Studies in Computational Intelligence 927

Modestus O. Okwu
Lagouge K. Tartibu

Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications

Springer

Studies in Computational Intelligence

Volume 927

Series Editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland

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

Indexed by SCOPUS, DBLP, WTI Frankfurt eG, zbMATH, SCImago.

All books published in the series are submitted for consideration in Web of Science.

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

Modestus O. Okwu · Lagouge K. Tartibu

Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications

Springer

Modestus O. Okwu
Department of Mechanical and Industrial Engineering Technology
University of Johannesburg
Auckland Park, South Africa

Lagouge K. Tartibu
Department of Mechanical and Industrial Engineering Technology
University of Johannesburg
Auckland Park, South Africa

ISSN 1860-949X ISSN 1860-9503 (electronic)
Studies in Computational Intelligence
ISBN 978-3-030-61110-1 ISBN 978-3-030-61111-8 (eBook)
https://doi.org/10.1007/978-3-030-61111-8

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Metaheuristics are global optimization techniques for solving real-life complex problems. The classical techniques for solving such realistic day-to-day problems is time-consuming and will not proffer an exact or feasible solution. Consequently, there is a need to shift from classical techniques to standard metaheuristic techniques inspired by nature. Metaheuristic techniques are capable of using search experience intelligently to explore and exploit the search space in a randomized manner, and the solution methods are inexact and near optimal. Nature-inspired algorithms swarm and computational intelligence with theory and applications are open text and resource designed for undergraduates, postgraduates and researchers in diverse field of studies like Professional and Applied Sciences (engineering and technology, military sciences, transportation, environmental studies, business, agriculture, consumer science and others); Natural Sciences (biology, chemistry, earth sciences, physics, space sciences and others); Formal Sciences (computer sciences, pure and applied mathematics, statistics and others); Economics; Humanities and Social Sciences. This book will guide students and research fellows who are developing problem-solving skills related to transportation and trans-shipment; planning and scheduling in product- and service-based systems like retail outlets, banking, airline industry and more; inbound and outbound supply chain management in industries; inventory management; maintenance management; warehousing; cost and quality control and others. It provides a guide to a better understanding of the development of nature-inspired algorithms and resolution of real-life complex stochastic problems as well as numerical problems. The organization of the book is as follows: Chapter 1—General Introduction to optimization—gives classical search techniques and limitations, metaheuristic techniques, nature-inspired algorithms and swarm intelligence. Chapter 2 discusses detailed information on particle swarm optimization and the swarming habit or behaviour of creatures, animals, or insects and the application in the field of evolutionary computation and application of PSO in numerical optimization. Chapter 3 provides an introduction to artificial bee colony algorithm, foraging behaviour and waggle

dance of honeybees while passing information about a given food source to the rest of the bee colony, such as the food source consistency, direction and distance from the nest, mathematical modelling of the bee algorithm and real-life application of the bee algorithm in fast moving grocery retailer system or big box facility. Chapter 4 gives detailed information related to the ant colony algorithm, with broad information on path navigation and behaviour of ants to solve the optimization problem, ants foraging pattern by depositing pheromones along the path taken until they get to the desired destination, especially during exploration for food source and model development with solved examples. Chapter 5 provides detailed information on grey wolf optimization (GWO) algorithm. GWO algorithm takes into consideration the motivation from the hunting pattern of the grey wolves, with solved numerical example. Chapter 6 gives an introduction to whale optimization algorithm (WOA). Feeding pattern of the whale which consist of strategically encircling prey. The humpback whale is considered in this context. Algorithm formation process is illustrated and solved with a numerical example. Chapter 7 gives an introduction to firefly algorithm. It describes converging nature of the firefly, especially where they find brighter light. How the less bright fireflies converge or swarm towards fireflies with brighter light. Rules for developing the firefly algorithm, advantages and disadvantages of firefly algorithm, area of application of fire fly algorithm and solved numerical problem are provided. Chapter 8 gives an introduction to bat algorithm. The communication and navigational pattern of bats and micro-bats echolocation (EL) are described. The great strength of a bat linked to the appealing wave of sound often exhibited while searching for prey. Algorithm development and solved numerical and real-life example are used to illustrate the approach. Chapter 9—Ant lion optimization algorithm—describes dangerous hunting technique of ant lion by digging holes in a cone-like shape while the insect goes down to the bottom of the hole and places its jaw at that end to ensure proper eating of prey falling into the trap. Model equation of the ant lion algorithm and solved examples are used to illustrate the approach. Chapter 10—Grasshopper optimization algorithm—describes important feature of grasshoppers, the search process of grasshopper during quest for food using exploration and exploitation procedure. Grasshopper algorithm for solving a real-life problem and numerical examples are used to illustrate the approach. Chapter 11 provides an introduction to butterfly optimization algorithm and power of fragrance and route-finding pattern of butterflies, and step-by-step mathematical model and application with numerical example are provided. Chapter 12 gives an introduction to moth flame optimization algorithm, in which steps for generating moths randomly within the neighbourhood or solution space and solved numerical example are provided. Chapter 13 presents detailed information on genetic algorithm and its application to real-life problem. Chapter 14 gives an introduction to artificial neural network (ANN). ANN is inspired by the pattern in which the biological nervous system (brain) process information and by mimicking how information flow through a set of interconnected neurons that work together to solve a problem. General mathematical model

and application of ANN model to open-pit mine vibration prediction are provided. Chapter 15 is the final chapter which is focused on the future of nature-inspired algorithm, swarm and computational intelligence, other area of application of nature-inspired algorithms, other algorithms inspired by nature and conclusion.

Auckland Park, South Africa

Modestus O. Okwu
Lagouge K. Tartibu

Contents

Chapter 1
Introduction to Optimization

Systems, be it manufacturing, product or service-based are faced with competition at every level as a result of limited resources. Competition is certain in systems with inadequate resources in all sectors of the economy such as agriculture, construction, oil and gas, product and service industries, other subdivisions are all subjected to limitations of available resources. Struggle for scarce resources has led to the implementation of strategic ways to change the status quo at hand. This investigation has led to the process referred to as an *optimization process* which is very necessary to address the issue of limited resources in product and service-based systems. In search for optimality, there are goals to be achieved and constraints to be satisfied [1, 2], the set goal is the objective function which vary from one form of problem to another [3, 4]. It is important to note that the goal of optimization in any system of constraints is to find values for a parameter that will optimize an established objective. The classical approach or local heuristic search can be used in search of a feasible or near-optimal solution. Recently, the development of metaheuristic techniques has made the approach much easier, reliable and faster.

