtures of 8 classes, **b** convergence of the proposed method (reduced Gaussian mixture)

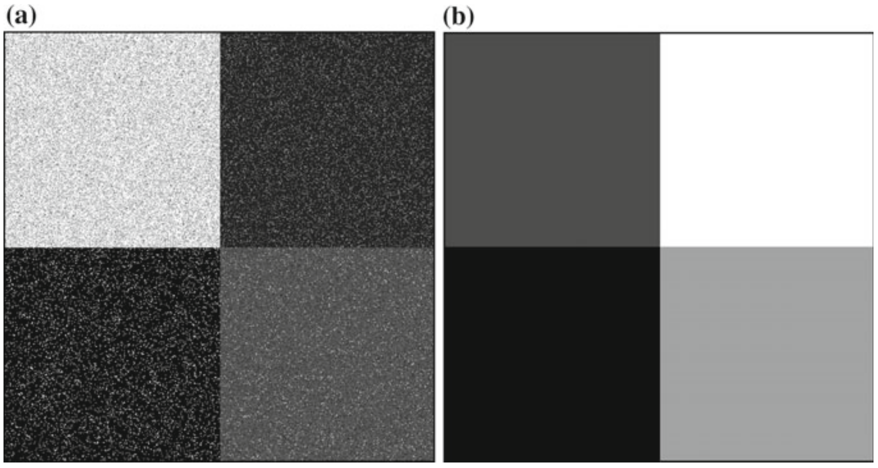


Fig. 8.13 Segmentation results obtained by **a** several methods including J + ABC, J + AIS and J + DE considering Gaussian mixtures of 8 classes; and **b** the proposed method (reduced Gaussian mixture)

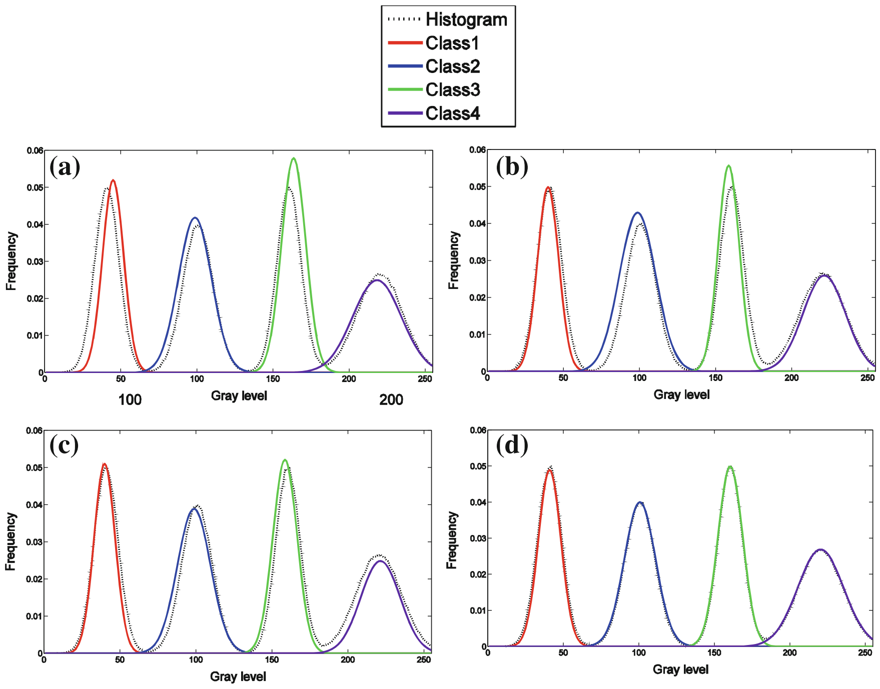


Fig. 8.14 Approximation results in terms of accuracy. **a** J + ABC, **b** J + AIS, **c** J + DE and **d** the proposed J^{new} + LS approach

In the segmentation problem, each Gaussian mixture represents a K-dimensional model where each dimension corresponds to a Gaussian function of the optimization problem to be solved. Since each Gaussian function possesses three parameters P_i, μ_i and σ_i, the complete number of parameters is $3 \cdot K$ dimensions. Therefore, the final error E produced by the final Gaussian mixture is:

$$E = \frac{1}{K \cdot 3} \sum_{v=1}^{K \cdot 3} \Delta\beta_v \%, \tag{8.18}$$

where $\beta_v \in (P_i, \mu_i, \sigma_i)$.

In order to compare accuracy, the algorithms, J + ABC, J + AIS, J + DE and the proposed approach are all executed over the image shown by Fig. 8.12a. The experiment aims to estimate the Gaussian mixture that better approximates the actual image histogram. Methods J + ABC, J + AIS and J + DE consider Gaussian mixtures composed of 4 functions ($K = 4$). In case of the J^{new} + LS method, although the algorithm finds a reduced Gaussian mixture of four functions, it is initially set with ten functions ($K = 10$). Table 8.4 presents the final Gaussian mixture parameters and the final error E. The final Gaussian mixture parameters

Table 8.4 Results of the reduced Gaussian mixture in terms of accuracy

Algorithm	Gaussian function	$\overline{P_i}$	$\overline{\mu_i}$	$\overline{\sigma_i}$	E (%)
J + ABC	(1)	0.052	44.5	6.4	11.79
	(2)	0.084	98.12	12.87	
	(3)	0.058	163.50	8.94	
	(4)	0.025	218.84	17.5	
J + AIS	(1)	0.072	31.01	6.14	22.01
	(2)	0.054	88.52	12.21	
	(3)	0.039	149.21	9.14	
	(4)	0.034	248.41	13.84	
J + DE	(1)	0.041	35.74	7.86	13.57
	(2)	0.036	90.57	11.97	
	(3)	0.059	148.47	9.01	
	(4)	0.020	201.34	13.02	
J^{new} + LS	(1)	0.049	40.12	7.5	3.98
	(2)	0.041	102.04	10.4	
	(3)	0.052	168.66	8.3	
	(4)	0.025	110.92	15.2	

have been averaged over 30 independent executions in order to assure consistency. A close inspection of Table 8.4 reveals that the proposed method is able to achieve the smallest error (E) in comparison to the other algorithms. Figure 8.15 presents the histogram approximations that are produced by each algorithm whereas Fig. 8.16 shows their correspondent segmented images. Both illustrations present the median case obtained throughout 30 runs. Figure 8.17 exhibits that J + ABC, J + AIS, and J + DE present different levels of misclassifications which are nearly absent in the proposed approach case.

Computational cost

The experiment aims to measure the complexity and the computing time spent by the J + ABC, the J + AIS, the J + DE and the J^{new} + LS algorithm while calculating the parameters of the Gaussian mixture in benchmark images (see Figs. 8.16a–d). J + ABC, J + AIS and J + DE consider Gaussian mixtures that are composed of 4 functions ($K = 4$). In case of the J^{new} + LS method, although the algorithm finds a reduced Gaussian mixture of four functions despite being initialized with ten functions ($K = 10$). Table 8.5 shows the averaged measurements after 30 experiments. It is evident that the J + ABC and J + DE are the slowest to converge (iterations) and the J + AIS shows the highest computational cost (time elapsed) because it requires operators which demand long times. On the other hand, the J^{new} + LS shows an acceptable trade-off between its convergence time and its computational cost. Therefore, although the implementation of J^{new} + LS in general requires more code than most of other evolution-based segmentators, such a fact is not reflected

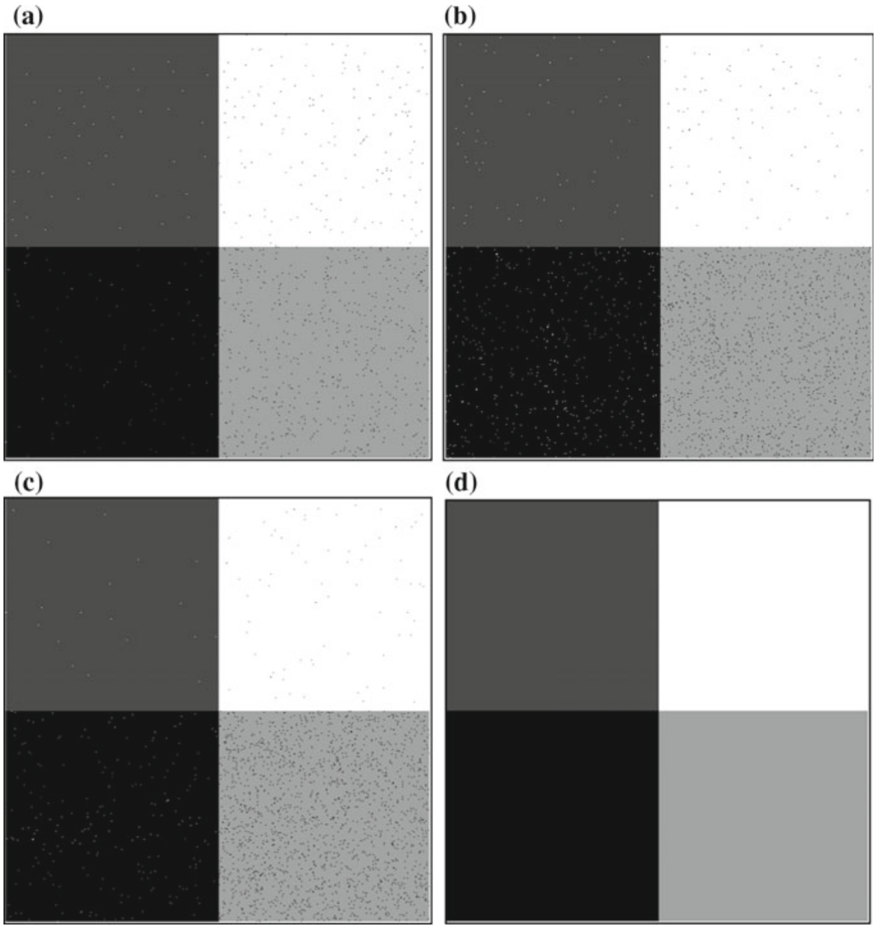


Fig. 8.15 Segmentation results in terms of accuracy. **a** J + ABC, **b** J + AIS, **c** J + DE and **d** the proposed J^{new} + LS approach

in the execution time. Finally, Fig. 8.16 shows the segmented images as they are generated by each algorithm. It can be seen that the proposed approach generate more homogeneous regions whereas J + ABC, J + AIS and J + DE present several artifacts that are produced by an incorrect pixel classification.

8.5.3 Performance Evaluation of the Segmentation Results

This section presents an objective evaluation of segmentation results that are produced by all algorithms in the comparisons. The ill-defined nature of the segmentation

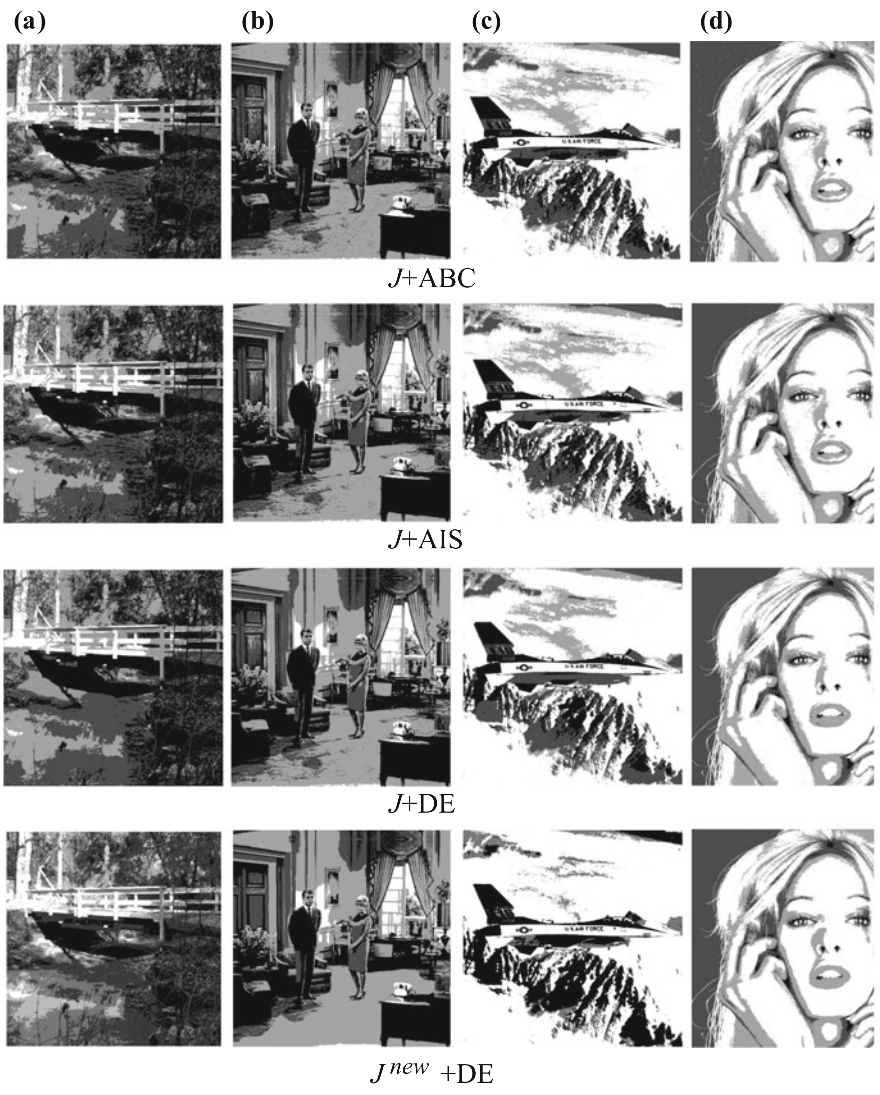


Fig. 8.16 Images employed in the computational cost analysis

problem makes the evaluation of a candidate algorithm difficult [38]. Traditionally, the evaluation has been conducted by using some supervised criteria [39] which are based on the computation of a dissimilarity measure between a segmentation result and a ground truth image. Recently, the use of unsupervised measures has substituted supervised indexes for the objective evaluation of segmentation results [40]. They enable the quantification of the quality of a segmentation result without a priori knowledge (ground truth image).

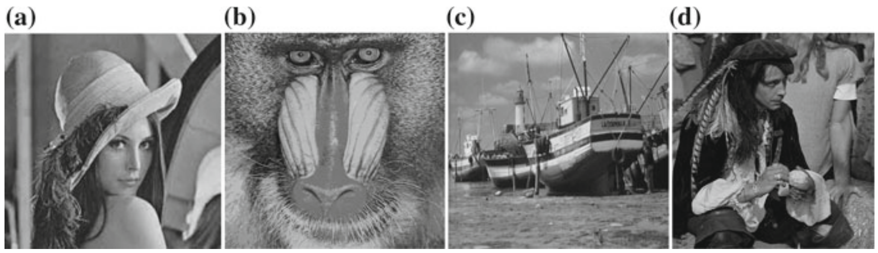


Fig. 8.17 Experimental set used in the evaluation of the segmentation results

Table 8.5 Iterations and time requirements of the J + ABC, the J + AIS, the J + DE and the J^{new} + LS algorithm as they are applied to segment benchmark images (see Fig. 8.16)

Iterations	(a)	(b)	(c)	(d)
Time elapsed				
J + ABC	855	833	870	997
	2.72 s	2.70 s	2.73 s	3.1 s
J + AIS	725	704	754	812
	1.78 s	1.61 s	1.41 s	2.01 s
J + DE	657	627	694	742
	1.25 s	1.12 s	1.45 s	1.88 s
J^{new} + **LS**	314	298	307	402
	0.98 s	0.84 s	0.72 s	1.02 s

Evaluation criteria

In this paper, the unsupervised index *ROS* proposed by Rosenberg [41] has been used to objectively evaluate the performance of each candidate algorithm. This index evaluates the segmentation quality in terms of the homogeneity within segmented regions and the heterogeneity among the different regions. *ROS* can be computed as follows:

$$ROS = \frac{\overline{D} - \underline{D}}{2}, \tag{8.19}$$

where $\underline{D}$ quantifies the homogeneity within segmented regions. Similarly, $\overline{D}$ measures the disparity among the regions. A segmentation result S_1 is considered better than S_2, if $ROS_{S_1} > ROS_{S_2}$. The inter-region homogeneity characterized by $\underline{D}$ is calculated considering the following formulation:

$$\underline{D} = \frac{1}{N_R} \sum_{c=1}^{N_R} \frac{R_c}{I} \cdot \frac{\sigma_c}{\left(\sum_{l=1}^{N_R} \sigma_l\right)}, \tag{8.20}$$

where N_R represents the number of partitions in which the image has been segmented. R_c symbolizes the number of pixels contained in the partition c whereas I considers the number of pixels that integrate the complete image. Similarly, σ_c represents the standard deviation from the partition c. On the other hand, disparity among the regions $\overline{D}$ is computed as follows:

$$\overline{D} = \frac{1}{N_R} \sum_{c=1}^{N_R} \frac{R_c}{I} \cdot \left[\frac{1}{(N_R - 1)} \sum_{l=1}^{N_R} \frac{|\mu_c - \mu_l|}{255} \right], \tag{8.21}$$

where μ_c is the average gray-level in the partition c.

Experimental protocol

In the comparison of segmentation results, a set of four classical images has been chosen to integrate the experimental set (Fig. 8.17). The segmentation methods used in the comparison are J + ABC [19], J + AIS [20] and J + DE [21].

From all segmentation methods used in the comparison, the proposed J^{new} + LS algorithm is the only one that has the capacity to automatically detect the number of segmentation partitions (classes). In order to conduct a fair comparison, all algorithms have been proved by using the same number of partitions. Therefore, in the experiments, the J^{new} + LS segmentation algorithm is firstly applied to detect the best possible number of partitions N_R. Once obtained the number of partitions N_R, the rest of the algorithms were configured to approximate the image histogram with this number of classes.

Figure 8.18 presents the segmentation results obtained by each algorithm considering the experimental set from Fig. 8.17. On the other hand, Table 8.6 shows the evaluation of the segmentation results in terms of the *ROS* index. Such values represent the averaged measurements after 30 executions. From them, it can be seen that the proposed J^{new} + LS method obtain the best *ROS* indexes. Such values indicate that the proposed algorithm maintains the best balance between the homogeneity within segmented regions and the heterogeneity among the different regions. From Fig. 8.18, it can be seen that the proposed approach generate more homogeneous regions whereas J + ABC, J + AIS and J + DE present several artifacts that are produced by an incorrect pixel classification.

Table 8.6 Evaluation of the segmentation results in terms of the ROS index

Number of classes	$N_R = 4$	$N_R = 3$	$N_R = 4$	$N_R = 4$
Image	(a)	(b)	(c)	(d)
J + ABC	0.534	0.328	0.411	0.457
J + AIS	0.522	0.321	0.427	0.437
J + DE	0.512	0.312	0.408	0.418
J^{new} + **LS**	0.674	0.401	0.514	0.527

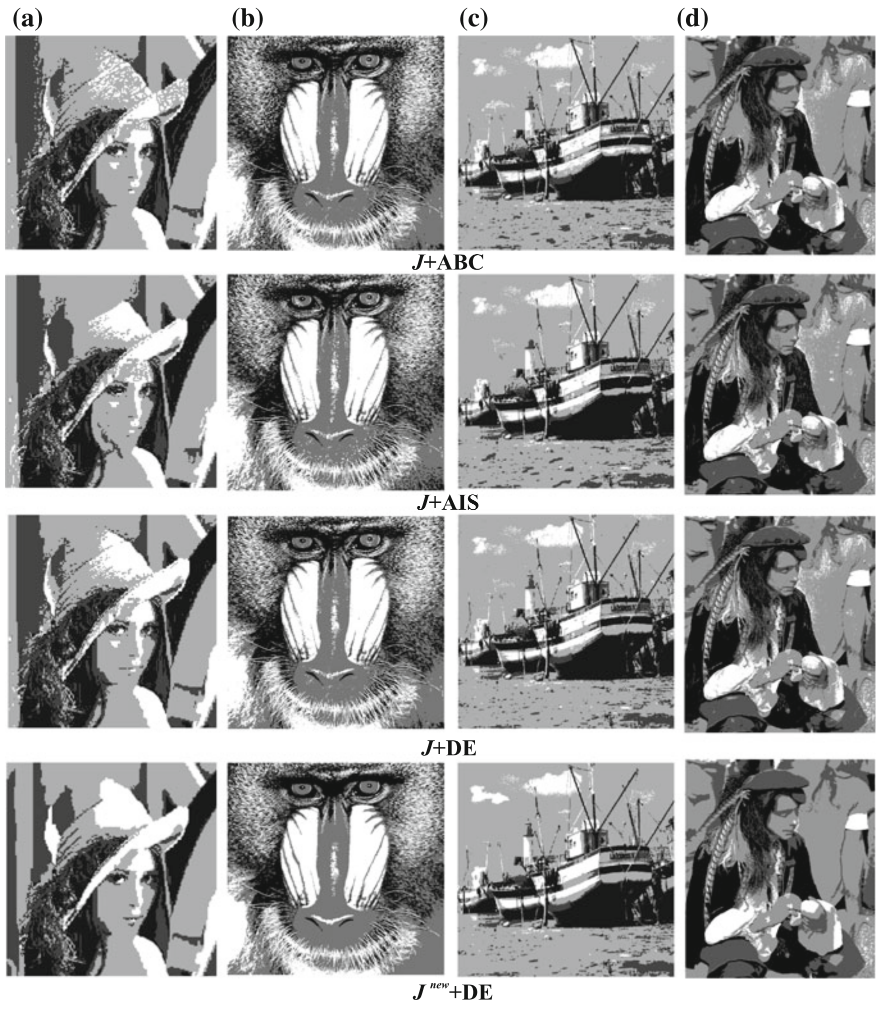


Fig. 8.18 Segmentation results using in the evaluation

8.6 Conclusions

Despite several evolutionary methods have been successfully applied to image segmentation with interesting results, most of them have exhibited two important limitations: (1) they frequently obtain sub-optimal results (misclassifications) as a consequence of an inappropriate balance between exploration and exploitation in their search strategies; (2) the number of classes is fixed and known in advance.

In this paper, a new swarm algorithm for the automatic image segmentation, called the Locust Search (LS) has been presented. The proposed method eliminates the typical flaws presented by previous evolutionary approaches by combining a novel evolutionary method with the definition of a new objective function that appropriately evaluates the segmentation quality with respect to the number of classes. In order to illustrate the proficiency and robustness of the proposed approach, several numerical experiments have been conducted. Such experiments have been divided into two parts. First, the proposed LS method has been compared to other well-known evolutionary techniques on a set of benchmark functions. In the second part, the performance of the proposed segmentation algorithm has been compared to other segmentation methods based on evolutionary principles. The results in both cases validate the efficiency of the proposed technique with regard to accuracy and robustness.

Several research directions will be considered for future work such as the inclusion of other indexes to evaluate similarity between a candidate solution and the image histogram, the consideration of spatial pixel characteristics in the objective function, the modification of the evolutionary LS operators to control the exploration-exploitation balance and the conversion of the segmentation procedure into a multi-objective problem.

References

1. Zhang, H., Fritts, J.E., Goldman, S.A.: Image segmentation evaluation: a survey of unsupervised methods. Comput. Vis. Image Underst. **110**(2), 260–280 (2008)
2. Uemura, T., Koutaki, G., Uchimura, K.: Image segmentation based on edge detection using boundary code. Int. J. Innov. Comput. Inf. Control **7**(10), 6073–6083 (2011)
3. Wang, L., Wu, H., Pan, C.: Region-based image segmentation with local signed difference energy. Pattern Recogn. Lett. **34**(6), 637–645 (2013)
4. Otsu, N.: A thresholding selection method from gray-level histogram. IEEE Trans. Syst. Man Cybern. **9**, 62–66 (1979)
5. Peng, R., Varshney, P.K.: On performance limits of image segmentation algorithms. Comput. Vis. Image Underst. **132**, 24–38 (2015)
6. Balafar, M.A.: Gaussian mixture model based segmentation methods for brain MRI images. Artif. Intell. Rev. **41**, 429–439 (2014)
7. McLachlan, G.J., Rathnayake, S.: On the number of components in a Gaussian mixture model. Data Min. Knowl. Disc. **4**(5), 341–355 (2014)
8. Oliva, D., Osuna-Enciso, V., Cuevas, E., Pajares, G., Pérez-Cisneros, M., Zaldívar, D.: Improving segmentation velocity using an evolutionary method. Expert Syst. Appl. **42**(14), 5874–5886 (2015)
9. Ye, Z.-W., Wang, M.-W., Liu, W., Chen, S.-B.: Fuzzy entropy based optimal thresholding using bat algorithm. Appl. Soft Comput. **31**, 381–395 (2015)
10. Sarkar, S., Das, S., Chaudhuri, S.S.: A multilevel color image thresholding scheme based on minimum cross entropy and differential evolution. Pattern Recogn. Lett. **54**, 27–35 (2015)
11. Bhandari, A.K., Kumar, A., Singh, G.K.: Modified artificial bee colony based computationally efficient multilevel thresholding for satellite image segmentation using Kapur's, Otsu and Tsallis functions. Expert Syst. Appl. **42**(3), 1573–1601 (2015)
12. Permuter, H., Francos, J., Jermyn, I.: A study of Gaussian mixture models of color and texture features for image classification and segmentation. Pattern Recogn. **39**, 695–706 (2006)
13. Dempster, A.P., Laird, A.P., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. Ser. B **39**(1), 1–38 (1977)

14. Zhang, Z., Chen, C., Sun, J., Chan, L.: EM algorithms for Gaussian mixtures with split-and-merge operation. Pattern Recogn. **36**, 1973–1983 (2003)
15. Park, H., Amari, S., Fukumizu, K.: Adaptive natural gradient learning algorithms for various stochastic models. Neural Netw. **13**, 755–764 (2000)
16. Park, H., Ozeki, T.: Singularity and slow convergence of the EM algorithm for Gaussian mixtures. Neural Process. Lett. **29**, 45–59 (2009)
17. Gupta, L., Sortrakul, T.: A Gaussian-mixture-based image segmentation algorithm. Pattern Recogn. **31**(3), 315–325 (1998)
18. Osuna-Enciso, V., Cuevas, E., Sossa, H.: A comparison of nature inspired algorithms for multi-threshold image segmentation. Expert Syst. Appl. **40**(4), 1213–1219 (2013)
19. Cuevas, E., Sención, F., Zaldivar, D., Pérez-Cisneros, M., Sossa, H.: A multi-threshold segmentation approach based on artificial bee colony optimization. Appl. Intell. **37**(3), 321–336 (2012)
20. Cuevas, E., Osuna-Enciso, V., Zaldivar, D., Pérez-Cisneros, M., Sossa, H.: Multithreshold segmentation based on artificial immune systems. Math. Probl. Eng. Article no. 874761 (2012)
21. Cuevas, E., Zaldivar, D., Pérez-Cisneros, M.: A novel multi-threshold segmentation approach based on differential evolution optimization. Expert Syst. Appl. **37**(7), 5265–5271 (2010)
22. Oliva, D., Cuevas, E., Pajares, G., Zaldivar, D., Osuna, V.: A multilevel thresholding algorithm using electromagnetism optimization. Neurocomputing **39**, 357–381 (2014)
23. Oliva, D., Cuevas, E., Pajares, G., Zaldivar, D., Perez-Cisneros, M.: Multilevel thresholding segmentation based on harmony search optimization. J. Appl. Math. Article no. 575414 (2013)
24. Cuevas, E., Zaldivar, D., Pérez-Cisneros, M.: Seeking multi-thresholds for image segmentation with learning automata. Mach. Vis. Appl. **22**(5), 805–818 (2011)
25. Tan, K.C., Chiam, S.C., Mamun, A.A., Goh, C.K.: Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization. Eur. J. Oper. Res. **197**, 701–713 (2009)
26. Chen, G., Low, C.P., Yang, Z.: Preserving and exploiting genetic diversity in evolutionary programming algorithms. IEEE Trans. Evol. Comput. **13**(3), 661–673 (2009)
27. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, Dec 1995
28. Storn, R., Price, K.: Differential evolution—a simple and efficient adaptive scheme for global optimisation over continuous spaces. Technical Report TR-95-012. ICSI, Berkeley, CA (1995)
29. Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B., Tian, Q.: Self-adaptive learning based particle swarm optimization. Inf. Sci. **181**(20), 4515–4538 (2011)
30. Tvrdík, J.: Adaptation in differential evolution: a numerical comparison. Appl. Soft Comput. **9**(3), 1149–1155 (2009)
31. Wang, H., Sun, H., Li, C., Rahnamayan, S., Jeng-shyang, P.: Diversity enhanced particle swarm optimization with neighborhood. Inf. Sci. **223**, 119–135 (2013)
32. Gong, W., Fialho, Á., Cai, Z., Li, H.: Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. Inf. Sci. **181**(24), 5364–5386 (2011)
33. Gordon, D.: The organization of work in social insect colonies. Complexity **8**(1), 43–46 (2003)
34. Kizaki, S., Katori, M.: A stochastic lattice model for locust outbreak. Phys. A **266**, 339–342 (1999)
35. Rogers, S.M., Cullen, D.A., Anstey, M.L., Burrows, M., Dodgson, T., Matheson, T., Ott, S.R., Stettin, K., Sword, G.A., Despland, E., Simpson, S.J.: Rapid behavioural gregarization in the desert locust, Schistocerca gregaria entails synchronous changes in both activity and attraction to conspecifics. J. Insect Physiol. **65**, 9–26 (2014)
36. Topaz, C.M., Bernoff, A.J., Logan, S., Toolson, W.: A model for rolling swarms of locusts. Eur. Phys. J. Special Topics **157**, 93–109 (2008)
37. Topaz, C.M., D'Orsogna, M.R., Edelstein-Keshet, L., Bernoff, A.J.: Locust dynamics: behavioral phase change and swarming. PLoS Comput. Biol. **8**(8), 1–11 (2012)
38. Unnikrishnan, R., Pantofaru, C., Hebert, M.: Toward objective evaluation of image segmentation algorithms. IEEE Trans. Pattern Anal. Mach. Intell. **29**(6), 929–944 (2007)

39. Unnikrishnan, R., Pantofaru, C., Hebert, M.: A measure for objective evaluation of image segmentation algorithms. In: Proceedings of CVPR Workshop Empirical Evaluation Methods in Computer Vision (2005)
40. Zhang, Y.J.: A survey on evaluating methods for image segmentation. Pattern Recogn. **29**(8), 1335–1346 (1996)
41. Chabrier, S., Emile, B., Rosenberger, C., Laurent, H.: Unsupervised performance evaluation of image segmentation. EURASIP J. Appl. Signal Process. **2006**, Article ID 96306, pp. 1–12 (2006)

Chapter 9
Multimodal Swarm Algorithm Based on the Collective Animal Behavior (CAB) for Circle Detection

Abstract In engineering problems due to physical and cost constraints, the best results, obtained by a global optimization algorithm, cannot be realized always. Under such conditions, if multiple solutions (local and global) are known, the implementation can be quickly switched to another solution without much interrupting the design process. This chapter presents a swarm multimodal optimization algorithm named as the Collective Animal Behavior (CAB). Animal groups, such as schools of fish, flocks of birds, swarms of locusts and herds of wildebeest, exhibit a variety of behaviors including swarming about a food source, milling around a central location or migrating over large distances in aligned groups. These collective behaviors are often advantageous to groups, allowing them to increase their harvesting efficiency to follow better migration routes, to improve their aerodynamic and to avoid predation. In the proposed algorithm, searcher agents emulate a group of animals which interact to each other based on simple biological laws that are modeled as evolutionary operators. Numerical experiments are conducted to compare the proposed method with the state-of-the-art methods on benchmark functions. The proposed algorithm has been also applied to the engineering problem of multi circle detection, achieving satisfactory results.

9.1 Introduction

A large number of real-world problems can be considered as multimodal function optimization subjects. An objective function may have several global optima, i.e. several points holding objective function values which are equal to the global optimum. Moreover, it may exhibit some other local optima points whose objective function values lay nearby a global optimum. Since the mathematical formulation of a real-world problem often produces a multimodal optimization issue, finding all global or even these local optima would provide to the decision makers multiple options to choose from [1].

E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_9

Several methods have recently been proposed for solving the multimodal optimization problem. They can be divided into two main categories: deterministic and stochastic (metaheuristic) methods. When facing complex multimodal optimization problems, deterministic methods, such as gradient descent method, the quasi-Newton method and the Nelder-Mead's simplex method, may get easily trapped into the local optimum as a result of deficiently exploiting local information. They strongly depend on a priori information about the objective function, yielding few reliable results.

Metaheuristic algorithms have been developed combined rules and randomness mimicking several phenomena. These phenomena include evolutionary processes (e.g., the evolutionary algorithm proposed by Fogel et al. [2], De Jong [3], and Koza [4] and the genetic algorithms (GAs) proposed by Holland [5] and Goldberg [6]), immunological systems (e.g., the artificial immune systems proposed by de Castro et al. [7]), physical processes (e.g., simulated annealing proposed by Kirkpatrick et al. [8], electromagnetism-like proposed by İlker et al. [9] and the gravitational search algorithm proposed by Rashedi et al. [10]) and the musical process of searching for a perfect state of harmony (proposed by Geem et al. [11], Lee and Geem [12], Geem [13] and Gao et al. [14]).

Traditional GA's perform well for locating a single optimum but fail to provide multiple solutions. Several methods have been introduced into the GA's scheme to achieve multimodal function optimization, such as sequential fitness sharing [15, 16], deterministic crowding [17], probabilistic crowding [18], clustering based niching [19], clearing procedure [20], species conserving genetic algorithm [21], and elitist-population strategies [22]. However, algorithms based on the GA's do not guarantee convergence to global optima because of their poor exploitation capability. GA's exhibit other drawbacks such as the premature convergence which results from the loss of diversity in the population and becomes a common problem when the search continues for several generations. Such drawbacks [23] prevent the GA's from practical interest for several applications.

Using a different metaphor, other researchers have employed Artificial Immune Systems (AIS) to solve the multimodal optimization problems. Some examples are the clonal selection algorithm [24] and the artificial immune network (AiNet) [25, 26]. Both approaches use some operators and structures which attempt to algorithmically mimic the natural immune system's behavior of human beings and animals.

Several studies have been inspired by animal behavior phenomena in order to develop optimization techniques such as the Particle swarm optimization (PSO) algorithm which models the social behavior of bird flocking or fish schooling [27]. In recent years, there have been several attempts to apply the PSO to multi-modal function optimization problems [28, 29]. However, the performance of such approaches presents several flaws when it is compared to the other multi-modal metaheuristic counterparts [26].

Recently, the concept of individual-organization [30, 31] has been widely used to understand collective behavior of animals. The central principle of individual-organization is that simple repeated interactions between individuals can produce complex behavioral patterns at group level [30, 32, 33]. Such inspiration comes from behavioral patterns seen in several animal groups, such as ant pheromone trail networks, aggregation of cockroaches and the migration of fish schools, which can be accurately described in terms of individuals following simple sets of rules [34]. Some examples of these rules [33, 35] include keeping current position (or location) for best individuals, local attraction or repulsion, random movements and competition for the space inside of a determined distance. On the other hand, new studies have also shown the existence of collective memory in animal groups [36–38]. The presence of such memory establishes that the previous history, of group structure, influences the collective behavior exhibited in future stages. Therefore, according to these new developments, it is possible to model complex collective behaviors by using simple individual rules and configuring a general memory.

On the other hand, the problem of detecting circular features holds paramount importance in several engineering applications. The circle detection in digital images has been commonly solved through the Circular Hough Transform (CHT) [39]. Unfortunately, this approach requires a large storage space that augments the computational complexity and yield a low processing speed. In order to overcome this problem, several approaches which modify the original CHT have been proposed. One well-known example is the Randomized Hough Transform (RHT) [40]. As an alternative to Hough Transform-based techniques, the problem of shape recognition has also been handled through optimization methods. In general, they have demonstrated to deliver better results than those based on the HT considering accuracy, speed and robustness [41]. Such approaches have produced several robust circle detectors using different optimization algorithms such as Genetic algorithms (GA) [41], Harmony Search (HSA) [42], Electromagnetism-Like (EMO) [43], Differential Evolution (DE) [44] and Bacterial Foraging Optimization (BFOA) [45]. Since such evolutionary algorithms are global optimizers, they detect only the global optimum (only one circle) of an objective function that is defined over a given search space. However, extracting multiple circle primitives falls into the category of multimodal optimization, where each circle represents an optimum which must be detected within a feasible solution space. The quality for such optima is characterized by the properties of their geometric primitives. Big and well drawn circles normally represent points in the search space with high fitness values (possible global maximum) whereas small and dashed circles describe points with fitness values which account for local maxima. Likewise, circles holding similar geometric properties, such as radius, size, etc., tend to represent locations with similar fitness values. Therefore, a multi-modal method must be applied in order to appropriately solve the problem of multi-shape detection. In this paper, a new multimodal optimization algorithm based on the collective animal behavior is proposed and also applied to multi-circle detection.

This paper proposes a new optimization algorithm inspired by the collective animal behavior. In this algorithm, the searcher agents emulate a group of animals that interact to each other based on simple behavioral rules which are modeled as evolutionary operators. Such operations are applied to each agent considering that the complete group has a memory which stores its own best positions seen so far by applying a competition principle. Numerical experiments have been conducted to compare the proposed method with the state-of-the-art methods on multi-modal benchmark functions. Besides, the proposed algorithm is also applied to the engineering problem of multi-circle detection, achieving satisfactory results.

This paper is organized as follows: Sect. 9.2 introduces the basic biological aspects of the algorithm. In Sect. 9.3, the proposed algorithm and its characteristics are described. A numerical study on different multi-modal benchmark function is presented in Sect. 9.4. Section 9.5 presents the application of the proposed algorithm to multi-circle detection whereas Sect. 9.6 shows the obtained results. Finally, in Sect. 9.7 the conclusions are discussed.

9.2 Biological Fundaments

The remarkable collective behavior of organisms such as swarming ants, schooling fish and flocking birds has long captivated the attention of naturalists and scientists. Despite a long history of scientific investigation, just recently we are beginning to decipher the relationship between individuals and group-level properties [46]. Grouping individuals often have to make rapid decisions about where to move or what behavior to perform, in uncertain and dangerous environments. However, each individual typically has only relatively local sensing ability [47]. Groups are, therefore, often composed of individuals that differ with respect to their informational status and individuals are usually not aware of the informational state of others [48], such as whether they are knowledgeable about a pertinent resource, or of a threat.

Animal groups are based on a hierarchic structure [49] which differentiates individuals according to a fitness principle known as Dominance [50]. Such concept represents the domain of some individuals within a group and occurs when competition for resources leads to confrontation. Several studies [51, 52] have found that such animal behavior lead to stable groups with better cohesion properties among individuals.

Recent studies have illustrated how repeated interactions among grouping animals scale to collective behavior. They have also remarkably revealed, that collective decision-making mechanisms across a wide range of animal group types, ranging from insects to birds (and even among humans in certain circumstances) seem to share similar functional characteristics [30, 34, 53]. Furthermore, at a certain level of description, collective decision-making in organisms shares essential common features such as a general memory. Although some differences may arise, there are good reasons to increase communication between researchers working in collective animal behavior and those involved in cognitive science [33].

Despite the variety of behaviors and motions of animal groups, it is possible that many of the different collective behavioral patterns are generated by simple rules followed by individual group members. Some authors have developed different models, such as the self-propelled particle (SPP) model which attempts to capture the collective behavior of animal groups in terms of interactions between group members following a diffusion process [54–57].

On other hand, following a biological approach, Couzin et al. [33, 34] have proposed a model in which individual animals follow simple rules of thumb: (1) keep the position of best individuals; (2) move from or to nearby neighbors (local attraction or repulsion); (3) move randomly and (4) compete for the space inside of a determined distance. Each individual thus admits three different movements: attraction, repulsion or random, while holds two kinds of states: preserve the position or compete for a determined position. In the model, the movement experimented by each individual is decided randomly (according to an internal motivation), meanwhile the states are assumed according to a fixed criterion.

The dynamical spatial structure of an animal group can be explained in terms of its history [54]. Despite this, the majority of the studies have failed in considering the existence of memory in behavioral models. However, recent researches [36, 58] have also shown the existence of collective memory in animal groups. The presence of such memory establishes that the previous history of the group structure, influences the collective behavior exhibited in future stages. Such memory can contain the position of special group members (the dominant individuals) or the averaged movements produced by the group.

According to these new developments, it is possible to model complex collective behaviors by using simple individual rules and setting a general memory. In this work, the behavioral model of animal groups is employed for defining the evolutionary operators through the proposed metaheuristic algorithm. A memory is incorporated to store best animal positions (best solutions) considering a competition-dominance mechanism.

9.3 Collective Animal Behavior Algorithm (CAB)

The CAB algorithm assumes the existence of a set of operations that resembles the interaction rules that model the collective animal behavior. In the approach, each solution within the search space represents an animal position. The "fitness value" refers to the animal dominance with respect to the group. The complete process mimics the collective animal behavior.

The approach in this paper implements a memory for storing best solutions (animal positions) mimicking the aforementioned biologic process. Such memory is divided into two different elements, one for maintaining the best found positions in each generation $\left(\mathbf{M}_g\right)$ and the other for storing best history positions during the complete evolutionary process $(\mathbf{M}_h)$.

9.3.1 Description of the CAB Algorithm

Likewise other metaheuristic approaches, the CAB algorithm is also an iterative process. It starts by initializing the population randomly, i.e. generating random solutions or animal positions. The following four operations are thus applied until the termination criterion is met, i.e. the iteration number *NI* is reached as follows:

1. Keep the position of the best individuals.
2. Move from or nearby neighbors (local attraction and repulsion).
3. Move randomly.
4. Compete for the space inside of a determined distance (updating the memory).

9.3.1.1 Initializing the Population

The algorithm begins by initializing a set **A** of N_p animal positions $\left(\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{N_p}\}\right)$. Each animal position $\mathbf{a}_i$ is a *D*-dimensional vector containing the parameter values to be optimized, which are randomly and uniformly distributed between the pre-specified lower initial parameter bound a_j^{low} and the upper initial parameter bound a_j^{high}.

$$\begin{aligned} a_{j,i} &= a_j^{low} + \text{rand}(0,1) \cdot (a_j^{high} - a_j^{low}); \\ j &= 1, 2, \ldots, D; \quad i = 1, 2, \ldots, N_p. \end{aligned} \tag{9.1}$$

with j and i being the parameter and individual indexes respectively. Hence, $a_{j,i}$ is the jth parameter of the ith individual.

All the initial positions **A** are sorted according to the fitness function (dominance) to form a new individual set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\}$, so that we can choose the best *B* positions and store them in the memory $\mathbf{M}_g$ and $\mathbf{M}_h$. The fact that both memories share the same information is only allowed at this initial stage.

9.3.1.2 Keep the Position of the Best Individuals

Analogously to the biological metaphor, this behavioral rule, typical in animal groups, is implemented as an evolutionary operation in our approach. In this operation, the first *B* elements of the new animal position set **A** ($\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_B\}$) are generated. Such positions are computed by the values contained in the historic memory $\mathbf{M}_h$ considering a slight random perturbation around them. This operation can be modelled as follows:

$$\mathbf{a}_l = \mathbf{m}_h^l + \mathbf{v} \tag{9.2}$$

where $l \in \{1, 2, \ldots, B\}$ while $\mathbf{m}_h^l$ represents the *l*-element of the historic memory $\mathbf{M}_h$ and $\mathbf{v}$ is a random vector holding an appropriate small length.

9.3.1.3 Move from or to Nearby Neighbours

From the biological inspiration, where animals experiment a random local attraction or repulsion according to an internal motivation, we implement the evolutionary operators that mimic them. For this operation, a uniform random number r_m is generated within the range [0,1]. If r_m is less than a threshold H, a determined individual position is moved (attracted or repelled) considering the nearest best historical value of the group (the nearest position contained in $\mathbf{M}_h$) otherwise it is considered the nearest best value in the group of the current generation (the nearest position contained in $\mathbf{M}_g$). Therefore such operation can be modeled as follows:

$$\mathbf{a}_i = \begin{cases} \mathbf{x}_i \pm r \cdot (\mathbf{m}_h^{nearest} - \mathbf{x}_i) \text{ with probability} H \\ \mathbf{x}_i \pm r \cdot (\mathbf{m}_g^{nearest} - \mathbf{x}_i) \text{ with probability} (1 - H) \end{cases} \tag{9.3}$$

where $i \in \{B + 1, B + 2, \ldots, N_p\}$, $\mathbf{m}_h^{nearest}$ and $\mathbf{m}_g^{nearest}$ represent the nearest elements of $\mathbf{M}_h$ and $\mathbf{M}_g$ to $\mathbf{x}_i$, while r is a random number between [−1,1]. Therefore, if $r > 0$, the individual position $\mathbf{x}_i$ is attracted to the position $\mathbf{m}_h^{nearest}$ or $\mathbf{m}_g^{nearest}$, otherwise such movement is considered as a repulsion.

9.3.1.4 Move Randomly

Following the biological model, under some probability P an animal randomly changes its position. Such behavioral rule is implemented considering the next expression:

$$\mathbf{a}_i = \begin{cases} \mathbf{r} \ \text{ with probability } P \\ \mathbf{x}_i \ \text{ with probability } (1 - P) \end{cases} \tag{9.4}$$

being $i \in \{B + 1, B + 2, \ldots, N_p\}$ and $\mathbf{r}$ a random vector defined within the search space. This operator is similar to re-initialize the particle in a random position as it is done by Eq. (9.1).

9.3.1.5 Compete for the Space Inside of a Determined Distance (Updating the Memory)

Once the operations to preserve the position of the best individuals, to move from or to nearby neighbors and to move randomly, have all been applied to the all N_p animal positions, generating N_p new positions, it is necessary to update the memory $\mathbf{M}_h$.

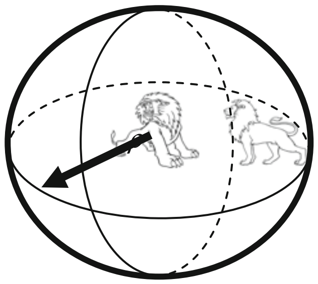


Fig. 9.1 Dominance concept, presented when two animals confront each other inside of a ρ distance

In order to update the memory $\mathbf{M}_h$, the concept of dominance is used. Animals that interact in a group keep a minimum distance among them. Such distance ρ depends on how aggressive the animal behaves [50, 58]. Hence, when two animals confront each other inside of such distance, the most dominant individual prevails as the other withdraws. Figure 9.1 shows this process.