Optimization problems can be categorized into two types based on the solutions produced [5], namely the classical or deterministic algorithms and the non-classical or non-deterministic also called the stochastic algorithms. The stochastic algorithm can be further classified into the heuristic and metaheuristic techniques.

1.1 Classical or Conventional Search Techniques

This search technique can be categorized into two types. The first is the *Gradient-based algorithms* [6] and the second class is the *Direct search algorithms* [7]. Gradient-based algorithm (GBA) are required when the objective function of a system of study is differentiable and continuous. In the GBA method, differential calculus is

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_1

employed while searching for an optimal solution to a problem. A direct search algorithm (DSA), unlike the GBA, is usually implemented in an optimization problem in conditions where the gradient tends to be unattainable as a result of non-differentiable or partially discontinuous objective function.

1.2 Limitations of Classical Search Methods

The use of classical or traditional techniques when addressing the problem of optimization, comes with several challenges. It usually depends on the nature of the problem at hand. For instance, challenges are encountered with problems involving quite a few local optimal solutions, discontinuous problems, challenges associated with a change in optimal solution as a result of system evaluation at different times, and constraints due to search space. There are often cases of classical search techniques with problems having a large "undulating" exploration or search space. Such limitations make classical techniques unable to explore all possible candidate solutions.

1.3 Metaheuristics

Trial and error have been used in most problem-solving techniques in time past, which is often regarded as heuristic technique(s) or short form of problem solving. In the field of Engineering, Science and mathematical optimization, metaheuristics are referred to as a high-level heuristic technique that is designed to provide an optimization problem with a sufficiently good solution, particularly with spick-and-span information containing limitless computing power which cannot be resolved by conventional means. Metaheuristics as a high-level technique cannot ascertain that global optimal solution can be obtained on any problem space. This is because most of the high level global techniques are presented as stochastic optimizations, meaning that the solution obtained is highly dependent on some type of generated random variables. Metaheuristics can also find good solutions in combinatorial optimization, with little or no computational effort in optimization algorithms, with a wide range of feasible solutions. They are considered useful approaches for optimization problems. In other words, the term metaheuristics can equally be referred to as a global search technique or all modern nature-inspired optimization algorithms, they are useful for a variety of problems. The algorithms are highly effective and have general applications with high performance and great driving force for optimality. In some cases, metaheuristic techniques are regarded as stochastic population-based or fuzzy search techniques which, when compared to the direct search algorithms or classical gradient-based techniques, provide a robust or high-level solution.

These are higher level or classes of algorithms developed to explore or search for practically good solutions to optimization problems by combining diverse concepts, perceptions, and ideas to balance exploitation and exploration of the search process. Exploration is also referred to as the divergence of search space, which has to do with searching for new possibilities by exploring the search space maximally. Exploitation, on the other hand, has to do with strengthening the search space, that is, the possibility of exploring further to find a better solution in a neighbourhood.

In soft computing and computational intelligence literature, good examples have been presented using biomimicry and nature-inspired systems and algorithms. Metaheuristic algorithms have equally been used by academic researchers and scholars in diverse fields to identify empirical findings based on analysis using the high-level techniques called from mathematics laboratory on computers. Many articles have been published without practical effectiveness and lack of conceptual elaboration. This book titled Metaheuristic Optimization: "Nature Inspired Algorithms, Swarm and computational Intelligence", focus on mimicking nature to solve real-life problems. Different systems inspired by nature has been identified and the behaviour of such systems explained in details and mathematical models of different systems have been developed and some example was used to illustrate the implementation of the approaches described. The codes for the listed algorithms are provided in the appendix section of the book.

1.4 Swarm Intelligence-Based Algorithms (SIA)

The use of classical search methods has never provided the best possible solutions especially while dealing with complex stochastic problems. Classical techniques have lower expectations in a dynamic system of the contemporary revolution. There are a lot of limitations of classical or traditional search techniques already discussed. The remedy lies in the use of global search techniques or metaheuristic techniques. Swarm intelligence-based Algorithm are specialized kind of biological based, nature inspired algorithms fashioned on the knowledge swarming behaviours of entities, especially insects and other animals known for gathering or forming of clusters or colonies. Scientists and researchers are very much interested in mimicking the behaviour of these intelligent entities to form collective intelligence by formulating step by step mathematical equations or building advanced algorithms to solve real life problems. Examples described in this text include: Bee Algorithm, Ant Colony Algorithm, Butterfly Algorithm, Grasshopper Algorithm, Ant Lion Algorithm and others.

References

1. Faturechi, R., and E. Miller-Hooks. 2014. A mathematical framework for quantifying and optimizing protective actions for civil infrastructure systems. *Computer-Aided Civil and Infrastructure Engineering* 29 (8): 572–589.
2. Aldwaik, M., and H. Adeli. 2014. Advances in optimization of high rise building structures. *Structural and Multidisciplinary Optimization* 50 (6): 899–919.
3. Gao, H., and X. Zhang. 2013. A Markov-based road maintenance optimization model considering user costs. *Computer-Aided Civil and Infrastructure Engineering* 28 (6): 451–464.
4. Zhang, G., and Y. Wang. 2013. Optimizing coordinated ramp metering—A preemptive hierarchical control approach. *Computer-Aided Civil and Infrastructure Engineering* 28 (1): 22–37.
5. Yang, X.-S. 2010. *Engineering optimisation: An introduction with metaheuristic application.* New York: Wiley.
6. Creswell, J. W. 2013. *Research design: Qualitative, quantitative, and mixed methods approaches*, 4th ed. Sage Publications.
7. Kolda, T., R. Lewis, and V. Torczon. 2003. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review* 45 (3): 385–482.