In the proposed algorithm, the historic memory $\mathbf{M}_h$ is updated considering the following procedure:

1. The elements of $\mathbf{M}_h$ and $\mathbf{M}_g$ are merged into $\mathbf{M}_U$ ($\mathbf{M}_U = \mathbf{M}_h \cup \mathbf{M}_g$).
2. Each element $\mathbf{m}_U^i$ of the memory $\mathbf{M}_U$, it is compared pair-wise with the remainder memory elements $\left(\left\{\mathbf{m}_U^1, \mathbf{m}_U^2, \ldots, \mathbf{m}_U^{2B-1}\right\}\right)$. If the distance between both elements is less than ρ, the element holding a better performance in the fitness function will prevail meanwhile the other will be removed.
3. From the resulting elements of $\mathbf{M}_U$ (as they are obtained in step 2), the B best value is selected to integrate the new $\mathbf{M}_h$.

Unsuitable values of ρ result in a lower convergence rate, longer computation time, larger function evaluation number, convergence to a local maximum or unreliability of solutions. The ρ value is computed considering the following equation:

$$\rho = \frac{\prod_{j=1}^{D} (a_j^{high} - a_j^{low})}{10 \cdot D} \tag{9.5}$$

where a_j^{low} and a_j^{high} represent the pre-specified lower bound and the upper bound of the j-parameter respectively, within an D-dimensional space.

9.3.1.6 Computational Procedure

The computational procedure for the proposed algorithm can be summarized as follows:

Step 1:	Set the parameters N_p, B, H, P and NI
Step 2:	Generate randomly the position set $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{N_p}\}$ using Eq. (9.1)
Step 3:	Sort **A,** according to the objective function (dominance), building $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\}$
Step 4:	Choose the first B positions of **X** and store them into the memory $\mathbf{M}_g$
Step 5:	Update $\mathbf{M}_h$ according to Sect. 3.1.5 (for the first iteration $\mathbf{M}_h = \mathbf{M}_g$)
Step 6:	Generate the first B positions of the new solution set $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_B\}$. Such positions correspond to elements of $\mathbf{M}_h$ making a slight random perturbation around them. $\mathbf{a}_l = \mathbf{m}_h^l + \mathbf{v}$; being $\mathbf{v}$ a random vector holding an appropriate small length.
Step 7:	Generate the rest of the **A** elements using the attraction, repulsion and random movements

```
for $i = B + 1 : N_p$
    if ( $r_1 < 1 - P$ ) then
    attraction and repulsion movement
        { if ( $r_2 < H$ ) then
            $\mathbf{a}_i = \mathbf{x}_i \pm r \cdot (\mathbf{m}_h^{nearest} - \mathbf{x}_i)$
          else if
            $\mathbf{a}_i = \mathbf{x}_i \pm r \cdot (\mathbf{m}_g^{nearest} - \mathbf{x}_i)$
        }
    else if
    random movement
        {
            $\mathbf{a}_i = \mathbf{r}$
        }
end for
```

where $r_1, r_2, r \in \text{rand}(0,1)$.

Step 8:	If NI is completed, the process is thus completed; otherwise go back to step 3

9.3.1.7 Optima Determination

Just after the optimization process has finished, an analysis of the final $\mathbf{M}_h$ memory is executed in order to find the global and significant local minima. For it, a threshold value T_h is defined to decide which elements will be considered as a significant local minimum. Such threshold is thus computed as:

$$T_h = \frac{\max\limits_{fitness} (\mathbf{M_h})}{6} \tag{9.6}$$

where $\max_{fitness}(\mathbf{M_h})$ represents the best fitness value among $\mathbf{M}_h$ elements. Therefore, memory elements whose fitness values are greater than T_h will be considered as global and local optima as other elements are discarded.

9.3.1.8 Capacities of CAB and Differences with PSO

Evolutionary algorithms (EA) have been widely employed for solving complex optimization problems. These methods are found to be more powerful than conventional methods based on formal logics or mathematical programming [59]. Exploitation and exploration are two main features of the EA [60]. The exploitation phase searches around the current best solutions and selects the best candidates or solutions. The exploration phase ensures that the algorithm seeks the search space more efficiently in order to analyze potential unexplored areas.

The EA do not have limitations in using different sources of inspiration (e.g., music-inspired [11] or physic-inspired charged system search [9]). However, nature is a principal inspiration for proposing new metaheuristic approaches and the nature-inspired algorithms have been widely used in developing systems and solving problems [61]. Biologically-inspired algorithms are one of the main categories of the nature-inspired metaheuristic algorithms. The efficiency of the bio-inspired algorithms is due to their significant ability to imitate the best features in nature. More specifically, these algorithms are based on the selection of the most suitable elements in biological systems which have evolved by natural selection.

Particle swarm optimization (PSO) is undoubtedly one of the most employed EA methods that use biologically-inspired concepts in the optimization procedure. Unfortunately, like others stochastic algorithms, PSO also suffers from the premature convergence [62], particularly in multi-modal problems. Premature convergence, in PSO, is produced by the strong influence of the best particle in the evolution process. Such particle is used by the PSO movement equations as a main individual in order to attract other particles. Under such conditions, the exploitation phase is privileged by allowing the evaluation of new search position around the best individual. However, the exploration process is seriously damaged, avoiding searching in unexplored areas.

As an alternative to PSO, the proposed scheme modifies some evolution operators for allowing not only attracting but also repelling movements among particles. Likewise, instead of considering the best position as reference, our algorithm uses a set of neighboring elements that are contained in an incorporated memory. Such improvements, allow increasing the algorithm's capacity to explore and to exploit the set of solutions which are operated during the evolving process.

In the proposed approach, in order to improve the balance between exploitation and exploration, we have introduced three new concepts. The first one is the "attracting and repelling movement", which outlines that one particle cannot be only attracted, but also repelled. The application of this concept to the evolution operators (Eq. 9.3) increases the capacity of the proposed algorithm to satisfactorily explore the search space. Since the process of attraction or repulsion of each particle is randomly determined, the possibility of prematurely convergence is very low, even for cases that hold an exaggerated number of local minima (excessive number of multimodal functions).

The second concept is the use of the main individual. In the approach, the main individual that is considered as pivot in the equations (in order to generate attracting and repulsive movements), is not the best (as in PSO), but one element ($\mathbf{m}_h^{nearest}$ or $\mathbf{m}_g^{nearest}$) of a set which is contained in memories that store the best individual seen so-far. Such pivot is the nearest element in memory with regard to the individual whose position is necessary to evolve. Under such conditions, the points considered to prompt the movement of a new individual are multiple. Such fact allows to maintain a balance between exploring new positions and exploiting the best positions seen so-far.

Finally, the third concept is the use of an incorporated memory which stores the best individuals seen so-far. As it has been discussed in Sect. 3.1.5, each candidate individual to be stored in the memory must compete with elements already contained in the memory in order to demonstrate that such new point is relevant. For the competition, the distance between each individual and the elements in the memory is used to decide pair-wise which individuals are actually considered. Then, the individual with better fitness value prevails whereas its pair is discarded. The incorporation of such concept allows simultaneously registering and refining the best-individual set seen-so-far. This fact guarantees a high precision for final solutions of the multi-modal landscape through an extensive exploitation of the solution set.

9.3.1.9 Numerical Example

In order to demonstrate the algorithm's step-by-step operation, a numerical example has been set by applying the proposed method to optimize a simple function which is defined as follows:

$$f(x_1, x_2) = e^{-((x_1-4)^2-(x_2-4)^2)} + e^{-((x_1+4)^2-(x_2-4)^2)} + 2 \cdot e^{-((x_1)^2+(x_2)^2)} + 2 \cdot e^{-((x_1)^2-(x_2+4)^2)} \tag{9.7}$$

Considering the interval of $-5 \leq x_1, x_2 \leq 5$, the function possesses two global maxima of value 2 at $(x_1, x_2) = (0, 0)$ and $(0, -4)$. Likewise, it holds two local minima of value 1 at $(-4, 4)$ and $(4, 4)$. Figure 9.2a shows the 3D plot of this function. The parameters for the CAB algorithm are set as: $N_p = 10$, $B = 4$, $H = 0.8$, $P = 0.1$, $\rho = 3$ and $NI = 30$.

Like all evolutionary approaches, CAB is a population-based optimizer that attacks the starting point problem by sampling the objective function at multiple, randomly chosen, initial points. Therefore, after setting parameter bounds that define the problem domain, 10 (N_p) individuals ($\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_{10}$) are generated using Eq. (9.1). Following an evaluation of each individual through the objective function (Eq. 9.7), all are sorted decreasingly in order to build vector $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{10})$. Figure 9.2b

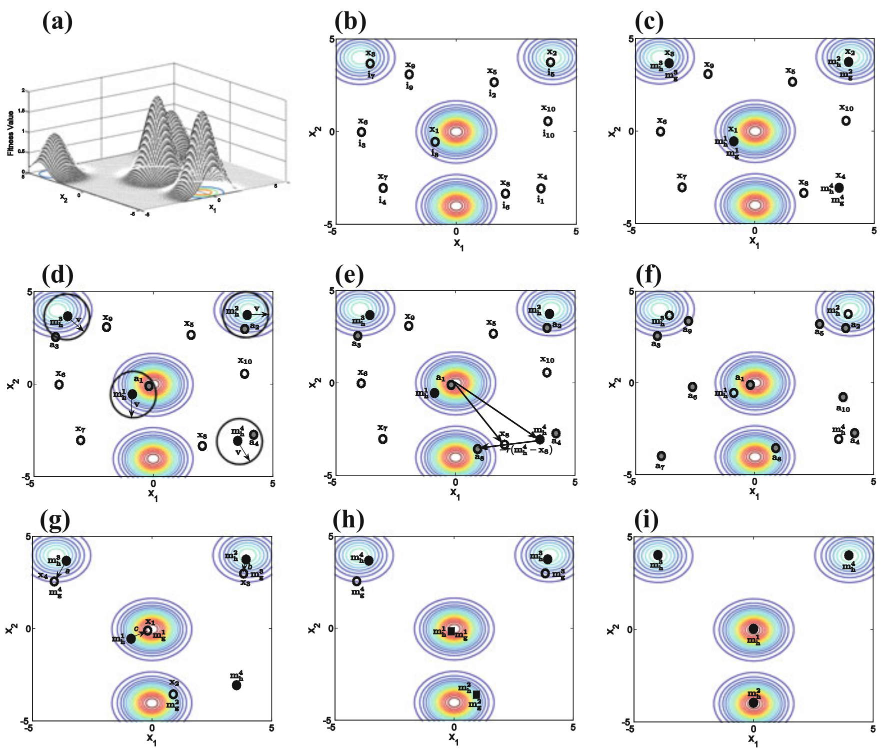


Fig. 9.2 CAB numerical example: **a** 3D plot of the function used as example. **b** Initial individual distribution. **c** Initial configuration of memories $\mathbf{M}_g$ and $\mathbf{M}_h$. **d** The computation of the first four individuals ($\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4$). **e** It shows the procedure employed by step 2 in order to calculate the new individual position $\mathbf{a}_8$. **f** Positions of all new individuals ($\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{10}$). **g** Application of the dominance concept over elements of $\mathbf{M}_g$ and $\mathbf{M}_h$. **h** Final memory configurations of $\mathbf{M}_g$ and $\mathbf{M}_h$ after the first iteration. **i** Final memory configuration of $\mathbf{M}_h$ after 30 iterations

depicts the initial individual distribution in the search space. Then, both memories $\mathbf{M}_g$ ($\mathbf{m}_g^1, \ldots, \mathbf{m}_g^4$) and $\mathbf{M}_h$ ($\mathbf{m}_h^1, \ldots, \mathbf{m}_h^4$) are filled with the first four (B) elements present in $\mathbf{X}$. Such memory elements are represented by solid points in Fig. 9.2c.

The new 10 individuals ($\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{10}$) are evolved at each iteration following three different steps: (1) Keep the position of best individuals. (2) Move from or nearby neighbors and (3) Move randomly. The first new four elements ($\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4$) are generated considering the first step (Keeping the position of best individuals). Following such step, new individual positions are calculated as perturbed versions of all the elements which are contained in the $\mathbf{M}_h$ memory (that represent the best individuals known so far). Such perturbation is done by using $\mathbf{a}_l = \mathbf{m}_h^l + \mathbf{v}$ ($l \in 1, \ldots, 4$). Figure 9.2d shows a comparative view between the memory element positions and the perturbed values of ($\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4$).

The remaining 6 new positions ($\mathbf{a}_5, \ldots, \mathbf{a}_{10}$) are individually computed according to step 2 and 3. For such operation, a uniform random number r_1 is generated within the range [0, 1]. If r_1 is less than $1 - P$, the new position $\mathbf{a}_j$ ($j \in 5, \ldots, 10$) is generated through step 2; otherwise, $\mathbf{a}_j$ is obtained from a random re-initialization (step 3) between search bounds.

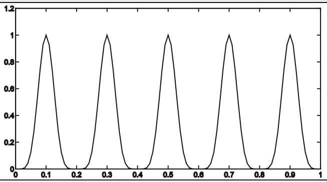
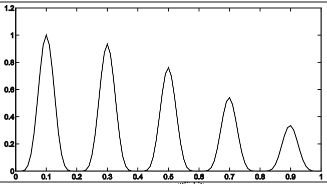
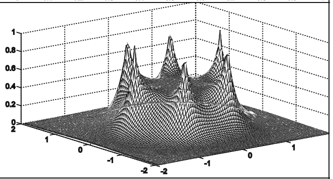
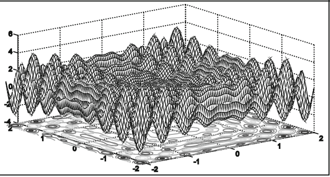
In order to calculate a new position $\mathbf{a}_j$ at step 2, a decision must be made on whether it should be generated by using the elements of $\mathbf{M}_h$ or $\mathbf{M}_g$. For such decision, a uniform random number r_2 is generated within the range [0, 1]. If r_2 is less than H, the new position $\mathbf{a}_j$ is generated by using $\mathbf{x}_j \pm r \cdot (\mathbf{m}_h^{nearest} - \mathbf{x}_j)$; otherwise, $\mathbf{a}_j$ is obtained by considering $\mathbf{x}_j \pm r \cdot (\mathbf{m}_g^{nearest} - \mathbf{x}_j)$. Where $\mathbf{m}_h^{nearest}$ and $\mathbf{m}_g^{nearest}$ represent the closest elements to $\mathbf{x}_j$ in memory $\mathbf{M}_h$ and $\mathbf{M}_g$ respectively. In the first iteration, since there is not available information from previous steps, both memories $\mathbf{M}_h$ and $\mathbf{M}_g$ share the same information which is only allowed at this initial stage. Figure 9.2e shows graphically the whole procedure employed by step 2 in order to calculate the new individual position $\mathbf{a}_8$ whereas Fig. 9.2 f presents the positions of all new individuals ($\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{10}$).

Finally, after all new positions ($\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{10}$) have been calculated, memories $\mathbf{M}_h$ and $\mathbf{M}_g$ must be updated. In order to update $\mathbf{M}_h$, new calculated positions ($\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{10}$) are arranged according to their fitness values by building vector $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{10})$. Then, the elements of $\mathbf{M}_h$ are replaced by the first four elements in $\mathbf{X}$ (the best individuals of its generation). In order to calculate the new elements of $\mathbf{M}_h$, current elements of $\mathbf{M}_h$ (the present values) and $\mathbf{M}_g$ (the updated values) are merged into $\mathbf{M}_U$. Then, by using the dominance concept (explained in Sect. 3.1.5) over $\mathbf{M}_U$, the best four values are selected to replace the elements in $\mathbf{M}_g$. Figure 9.2g and 2 h show the updating procedure for both memories. Applying the dominance (see Fig. 9.2g), since the distances $a = dist(\mathbf{m}_h^3, \mathbf{m}_g^4)$, $b = dist(\mathbf{m}_h^2, \mathbf{m}_g^3)$ and $c = dist(\mathbf{m}_h^1, \mathbf{m}_g^1)$ are less than $\rho = 3$, elements with better fitness evaluation will build the new memory $\mathbf{M}_h$. Figure 9.2h depicts final memory configurations. The circles and solid circles points represent the elements of $\mathbf{M}_g$ and $\mathbf{M}_h$ respectively whereas the bold squares perform as elements shared by both memories. Therefore, if the complete procedure is repeated over 30 iterations, the memory $\mathbf{M}_h$ will contain the 4 global and local maxima as elements. Figure 9.2i depicts the final configuration after 30 iterations.

9.4 Results on Multi-modal Benchmark Functions

In this section, the performance of the proposed algorithm is tested. Section 9.4.1 describes the experiment methodology. Sections 9.4.2, and 9.4.3 report on a comparison between the CAB experimental results and other multimodal metaheuristic algorithms for different kinds of optimization problems.

Table 9.1 The test suite of multimodal functions for Experiment 4.2

Function	Search space	Sketch
$f_1 = \sin^6(5\pi x)$ Deb's function 5 optima	$x \in [0,1]$	
$f_2(x) = 2^{-2((x-0.1)/0.9)^2} \cdot \sin(5\pi x)$ Deb's decreasing function 5 optima	$x \in [0,1]$	
$f_3(z) = \frac{1}{1+\lvert z^6+1\rvert}$ Roots function 6 optima	$z \in C,\ z = x_1 + ix$ $x_1, x_2 \in [-2,2]$	
$f_4(x_1,x_2) = x_1\sin(4\pi x_1) - x_2\sin(4\pi x_2 + \pi) + 1$ Two dimensional multi-modal function 100 optima	$x_1, x_2 \in [-2,2]$	

9.4.1 Experiment Methodology

In this section, we will examine the search performance of the proposed CAB by using a test suite of 8 benchmark functions with different complexities. They are listed in Tables 9.1 and 9.2. The suite mainly contains some representative, complicated and multimodal functions with several local optima. These functions are normally regarded as difficult to be optimized as they are particularly challenging to the applicability and efficiency of multimodal metaheuristic algorithms. The performance measurements considered at each experiment are the following:

- The consistency of locating all known optima; and
- The averaged number of objective function evaluations that are required to find such optima (or the running time under the same condition).

The experiments compare the performance of CAB against the Deterministic Crowding [17], the Probabilistic Crowding [18], the Sequential Fitness Sharing [15], the Clearing Procedure [20], the Clustering Based Niching (CBN) [19], the Species Conserving Genetic Algorithm (SCGA) [21], the Elitist-population strategy (AEGA)

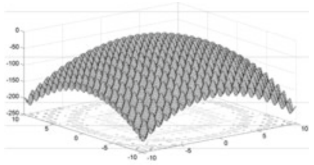
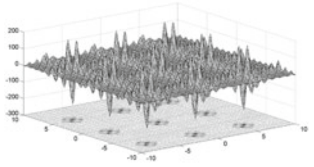
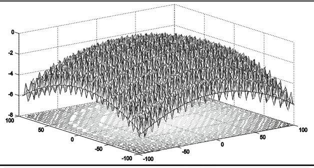
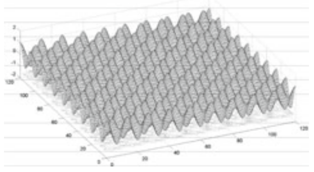
[22], the Clonal Selection algorithm [24] and the artificial immune network (AiNet) [25].

Since the approach solves real-valued multimodal functions, we have used, in the GA-approaches, consistent real coding variable representation, uniform crossover and mutation operators for each algorithm seeking a fair comparison. The crossover probability Pc = 0.8 and the mutation probability Pm = 0.1 have been used. We use the standard tournament selection operator with a tournament size = 2 in our implementation of Sequential Fitness Sharing, Clearing Procedure, CBN, Clonal Selection algorithm, and SCGA. On the other hand, the parameter values for the aiNet algorithm have been defined as suggested in [25], with the mutation strength $\beta = 100$, the suppression threshold $\sigma_{s(aiNet)} = 0.2$ and the update rate $d = 40\%$.

In the case of the CAB algorithm, the parameters are set to $N_p = 200$, $B = 100$, $P = 0.8$ and $H = 0.6$. Once they have been all experimentally determined, they are kept for all the test functions through all experiments.

To avoid relating the optimization results to the choice of a particular initial population and to conduct fair comparisons, we perform each test 50 times, starting from various randomly selected points in the search domain as it is commonly given in the

Table 9.2 The test suite of multimodal functions used in the Experiment 4.3

Function	Search space	Sketch
$f_5(x_1,x_2) = -(20 + x_1^2 + x_2^2 - 10(\cos(2\pi x_1) + \cos(2\pi x_2))$ Rastringin's function 100 optima	$x_1, x_2 \in [-10,10]$	
$f_6(x_1,x_2) = -\prod_{i=1}^{2} \sum_{j=1}^{5} \cos((j+1)x_i + j)$ Shubert function 18 optima	$x_1, x_2 \in [-10,10]$	
$f_7(x_1,x_2) = \frac{1}{4000}\sum_{i=1}^{2} x_i^2 - \prod_{i=1}^{2}\cos\left(\frac{x_i}{\sqrt{2}}\right) + 1$ Griewank function 100 optima	$x_1, x_2 \in [-100,10]$	
$f_8(x_1,x_2) = \frac{\cos(0.5x_1) + \cos(0.5x_2)}{4000} + \cos(10x_1)\cos(10x$ Modified Griewank function 100 optima	$x_1, x_2 \in [0,120]$	

literature. An optimum o_j is considered as found if $\exists x_i \in Pop(k = T)|d(x_i, o_j) < 0.005$, where $Pop(k = T)$ is the complete population at the end of the run T and x_i is an individual in $Pop(k = T)$.

All algorithms have been tested in MatLAB© over the same Dell Optiplex GX260 computer with a Pentium-4 2.66G-HZ processor, running Windows XP operating system over 1 Gb of memory. Next sections present experimental results for multimodal optimization problems which have been divided into two groups with different purposes. The first one consists of functions with smooth landscapes and well defined optima (local and global values), while the second gathers functions holding rough landscapes and complex location optima.

9.4.2 Comparing CAB Performance for Smooth Landscapes Functions

This section presents a performance comparison for different algorithms solving multimodal problems $f_1 - f_4$ in Table 9.1. The aim is to determine whether CAB is more efficient and effective than other existing algorithms for finding all multiple optima of $f_1 - f_4$. The stopping criterion analyzes if the number identified optima cannot be further increased over 10 successive generations after the first 100 generations, then the execution will be stopped. Four measurements have been employed to evaluate the performance:

- The average of optima found within the final population (NO);
- The average distance between multiple optima detected by the algorithm and their closest individuals in the final population (DO);
- The average of function evaluations (FE); and
- The average of execution time in seconds (ET).