Chapter 2
Particle Swarm Optimisation

2.1 Introduction

Particle Swarm Optimization (PSO) is a robust method of stochastic optimization developed by Kennedy and Eberhart in 1995 [1]. It is used to analyse the search space of a problem to find the structure or parameters necessary for optimizing a critical target or defined objective. PSO originates from the idea of swarm intelligence by looking deep into nature to develop algorithms for solving real-life stochastic problems. Experts looked at the swarming habit or behaviour of creatures, animals, or insects and the application in the field of evolutionary computation. The PSO algorithm is gaining popularity in the field of computational intelligence and it has been successfully applied to a variety of optimization and search problems. PSO is one of the most widely used and highly regarded algorithms in the literature of computational intelligence, metaheuristics, and optimization. The algorithm has been broadly used in a variety of fields like science and engineering [2].

2.2 Inspiration

PSO was built by abstracting the working mechanism of the natural phenomenon through mimicking the navigation pattern and foraging swarms in nature, such as flock of birds or school of fishes and many more nature-inspired swarms [3]. The description of PSO is such that, group of individuals or particles interconnect, link together, interact and communicate reliably or meanderingly with one another using search directions or gradients. The PSO algorithm uses established particles that wing over a search space for global optima location. Throughout the PSO iteration process, each particle updates its location based on the preceding knowledge or experience as well as the knowledge obtained from the neighbourhood search. That means looking

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_2

at a given particle bird, bat, or fish, they maintain position and learn from experience encountered when they navigate as a flock of bats, swarm of birds or school of fishes. The pattern of navigation of the particles is of importance as communication is vital during the navigational process. Feedback is received from both the local and global best during the global best search process.

2.3 Three Vectors Associated with Particle Swarm Optimization (PSO)

According to [1], there are three vectors associated with Particle swarm optimization:

(i) The first vector is the vector which documents the location of particles, often referred to as the X-Vector.
(ii) The second vector is the vector known for documentation of the search for the best solution introduced by individual or particle, often referred to as the P-Vector.
(iii) The third vector is the vector which maintains the direction or gradient for navigation of particles in an undisturbed manner.

The velocity vector is referred to as the position vector which helps in the search for the best solution. For instance, considering six (6) particles (P1–P6) having different velocities. These particles move in search of a solution. There are global best solutions and local best. Particles try as much as possible to move from the current position towards the local best or global best position for the entire group promptly. The flow chart of the Particle Swarm Optimization algorithm (PSO) is illustrated in Fig. 2.1.

PSO is no doubt a novel population-based putative search algorithm and a preference solution to the complex stochastic and non-linear optimization problem. As earlier mentioned, the PSO algorithm erudite from animals' activity or behaviour to solve optimization problems. Every member of the population of PSOs based on navigation dynamics is referred to as the swarm process. Originating with a randomly initialized population and traveling in a randomly chosen direction, each particle moves through the penetrating space and reminds itself and its neighbors of the best previous positions. Particle of a swarm confabulates good positions as well as acclimatize dynamically at any location and velocity resulting from the best position of all particles.

The next step begins when all particles tend to fly towards better positions over the searching mechanism until the swarm moves closer to an optimum of the fitness function. A fitness value is assigned to every particle which is a function of the target function and the destination of the arrangement is determined by the swiftness of the system. In search for the best position, particles explore the solution space, and the locations of the best particle in the swarm. In most cases, optimum solutions are located by persistent searching. From a group of random particles, the optimum

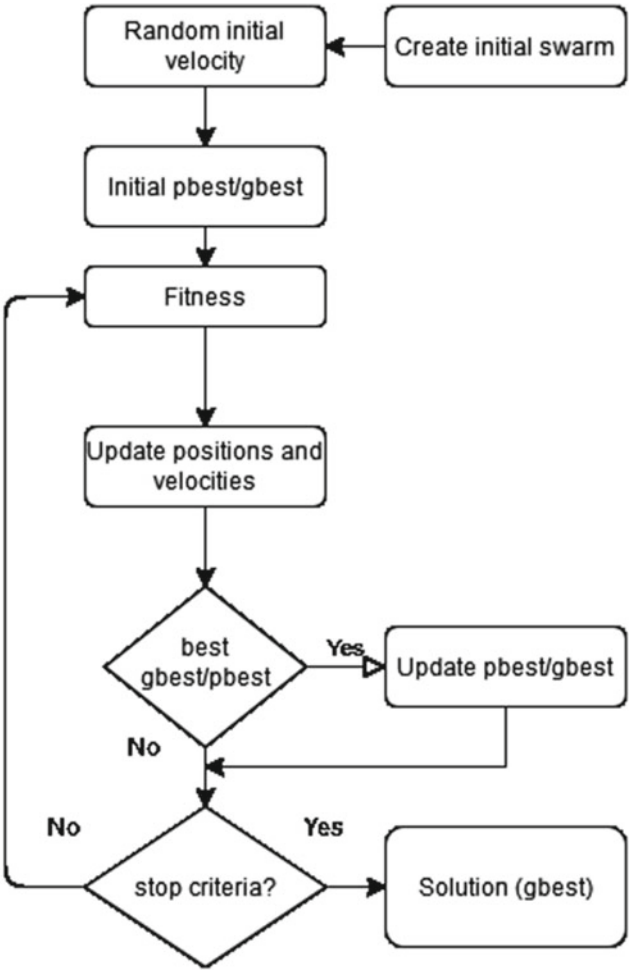


Fig. 2.1 PSO algorithm flowchart

solutions are located during the search through an iterative process. In every iteration, the best position pbest and gbest [4]. The following equations summarize the iteration process:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.1}$$

where x_i^t and x_i^{t+1} are the positions of particles in the search space at time step t and $t + 1$;

v_i^{t+1} is the velocity vector of particle I at time step $t + 1$.

$$v_i^{t+1} = v_i^t + c_1 r_1^t \left[P_{best,i}^t - x_i^t\right] + c_2 r_2^t \left[G_{best,i}^t - x_i^t\right] \tag{2.2}$$

where v_i^t is the velocity vector of particle I;

$G_{best,i}^t$ and $P_{best,i}^t$ are the global and local best position;

c_1 and c_2 are the positive acceleration constants;

r_1^t and r_2^t are random numbers between [0, 1].

In other to illustrate PSO optimization, a numerical example was used to describe the iteration as describe in the nest section.

2.4 Application of PSO for Numerical Optimization

$$\text{Minimize } f(x) = 5(x_1 - 1)^2 + 10\, x_1 x_2 + (x_3 - 3)^2$$

Subject to

$$x_1 + x_2 + x_3 \leq 5$$
$$x_1^2 + x_2^2 - 5 \leq 0$$
$$-5 \leq x_1 \leq 5$$
$$-5 \leq x_2 \leq 5$$
$$-5 \leq x_3 \leq 5$$

Population size: 5

$$c_1 = c_2 = 1$$

Maximum number of iterations: 20

Dimensions of the problem: 3.

Solution

Initialize the initial position (x_i^t) and velocity (v_i^t);

The initial velocities (v_i^t) of each particle are set to zero;

We initialize the position of 5 particles randomly in the range (−5, 5), (−5, 5) and (−5, 5) corresponding to x_1, x_2 and x_3 respectively.

x_i^t	$x_{11} = -4$	$x_{12} = -2$	$x_{13} = 2$
	$x_{21} = 4$	$x_{22} = 3$	$x_{23} = -1$
	$x_{31} = 3$	$x_{32} = -5$	$x_{33} = -4$
	$x_{41} = 3.5$	$x_{42} = -2$	$x_{43} = 3$
	$x_{51} = -5$	$x_{52} = 3.5$	$x_{53} = 2.5$

The corresponding initial fitness function values are:

$$f_i^t = [206 \quad 181 \quad -81 \quad -38.75 \quad 5.25]$$

Minimum fitness value and corresponding solutions are -81 and [3 -5 -4] respectively. This solution is called **gbest**. Also, since no previous iteration exists, this initial particle $P^t_{best} = x^t_i$.

a. **Iteration 1**

P^t_{best}	-4	-2	2
	4	3	-1
	3	-5	-4
	3.5	-2	3
	-5	3.5	2.5

Consider the random numbers in the range (0, 1) as $r^1_1 = 0.25$ and $r^1_2 = 0.85$ and find the velocities of the particles as follows

$$v^{t+1}_i = v^t_i + c_1 r^t_1 \left[P^t_{best,i} - x^t_i\right] + c_2 r^t_2 \left[G^t_{best,i} - x^t_i\right]$$

v^{t+1}_i	$v_{11} = 5.95$	$v_{12} = -2.55$	$v_{13} = -5.1$
	$v_{21} = -0.85$	$v_{22} = -6.8$	$v_{23} = -2.55$
	$v_{31} = 0$	$v_{32} = 0$	$v_{33} = 0$
	$v_{41} = -0.425$	$v_{42} = -2.55$	$v_{43} = -5.95$
	$v_{51} = 6.8$	$v_{52} = -7.225$	$v_{53} = -5.525$

The new values of x^{t+1}_i can be obtained as follows:

$$x^{t+1}_i = x^t_i + v^{t+1}_i$$

Hence the updated swarm's are:

x^{t+1}_i	$x_{11} = 1.95$	$x_{12} = -4.55$	$x_{13} = -3.1$
	$x_{21} = 3.15$	$x_{22} = -3.8$	$x_{23} = -3.55$
	$x_{31} = 3$	$x_{32} = -5$	$x_{33} = -4$
	$x_{41} = 3.075$	$x_{42} = -4.55$	$x_{43} = -2.95$
	$x_{51} = 1.8$	$x_{52} = -3.725$	$x_{53} = -3.025$

The corresponding initial fitness vectors are:

$$f_i^{t+1} = [-47.0025 \quad -53.685 \quad -81 \quad -82.9819 \quad -27.5494]$$

The minimum is −82.9819 and its position is x_4. Therefore, **gbest** $= x_4 = (3.075 \; -4.55 \; -2.95)$.

Now, compare this **gbest** $(=x_4)$ with previous **gbest** $(=x_3)$. We concluded that the updated **gbest** is better than the previous one as the fitness value at the new **gbest** is better than the previous one. So we replace old **gbest** $(=x_3)$ with new **gbest** $(=x_4)$. Therefore, new **gbest** $= x_4 = (3.075 \; -4.55 \; -2.95)$.

In case the new updated **gbest** is not better than the previous **gbest;** then carry forward with old **gbest** for the new iteration.

For the first particle:

- Fitness in previous swarm = 206;
- Fitness in current swarm = −47.0025;

b. **Iteration 2**

Since fitness in current swarm < fitness in previous swarm, we set **Pbest1** $= x_1 = (1.95 \; -4.55 \; -3.1)$. A similar comparison has been made for the other particles. In summary, we obtain:

P_{best}^{t+1}	1.95	−4.55	−3.1
	3.15	−3.8	−3.55
	3	−5	−4
	3.075	−4.55	−2.95
	1.8	−3.725	−3.025

Considering the random numbers in the range (0, 1) as $r_1^2 = 0.3$ and $r_2^2 = 0.7$. The velocities of the particles are given as follows:

v_i^{t+2}	$v_{11} = 6.7375$	$v_{12} = -2.55$	$v_{13} = -4.995$
	$v_{21} = -0.9025$	$v_{22} = -7.325$	$v_{23} = -2.13$
	$v_{31} = 0.0525$	$v_{32} = 0.315$	$v_{33} = 0.735$
	$v_{41} = -0.425$	$v_{42} = -2.55$	$v_{43} = -5.95$
	$v_{51} = 7.6925$	$v_{52} = -7.8025$	$v_{53} = -5.4725$

Hence the updated swarm's are:

x_i^{t+2}	$x_{11} = 8.6875$ ℂ (5, −5) thus $x_{11} = 5$	$x_{12} = -7.1$ ℂ (5, −5) thus $x_{12} = -5$	$x_{13} = -8.095$ ℂ (5, −5) thus $x_{13} = -5$
	$x_{21} = 2.2475$	$x_{22} = -11.125$ ℂ (5, −5) thus $x_{22} = -5$	$x_{23} = -5.68$ ℂ (5, −5) thus $x_{13} = -5$

(continued)

(continued)

	$x_{31} = 3.0525$	$x_{32} = -4.685$	$x_{33} = -3.265$
	$x_{41} = 2.65$	$x_{42} = -7.1$ ℂ (5, −5) thus $x_{42} = -5$	$x_{43} = -8.9$ ℂ (5, −5) thus $x_{43} = -5$
	$x_{51} = 9.4925$ ℂ (5, −5) thus $x_{51} = 5$	$x_{52} = -11.5275$ ℂ (5, −5) thus $x_{52} = -5$	$x_{53} = -8.4975$ ℂ (5, −5) thus $x_{11} = -5$