Table 9.3 provides a summarized performance comparison among several algorithms. Best results have been bold-faced. From the NO measure, CAB always finds better or equally optimal solutions for the multimodal problems $f_1 - f_4$. It is evident that each algorithm can find all optima of f_1. For function f_2, only AEGA, Clonal Selection algorithm, aiNet, and CAB can eventually find all optima each time. For function f_3, Clearing Procedure, SCGA, AEGA and CAB can get all optima at each run. For function f_4, Deterministic Crowding leads to premature convergence and all other algorithms cannot get any better results but CAB yet can find all multiple optima 48 times in 50 runs and its average successful rate for each run is higher than 99%. By analyzing the DO measure in Table 9.3, CAB has obtained the best score for all the multimodal problems except for f_3. In the case of f_3, the solution precision of CAB is only worse than that of Clearing Procedure. On the other hand, CAB has smaller standard deviations in the NO and DO measures than all other algorithms and hence its solution is more stable.

From the FE measure in Table 9.3, it is clear that CAB needs fewer function evaluations than other algorithms considering the same termination criterion. Recall

Table 9.3 Performance comparison among the multimodal optimization algorithms for the test functions $f_1 - f_4$

Function	Algorithm	NO	DO	FE	ET
f_1	Deterministic crowding	**5 (0)**	$1.52 \times 10^{-4}(1.38 \times 10^{-4})$	7153 (358)	0.091 (0.013)
	Probabilistic crowding	**5 (0)**	$3.63 \times 10^{-4}(6.45 \times 10^{-5})$	10,304 (487)	0.163 (0.011)
	Sequential fitness sharing	**5 (0)**	$4.76 \times 10^{-4}(6.82 \times 10^{-5})$	9927 (691)	0.166 (0.028)
	Clearing procedure	**5 (0)**	$1.27 \times 10^{-4}(2.13 \times 10^{-5})$	5860 (623)	0.128 (0.021)
	CBN	**5 (0)**	$2.94 \times 10^{-4}(4.21 \times 10^{-5})$	10,781 (527)	0.237 (0.019)
	SCGA	**5 (0)**	$1.16 \times 10^{-4}(3.11 \times 10^{-5})$	6792 (352)	0.131 (0.009)
	AEGA	**5 (0)**	$4.6 \times 10^{-5}(1.35 \times 10^{-5})$	2591 (278)	0.039 (0.007)
	Clonal selection algorithm	**5 (0)**	$1.99 \times 10^{-4}(8.25 \times 10^{-5})$	15,803 (381)	0.359 (0.015)
	AiNet	**5 (0)**	$1.28 \times 10^{-4}(3.88 \times 10^{-5})$	12,369 (429)	0.421 (0.021)
	CAB	**5 (0)**	$\mathbf{1.69 \times 10^{-5}(5.2 \times 10^{-6})}$	**1776 (125)**	**0.020 (0.009)**
f_2	Deterministic crowding	3.53 (0.73)	$3.61 \times 10^{-3}(6.88 \times 10^{-4})$	6026 (832)	0.271 (0.06)
	Probabilistic crowding	4.73 (0.64)	$2.82 \times 10^{-3}(8.52 \times 10^{-4})$	10,940 (9517)	0.392 (0.07)
	Sequential fitness sharing	4.77 (0.57)	$2.33 \times 10^{-3}(4.36 \times 10^{-4})$	12,796 (1430)	0.473 (0.11)
	Clearing procedure	4.73 (0.58)	$4.21 \times 10^{-3}(1.24 \times 10^{-3})$	8465 (773)	0.326 (0.05)
	CBN	4.70 (0.53)	$2.19 \times 10^{-3}(4.53 \times 10^{-4})$	14,120 (2187)	0.581 (0.14)
	SCGA	4.83 (0.38)	$3.15 \times 10^{-3}(4.71 \times 10^{-4})$	10,548 (1382)	0.374 (0.09)
	AEGA	**5 (0)**	$1.38 \times 10^{-4}(2.32 \times 10^{-5})$	3605 (426)	0.102 (0.04)
	Clonal selection algorithm	**5 (0)**	$1.37 \times 10^{-3}(6.87 \times 10^{-4})$	21,922 (746)	0.728 (0.06)
	AiNet	**5 (0)**	$1.22 \times 10^{-3}(5.12 \times 10^{-4})$	18,251 (829)	0.664 (0.08)
	CAB	**5 (0)**	$\mathbf{4.5 \times 10^{-5}(8.56 \times 10^{-6})}$	**2065 (92)**	**0.08 (0.007)**

(continued)

Table 9.3 (continued)

Function	Algorithm	NO	DO	FE	ET
f_3	Deterministic crowding	4.23 (1.17)	$7.79 \times 10^{-4}(4.76 \times 10^{-4})$	11,009 (1137)	1.07 (0.13)
	Probabilistic crowding	4.97 (0.64)	$2.35 \times 10^{-3}(7.14 \times 10^{-4})$	16,391 (1204)	1.72 (0.12)
	Sequential fitness sharing	4.87 (0.57)	$2.56 \times 10^{-3}(2.58 \times 10^{-3})$	14,424 (2045)	1.84 (0.26)
	Clearing procedure	**6 (0)**	$\mathbf{7.43 \times 10^{-5}(4.07 \times 10^{-5})}$	12,684 (1729)	1.59 (0.19)
	CBN	4.73 (1.14)	$1.85 \times 10^{-3}(5.42 \times 10^{-4})$	18,755 (2404)	2.03 (0.31)
	SCGA	**6 (0)**	$3.27 \times 10^{-4}(7.46 \times 10^{-5})$	13,814 (1382)	1.75 (0.21)
	AEGA	**6 (0)**	$1.21 \times 10^{-4}(8.63 \times 10^{-5})$	6218 (935)	0.53 (0.07)
	Clonal selection algorithm	5.50 (0.51)	$4.95 \times 10^{-3}(1.39 \times 10^{-3})$	25,953 (2918)	2.55 (0.33)
	AiNet	4.8 (0.33)	$3.89 \times 10^{-3}(4.11 \times 10^{-4})$	20,335 (1022)	2.15 (0.10)
	CAB	**6 (0)**	$9.87 \times 10^{-5}(1.69 \times 10^{-5})$	**4359 (75)**	**0.11 (0.023)**
f_4	Deterministic crowding	76.3 (11.4)	$4.52 \times 10^{-3}(4.17 \times 10^{-3})$	1,861,707 (329,254)	21.63 (2.01)
	Probabilistic crowding	92.8 (3.46)	$3.46 \times 10^{-3}(9.75 \times 10^{-4})$	2,638,581 (597,658)	31.24 (5.32)
	Sequential fitness sharing	89.9 (5.19)	$2.75 \times 10^{-3}(6.89 \times 10^{-4})$	2,498,257 (374,804)	28.47 (3.51)
	Clearing procedure	89.5 (5.61)	$3.83 \times 10^{-3}(9.22 \times 10^{-4})$	2,257,964 (742,569)	25.31 (6.24)
	CBN	90.8 (6.50)	$4.26 \times 10^{-3}(1.14 \times 10^{-3})$	2,978,385 (872,050)	35.27 (8.41)
	SCGA	91.4 (3.04)	$3.73 \times 10^{-3}(2.29 \times 10^{-3})$	2,845,789 (432,117)	32.15 (4.85)
	AEGA	95.8 (1.64)	$1.44 \times 10^{-4}(2.82 \times 10^{-5})$	1,202,318 (784,114)	12.17 (2.29)
	Clonal selection algorithm	92.1 (4.63)	$4.08 \times 10^{-3}(8.25 \times 10^{-3})$	3,752,136 (191,849)	45.95 (1.56)
	AiNet	93.2 (7.12)	$3.74 \times 10^{-3}(5.41 \times 10^{-4})$	2,745,967 (328,176)	38.18 (3.77)
	CAB	**100 (2)**	$\mathbf{2.31 \times 10^{-5}(5.87 \times 10^{-6})}$	**697,578 (57,089)**	**5.78 (1.26)**

The standard unit in the column ET is seconds. For all the parameters, numbers in parentheses are the standard deviations. Bold-cased letters represents best obtained results that all algorithms use the same conventional crossover and mutation operators. It can be easily deduced from results that the CAB algorithm is able to produce better search positions (better compromise between exploration and exploitation), in a more efficient and effective way than other multimodal search strategies.

To validate that CAB improvement over other algorithms as a result of CAB producing better search positions over iterations, Fig. 9.3 shows the comparison of CAB and other multimodal algorithms for f_4. The initial populations for all algorithms have 200 individuals. In the final population of CAB, the 100 individuals belonging to the $\mathbf{M}_h$ memory correspond to the 100 multiple optima, while, on the contrary, the final population of the other nine algorithms fail consistently in finding all optima, despite they have superimposed several times over some previously found optima.

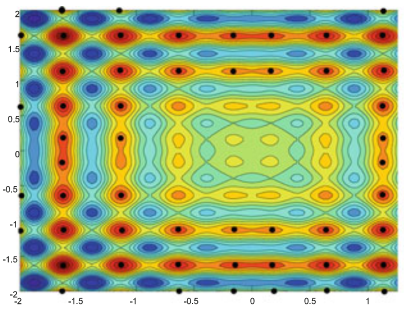
When comparing the execution time (ET) in Table 9.3, CAB uses significantly less time to finish than other algorithms. The situation can be registered by the reduction of the redundancy in the $\mathbf{M}_h$ memory due to competition (dominance) criterion. All these comparisons show that CAB generally outperforms all other multimodal algorithms regarding efficacy and efficiency.

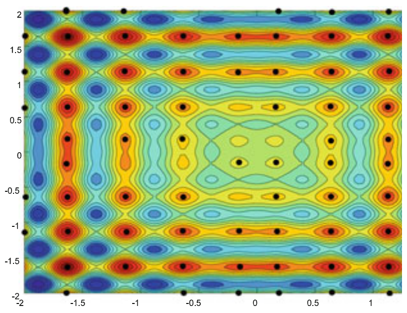
9.4.3 Comparing CAB Performance in Rough Landscapes Functions

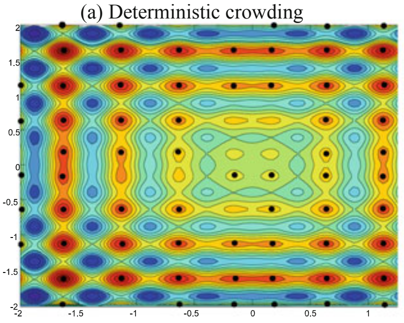
This section presents the performance comparison among different algorithms solving multimodal optimization problems which are listed in Table 9.2. Such problems hold lots of local optima and very rugged landscapes. The goal of multimodal optimizers is to find as many as possible global optima and possibly good local optima. Rastrigin's function f_5 and Griewank's function f_7 have 1 and 18 global optima respectively, becoming practical as to test to whether a multimodal algorithm can find a global optimum and at least 80 higher fitness local optima to validate the algorithms' performance.

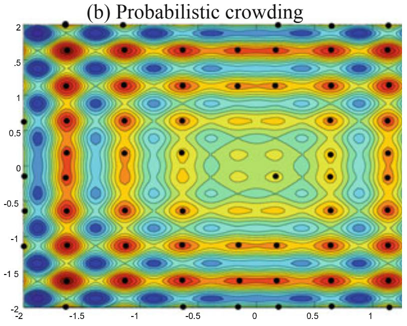
Our main objective in these experiments is to determine whether CAB is more efficient and effective than other existing algorithms for finding the multiple high fitness optima of functions $f_5 - f_8$. In the experiments, the initial population size for all algorithms has been set to 1000. For Sequential Fitness Sharing, Clearing Procedure, CBN, Clonal Selection, SCGA, and AEGA, we have set the distance threshold σ_s to 5. The algorithms' stopping criterion checks whenever the number of optima found cannot be further increased in 50 successive generations after the first 500 generations. If such condition prevails then the algorithm is halted. We still evaluate the performance of all algorithms using the aforementioned four measures NO, DO, FE, and ET.

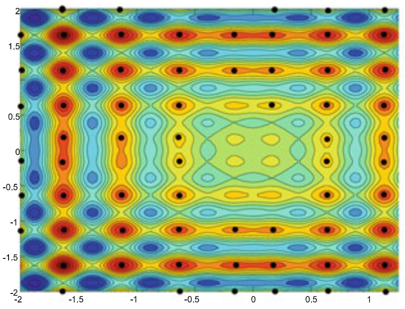
Table 9.4 provides a summary of the performance comparison among different algorithms. From the NO measure, we observe that CAB could always find more optimal solutions for the multimodal problems $f_5 - f_8$. For Rastrigin's function f_5, only CAB can find all multiple high fitness optima 49 times out of 50 runs

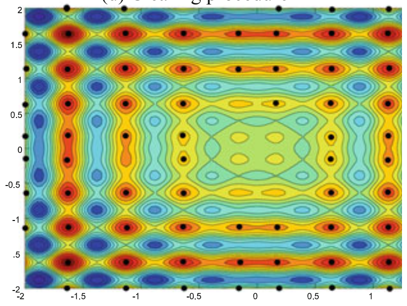

(a) Deterministic crowding

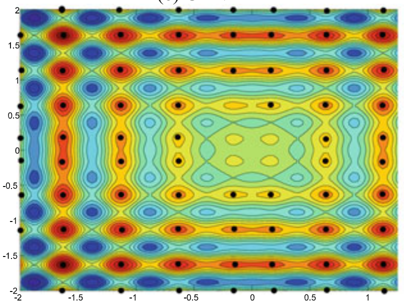

(b) Probabilistic crowding


(c) Sequential fitness sharing


(d) Clearing procedure


(e) CBN


(f) SCGA


(g) AEGA

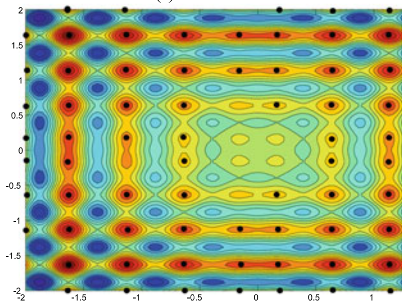
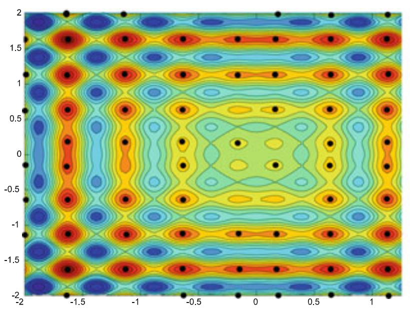

(h) Clonal selction algorithm

Fig. 9.3 Typical results of the maximization of f_4. **a–j** Local and global optima located by all ten algorithms in the performance comparison


(i) AiNet

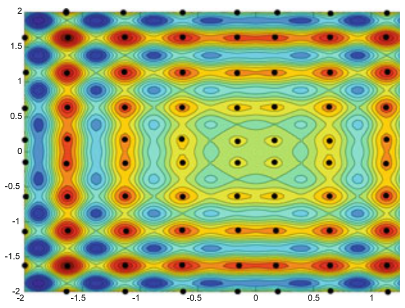

(j) CAB

Fig. 9.3 (continued)

and its average successful rate for each run is higher than 97%. On the contrary, other algorithms cannot find all multiple higher fitness optima for any run. For f_6, 5 algorithms (Clearing Procedure, SCGA, AEGA, clonal selection algorithm, AiNet and CAB) can get all multiple higher fitness maxima for each run respectively. For Griewank's function (f_7), only CAB can get all multiple higher fitness optima for each run. In case of the modified Griewank's function (f_8), it has numerous optima whose value is always the same. However, CAB still can find all global optima with a effectiveness rate of 95%.

From the FE and ET measures in Table 9.4, we can clearly observe that CAB uses significantly fewer function evaluations and a shorter running time than all other algorithms under the same termination criterion. Moreover, Deterministic Crowding leads to premature convergence as CAB is at least 2.5, 3.8, 4, 3.1, 4.1, 3.7, 1.4, 7.9 and 4.9 times faster than all others respectively according to Table 9.4 for functions $f_5 - f_8$.

9.5 Application of CAB in Multi-circle Detection

9.5.1 *Individual Representation*

In order to detect circle shapes, candidate images must be preprocessed first by the well-known Canny algorithm which yields a single-pixel edge-only image. Then, the (x_i, y_i) coordinates for each edge pixel p_i are stored inside the edge vector $P = \left\{p_1, p_2, \ldots, p_{N_p}\right\}$, with N_p being the total number of edge pixels. Each circle C uses three edge points as individuals in the optimization algorithm. In order to construct such individuals, three indexes p_i, p_j and p_k, are selected from vector P, considering the circle's contour that connects them. Therefore, the circle $C = \{p_i, p_j, p_k\}$ that crosses over such points may be considered as a potential solution for the detection

Table 9.4 Performance comparison among multimodal optimization algorithms for the test functions $f_5 - f_8$

Function	Algorithm	NO	DO	FE	ET
f_5	Deterministic crowding	62.4 (14.3)	$4.72 \times 10^{-3}(4.59 \times 10^{-3})$	1,760,199 (254,341)	14.62 (2.83)
	Probabilistic crowding	84.7 (5.48)	$1.50 \times 10^{-3}(9.38 \times 10^{-4})$	2,631,627 (443,522)	34.39 (5.20)
	Sequential fitness sharing	76.3 (7.08)	$3.51 \times 10^{-3}(1.66 \times 10^{-3})$	2,726,394 (562,723)	36.55 (7.13)
	Clearing procedure	93.6 (2.31)	$2.78 \times 10^{-3}(1.20 \times 10^{-3})$	2,107,962 (462,622)	28.61 (6.47)
	CBN	87.9 (7.78)	$4.33 \times 10^{-3}(2.82 \times 10^{-3})$	2,835,119 (638,195)	37.05 (8.23)
	SCGA	97.4 (4.80)	$1.34 \times 10^{-3}(8.72 \times 10^{-4})$	2,518,301 (643,129)	30.27 (7.04)
	AEGA	99.4 (1.39)	$6.77 \times 10^{-4}(3.18 \times 10^{-4})$	978,435 (71,135)	10.56 (4.81)
	Clonal selection algorithm	90.6 (9.95)	$3.15 \times 10^{-3}(1.47 \times 10^{-3})$	5,075,208 (194,376)	58.02 (2.19)
	AiNet	93.8 (7.8)	$2.11 \times 10^{-3}(3.2 \times 10^{-3})$	3,342,864 (549,452)	51.65 (6.91)
	CAB	**100 (2)**	$2.22 \times \mathbf{10^{-4}}(3.1 \times \mathbf{10^{-5}})$	**680,211 (12,547)**	**7.33 (1.84)**
f_6	Deterministic crowding	9.37 (1.91)	$3.26 \times 10^{-3}(5.34 \times 10^{-4})$	832,546 (75,413)	4.58 (0.57)
	Probabilistic crowding	15.17 (2.43)	$2.87 \times 10^{-3}(5.98 \times 10^{-4})$	1,823,774 (265,387)	12.92 (2.01)
	Sequential fitness sharing	15.29 (2.14)	$1.42 \times 10^{-3}(5.29 \times 10^{-4})$	1,767,562 (528,317)	14.12 (3.51)
	Clearing procedure	**18 (0)**	$1.19 \times 10^{-3}(6.05 \times 10^{-4})$	1,875,729 (265,173)	11.20 (2.69)
	CBN	14.84 (2.70)	$4.39 \times 10^{-3}(2.86 \times 10^{-3})$	2,049,225 (465,098)	18.26 (4.41)
	SCGA	4.83 (0.38)	$1.58 \times 10^{-3}(4.12 \times 10^{-4})$	2,261,469 (315,727)	13.71 (1.84)
	AEGA	**18 (0)**	$3.34 \times 10^{-4}(1.27 \times 10^{-4})$	656,639 (84,213)	3.12 (1.12)
	Clonal selection algorithm	**18 (0)**	$3.42 \times 10^{-3}(1.58 \times 10^{-3})$	4,989,856 (618,759)	33.85 (5.36)
	AiNet	**18 (0)**	$2.11 \times 10^{-3}(3.31 \times 10^{-3})$	3,012,435 (332,561)	26.32 (2.54)
	CAB	**18 (0)**	$\mathbf{1.02 \times 10^{-4}(4.27 \times 10^{-5})}$	**431,412 (21,034)**	**2.21 (0.51)**

(continued)

Table 9.4 (continued)

Function	Algorithm	NO	DO	FE	ET
f_7	Deterministic crowding	52.6 (8.86)	$3.71 \times 10^{-3}(1.54 \times 10^{-3})$	2,386,960 (221,982)	19.10 (2.26)
	Probabilistic crowding	79.2 (4.94)	$3.48 \times 10^{-3}(3.79 \times 10^{-3})$	3,861,904 (457,862)	43.53 (4.38)
	Sequential fitness sharing	63.0 (5.49)	$4.76 \times 10^{-3}(3.55 \times 10^{-3})$	3,619,057 (565,392)	42.98 (6.35)
	Clearing procedure	79.4 (4.31)	$2.95 \times 10^{-3}(1.64 \times 10^{-3})$	3,746,325 (594,758)	45.42 (7.64)
	CBN	71.3 (9.26)	$3.29 \times 10^{-3}(4.11 \times 10^{-3})$	4,155,209 (465,613)	48.23 (5.42)
	SCGA	94.9 (8.18)	$2.63 \times 10^{-3}(1.81 \times 10^{-3})$	3,629,461 (373,382)	47.84 (0.21)
	AEGA	98 (2)	$1.31 \times 10^{-3}(8.76 \times 10^{-4})$	1,723,342 (121,043)	12,54 (1.31)
	Clonal selection algorithm	89.2 (5.44)	$3.02 \times 10^{-3}(1.63 \times 10^{-3})$	5,423,739 (231,004)	47.84 (6.09)
	AiNet	92.7 (3.21)	$2.79 \times 10^{-3}(3.19 \times 10^{-4})$	4,329,783 (167,932)	41.64 (2.65)
	CAB	**100 (1)**	$\mathbf{3.32 \times 10^{-4}(5.25 \times 10^{-5})}$	**953,832 (9,345)**	**8.82 (1.51)**
f_8	Deterministic crowding	44.2 (7.93)	$4.45 \times 10^{-3}(3.63 \times 10^{-3})$	2,843,452 (353,529)	23.14 (3.85)
	Probabilistic crowding	70.1 (8.36)	$2.52 \times 10^{-3}(1.47 \times 10^{-3})$	4,325,469 (574,368)	49.51 (6.72)
	Sequential fitness sharing	58.2 (9.48)	$4.14 \times 10^{-3}(3.31 \times 10^{-3})$	4,416,150 (642,415)	54.43 (12.6)
	Clearing procedure	67.5 (10.11)	$2.31 \times 10^{-3}(1.43 \times 10^{-3})$	4,172,462 (413,537)	52.39 (7.21)
	CBN	53.1 (7.58)	$4.36 \times 10^{-3}(3.53 \times 10^{-3})$	4,711,925 (584,396)	61.07 (8.14)
	SCGA	87.3 (9.61)	$3.15 \times 10^{-3}(2.07 \times 10^{-3})$	3,964,491 (432,117)	53.87 (8.46)
	AEGA	90.6 (1.65)	$2.55 \times 10^{-3}(9.55 \times 10^{-4})$	2,213,754 (412,538)	16.21 (3.19)
	Clonal selection algorithm	74.4 (7.32)	$3.52 \times 10^{-3}(2.19 \times 10^{-3})$	5,835,452 (498,033)	74.26 (5.47)
	AiNet	83.2 (6.23)	$3.11 \times 10^{-3}(2.41 \times 10^{-4})$	4,123,342 (213,864)	60.38 (5.21)
	CAB	**97 (2)**	$\mathbf{1.54 \times 10^{-3}(4.51 \times 10^{-4})}$	**1,121,523 (51,732)**	**12.21 (2.66)**

The standard unit of the column ET is seconds (numbers in parentheses are standard deviations). Bold-case letters represent best results

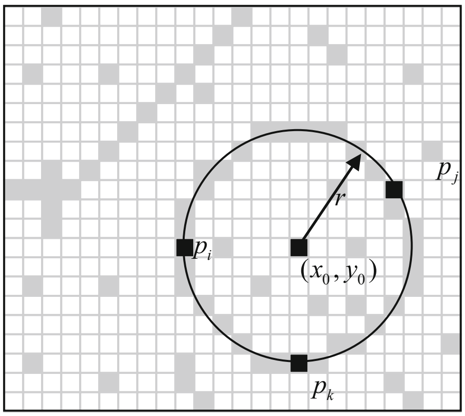


Fig. 9.4 Circle candidate (individual) built from the combination of points p_i, p_j and p_k

problem. Considering the configuration of the edge points shown by Fig. 9.4, the circle center (x_0, y_0) and the radius r of C can be computed as follows:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{9.8}$$

considering

$$\mathbf{A} = \begin{bmatrix} x_j^2 + y_j^2 - (x_i^2 + y_i^2) & 2 \cdot (y_j - y_i) \\ x_k^2 + y_k^2 - (x_i^2 + y_i^2) & 2 \cdot (y_k - y_i) \end{bmatrix}$$
$$\mathbf{B} = \begin{bmatrix} 2 \cdot (x_j - x_i) & x_j^2 + y_j^2 - (x_i^2 + y_i^2) \\ 2 \cdot (x_k - x_i) & x_k^2 + y_k^2 - (x_i^2 + y_i^2) \end{bmatrix}, \tag{9.9}$$

$$x_0 = \frac{\det(\mathbf{A})}{4((x_j - x_i)(y_k - y_i) - (x_k - x_i)(y_j - y_i))},$$
$$y_0 = \frac{\det(\mathbf{B})}{4((x_j - x_i)(y_k - y_i) - (x_k - x_i)(y_j - y_i))}, \tag{9.10}$$

and

$$r = \sqrt{(x_0 - x_d)^2 + (y_0 - y_d)^2}, \tag{9.11}$$

being det(.) the determinant and $d \in \{i, j, k\}$. Figure 9.2 illustrates the parameters defined by Eqs. (9.8–9.11).