The corresponding initial fitness vector are:

$$f_i^{t+2} = [-106 \quad -40.5937 \quad -82.6956 \quad -54.8875 \quad -106]$$

The minimum is −106 and their positions are x_1 and x_5. Therefore, **gbest** $= x_1 = x_5 = (5\ -5\ -5)$. This new updated **gbest** is better than the previous ones. Therefore, new **gbest** $= x_1 = x_5 = (5\ -5\ -5)$. Since fitness in current swarm < fitness in previous swarm, we set **Pbest1** $= x_1 = x_5 = (5\ -5\ -5)$.

c. **Iteration 3**

After comparing the fitness of each particle with the values calculated in the previous iteration, we obtain:

P_{best}^{t+3}	5	−5	−5
	3.15	−3.8	−3.55
	3.0525	−4.685	−3.265
	3.075	−4.55	−2.95
	5	−5	−5

Considering the random numbers in the range (0, 1) as $r_1^3 = 0.15$ and $r_2^3 = 0.9$, the velocities of the particles are calculated.

v_i^{t+3}	$v_{11} = 6.7375$	$v_{12} = -2.55$	$v_{13} = -4.995$
	$v_{21} = 1.295$	$v_{22} = -6.965$	$v_{23} = -1.695$
	$v_{31} = 1.41575$	$v_{32} = 0.0945$	$v_{33} = -0.4795$
	$v_{41} = 1.3475$	$v_{42} = -2.415$	$v_{43} = -5.335$
	$v_{51} = 7.6925$	$v_{52} = -7.8025$	$v_{53} = -5.4725$

Hence the updated swarm's are:

x_i^{t+3}	$x_{11} = 11.7375$ ℂ (5, −5) thus $x_{11} = 5$	$x_{12} = -7.55$ ℂ (5, −5) thus $x_{12} = -5$	$x_{13} = -9.995$ ℂ (5, −5) thus $x_{13} = -5$
	$x_{21} = 3.5425$	$x_{22} = -11.965$ ℂ (5, −5) thus $x_{22} = -5$	$x_{23} = -6.695$ ℂ (5, −5) thus $x_{13} = -5$
	$x_{31} = 4.46825$	$x_{32} = -4.5905$	$x_{33} = -3.7445$

(continued)

(continued)

	$x_{41} = 3.9975$	$x_{42} = -7.415$ ℂ $(5, -5)$ thus $x_{42} = -5$	$x_{43} = -10.335$ ℂ $(5, -5)$ thus $x_{43} = -5$
	$x_{51} = 12.6925$ ℂ $(5, -5)$ thus $x_{51} = 5$	$x_{52} = -12.8025$ ℂ $(5, -5)$ thus $x_{52} = -5$	$x_{53} = -10.4725$ ℂ $(5, -5)$ thus $x_{11} = -5$

The corresponding initial fitness vector are:

$$f_i^{t+2} = [-106 \quad -80.8035 \quad -99.4829 \quad -90.95 \quad -106]$$

The iteration process will continue until all particles meet a single value. The approach described in this example was implemented in the software MATLAB. The PSO codes are provided in Appendix A and adapted from Ref. [5]. The best fitness function value obtained with the PSO model was -170 corresponding to the updated **gbest** $(5\ -5\ 5)$. The PSO convergence characteristic is shown in Fig. 2.2.

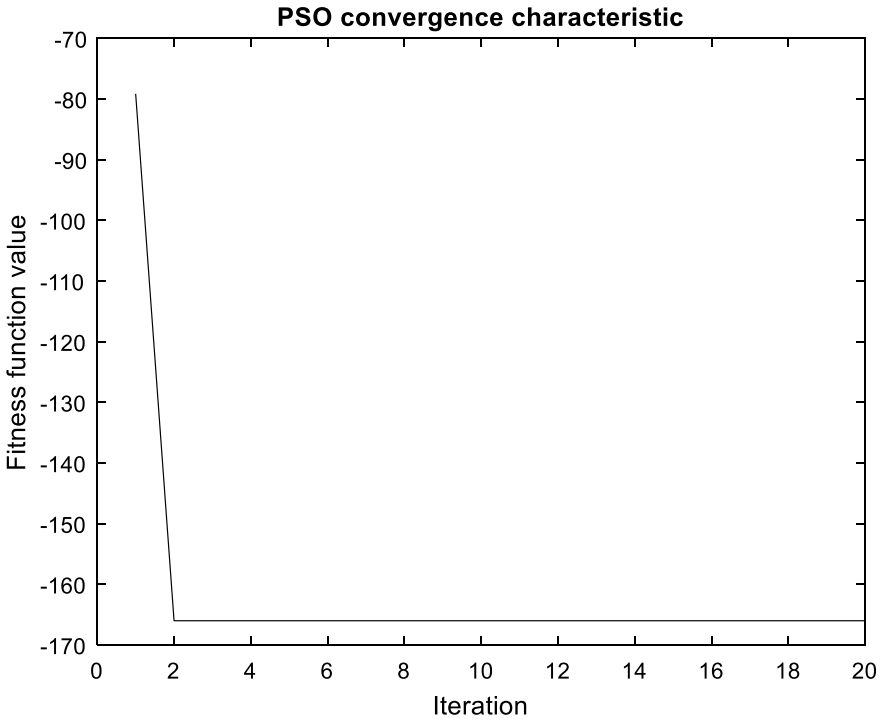


Fig. 2.2 Variation of fitness function as a function of the number of iteration

References

1. Kennedy, J., and R.C. Eberhart. 1995. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Perth, Australia, 1942–1948. Piscataway, NJ: IEEE Service Center.
2. Machesa, M.G., L.K. Tartibu, F.K. Tekweme, and M.O. Okwu. 2019. Evaluation of the Stirling heat engine performance prediction using ANN-PSO and ANFIS models. In *2019 6th International Conference on Soft Computing and Machine Intelligence (ISCMI)*, 19 Nov 2019, 217–222. IEEE. https://doi.org/10.1109/ISCMI47871.2019.9004406.
3. Kennedy, J., and R. Eberhart. 2001. *Swarm intelligence*. San Francisco, USA: Morgan Kaufmann.
4. Poli, R., J. Kennedy, and T. Blackwell. 2007. Particle swarm optimization: An overview. *Swarm Intelligence* 1 (1): 33–57. https://doi.org/10.1007/s11721-007-0002-0.
5. Alam, M.N. 2016. Particle swarm optimization: Algorithm and its codes in matlab. *ResearchGate* 8: 1–10.