9.5.2 Objective Function

In order to calculate the error produced by a candidate solution C, a set of test points is calculated as a virtual shape which, in turn, must be validated, i.e. if it really exists in the edge image. The test set is represented by $S = \{s_1, s_2, \ldots, s_{N_s}\}$, where N_s is the number of points over which the existence of an edge point, corresponding to C, should be validated. In our approach, the set S is generated by the Midpoint Circle Algorithm (MCA) [63]. The MCA is a searching method which seeks the required points for drawing a circle digitally. Therefore MCA calculates the necessary number of test points N_s to totally draw the complete circle. Such a method is considered the fastest because MCA avoids computing square-root calculations by comparing the pixel separation distances among them.

The objective function $J(C)$ represents the matching error produced between the pixels S of the circle candidate C (animal position) and the pixels that actually exist in the edge image, yielding:

$$J(C) = 1 - \frac{\sum_{v=1}^{Ns} E(x_v, y_v)}{Ns} \tag{9.12}$$

where $E(x_i, y_i)$ is a function that verifies the pixel existence in (x_v, y_v), with $(x_v, y_v) \in S$ and N_s being the number of pixels lying on the perimeter corresponding to C currently under testing. Hence, function $E(x_v, y_v)$ is defined as:

$$E(x_v, y_v) = \begin{cases} 1 \text{ if the pixel } (x_v, y_v) \text{ is an edge point} \\ 0 \text{ otherwise} \end{cases} \tag{9.13}$$

A value near to zero of $J(C)$ implies a better response from the "circularity" operator. Figure 9.5 shows the procedure to evaluate a candidate solution C with its representation as a virtual shape S. In Fig. 9.5b, the virtual shape is compared to the original image, point by point, in order to find coincidences between virtual and edge points. The virtual shape is built from points p_i, p_j and p_k shown by Fig. 9.5a. The virtual shape S gathers 56 points ($Ns = 56$) with only 18 of such points existing in both images (shown as blue points plus red points in Fig. 9.5c) yielding: $\sum_{v=1}^{Ns} E(x_v, y_v) = 18$ and therefore $J(C) \approx 0.67$.

9.5.3 The Multiple Circle Detection Procedure

In order to detect multiple circles, most detectors simply apply a one-minimum optimization algorithm, which is able to detect only one circle at a time, repeating the same process several times as previously detected primitives are removed from the image. Such algorithms iterate until there are no more candidates left in the image.

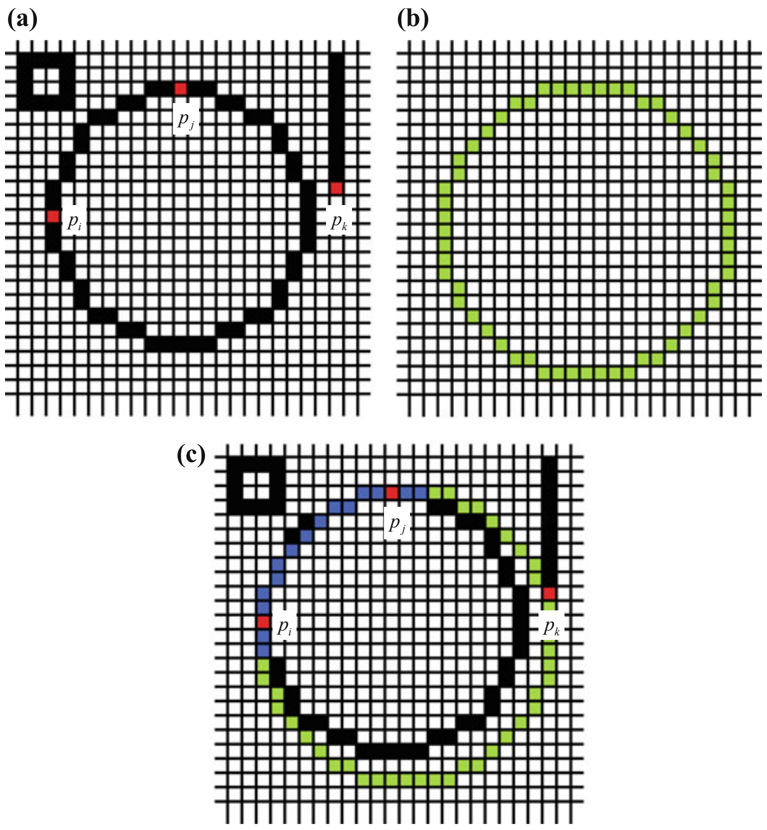


Fig. 9.5 Evaluation of candidate solutions C: the image in (**a**) shows the original image while **b** presents the virtual shape generated including points p_i, p_j and p_k. The image in (**c**) shows coincidences between both images marked by blue or red pixels while the virtual shape is also depicted in green

On the other hand, the method at this paper is able to detect single or multiples circles through only one optimization step. The multi-detection procedure can be summarized as follows: guided by the values of a matching function, the whole group of encoded candidate circles is evolved through the set of evolutionary operators. The best circle candidate (global optimum) is considered to be the first detected circle over the edge-only image. An analysis of the historical memory $\mathbf{M}_h$ is thus executed in order to identify other local optima (other circles).

In order to find other possible circles contained in the image, the historical memory $\mathbf{M}_h$ is carefully examined. The approach aims to explore all elements, one at a time, assessing which of them represents an actual circle in the image. Since several elements can represent the same circle (i.e. circles slightly shifted or holding small deviations), a distinctiveness factor $D_{A,B}$ is required to measure the mismatch between two given circles (A and B). Such distinctiveness factor is defined as follows:

$$D_{A,B} = |x_A - x_B| + |y_A - y_B| + |r_A - r_B| \tag{9.14}$$

being (x_A, y_A) and r_A, the central coordinates and radius of the circle C_A respectively, while (x_B, y_B) and r_B represent the corresponding parameters of the circle C_B. One threshold value $E_{s_{TH}}$ is also calculated to decide whether two circles must be considered different or not. *Th* is computed as:

$$Th = \frac{r_{\max} - r_{\min}}{d} \tag{9.15}$$

where $\left[r_{\min}, r_{\max}\right]$ is the feasible radii's range and d is a sensitivity parameter. By using a high d value, two very similar circles would be considered different while a smaller value for d would consider them as similar shapes. In this work, after several experiments, the d value has been set to 2.

Thus, since the historical memory $\mathbf{M}_h$ $\left\{C_1^{\mathbf{M}}, C_2^{\mathbf{M}}, \ldots, C_B^{\mathbf{M}}\right\}$ groups the elements in descending order according to their fitness values, the first element $C_1^{\mathbf{M}}$, whose fitness value represents the best value $J(C_1^{\mathbf{M}})$, is assigned to the first circle. Then, the distinctiveness factor $(D_{C_1^{\mathbf{M}},C_2^{\mathbf{M}}})$ over the next element $C_2^{\mathbf{M}}$ is evaluated with respect to the prior $C_1^{\mathbf{M}}$. If $D_{C_1^{\mathbf{M}},C_2^{\mathbf{M}}} > Th$, then $C_2^{\mathbf{M}}$ is considered as a new circle otherwise the next element $C_3^{\mathbf{M}}$ is selected. This process is repeated until the fitness value $J(C_i^{\mathbf{M}})$ reaches a minimum threshold J_{TH}. According to such threshold, other values above J_{TH} represent individuals (cirlces) that are considered as significant while other values lying below such boundary are considered as false circles and hence they are not contained in the image. After several experiments the value of J_{TH} is set to $(J(C_1^{\mathbf{M}})/10)$.

The fitness value of each detected circle is characterized by its geometric properties. Big and well-drawn circles normally represent points in the search space with higher fitness values whereas small and dashed circles describe points with lower fitness values. Likewise, circles with similar geometric properties, such as radius, size, etc., tend to represent locations holding similar fitness values. Considering that the historical memory $\mathbf{M}_h$ groups the elements in descending order according to their fitness values, the proposed procedure allows the cancelling of those circles which belong to the same circle and hold a similar fitness value.

9.5.4 Implementation of CAB Strategy for Circle Detection

The implementation of the proposed algorithm can be summarized in the following steps:

Step 1:	Adjust the algorithm parameters N_p, B, H, P, NI and d.
Step 2:	Randomly generate a set of N_p candidate circles (position of each animal) $\mathbf{C} = \{C_1, C_2, \ldots, C_{N_p}\}$ set using Eq. (9.1)
Step 3:	Sort $\mathbf{C}$ according to the objective function (dominance) to build $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\}$
Step 4:	Choose the first B positions of $\mathbf{X}$ and store them into the memory $\mathbf{M}_g$
Step 5:	Update $\mathbf{M}_h$ according to Sect. 2.1.5. (during the first iteration:$\mathbf{M}_h = \mathbf{M}_g$)
Step 6:	Generate the first B positions of the new solution set $\mathbf{C}$ ($\{C_1, C_2, \ldots, C_B\}$. Such positions correspond to the elements of $\mathbf{M}_h$ making a slight random perturbation around them $C_l = \mathbf{m}_h^l + \mathbf{v}$; being $\mathbf{v}$ a random vector of a small enough length
Step 7:	Generate the rest of the $\mathbf{C}$ elements using the attraction, repulsion and random movements

```
for $i$=$B$+1: $N_p$
    if ($r_1 < P$) then
    attraction and repulsion movement
        { if ($r_2 < H$) then
            $C_i = \mathbf{x}_i \pm r \cdot (\mathbf{m}_h^{nearest} - \mathbf{x}_i)$
            else if
            $C_i = \mathbf{x}_i \pm r \cdot (\mathbf{m}_g^{nearest} - \mathbf{x}_i)$
        }
    else if
    random movement
        {
            $C_i = \mathbf{r}$
        }
end for   where $r_1, r_2, r \in \text{rand}(0,1)$
```

Step 8:	If NI is not completed, the process go back to step 3. Otherwise, the best values in $\mathbf{M}_h$ $\{C_1^{\mathbf{M}}, C_2^{\mathbf{M}}, \ldots, C_B^{\mathbf{M}}\}$ represents the best solutions (the best found circles)
Step 9:	The element with the highest fitness value $J(C_1^{\mathbf{M}})$ is identified as the first circle C_1
Step 10:	The distinctiveness factor $D_{C_m^{\mathbf{M}}, C_{m-1}^{\mathbf{M}}}$ of circle $C_m^{\mathbf{M}}$ (element m) with the next highest probability is evaluated with respect to $C_{m-1}^{\mathbf{M}}$. If $D_{C_m^{\mathbf{M}}, C_{m-1}^{\mathbf{M}}} > Th$, then it is considered $C_m^{\mathbf{M}}$ as a new circle otherwise the next action is evaluated
Step 11:	The step 10 is repeated until the element's fitness value reaches $(J(C_1^{\mathbf{M}})/10)$

The number of candidate circles N_p is set considering a balance between the number of local minima to be detected and the computational complexity. In general terms, a large value of N_p suggests the detection of a great amount of circles at the cost of excessive computer time. After exhaustive experimentation, it has been found that a value of $N_p = 30$ represents the best trade-off between computational overhead and accuracy and therefore such value is used throughout the study.

Table 9.5 CAB detector parameters

N_p	H	P	B	NI
30	0.5	0.1	12	200

9.6 Results on Multi-circle Detection

In order to achieve the performance analysis, the proposed approach is compared to the BFAO detector, the GA-based algorithm and the RHT method over an image set.

The GA-based algorithm follows the proposal of Ayala-Ramirez et al. [41], which considers the population size as 70, the crossover probability as 0.55, the mutation probability as 0.10 and the number of elite individuals as 2. The roulette wheel selection and the 1-point crossover operator are both applied. The parameter setup and the fitness function follow the configuration suggested in [41]. The BFAO algorithm follows the implementation from [45] considering the experimental parameters as: $S = 50$, $N_c = 350$, $N_s = 4$, $N_{\text{ed}} = 1$, $P_{\text{ed}} = 0.25$, $d_{\text{attract}} = 0.1$, $w_{\text{attract}} = 0.2$, $w_{\text{repellant}} = 10$ $h_{\text{repellant}} = 0.1$, $\lambda = 400$ and $\psi = 6$. Such values are found to be the best configuration set according to [45]. Both, the GA-based algorithm and the BAFO method use the same objective function that is defined by Eq. (9.12). Likewise, the RHT method has been implemented as it is described in [40]. Finally, Table 9.5 presents the parameters for the CAB algorithm used in this work. They have been kept for all test images after being experimentally defined.

Images rarely contain perfectly-shaped circles. Therefore, with the purpose of testing accuracy for a single-circle, the detection is challenged by a ground-truth circle which is determined from the original edge map. The parameters $(x_{true}, y_{true}, r_{true})$ representing the testing circle are computed using Eqs. (9.6–9.9) for three circumference points over the manually-drawn circle. Considering the centre and the radius of the detected circle are defined as (x_D, y_D) and r_D, the Error Score (Es) can be accordingly calculated as:

$$Es = \eta \cdot (|x_{true} - x_D| + |y_{true} - y_D|) + \mu \cdot |r_{true} - r_D| \tag{9.16}$$

The central point difference $(|x_{true} - x_D| + |y_{true} - y_D|)$ represents the centre shift for the detected circle as it is compared to a benchmark circle. The radio mismatch $(|r_{true} - r_D|)$ accounts for the difference between their radii. η and μ represent two weighting parameters which are to be applied separately to the central point difference and to the radio mismatch for the final error Es. At this work, they are chosen as $\eta = 0.05$ and $\mu = 0.1$. Such particular choice ensures that the radii difference would be strongly weighted in comparison to the difference of central circular positions between the manually detected and the machine-detected circles. Here we assume that if Es is found to be less than 1, then the algorithm gets a success, otherwise, we say that it has failed to detect the edge-circle. Note that for $\eta = 0.05$ and $\mu = 0.1$; $Es < 1$ means the maximum difference of radius tolerated is 10 while the maximum mismatch in the location of the center can be 20 (in number of pixels).

In order to appropriately compare the detection results, the Detection Rate (DR) is introduced as a performance index. DR is defined as the percentage of reaching detection success after a certain number of trials. For "success" it does mean that the compared algorithm is able to detect all circles contained in the image, under the restriction that each circle must hold the condition $Es < 1$. Therefore, if at least one circle does not fulfil the condition of $Es < 1$, the complete detection procedure is considered as a failure.

In order to use an error metric for multiple-circle detection, the averaged *Es* produced from each circle in the image is considered. Such criterion, defined as the Multiple Error (ME), is calculated as follows:

$$\mathrm{ME} = \left(\frac{1}{NC}\right) \cdot \sum_{R=1}^{NC} \mathrm{Es}_R \tag{9.17}$$

where $\boldsymbol{NC}$ represents the number of circles within the image according to a human expert.

Figure 9.6 shows three synthetic images and the resulting images after applying the GA-based algorithm [41], the BFOA method [45] and the proposed approach. Figure 9.7 presents experimental results considering three natural images. The performance is analyzed by considering 35 different executions for each algorithm. Table 9.6 shows the averaged execution time, the detection rate in percentage and the averaged multiple error (ME), considering six test images (shown by Figs. 9.6 and 9.7). The best entries are bold-cased in Table 9.6. Close inspection reveals that the proposed method is able to achieve the highest success rate keeping the smallest error, still requiring less computational time for the most cases.

In order to statistically analyze the results in Table 9.6, a non-parametric significance proof known as the Wilcoxon's rank test [64–66] for 35 independent samples has been conducted. Such proof allows assessing result differences among two related methods. The analysis is performed considering a 5% significance level over multiple error (ME) data. Table 9.7 reports the *p*-values produced by Wilcoxon's test for a pair-wise comparison of the multiple error (ME), considering two groups gathered as CAB versus GA and CAB versus BFOA. As a null hypothesis, it is assumed that there is no difference between the values of the two algorithms. The alternative hypothesis considers an existent difference between the values of both approaches. All *p*-values reported in Table 9.7 are less than 0.05 (5% significance level) which is a strong evidence against the null hypothesis, indicating that the best CAB mean values for the performance are statistically significant which has not occurred by chance.

Figure 9.8 demonstrates the relative performance of CAB in comparison with the RHT algorithm as it is described in [40]. All images belonging to the test are complicated and contain different noise conditions. The performance analysis is achieved by considering 35 different executions for each algorithm over the three images. The results, exhibited in Fig. 9.8, present the median-run solution (when the runs were ranked according to their final ME value) obtained throughout the

Table 9.6 The averaged execution-time, detection rate and the averaged multiple error for the GA-based algorithm, the BFOA method and the proposed CAB algorithm, considering six test images (shown by Figs. 8 and 9)

Image	Averaged execution time ± Standard deviation (s)			Success rate (DR) (%)			Averaged ME ± Standard deviation		
	GA	BFOA	CAB	GA	BFOA	CAB	GA	BFOA	CAB
Synthetic images									
(a)	2.23 ± (0.41)	1.71 ± (0.51)	**0.21 ± (0.22)**	88	99	**100**	0.41 ± (0.044)	0.33 ± (0.052)	**0.22 ± (0.033)**
(b)	3.15 ± (0.39)	2.80 ± (0.65)	**0.36 ± (0.24)**	79	92	**99**	0.51 ± (0.038)	0.37 ± (0.032)	**0.26 ± (0.041)**
(c)	4.21 ± (0.11)	3.18 ± (0.36)	**0.20 ± (0.19)**	74	88	**100**	0.48 ± (0.029)	0.41 ± (0.051)	**0.15 ± (0.036)**
Natural images									
(a)	5.11 ± (0.43)	3.45 ± (0.52)	**1.10 ± (0.24)**	90	96	**100**	0.45 ± (0.051)	0.41 ± (0.029)	**0.25 ± (0.037)**
(b)	6.33 ± (0.34)	4.11 ± (0.14)	**1.61 ± (0.17)**	83	89	**100**	0.81 ± (0.042)	0.77 ± (0.051)	**0.37 ± (0.055)**
(c)	7.62 ± (0.97)	5.36 ± (0.17)	**1.95 ± (0.41)**	84	92	**99**	0.92 ± (0.075)	0.88 ± (0.081)	**0.41 ± (0.066)**

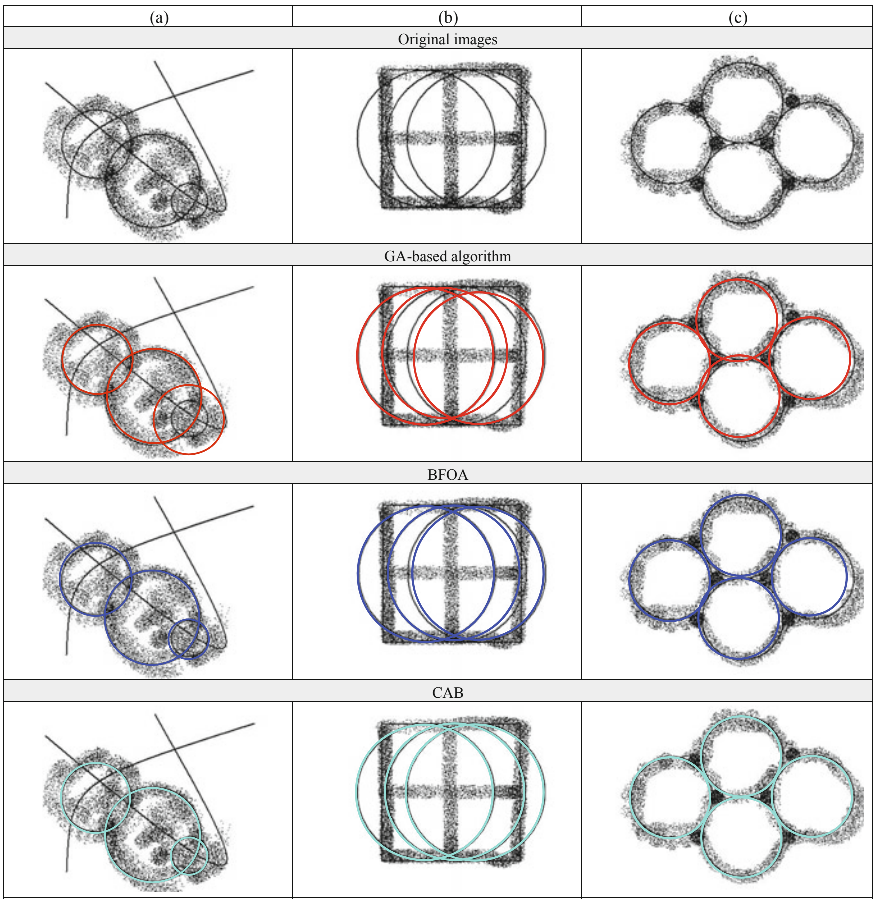


Fig. 9.6 Synthetic images and their detected circles for: GA-based algorithm, the BFOA method and the proposed CAB algorithm

Table 9.7 *p*-values produced by Wilcoxon's test comparing CAB to GA and BFOA over the averaged ME from Table 9.2

Image	*p*-value	
	CAB versus GA	CAB versus BFOA
Synthetic images		
(a)	1.8061e−004	1.8288e−004
(b)	1.7454e−004	1.9011e−004
(c)	1.7981e−004	1.8922e−004
Natural images		
(a)	1.7788e−004	1.8698e−004
(b)	1.6989e−004	1.9124e−004
(c)	1.7012e−004	1.9081e−004

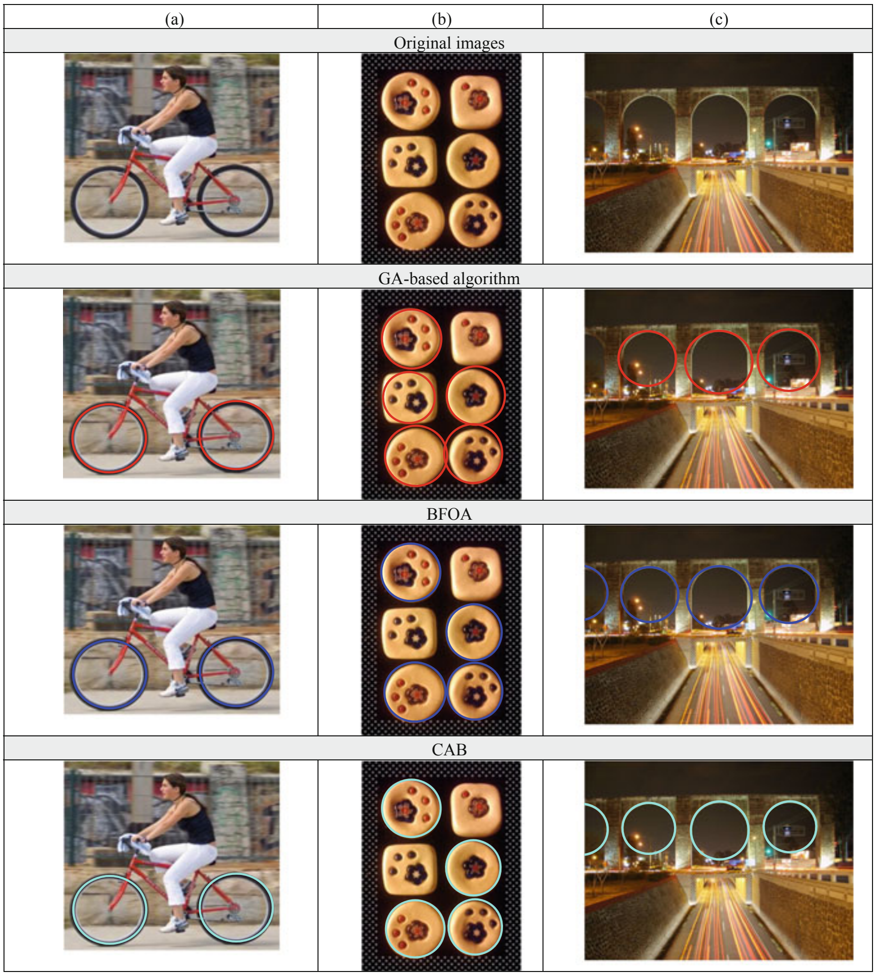


Fig. 9.7 Real-life images and their detected circles for: GA-based algorithm, the BFOA method and the proposed CAB algorithm

35 runs. On the other hand, Table 9.4 reports the corresponding averaged execution time, detection rate (in %), and average multiple error [using (10)]for CAB and RHT algorithms over the set of images (the best results are bold-cased). Table 9.8 shows a decrease in performance of the RHT algorithm as noise conditions change. Yet the CAB algorithm holds its performance under the same circumstances.

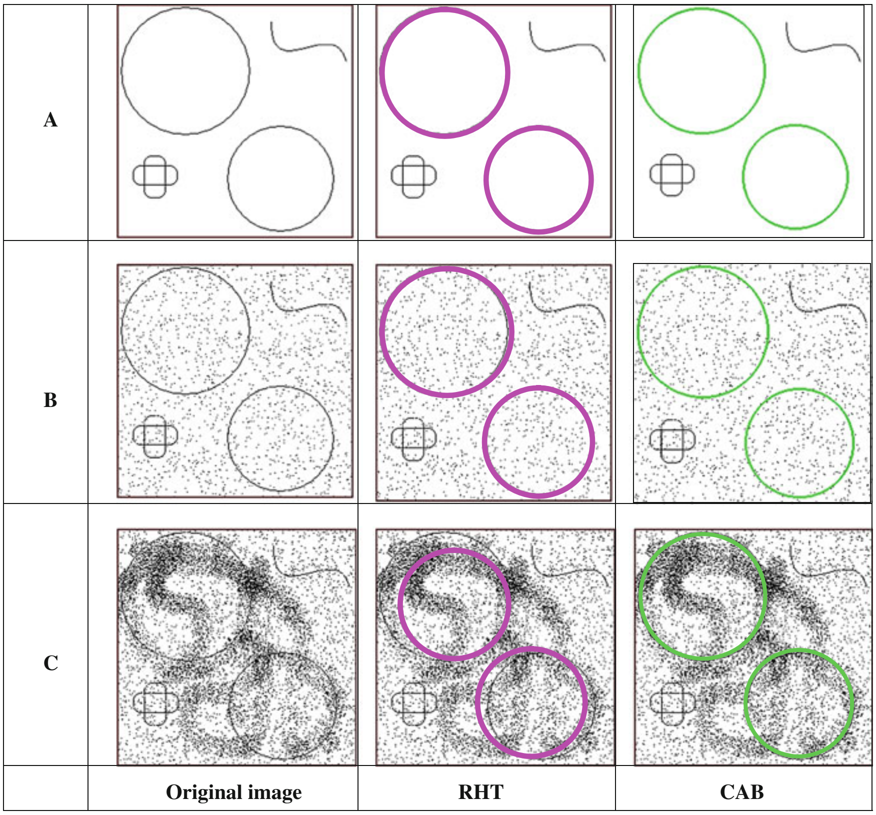


Fig. 9.8 Relative performance of the RHT and the CAB

Table 9.8 Average time, detection rate and averaged error for CAB and HT, considering three test images

Image	Average time ± Standard deviation (s)		Success rate (DR) (%)		Average ME ± Standard deviation	
	RHT	CAB	RHT	CAB	RHT	CAB
(A)	7.82 ± (0.34)	**0.30 ± (0.10)**	**100**	**100**	0.19 ± (0.041)	**0.11 ± (0.017)**
(B)	8.65 ± (0.48)	**0.22 ± (0.13)**	64	**100**	0.47 ± (0.037)	**0.13 ± (0.019)**
(C)	10.65 ± (0.48)	**0.25 ± (0.12)**	11	**100**	1.21 ± (0.033)	**0.15 ± (0.014)**

Bold data represent the best results

9.7 Conclusions

In recent years, several metaheuristic optimization methods have been inspired from nature-like phenomena. In this article, a new multimodal optimization algorithm known as the Collective Animal Behavior Algorithm (CAB) has been introduced. In CAB, the searcher agents emulate a group of animals that interact to each other depending on simple behavioral rules which are modeled as mathematical operators. Such operations are applied to each agent considering that the complete group hold a memory to store its own best positions seen so far, using a competition principle.

CAB has been experimentally evaluated over a test suite consisting of 8 benchmark multimodal functions for optimization. The performance of CAB has been compared to some other existing algorithms including Deterministic Crowding [17], Probabilistic Crowding [18], Sequential Fitness Sharing [15], Clearing Procedure [20], Clustering Based Niching (CBN) [19], Species Conserving Genetic Algorithm (SCGA) [21], elitist-population strategies (AEGA) [22], Clonal Selection algorithm [24] and the artificial immune network (aiNet) [25]. All experiments have demonstrated that CAB generally outperforms all other multimodal metaheuristic algorithms regarding efficiency and solution quality, typically showing significant efficiency speedups. The remarkable performance of CAB is due to two different features: (i) operators allow a better exploration of the search space, increasing the capacity to find multiple optima; (ii) the diversity of solutions contained in the $\mathbf{M}_h$ memory in the context of multimodal optimization, is maintained and even improved through of the use of a competition principle (dominance concept).

The proposed algorithm is also applied to the engineering problem of multi-circle detection. Such a process is faced as a multi-modal optimization problem. In contrast to other heuristic methods that employ an iterative procedure, the proposed CAB method is able to detect single or multiple circles over a digital image by running only one optimization cycle. The CAB algorithm searches the entire edge-map for circular shapes by using a combination of three non-collinear edge points as candidate circles (animal positions) in the edge-only image. A matching function (objective function) is used to measure the existence of a candidate circle over the edge-map. Guided by the values of such matching function, the set of encoded candidate circles is evolved using the CAB algorithm so that the best candidate can be fitted into an actual circle. After the optimization has been completed, an analysis of the embedded memory is executed in order to find the significant local minima (remaining circles). The overall approach generates a fast sub-pixel detector which can effectively identify multiple circles in real images despite some circular objects exhibit a significant occluded portion.

In order to test the circle detection performance, both speed and accuracy have been compared. Score functions are defined by Eqs. (9.16) and (9.17) in order to measure accuracy and effectively evaluate the mismatch between manually detected and machine-detected circles. We have demonstrated that the CAB method outperforms both the GA (as described in [41]) and the BFOA (as described in [45]) within a statistically significant framework (Wilcoxon test). In contrast to the CAB method, the RHT algorithm [40] shows a decrease in performance under noisy conditions. Yet the CAB algorithm holds its performance under the same circumstances. Finally, Table 9.6 indicates that the CAB method can yield better results on complicated and noisy images compared with the GA and the BFOA methods.

References

1. Ahrari, A., Shariat-Panahi, M., Atai, A.A.: GEM: a novel evolutionary optimization method with improved neighbourhood search. Appl. Math. Comput. **210**(2), 376–386 (2009)
2. Fogel, L.J., Owens, A.J., Walsh, M.J.: Artificial Intelligence through Simulated Evolution. John Wiley, Chichester, UK (1966)
3. De Jong, K.: Analysis of the behavior of a class of genetic adaptive systems, Ph.D. thesis, University of Michigan, Ann Arbor, MI (1975)
4. Koza, J.R.: Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems, Rep. No. STAN-CS-90-1314, Stanford University, CA (1990)
5. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975)
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Boston, MA (1989)
7. de Castro, L.N., Von Zuben, F.J.: Artificial immune systems: Part I—Basic theory and applications. Technical report, TR-DCA 01/99 (1999)
8. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
9. İlker, B., Birbil, S., Shu-Cherng, F.: An electromagnetism-like mechanism for global optimization. J. Global Optim. **25**, 263–282 (2003)
10. Rashedi, E., Nezamabadi-Pour, H., Saryazdi, S.G.S.A.: A gravitational search algorithm. Inf. Sci. **179**, 2232–2248 (2009)
11. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: harmony search. Simulation **76**(2), 60–68 (2001)
12. Lee, K.S., Geem, Z.W.: A new meta-heuristic algorithm for continues engineering optimization: harmony search theory and practice. Comput. Methods Appl. Mech. Eng. **194**, 3902–3933 (2004)
13. Geem, Z.W.: Novel derivative of harmony search algorithm for discrete design variables. Appl. Math. Comput. **199**, 223–230 (2008)
14. Gao, X.Z., Wang, X., Ovaska, S.J.: Uni-modal and multi-modal optimization using modified harmony search methods. Int. J. Innov. Comput. Inf. Control. 5(10A), 2985–2996 (2009)
15. Beasley, D., Bull, D.R., Matin, R.R.: A sequential niche technique for multimodal function optimization. Evol. Comput. **1**(2), 101–125 (1993)
16. Miller, B.L., Shaw, M.J.: Genetic algorithms with dynamic niche sharing for multimodal function optimization. In: Proceedings of the 3rd IEEE Conference on Evolutionary Computation, pp. 786–791 (1996)
17. Mahfoud, S.W.: Niching methods for genetic algorithms. Ph.D. dissertation, Illinois Genetic Algorithm Laboratory, University of Illinois, Urbana, IL (1995)
18. Mengshoel, O.J., Goldberg, D.E.: Probability crowding: deterministic crowding with probabilistic replacement. In: Banzhaf, W. (ed.), Proceedings of International Conferences GECCO-1999, Orlando, FL, pp. 409–416 (1999)
19. Yin, X., Germay, N.: A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In: Proceedings of the 1993 International Conference on Artificial Neural Networks and Genetic Algorithms, pp. 450–457 (1993)

20. Petrowski, A.: A clearing procedure as a niching method for genetic algorithms,. In: Proceeding of the 1996 IEEE International Conference on Evolutionary Computation, pp. 798–803. IEEE Press, New York (1996)
21. Li, J.P., Balazs, M.E., Parks, G.T., Glarkson, P.J.: A species conserving genetic algorithms for multimodal function optimization. Evol. Comput. **10**(3), 207–234 (2002)
22. Lianga, Y., Kwong-Sak, L.: Genetic Algorithm with adaptive elitist-population strategies for multimodal function optimization. Appl. Soft Comput. **11**, 2017–2034 (2011)
23. Wei, L.Y., Zhao, M.: A niche hybrid genetic algorithm for global optimization of continuous multimodal functions. Appl. Math. Comput. **160**(3), 649–661 (2005)
24. Castro, L.N., Zuben, F.J.: Learning and optimization using the clonal selection principle. IEEE Trans. Evol. Comput. **6**, 239–251 (2002)
25. Castro, L.N., Timmis, J.: An artificial immune network for multimodal function optimization. In: Proceedings of the 2002 IEEE International Conference on Evolutionary Computation. IEEE Press, New York, pp. 699–704 (2002)
26. Xu, Q., Lei, W., Si, J.: Predication based immune network for multimodal function optimization. Eng. Appl. Artif. Intell. **23**, 495–504 (2010)
27. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
28. Liang, Jj, Qin, A.K., Suganthan, P.N.: Comprehensive learning particle swarm optimizer for global optimization of multi-modal functions. IEEE Trans. Evol. Comput. **10**(3), 281–295 (2006)
29. Chen, D.B., Zhao, C.X.: Particle swarm optimization with adaptive population size and its application. Appl. Soft Comput. **9**(1), 39–48 (2009)
30. Sumper, D.: The principles of collective animal behaviour. Philos. Trans. R. Soc. Lond. B Biol. Sci. **36**(1465), 5–22 (2006)
31. Petit, O., Bon, R.: Decision-making processes: the case of collective movements. Behav. Proc. **84**, 635–647 (2010)
32. Kolpas, A., Moehlis, J., Frewen, T., Kevrekidis, I.: Coarse analysis of collective motion with different communication mechanisms. Math. Biosci. **214**, 49–57 (2008)
33. Couzin, I.: Collective cognition in animal groups. Trends in Cogn. Sci. **13**(1), 36–43 (2008)
34. Couzin, I.D., Krause, J.: Self-organization and collective behavior in vertebrates. Adv. Stud. Behav. **32**, 1–75 (2003)
35. Bode, N., Franks, D., Wood, A.: Making noise: emergent stochasticity in collective motion. J. Theor. Biol. **267**, 292–299 (2010)
36. Couzi, I., Krause, I., James, R., Ruxton, G., Franks, N.: Collective memory and spatial sorting in animal groups. J. Theor. Biol. **218**, 1–11 (2002)
37. Couzin, I.D.: Collective minds. Nature **445**, 715–728 (2007)
38. Bazazi, S., Buhl, J., Hale, J.J., Anstey, M.L., Sword, G.A., Simpson, S.J., Couzin, I.D.: Collective motion and cannibalism in locust migratory bands. Curr. Biol. **18**, 735–739 (2008)
39. Atherton, T.J., Kerbyson, D.J.: Using phase to represent radius in the coherent circle Hough transform. In: Proceedings of IEE Colloquium on the Hough Transform. IEE, London (1993)
40. Xu, L., Oja, E., Kultanen, P.: A new curve detection method: Randomized Hough transform (RHT). Pattern Recognit. Lett. **11**(5), 331–338 (1990)
41. Ayala-Ramirez, V., Garcia-Capulin, C.H., Perez-Garcia, A., Sanchez-Yanez, R.E.: Circle detection on images using genetic algorithms. Pattern Recognit. Lett. **27**, 652–657 (2006)
42. Cuevas, Erik, Ortega-Sánchez, Noé, Zaldivar, Daniel, Pérez-Cisneros, Marco: Circle detection by harmony search optimization. J. Intell. Rob. Syst. **66**(3), 359–376 (2012)
43. Cuevas, Erik, Oliva, Diego, Zaldivar, Daniel, Pérez-Cisneros, Marco, Sossa, Humberto: Circle detection sing electro-magnetism optimization. Inf. Sci. **182**(1), 40–55 (2012)
44. Cuevas, Erik, Zaldivar, Daniel, Pérez-Cisneros, Marco, Ramírez-Ortegón, Marte: Circle detection using discrete differential evolution optimization. Pattern Anal. Appl. **14**(1), 93–107 (2011)
45. Dasgupta, Sambarta, Das, Swagatam, Biswas, Arijit, Abraham, Ajith: Automatic circle detection on digital images with an adaptive bacterial foraging algorithm. Soft. Comput. **14**(11), 1151–1164 (2010)

46. Bode, N., Wood, A., Franks, D.: The impact of social networks on animal collective motion. Anim. Behav. **82**(1), 29–38 (2011)
47. Lemasson, B., Anderson, J., Goodwin, R.: Collective motion in animal groups from a neurobiological perspective: The adaptive benefits of dynamic sensory loads and selective attention. J. Theor. Biol. **261**(4), 501–510 (2009)
48. Bourjade, M., Thierry, B., Maumy, M., Petit, O.: Decision-making processes in the collective movements of Przewalski horses families *Equus ferus Przewalskii*: influences of the environment. Ethology **115**, 321–330 (2009)
49. Banga, A., Deshpande, S., Sumanab, A., Gadagkar, R.: Choosing an appropriate index to construct dominance hierarchies in animal societies: a comparison of three indices. Anim. Behav. **79**(3), 631–636 (2010)
50. Hsu, Y., Earley, R., Wolf, L.: Modulation of aggressive behaviour by fighting experience: mechanisms and contest outcomes. Biol. Rev. **81**(1), 33–74 (2006)
51. Broom, M., Koenig, A., Borries, C.: Variation in dominance hierarchies among group-living animals: modeling stability and the likelihood of coalitions. Behav. Ecol. **20**, 844–855 (2009)
52. Bayly, K.L., Evans, C.S., Taylor, A.: Measuring social structure: a comparison of eight dominance indices. Behav. Proc. **73**, 1–12 (2006)
53. Conradt, L., Roper, T.J.: Consensus decision-making in animals. Trends Ecol. Evol. **20**, 449–456 (2005)
54. Okubo, A.: Dynamical aspects of animal grouping. Adv. Biophys. **22**, 1–94 (1986)
55. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioural model. Comp. Graph. **21**, 25–33 (1987)
56. Gueron, S., Levin, S.A., Rubenstein, D.I.: The dynamics of mammalian herds: from individual to aggregations. J. Theor. Biol. **182**, 85–98 (1996)
57. Czirok, A., Vicsek, T.: Collective behavior of interacting self-propelled particles. Phys. A **281**, 17–29 (2000)
58. Ballerini, M.: Interaction ruling collective animal behavior depends on topological rather than metric distance: evidence from a field study. Proc. Natl. Acad. Sci. U.S.A. **105**, 1232–1237 (2008)
59. Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press (2008)
60. Gandomi, A.H., Yang, X.-S., Alavi, A.H.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng. Comput. https://doi.org/10.1007/s00366-011-0241-y
61. Zang, H., Zhang, S., Hapeshi, K.: A review of nature-inspired algorithms. J. Bionic Eng. **7**, S232–S237 (2010)
62. Gandomi, A., Alavi, A.: Krill herd: a new bio-inspired optimization algorithm. Commun. Nonlinear Sci. Numer. Simul. **17**, 4831–4845 (2012)
63. Bresenham, J.E.: A linear algorithm for incremental digital display of circular arcs. Commun. ACM **20**, 100–106 (1977)
64. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics **1**, 80–83 (1945)
65. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special session on real parameter optimization. J. Heurist. (2008). https://doi.org/10.1007/s10732-008-9080-4
66. Santamaría, J., Cordón, O., Damas, S., García-Torres, J.M., Quirin, A.: Performance evaluation of memetic approaches in 3D reconstruction of forensic objects. Soft Comput. (2008). https://doi.org/10.1007/s00500-008-0351-7

Chapter 10
Locust Search Algorithm Applied for Template Matching

Abstract For a wide-range of image processing tasks (such as feature tracking, object recognition, stereo matching and remote sensing), Template Matching (TM) serves as a crucial strategy to allow the localization and recognition of objects and/or patterns within digital images. The purpose of a TM approach is to find the location (region) within a source image which manifest the best possible resemblance to a given sub-image (commonly known as image template). Any TM algorithm (independently of their particularities) incorporates two important elements: 1. a search strategy; and 2. a similarity measurement. In its simplest form, a TM method involves exhaustive search processes were a similarity measurement (such as the Normalized Cross-Correlation) is applied for each available location within the source image. Intuitively, such an extensive amount of function evaluations has a strong impact on the algorithms computational cost, an issue that happens to be extremely detrimental for many real-world applications. In the last few years, Evolutionary Algorithms (EAs) have been proposed as an alternative to aid on the search process of TM approaches by reducing the amount of locations that are evaluated within a given source image; however, it is known that many of these methods carry with them several critical flaws which could negatively impact the TM process, including an insufficient exploration of the source image (which often leads to premature convergence). In this chapter, the swarm optimization algorithm known as Locust Search (LS) is applied to aid on solving the of template matching. The LS method includes a series of unique evolutionary operators that allows to explicitly avoid the concentration of search agents toward the best-known solutions, which in turn allows these agents to better explore of the available image's search region. According to a series of experimental results, when compared to other TM approaches which integrates EAs as a part of their search strategy, LS achieves the best between estimation accuracy and computational load.

10.1 Introduction

The localization and recognition of objects or patterns in digital images represents one of the most important tasks for several image processing and computer vision

© Springer Nature Switzerland AG 2020
E. Cuevas et al., *New Advancements in Swarm Algorithms: Operators and Applications*, Intelligent Systems Reference Library 160,
https://doi.org/10.1007/978-3-030-16339-6_10

applications, such as industrial inspection, target classification, digital photometry, remote sensing, among others [1].

Template matching (TM) is an image processing technique which aims to define the localization of objects or patters within a digital image by finding the best-possible resemblance between a sample sub-image (usually known as template) and a coincident region within a source image. On a typical TM procedure, the similarity between a template and a region within a given digital image is determined by applying a specific similarity measurement over a neighborhood around a given pixel location within such image. Several measurements, such as the Sum of Absolute Differences (SAD), the Sum of Squared Differences (SSD), and the Normalized Cross-Correlation (NCC) are among the most common metrics used to evaluate the similarity between the template and the source image. However, the calculation of such similarity measurements comes at the expense of a high computational cost, and as such they usually represent the most time-consuming operation on a TM process; this, added to the fact that most traditional TM algorithms exhaustively search on every single pixel location of the source image, limits the use of such recognition techniques in most real-time computer vision applications [2–4]. With that being said, research on modern TM techniques aim to improve the template's detection performance in terms of two particular tasks: (1) the image search strategy, and (2) the similarity measurement criteria [5, 6].