Chapter 3
Artificial Bee Colony Algorithm

3.1 Introduction

The Artificial Bee Colony (or ABC) algorithm falls under the class of optimization methodology referred to as the Swarm optimization algorithm. Swarm Algorithms are inspired to look for optimal solutions by the behavioral nature of swarms. Some advantages of bee algorithms include the ease of use and implementation; ability to integrate with other algorithms and increase the performance of solved optimization problems; ability to perform a global and local search and possibility to solve a variety of optimization issues. The drawbacks include parameter adjustment, time, and accuracy required fur tuning as well as the need for multiple and random initialization. A crucial difference in Swarm-based Optimization algorithms' characteristics is their use of a whole population of solutions rather than a single solution [1]. Through swarm intelligence, multiple optimal solutions can also be obtained, using a greedy criterion to determine which solution to implement. The solution is accepted if the importance of the specified objective functions increases [2]. Existing studies described how patterns of knowledge mapping decide the foraging behavior of a colony of honeybees as well as collaborative decision-making in the dynamics of honeybee foraging [3]. Pham et al. conducted an experimental analysis of the Bee Algorithm's performance by evaluating the algorithm using eighteen (18) custom minimization tests and the result was compared with two other common swarm-based optimization algorithms [4]. Their study pointed out that BA has a wide variety of applications and is very successful in solving complex problems with real-world optimization.

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_3

3.2 Foraging Behavior of Honeybee

The foragers are divided into two groups namely the scouts and the onlookers [5]. The scout bees move out of the nest and search for good food sources and the onlookers stay back in the nest and through dancing (waggle dance) evaluate the information provided by the scouts. A good source of food represents flower patches with good amounts of nectar or pollen that can be accessed for collection with the least effort possible. More bees visit this type of food source while fewer bees receive patches with less nectar or pollen [6]. The cycle of foraging starts with the scout bees going around the hive in search of attractive flower patches. Scout bees keep going around the hives in a random search manner from one patch to another, determining their feasibility based on some predetermined level of consistency. Upon returning to the nest, the scouts who find a patch that suits the standard deposit would remove the nectar or pollen from these patches and perform the waggle dance to create awareness [1]. The next section describes the waggle dance in detail. The dance witnessed by the bee is usually a communication pattern within the colony.

3.3 Waggle Dance of Honeybees

Waggle dance is a phenomenon through which the scout bees dance in wagging pattern. Meaning, they move their bodies from side to side, producing a loud buzzing sound. This dance is used to pass on information about a given food source to the rest of the bee colony, such as the food source consistency, direction, and distance from the nest [7]. The waggle dance is a path which has the form of an eight-figure.

This starts with the vibration of wing muscles for scout, which in effect creates a loud buzz and runs on a straight line in a direction showing the position of the food source relative to both the vertical of the hive and the azimuth area of the sun. It then circles back, from left to right, changing the path. The distance from the food source is determined by the speed or duration of the dance, and the buzzing determines the quality of the source, which influences the number of follower bees [8]. An illustration of the bee waggle dance is shown in Fig. 3.1.

3.4 Mathematical Modelling of the ABC Algorithm

The ABC algorithm of the bee generates an arbitrarily distributed solution of the initial population ($f = 1, 2 \ldots E_b$) where the population size is denoted by "f", "E_b" represent the number of working bees. Individual solution x_f represents a dimensional vector D. The value D represents the number of objective functions and constraints presented for optimization. It is equally important to note the position of a food source while handling optimization problem. For instance, in solving the problem using

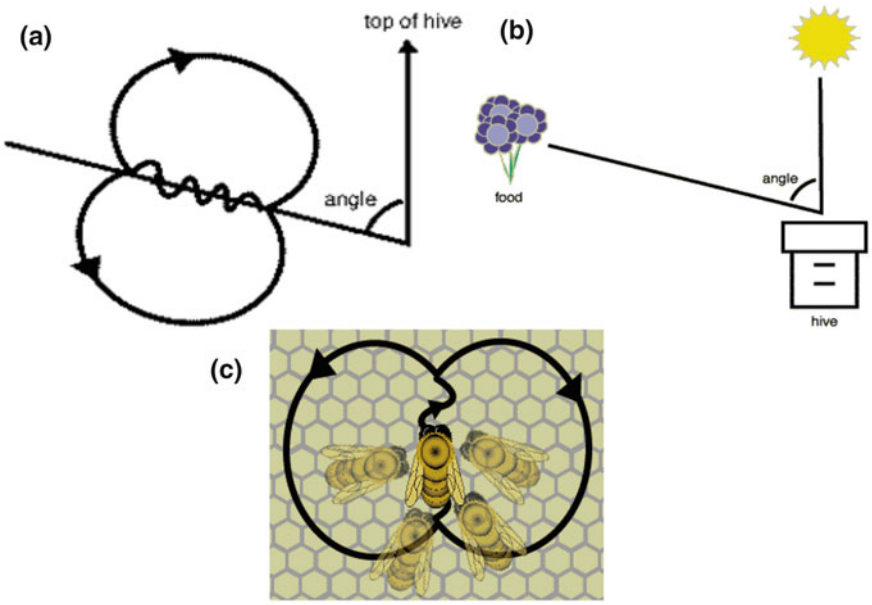


Fig. 3.1 **a** Waggle dance orientation of honeybee as a result of the sun **b** Waggle Dance orientation in search for hive, food and sun **c** Honeybee followers and waggle dance [5]

the ABC algorithm, proper positioning of food source create a quick and probable solution to the optimization problem. Also, another parameter to note is the amount of nectar present in a food source that represents the quality of the connected solution which is equally referred to as the fitness value. The bee algorithm process is such that after the process of initialization, the solution to the population undergoes repetitive sequences of search process targeted at the employed bees in the classification, the onlooker bee, and the scout bees. The cycle process starts from one (1) and ends at the search process maximum cycle number. The selection process is referred to as the Greedy-Selection Process (GSP). The GSP is appropriate for an unrestrained optimization process. To better explain the concept, it is important to take note of the mathematical process for computing the probability (P_i) of selecting a food source by an observer or onlooker bees can be is computed from the system of equations:

$$P_i = \frac{\text{fitness}}{\sum_{f=1}^{E_b} \text{fitness}_f} \tag{3.1}$$

where:

fitness_f represent the value of fitness for solution f.

E_b represent the position of available food-source in totality.

Based on Eq. (3.1), the better the food source is, the more onlooker bees will be attracted while such bees stay away from bad food sources. Consequent to onlooker bees opting for their favorite food source, they introduce a neighbor position of food-source represented by f + 1 to fit into the selected one f, and equate the fitness value or nectar amount of the introduced neighbor f + 1 location with the long-standing location. An unrestrained optimization problem was considered to illustrate the ABC algorithm. A candidate solution $v_{i,j}$ will be generated by each bee x_i in the neighborhood of its current position as described by the Eq. (3.2):

$$v_{i,j} = \begin{cases} x_{ij} + \Phi_{ij}\left(x_{ij} - x_{kj}\right), if R_j < MR \\ x_{ij}, \text{otherwise} \end{cases} \tag{3.2}$$

where $k \in [1, 2, \ldots, SN]$ is an index determined arbitrarily. Its value must be dissimilar from the "i" value. R_j is a real number randomly chosen in the range [0, 1] and $j \in \{1, 2, 3, 4, \ldots, D\}$. The modification rate MR is a parameter that is controlled, and it determines whether the constraint x_j is fit for modification. Φ_{ij} is a random number selected in the interval [−1,1].

Considering the ABC algorithm version proposed for constrained optimization problems, scouts in the system are formed based on a prearranged period of cycles. This is another stage of the process with a defined control parameter referred to as the production period of the scout or Scout Production Period (SPP) of the bee algorithm. The ABC algorithm has a well-defined control parameters repeatedly defined SN equal total number of available food source which is also equivalent to onlooker bees or the number of employed. The cycle number or Maximum Cycle Number is presented as MCN.

A summary of the steps of the ABC algorithm is described by the following pseudo-code:

1. Initialization of the population with haphazard or random solutions.
2. Evaluation of the appropriateness of the defined population.
3. Presentation of cycle as unity.
4. Process repetition
5. Formation of a new population in the system.
6. Application of greedy selection process.
7. Selection of sites appropriate for neighborhood exploration.
8. Recruitment of more bees for the selected site and fitness evaluation.
9. Selecting the most appropriate or suitable bee from the respective area.
10. Assigning outstanding bees for random exploration and evaluating their sensitivities.
11. Finish process.

The first step is the initialization process has to do with fine-tuning the solution population x_{ij}, where i equal 1,2,3 … SN, j = 1, 2, 3 … D. The second step is the evaluation of the defined population; with a single cycle considered. The process is

repeated; then a new solution is provided ($v_{i,j}$) for the bees employed subject to evaluation. The greedy selection process is further applied and the values of probability P_i is calculated for the obtained solutions x_{ij} and new solutions are computed. The solutions for the onlookers x_{ij} (carefully chosen) depend on $P_{i,j}$. After this process, evaluation is conducted and the greedy process of selection is activated. It is equally very important to remember the abandonment (control of the abandoned solution) is very crucial for the scout bee. Where abandonment exist, it is preferable to replace such with innovative solution obtained randomly in the form x_{ij}.. The best solution is memorized at the completion stage. After the process of initialization, the entire population in the system of solutions is then subjected to repetitive cycle for the scout bees, employed bees, and onlooker bees. The cycle is defined and runs from the value of one to the maximum cycle number (MCN) of the process of exploration.

3.5 Application of ABC Algorithm in a Fast-Moving Consumer Goods

Data was obtained from a Fast-Moving Consumers Goods (FCMG). The system has sixteen sections as presented in Table 3.1, which shows clearly product sections and availability of products. Also, there are 4 serving units namely: butchery, seafood, delicatessen, and bakery. Information containing the number of customers visiting

Table 3.1 Sections of the system and products availability

Sections	Products
Section 1	Detergents, brush ware, air fresheners, all-purpose cleaners
Section 2	Baby foods, toilet paper, baby toiletries, Manchester goods
Section 3	Bath soap, men deodorants, ladies deodorants, toiletries
Section 4	Eggs, pasta, bakery aids, canned fruits
Section 5	Spices, crockery, glassware, condiments
Section 6	Cooking oil, kitchenware, kitchen utensil, canned vegetables), section
Section 7	Tea, coffee, creamers, small appliance), section
Section 8	Yogurts, breakfast cereal, breakfast bars), section
Section 9	Juices, biscuits, fruit juices,
Section 10	Sweet, chocolates,
Section 11	Chips, snack, outdoor toys, puzzles
Section 12	Girls toys, boys toys
Section 13	Stationery, schoolbag, school stationery, party accessories
Section 14	Luggage, electric ware, motor accessories, bathroom
Section 15	D.I.Y, pet food, pool care, pet accessories
Section 16	Braai, garden, outdoor living, gardening tool

the various sections of the product-based system is shown in Fig. 3.2. Figure 3.3 represents customers visiting the service-based sections within the period of investigation. The objective is to know the most highly rated net source of optimization by onlookers using Bee Algorithm.

Solution Procedure

The probability P_i of selecting the food source i is determined by Equations

$$P_i = \frac{f_i}{\sum_{f=1}^{NS} f_i}$$

where f_i = Number of visit per set, NS = Total Number of Set;
Total Number of Sets, NS = 9;

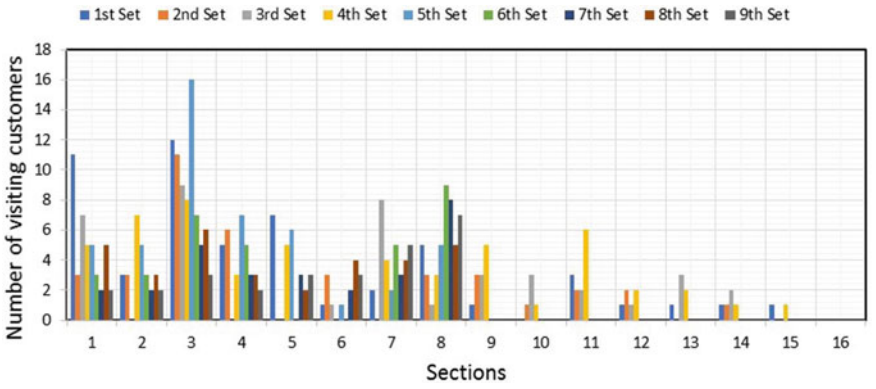


Fig. 3.2 Chart illustrating the customers visiting various sections of the retail outlet