Recently, several TM algorithms based on evolutionary optimization techniques have been proposed as an alternative to reduce the computational cost that is inherently related to the matching process; this is achieved by strategically searching over a limited subset of pixel locations within the image plane, which in turn allows a substantial reduction on the number of function evaluations (similarity measurements) that are required by such matching process. Such approaches have yield to several robust detectors based on many different optimization methods such as the well-known Genetic Algorithms (GA) approach [7], the popular Particle Swarm Optimization (PSO) algorithm [8], and the interesting Imperialist Competitive Algorithm (ICA) method [9]. Although such optimization approaches allow a significant reduction in the number of searched image locations, they often suffer from premature convergence as a result of an ineffective exploration strategy, yielding to sub-optimal (or even erroneous) detections [10]. Such difficulties are often related to both, the operators used to guide de algorithm's search process, and the lack of an appropriate exploitation mechanism [11, 12].

Recently, a swarm optimization method known as Locust Search (LS) has been proposed to solve unconstrained optimization problems [13]. Such approach emulates the exotic and distinctive behavior manifested by swarms of locust when looking for food sources. In LS, search agents are modeled as individuals within a swarm of locust which interact to each other based on series of evolutionary operators which mimic several distinctive biological behaviors commonly found in such groups of insects. In the LS method, the entire search space is represented as a plantation, which all locusts explore in search for appropriate food sources. At each stage of the LS's evolutionary process, every locust within such plantation receives a food quality index according to the fitness value corresponding to its current position. Different to most existent Evolutionary Algorithms (EA), the behavioral model proposed in LS explicitly avoids the concentration of individuals over the current best solutions, which in turn allows to avoid several critical flaws commonly found in other EA, such as premature convergence and the lack of balance between exploration and exploitation [14].

In this paper, the LS method is proposed for solving the previously illustrated TM matching problem. In the proposed approach, individuals within a swarm of locust are represented by pixel positions defined over a specific search region within a given digital image. Each individual within the swarm moves through several positions of the source image in search of the better food sources, which are further represented by the positions in which the value of NCC coefficient (computed with respect to a given image template) yields to higher values. In this sense, the best food source (optimal solution) is represented by the location within the image in which the NCC score attains its maximum value.

In order to prove the feasibility of the proposed TM method, our experimental results are further compared with several other TM approaches based on popular EA methods, such as the Genetic Algorithms (GA) [7], Particle Swarm Optimization (PSO) [8], Artificial Bee Colony (ABC) [15], and Imperialist Competitive Algorithm (ICA) [9].

10.2 Template Matching Process

In order to illustrate the process known as Template Matching (TM), let I denote an intensity image with a certain spatial resolution (size) $M \times N$, and let R represent a reference image (or image template) of size $m \times n$. Furthermore, let (x, y) denote a coordinate pair of positions of R. If we consider the shifted reference image $R_{u,v}(x, y) = R(x - u, y - v)$ such that u and v represent and horizontal and vertical displacement over the source image I respectively, then the matching problem may be summarized as follows: Given a source image I and a reference image R, find the offset (u, v) within a search region $\mathbf{S} \in I$ such that the similarity between the shifted reference image $R_{u,v}(x, y)$ and the corresponding sub-image of I is maximum. In order to solve such a problem, two important issues must first be addressed: (1) determining and appropriate similarity measurement to validate a match occurrence, and (2) develop an efficient search strategy to find the optimal template displacement (u, v) (Fig. 10.1).

Although there exist several metrics used to evaluate the similarity between two images, in the TM process the most commonly used measurements include the Sum of Absolute Differences (SAD), the Sum of Squared Differences (SSD), and the Normalizes Cross-Correlation (NCC). However, it is known that the calculation of any of such similarity measurements demands a high computational cost, and as such, it represents the most time-consuming operation on a typical TM process [16]. Furthermore, although such metrics allow an adequate measurement of the similarity

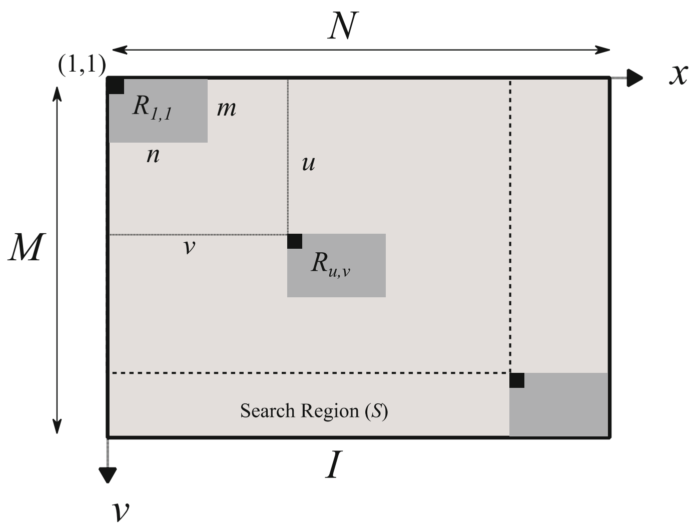


Fig. 10.1 Template matching process. The reference image R is shifted by an offset (u, v) across the a search region **S** defined within a given source image I. The total search region **S** depends on both, the size of source image I $(M \times N)$ and the size of the reference image R $(m \times n)$

between a given pair of images, the NCC coefficient is considered to be the most robust measurement among them, and as such is most commonly used [16].

The NCC value between a given source image I of size $N \times M$ and an image template R of size $m \times n$, at a given image displacement (u, v), is given as follows:

$$\mathrm{NCC}(u, v) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \left[I(u+i, v+j) - \overline{I}(u, v) \right] \cdot \left[R(i, j) - \overline{R} \right]}{\sqrt{\left[\sum_{i=1}^{m} \sum_{j=1}^{n} \left[I(u+i, v+j) - \overline{I}(u, v) \right] \right]^2 \cdot \left[\sum_{i=1}^{m} \sum_{j=1}^{n} \left[R(i, j) - \overline{R} \right] \right]^2}} \tag{10.1}$$

where $\overline{I}(u, v)$ denotes the gray-scale average intensity of the source image I for the coincident region of the image template R, while $\overline{R}$ stand for the gray-scale average intensity of the image template R. Such values are given as follows:

$$\overline{I}(u, v) = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} I(u+i, v+j) \quad \overline{R} = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} R(i, j) \tag{10.2}$$

The NCC computation yields to values defined between the interval of $[-1, 1]$, where score of NCC $= 1$ implies the best possible similarity between the image template R and its corresponding sub-image on I, whereas a value of NCC $= -1$ means that both of such images are completely different.

Furthermore, in the context of the TM process, let $(\hat{u}, \hat{v})$ denote the position within the source image I in which the best-best possible resemblance (maximum NCC value) between R and I is found. Such image position is defined as follows:

$$\left(\hat{u},\hat{v}\right) = \arg\max_{(u,v)\in\mathbf{S}} \mathrm{NCC}\left(\hat{u},\hat{v}\right) \tag{10.3}$$

where $\mathbf{S} = \{(u,v)|1 \le u \le M - m,\ 1 \le v \le N - n\}$, as previously stated, denotes a search region defined within the source image I.

In a typical TM algorithm, the process to find the image displacement $\left(\hat{u},\hat{v}\right)$ which satisfies a maximum degree of resemblance involves an exhaustive search over all valid pixel positions within the source image I. While this approach yield to an optimal detection with respect to the NCC score, such exhaustive search, in conjunction with the NCC coefficient's high computational cost, seriously constrains the use of classic TM approaches in many image processing and computer vision applications.

With the previous being said, it is clear that developing an adequate search strategy to find the optimal image displacement $\left(\hat{u},\hat{v}\right)$ is important to increase the efficiency of the TM process. In fact, as illustrated by Eq. (10.3), the modeled as a global optimization problem, in which we aim to find the optimal combination of discrete horizontal and vertical displacements, $\hat{u}$ and $\hat{v}$ respectively, such that the similarity between the image template and its corresponding sub-image in position $\left(\hat{u},\hat{v}\right)$ yields to a maximum NCC value. In this sense, the use of optimization techniques, as an alternative to solve the problem of efficient TM matching, become intuitive [17].

Figure 10.2 illustrates the TM process with regard to the NCC coefficient: Fig. 10.2a and b illustrate both, an example source image and an image template, respectively; Fig. 10.2c shows the color-encoded NCC values corresponding to all locations within the valid search region $\mathbf{S}$ on the source image (full search strategy); finally, Fig. 10.2d presents the NCC surface, which exhibits the highly multimodal nature of a typical TM problem. Furthermore, by observing both, Fig. 10.2c and d, it is clear that the surface generated by the NCC values has several local maxima positions, while it only has a single global maximum. In such situations, classical optimization methods (particularly those founded on gradient-based techniques) are subject to be trapped in local optimal values, and as such, they are unfeasible for solving this kind of optimization problems.

10.3 The Locust Search (LS) Algorithm

The Locust Search (LS) algorithm is a swarm optimization method inspired in several behaviors commonly found in swarms of locust [18]. In the LS method, the entire search space is assumed as a plantation, where all individuals within the swarm interact to each other. In the LS approach, each solution within the search space represents a locust position within the plantation. Also, each locust receives a food quality index based on the fitness value related to the solution that is represented by each of such individuals [19]. Furthermore, and unique to the LS method, individuals within the swarm's population are guided by a set of evolutionary operators based

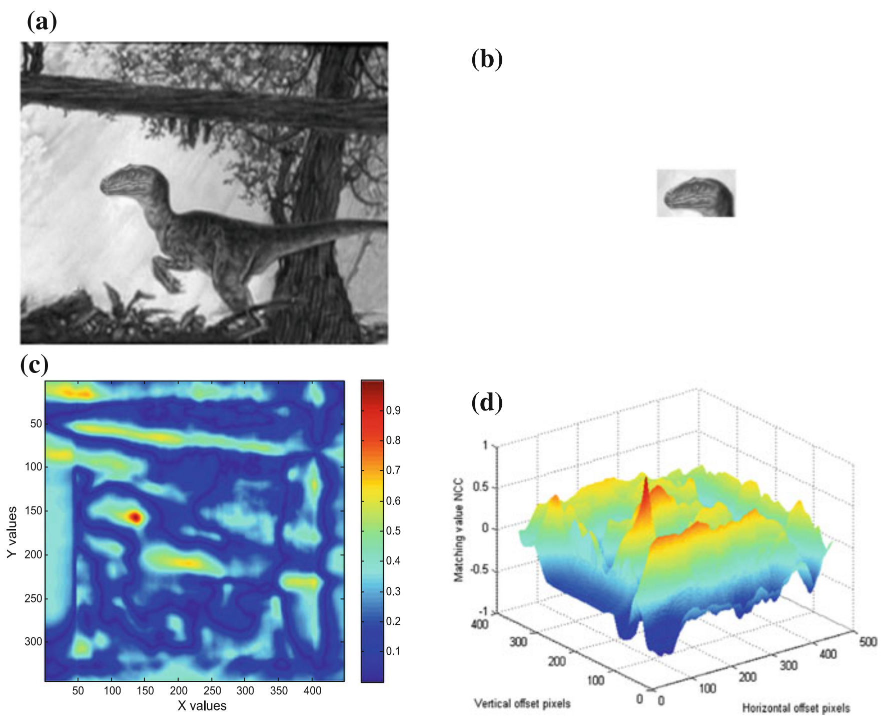


Fig. 10.2 **a** Example of a source image, **b** a reference image (image template), **c** color-encoded NCC values corresponding to the template matching process between (**a**) and (**b**), and **d** the NCC multimodal surface of (**c**)

on two distinctive behaviors that are commonly observed in swarms of locust: (1) a solitary phase, and (2) a social phase.

10.3.1 LS Solitary Phase

One distinctive feature of the LS method is the integration of a unique behavior known as solitary phase. Under this behavioral model, each locust within the swarm is assumed to be displaced as a result of a social force, which is related to their positional relationships with respect to other members of the aggregation. Therefore, the net effect caused by such social force are: (1) an attraction toward distant individuals, or (2) a repulsion between nearer individuals.

In the LS approach, the concept of social force is applied to develop a solitary operator specifically devised to explicitly avoid the concentration of individual toward the best solutions found during the evolutionary process, which in turn allows a sufficient exploration of the entire search space [20]. In order to illustrate such operator, let $\mathbf{L}^k = \{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\}$ denote a population (set of solutions) comprised by N locusts, where $k = 1, 2, \ldots, gen$ denotes the current iteration number of whole the evolutionary process (with gen denoting the maximum number of iterations). At each iteration k, the solitary operation produces a new position $\mathbf{p}_i^k$ by perturbing the current locust position $\mathbf{l}_i^k$ with a change of position $\Delta\mathbf{l}_i^k$, such that:

$$\mathbf{p}_i^k = \mathbf{l}_i^k + \Delta\mathbf{l}_i^k \tag{10.4}$$

The position change $\Delta\mathbf{l}_i^k$ results from the social force experimented by the individual $\mathbf{l}_i^k$ with respect to the other $N-1$ individuals in the entire locust population. With that being said, the social force exerted between a given individual $\mathbf{l}_i^k$ and any other locust $\mathbf{l}_j^k$ within the swarm is calculated as follows:

$$\mathbf{s}_{ij}^k = \rho\left(\mathbf{l}_i^k, \mathbf{l}_j^k\right) \cdot s\left(r_{ij}\right) \cdot \mathbf{d}_{ij} + \text{rand}(1, -1) \tag{10.5}$$

where $r_{ij} = \left\|\mathbf{l}_i^k - \mathbf{l}_j^k\right\|$ denotes the Euclidian distance between individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$, Furthermore, $\mathbf{d}_{ij} = \left(\mathbf{l}_j^k - \mathbf{l}_i^k\right)/r_{ij}$ represent a unit vector which points from $\mathbf{l}_i^k$ to $\mathbf{l}_j^k$, while rand$(1, -1)$ stands for a randomly generated number from within the interval $[1, -1]$. Also, $s\left(r_{ij}\right)$ represents the so called social relation between $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$, as defined as follows:

$$s\left(r_{ij}\right) = F \cdot e^{-r_{ij}/L} - e^{-r_{ij}} \tag{10.6}$$

where F and L denote, an attraction magnitude and an attractive length scale, respectively [21]. On the other hand, $\rho\left(\mathbf{l}_i^k, \mathbf{l}_j^k\right)$ stand for what is referred as the dominance value between $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$. In the LS approach, each individual from $\mathbf{L}^k$ $\left(\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\}\right)$ is ranked according to their fitness values, with the best individual (most dominant locust) receiving a rank of 0 (zero), whereas the worst individual (least dominant locust) gains the rank $N-1$. With that being said, the value of $\rho\left(\mathbf{l}_i^k, \mathbf{l}_j^k\right)$ is computed by considering the rank corresponding to the individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$, as defined as follows:

$$\rho\left(\mathbf{l}_i^k, \mathbf{l}_j^k\right) = \begin{cases} e^{-\left(\text{rank}\left(\mathbf{l}_i^k\right)/N\right)} & \text{if}\,\text{rank}\left(\mathbf{l}_i^k\right) < \text{rank}\left(\mathbf{l}_j^k\right) \\ e^{-\left(\text{rank}\left(\mathbf{l}_j^k\right)/N\right)} & \text{if}\,\text{rank}\left(\mathbf{l}_i^k\right) > \text{rank}\left(\mathbf{l}_j^k\right) \end{cases} \tag{10.7}$$

where rank$\left(\mathbf{l}_i^k\right)$ and rank$\left(\mathbf{l}_j^k\right)$ stand for the ranks of the individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$ respectively. Intuitively, the value $\rho\left(\mathbf{l}_i^k, \mathbf{l}_j^k\right)$ has the property to magnify or weaken the social force experimented between the individuals $\mathbf{l}_i^k$ and $\mathbf{l}_j^k$ depending on the fitness of the most dominant member among them.

Finally, the total social force experimented by individual $\mathbf{l}_i^k$ is computed as a superposition of all pairwise interactions exerted on it, as given as follows:

$$\mathbf{S}_i^k = \sum_{\substack{j=1 \\ j \neq 1}}^{N} \mathbf{s}_{ij}^k \tag{10.8}$$

Therefore, the change of position $\Delta \mathbf{l}_i^k$ is given by the total social force $\mathbf{S}_i^k$ experimented by $\mathbf{l}_i^k$, such that (Fig. 10.3):

$$\Delta \mathbf{l}_i^k = \mathbf{S}_i^k \tag{10.9}$$

Once a new set of positions $\mathbf{P}^k = \left\{\mathbf{p}_1^k, \mathbf{p}_2^k, \ldots, \mathbf{p}_N^k\right\}$, corresponding to the individuals within the population $\mathbf{L}^k\left(\left\{\mathbf{l}_1^k, \mathbf{l}_2^k, \ldots, \mathbf{l}_N^k\right\}\right)$, has been computed, each individual's position $\mathbf{l}_i^k$ for the next iteration of the evolutionary process is updated as follows:

$$\mathbf{l}_i^{k+1} = \begin{cases} \mathbf{p}_i^k \ \text{if} \ f\left(\mathbf{p}_i^k\right) > f\left(\mathbf{l}_i^k\right) \\ \mathbf{l}_i^k \quad \text{otherwise} \end{cases} \tag{10.10}$$

where $f\left(\mathbf{p}_i^k\right)$ and $f\left(\mathbf{l}_i^k\right)$ denotes the fitness evaluation function with respect to positions $\mathbf{p}_i^k$ and $\mathbf{l}_i^k$ respectively. In other words, the position of a given individual $\mathbf{l}_i^k$ for the iteration $k+1$ is updated only if its respective position $\mathbf{p}_i^k$ promotes such individual to get a better fitness value than that on its current position; otherwise, its position remains unchanged. It is important to note that the previous is illustrated by considering a maximization optimization problem.

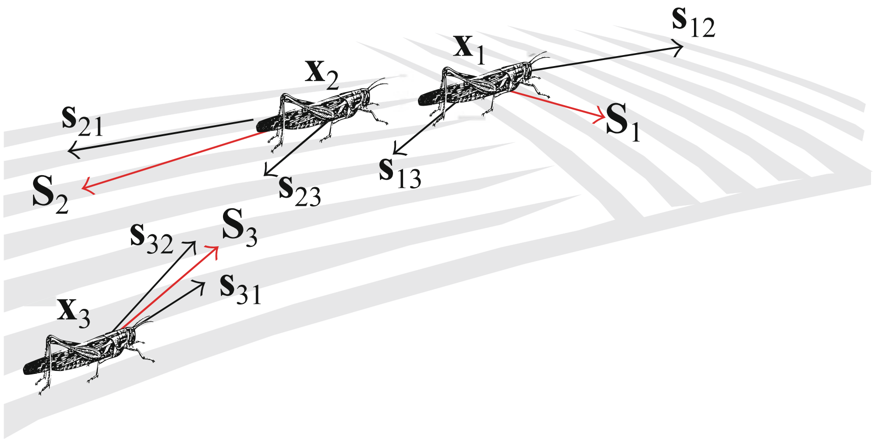


Fig. 10.3 LS solitary phase. Under this behavioral model, each locust's movement is computed respective to the total social force experimented by such individual

10.3.2 LS Social Phase

Different to the solitary phase illustrated in Sect. 10.3.1, in the LS approach, the so called social phase represents a selective operator used to refine a particular subset of individuals $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_q\}$ in order to improve their solution quality. Such subset $\mathbf{B}$ is formed by the q best individuals within the set of solutions $\mathbf{L}^{k+1} = \{\mathbf{l}_1^{k+1}, \mathbf{l}_2^{k+1}, \ldots, \mathbf{l}_N^{k+1}\}$, which correspond to the individuals' positions for the next iteration of the evolutionary process. Therefore, for each individual within the subset $\mathbf{B} \in \mathbf{L}^{k+1}$, a subspace C_j around each of such individuals is created. The limits of each of such subspaces depend on distance r, as defined as follows:

$$r = \frac{\sum_{d=1}^{n}\left(b_d^{high} - b_d^{low}\right)}{n} \cdot \beta \tag{10.11}$$

where b_d^{low} and b_d^{high} denote the lower and upper bounds in the d-th dimension, while n stand for total number of decision variables (dimensions). Furthermore, $\beta \in [0, 1]$ represents a scalar factor used to modulate the size of the subspace. Therefore, for each individual $\mathbf{l}_j^{k+1} = \left[l_{j,1}^{k+1}, l_{j,2}^{k+1}, \ldots, l_{j,n}^{k+1}\right]$ (where $\mathbf{l}_j^{k+1} \in \mathbf{B}$), the limits of each subspace C_j is given as follows:

$$\begin{aligned} C_{j,d}^{low} &= l_{j,d}^{k+1} - r \\ C_{j,d}^{high} &= l_{j,d}^{k+1} + r \end{aligned} \tag{10.12}$$

where $C_{j,d}^{low}$ and $C_{j,d}^{high}$ represent the upper and lower bounds of each subspace C_j at the d-th dimension, respectively. Finally, for each of subspace C_j, a new set of h new solutions $\mathbf{M}^j = \left\{\mathbf{m}_1^j, \mathbf{m}_2^j, \ldots, \mathbf{m}_h^j\right\}$ is generated within the bounds of each of such subspaces (see Fig. 10.4). If any of the fitness value of any of the solutions within $\mathbf{M}^j$ is better than that of their corresponding individual $\mathbf{l}_j^{k+1} \in \mathbf{B}$, then $\mathbf{l}_j^{k+1}$ is replaced with such better solution; otherwise, no changes are made to $\mathbf{l}_j^{k+1}$.

10.4 Template Matching (TM) Algorithm Based on the Locust Search (LS) Method

As illustrated in Sect. 10.2, a typical Template Matching (TM) method is able to find the accurate detection position (u, v) by computing a certain similarity metric over all valid pixel locations within a specific search region $\mathbf{S}$ of a given source image. However, such approach demands a high computational cost for practical use [14]. In an effort to overcome such issue, several TM algorithms [21–23] have been proposed to accelerate the search process by computing only a reduced subset of search locations. Although these methods allow a significant reduction on the

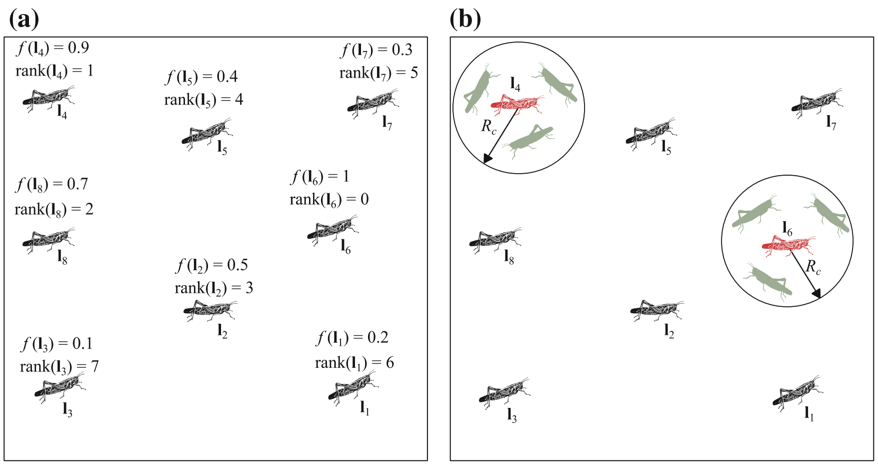


Fig. 10.4 LS social phase. **a** Initial locusts' configuration and rank according to their respective food quality indexes, and **b** social phase operator applied by considering $q = 2$ and $h = 3$

average TM's computational cost, they lack the ability to explore the whole search region **S** effectively and, as a result, they often suffer from premature convergence, which inevitably yields to sub-optimal solutions. Such problems are usually related to the operators that are used to modify the particles positions during the evolutionary process. Furthermore, in most of these methods, the position of each search agent for the next iteration is updated by considering an attraction toward the best individual seen-so-far [24]. Due to this particularity, the entire population has a tendency to concentrate around the best-known coincident location within the source image, which in turn, favors a premature convergence toward a possible local optima of the image's multi-modal surface.

Different to most recent TM algorithms, the unique operators implemented in the LS approach explicitly avoid the concentration of individuals over the current best solutions. This important trait allows the LS method to keep an appropriate balance between exploration and exploitation, and, as a result, avoid premature convergence [13]. In the proposed LS-based TM algorithm, individuals a represented by a pair of coordinated search positions (u, v) within the search space **S** of a given source image I. Each of such individuals moves though the search space **S** while looking for the optimal match position with respect to a given image template T. Furthermore, the NCC coefficient, which evaluates the matching quality between the image template T and the source image I, is computed for each individual's search position and used as a fitness value.

Formally speaking, the search space **S** consists of a set of 2-D discrete search positions u and v, each representing the horizontal and vertical components of the detection locations, respectively. With that being said, in the proposed LS-based TM approach, each individual (locust) is defined as follows:

$$\mathbf{l}_i = \{(u_i, v_i)|1 \leq u_i \leq M - m, 1 \leq v_i \leq N - n\} \tag{10.13}$$

where m and n, as illustrated in Sect. 10.2, stand for the horizontal and vertical sizes of the image template T, respectively. Furthermore, for each individual position $\mathbf{l}_i$ within the search space $\mathbf{S}$, a corresponding fitness value f_i is assigned by computing the NCC coefficient (see Eq. 10.1) with respect to their individual positions (u_i, v_i), such that:

$$f_i = NCC(u_i, v_i) \tag{10.14}$$

In the context of the LS method, the fitness value f_i represents the quality of the food source represented by a particular locust $\mathbf{l}_i$. Furthermore, as illustrated in Sect. 10.3, each individual $\mathbf{l}_i$ within a locust population $\mathbf{L}$ is ranked according to the quality of their respective food sources and then operated by the LS algorithm's solitary and social operators in order to update each individual's position as the process evolves.

Therefore, the proposed LS-TM approach may be summarized as follows:

Step 1 Read a gray-scale image I.
Step 2 Select an image template T.
Step 3 Initialize a set $\mathbf{L} = \{\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_N\}$ of N locust (particles) within the search space $\mathbf{S}$.
Step 4 Initialize global memory with the best individual $\mathbf{l}_i$ of the entire population $\mathbf{L}$.
Step 5 Evaluate the NCC coefficient (see Eq. 10.1) for each individual $\mathbf{l}_i$ within the entire population $\mathbf{L}$ and assign a rank to each of such individuals (see Sect. 10.3).
Step 6 Apply the LS algorithm's solitary operator (see Sect. 10.3.1).
Step 7 Apply the LS algorithm's social operator (see Sect. 10.3.2).
Step 8 Update global memory with the new best individual (matching position) $\mathbf{l}_i$ of $\mathbf{L}$.
Step 9 If the maximum number of iterations hasn't been reached, go to Step 5; otherwise, the process ends.

10.5 Experimental Setup and Results

In order to verify the feasibility and effectiveness of the proposed LS-TM approach, a set of comparative experiments against other state-of-the-art TM methods, which consider a set of several different image cases, were performed. In Appendix we illustrate the image dataset that was employed on our experimental setup. Furthermore, for each of such experiments, the evolution of the NCC coefficient's value is used to evaluate the performance of each of the compared TM methods, which include the PSO-TM algorithm [8], ABC-TM [15] and the ICA-TM approach [9].

Also, for each individual experiment, the maximum iteration number has been set to 500 iterations. Such stop criterion has been selected in order to keep compatibility to other similar works reported on the literature [9, 8].

The parameter setup for each of the compared TM approaches is described as follows:

1. PSO-TM: The algorithm's learning factors are set to $c_1 = 2$ and $c_2 = 2$; also, the inertia weight factor is set to decreases linearly from 0.9 to 0.2 as the process evolves [8].
2. ABC-TM: The algorithm was implemented by considering the guidelines provided by [15], with the parameter limit = 500.
3. ICA-TM: The parameter are set to:$N_{\text{countries}} = 100$, $N_{\text{Imper}} = 10$, $N_{\text{Colony}} = 90$, $T_{\max} = 500$, $\xi = 0.1$, $\varepsilon_1 = 0.15$, $\varepsilon_2 = 0.9$. Such values represent the best parameter setup for this approach, as illustrated by [9].
4. SMS-TM: This algorithm was implemented by considering the guidelines provided in [17]. The parameter setup for each individual stage of the SMS method are:

    a. Gas state: $\alpha = 0.8$, $\beta = 0.8$, $H = 0.9$, $\rho = [0.8, 1.0]$.
    b. Liquid state: $\alpha = 0.2$, $\beta = 0.4$, $H = 0.2$, $\rho = [0.3, 0.6]$.
    c. Solid state: $\alpha = 0.0$, $\beta = 0.1$, $H = 0.0$, $\rho = [0.0, 0.1]$.

5. LS-TM: Our proposed approach was implemented by considering the following parameter setup: $N = 50$, $q = 10$, and $h = 3$.

The parameter setup for each of the compared methods is kept with no modifications during all experimental work. The parameter configurations chosen for each of the compared methods were obtained though exhaustive experimentation over the proposed template matching approach, and as such, represent the best possible parameters found on our experiments. All experiments were performed on MatLAB® R2016a, running on a computer with an Intel® Core™ i7—3.40 GHz processor, and Windows 8 (64-bit, 8 GB of memory) as it operating system.

The results, corresponding to 30 individual runs for each compared method, are reported in Table 10.1. The comparisons are analyzed by considering the following performance indexes: (1) the Average-Best NCC value (AB_{NCC}), (2) the Median-Best NCC value (MB_{NCC}), (3) the Standard Deviation of the NCC value (SD_{NCC}), (4) the Average Number of Iterations in which the best match is found (NI) and (5) the Average Success Rate (SR) which represent the percent of all test runs in which a given image template was successfully detected. According to Table 10.1, for the proposed test images (see Appendix), LS-TM delivers significantly better results in comparison to those of PSO-TM, ICA-TM, ABC-TM and SMS-TM. Also, in Fig. 10.5, the NCC values evolution curves corresponding to each of the compared TM method are shown.

Furthermore, the non-parametric statistical significance proof known as the Wilcoxon's rank sum test for independent samples [25, 26] was also conducted. Such test is performed over the Average-Best NCC values (AB_{NCC}) provided by

Table 10.1 Performance comparison of PSO-TM, ICA-TM, ABC-TM, SMS-TM and the proposed method for the experimental set show in Appendix

Algorithms	Images	AB_{NCC}	MB_{NCC}	SD_{NCC}	NI	%SR
LS-TM	(a)	**9.989E−01**	**0.9989**	**1.00E−04**	**349**	**100**
	(b)	**9.985E−01**	**0.9985**	**3.83E−02**	**70**	**100**
	(c)	**9.978E−01**	**0.9978**	**1.38E−03**	**206**	**100**
	(d)	**9.990E−01**	**0.9990**	**7.66E−03**	**317**	**100**
	(e)	**9.991E−01**	**0.9991**	**7.87E−05**	**113**	**100**
	(f)	**9.998E−01**	**0.9998**	**5.37E−03**	**111**	**97**
PSO-TM	(a)	8.403E−01	0.8304	3.79E+03	397	92
	(b)	8.641E−01	0.8159	1.59E+01	451	96
	(c)	8.322E−01	0.8357	1.35E+04	430	93
	(d)	8.658E−01	0.8987	4.26E+00	464	95
	(e)	8.252E−01	0.8321	4.26E+04	465	91
	(f)	7.651E−01	0.7741	7.42E+02	489	88
ICA-TM	(a)	8.984E−01	0.8321	7.13E+02	489	95
	(b)	9.252E−01	0.9321	3.19E+01	412	98
	(c)	8.653E−01	0.8357	7.44E+00	432	96
	(d)	8.046E−01	0.8123	3.12E+01	496	97
	(e)	8.655E−01	0.8564	7.07E−01	431	94
	(f)	7.985E−01	0.7864	5.74E−01	395	90
ABC-TM	(a)	8.111E−01	0.8403	2.08E−01	367	96
	(b)	8.985E−01	0.8252	7.96E+02	361	98
	(c)	8.068E−01	0.8266	5.00E−01	254	96
	(d)	8.173E−01	0.8173	1.27E−02	310	95
	(e)	8.895E−01	0.8252	6.66E+01	398	96
	(f)	7.213E−01	0.7776	1.55E−01	420	92
SMS-TM	(a)	8.931E−01	0.8950	1.99E−01	380	98
	(b)	9.518E−01	0.9598	2.96E−02	331	98
	(c)	9.687E−01	0.9612	3.11E−01	360	98
	(d)	9.656E−01	0.9756	2.74E−02	456	98
	(e)	9.315E−01	0.9499	3.66E−01	339	98
	(f)	8.277E−01	0.8841	2.98E−01	135	95

Bold data represent the best results

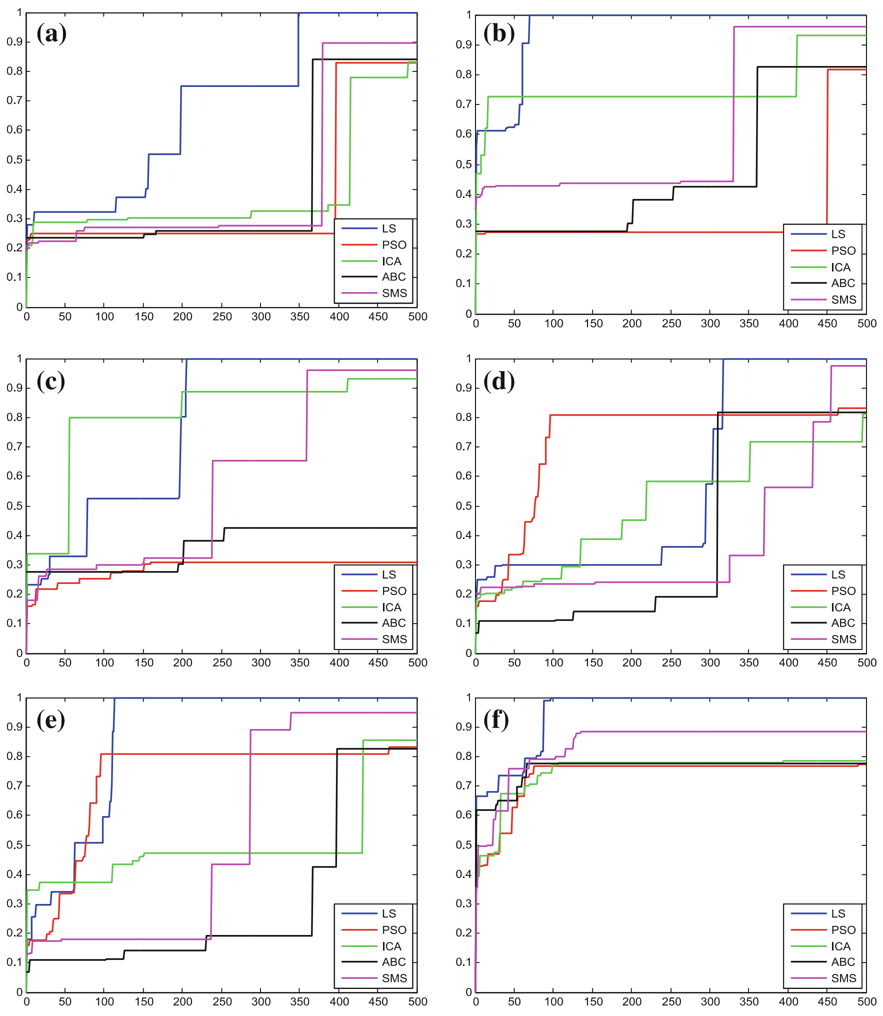


Fig. 10.5 Evolution curves for PSO-TM, ICA-TM, ABC-TM, SMS-TM and the proposed approach, by considering the image cases illustrated in Appendix

Table 10.1, by considering a 5% of significance level. Table 10.2 reports the p-values corresponding to the Wilcoxon's test pair-wise comparison between the AB_{NCC} values of three particular groups. Such groups are constituted by: (1) LS-TM versus PSO-TM, (2) LS-TM versus ABC-TM, (3) LS-TM versus ICA-TM, and (4) LS-TM versus SMS-TM. With respect to so called Wilcoxon's test significance level, a null hypothesis assumes that there is no significant difference between the $\mathrm{AB}_{\mathrm{NCC}}$ values of two different methods. On the other hand, an alternative hypothesis considers that there exist a significant difference between the $\mathrm{AB}_{\mathrm{NCC}}$ values of both of the compared approaches. The fact that all p-values reported on Table 10.2 are less than

Table 10.2 *p*-values corresponding to the Wilcoxon's test

Image cases	PSO-TM versus LS-TM	ICA-TM versus LS-TM	ABC-TM versus LS-TM	SMS-TM versus LS-TM
(a)	1.83E−04	1.73E−02	1.83E−04	7.65E−05
(b)	3.85E−01	1.83E−04	1.83E−04	6.37E−05
(c)	1.83E−04	6.23E−01	3.12E−02	1.98E−03
(d)	2.57E−02	5.21E−01	1.83E−04	7.46E−05
(e)	4.73E−01	1.83E−04	1.40E−01	6.88E−08
(f)	6.39E−05	2.15E−03	4.73E−01	2.69E−02

Table 10.3 Computational times (in seconds) for each of the compared template matching approaches

Image cases	LS-TM	PSO-TM	ICA-TM	ABC-TM	SMS-TM	Exhaustive-TM
(a)	13.309 seg.	3.950 seg.	5.951 seg.	14.01 seg.	4.038 seg.	25.719 seg.
(b)	10.034 seg.	3.385 seg.	5.739 seg.	13.92 seg.	4.063 seg.	22.547 seg.
(c)	12.345 seg.	3.995 seg.	5.152 seg.	15.54 seg.	5.109 seg.	26.107 seg.
(d)	12.672 seg.	3.813 seg.	5.913 seg.	17.98 seg.	4.287 seg.	25.013 seg.
(e)	11.806 seg.	3.546 seg.	5.164 seg.	13.29 seg.	4.333 seg.	107.123 seg.
(f)	11.225 seg.	4.225 seg.	6.866 seg.	22.31 seg.	5.984 seg.	138.682 seg.

0.05 (and thus, below the significance level value of 5%) provides a strong evidence against the null hypothesis. With that being said, it is clear that the results provided by the proposed LS-TM approach are not a product of coincidence, and as such, are statistically significant.

Finally, in Table 10.3, the computational times (in units of seconds) corresponding to each of the compared methods are shown. In addition to this, the computational times corresponding to a typical exhaustive search template matching approach [3] are also shown. As evidenced by such table, the PSO-TM approach takes much less time to find a solution in comparison to the other methods, whereas ABC-TM takes the most time (without accounting on the exhaustive search approach). Also, it could be appreciated that our proposed method (LS-TM) has similar computation times to those reported for the ABC-TM approach. While our proposed template matching approach is clearly outperformed by other methods in terms of computational time, it must be noted that, as evidenced by the data reported in Table 10.1, our proposed approach yields to significantly better results in terms of solution quality. Furthermore it is also worth noting that, in exchange for some accuracy, all of the compared approaches are easily able to outperform the classic exhaustive search approach in terms of computational time. This further evidence the viability of such approaches for solving such a complex problem.

10.6 Conclusions

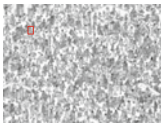
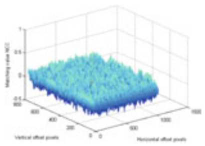
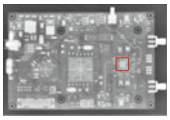
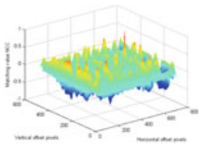
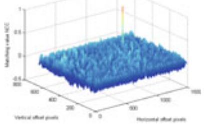
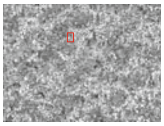
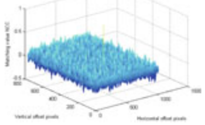
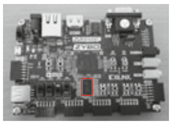
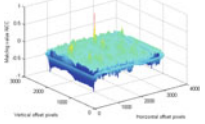
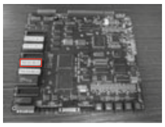
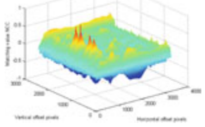
In this paper, the swarm optimization method known as Locust Search (LS) is applied to solve the problem of template matching (TM). The operators employed by LS during the exploration and exploitation of new solutions are devised to explicitly avoid the concentration of search agents around the best-known solutions, which in turn prevents several issues commonly found in other swarm optimization methods, such as those related to premature convergence.

In the proposed approach, individuals represent search positions within an image search region. The NCC coefficient (represented as a fitness value) is computed for each individual search agent, and then, used to evaluate the matching quality between a given image template and a coincident region of the source image. Guided by the LS unique evolutionary operators, individuals move through several positions within the source image while looking for the locations corresponding to an optimal degree of resemblance with respect to the image template. The exploration strategy employed by LS algorithm allows a better exploration of an image's search region, while considering only a limited number of search locations. This fact allows both, a more accurate detection and a significant reduction on the TM's computational cost.

The performance of the proposed approach has been compared to other state-of-the-art TM algorithms, by considering a set of several different image cases. Experimental results show that the LS-TM approach yields to a high performance in terms of both, precision and computational cost.

Appendix

Images dataset used on our experimental setup. **a** Waldo 1, **b** Circuit board 1, **c** Satellite image, **d** Waldo 2, **e** Circuit board 2, and **f** Circuit board 3.

	Image	Template	NCC surface	Image resolution
(a)				768 × 1024
(b)				474 × 700
(c)				641 × 1483
(d)				768 × 1024
(e)				2137 × 3027
(f)				2448 × 3264

References

1. Brunelli, R.: Template matching techniques in computer vision: theory and practice. Wiley, London (2009). ISBN 978-0-470-51706-2
2. Hadi, G., Mojtaba, L., Hadi, S.Y.: An improved pattern matching technique for lossy/lossless compression of binary printed Farsi and Arabic textual images. Int. J. Intell. Comput. Cybern **2**(1), 120–147 (2009)

3. Krattenthaler, W., Mayer, K.J., Zeiler, M.: Point correlation: a reduced-cost template matching technique. In: Proceedings of the First IEEE International Conference on Image Processing, pp. 208–212 (1994)
4. Rosenfeld, A., VanderBrug, G.J.: Coarse-fine template matching. IEEE Trans. Syst. Man Cybern. SMC-**7**(2), 104–107 (1977)
5. Tanimoto, S.L.: Template matching in pyramids. Comput. Vision Graph. Image Proc. **16**(4), 356–369 (1981)
6. Dong, N., Wu, C.-H., Ip, W.-H., Chen, Z.-Q., Chan, C.-Y., Yung, K.-L.: An improved species based genetic algorithm and its application in multiple template matching for embroidered pattern inspection. Expert Syst. Appl. **38**, 15172–15182 (2011)
7. Mitchell, M.: An Introduction to Genetic Algorithms. Cambridge, MA (1996)
8. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1248 (1995)
9. Haibin, D., Chunfang, X., Senqi, L., Shan, S.: Template matching using chaotic imperialist competitive algorithm. Pattern Recogn. Lett. **31**, 1868–1875 (2010)
10. Chen, G., Low, C.P., Yang, Z.: Preserving and exploiting genetic diversity in evolutionary programming algorithms. IEEE Trans. Evol. Comput. **13**(3), 661–673 (2009)
11. Adra, S.F., Fleming, P.J.: Diversity management in evolutionary many-objective optimization. IEEE Trans. Evol. Comput. **15**(2), 183–195 (2011)
12. Tan, K.C., Chiam, S.C., Mamun, A.A., Goh, C.K.: Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization. Eur. J. Oper. Res. **197**, 701–713 (2009)
13. Cuevas, E., González, A., Zaldívar, D., Pérez-Cisneros, M.: An optimisation algorithm based on the behaviour of locust swarms. Int. J. Bio-Inspir. Comput. **7**(6), 402–407 (2015)
14. Sadoghi Yazdi, H.: An improved pattern matching technique for lossy/lossless compression of binary printed farsi and arabic textual images. Int. J. Intell. Comput. Cybern., 120–147 (2009)
15. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report-TR06. Engine, Engineering Faculty, Computer Engineering Department, Erciyes University (2005)
16. Hossein, A., Hossein-Alavi, A.: Krill herd: a new bio-inspired optimization algorithm. Commun. Nonlinear Sci. Numer. Simul. **17**, 4831–4845 (2012)
17. Cuevas, E., Echavarría, A., Zaldívar, D., Pérez-Cisneros, M.: A novel evolutionary algorithm inspired by the states of matter for template matching. Expert Syst. Appl. **40**, 6359 (2013)
18. Topaz, C.M., Bernoff, A.J., Logan, S., Toolson, W.: A model for rolling swarms of locusts. Eur. Phys. J. Spec. Top. **157**, 93–109 (2008)
19. Topaz, C.M., D'Orsogna, M.R., Edelstein-Keshet, L., Bernoff, A.J.: Locust dynamics: behavioral phase change and swarming. Plos Comput. Biol. **8**(8), 1–11
20. Cuevas, E., González, A., Fausto, F., Zaldívar, D., Pérez-Cisneros, M.: Multithreshold segmentation by using an algorithm based on the behavior of locust swarms. Math. Probl. Eng. **2015**, 25 (2015). Article ID 805357. https://doi.org/10.1155/2015/805357
21. Dong, N., Wu, C.-H., Ip, W.-H., Chen, Z.-Q., Chan, C.-Y., Yung, K.-L.: An improved species based genetic algorithm and its application in multiple template matching for embroidered pattern inspection. Expert Syst. Appl. **38**(12), 15172–15182 (2011)
22. Liu, F., Duan, H., Deng, Y.: A chaotic quantum-behaved particle swarm optimization based on lateral inhibition for image matching. Optik-Int. J. Light Electron Opt. **123**(21), 1955–1960 (2012)
23. Wu, C.-H., Wang, D.-Z., Ip, A., Wang, D.-W., Chan, C.-Y., Wang, H.-F.: A particle swarm optimization approach for components placement inspection on printed circuit boards. J. Intell. Manuf. **20**(5), 535–549 (2009)
24. Chen, G., Low, C.P., Yang, Z.: Preserving and exploiting genetic diversity in evolutionary programming algorithms. IEEE Trans. Evol. Comput. **13**(3), 661–673 (2009)
25. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics **1**, 80–83 (1945)
26. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of nonparametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special session on real parameter optimization. J. Heuristics (2008)