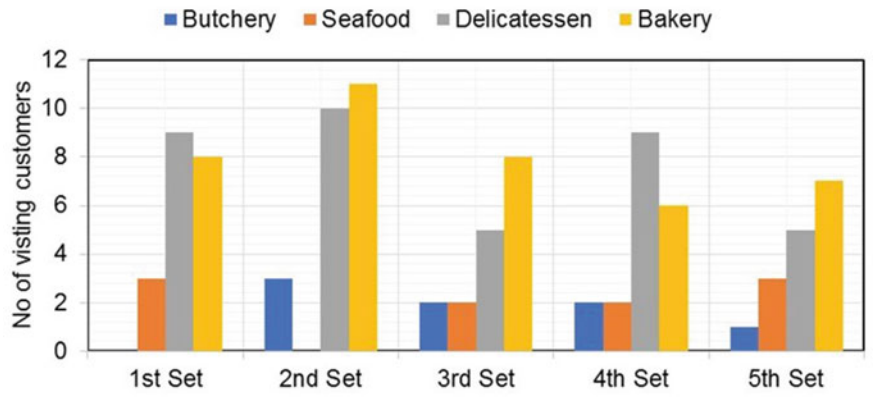


Fig. 3.3 Chart illustrating customers visiting the service units

For the first section; the number of visits per set f_i;

- For the 1st Set ==> Probability, Pi $= \frac{11}{\sum_{i=1}^{9} 11} = 0.2558$
- For 2nd Set ==> Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0697$
- For 3rd Set ==> Probability, Pi $= \frac{7}{\sum_{i=1}^{9} 7} = 0.1627$

For the second section;
Total No. of Sets, NS = 9, No. of visit per set fi;

- For 1st Set ==> Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.1071$,
- For 2nd Set ==> Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.1071$
- For 3rd Set ==> Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0$

Table 3.2 summarizes the results in terms of the probability of selection of food sources by onlookers.

Analysis and Result

From Figure 3.4, it was observed that Section 3(Bath soap, men deodorant, ladies deodorant, toiletries) was the most visited in the retail outlet. Also from Table 3.1, it was observed that Section 1 to 8 has more visits compared to Sections 9 to 16. However, visit to each of the sections is dependent on certain observable factors taken into consideration during the study namely: Location of the sections, number of

Table 3.2 Probability of selecting food Source by onlookers, P_i

Sections	1st Set P_1	2nd Set P_2	3rd Set P_3	4th Set P_4	5th Set P_5	6th Set P_6	7th Set P_7	8th Set P_8	9th Set P_9
1	0.2558	0.0697	0.1627	0.1162	0.1162	0.0697	0.0465	0.1167	0.0465
2	0.1071	0.1071	0	0.25	0.1785	0.1071	0.0714	0.1071	0.0714
3	0.1558	0.1429	0.1169	0.1039	0.1078	0.0909	0.0649	0.0779	0.0389
4	0.1470	0.1764	0	0.0882	0.1058	0.1470	0.0882	0.0882	0.0588
5	0.2692	0	0	0.1923	0.2307	0	0.1153	0.0769	0.1153
6	0.0666	0.2	0.0666	0	0.0666	0	0.1333	0.2666	0.2
7	0.0606	0	0.2424	0.1212	0.0606	0.1515	0.0909	0.1212	0.1515
8	0.1086	0.0652	0.0217	0.0652	0.1086	0.1956	0.1739	0.1086	0.1521
9	0.0833	0.25	0.25	0.4166	–	–	–	–	–
10	0	0.2	0.6	0.2	–	–	–	–	–
11	0.2307	0.1538	0.1538	0.4615	–	–	–	–	–
12	0.1666	0.3333	0.1666	0.3333	–	–	–	–	–
13	0.1666	0	0.5	0.3333	–	–	–	–	–
14	0.2	0.2	0.4	0.2	–	–	–	–	–
15	0.5	0	0	0.5	–	–	–	–	–
16	0	0	0	0	–	–	–	–	–

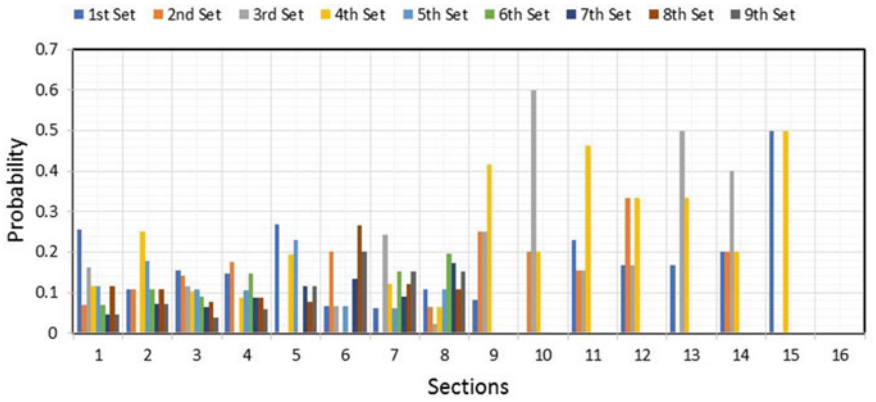


Fig. 3.4 Chart showing the probability of selecting foodstuff

children that come along with their guardian or parent which had a great influence in Section 3, 4, 8, 9, 10 and 12; number of probable parents which influenced Section 1, 2, 3, 4, 6, and 8 and number of average singles which influenced Section 1,3, 4, 6, 7, 8, 9 and 11. Considering Table 3.2, it is observed that the serving unit with many visits is between the delicatessen and the bakery serving unit. Other units have less number of visits within the period of investigation. Some of the observable factors that influence this number of visits are information about the available products and how fresh the products are. The probability of selecting the food source by onlookers is presented in Fig. 3.4.

From the bar chart of Figs. 3.4 and 3.5 which represent both the product sections and serving unit respectively, it was observed that the most highly rated section at this retail store is Section 3 among all sections while the most highly rated section considering the serving unit, is the bakery. However, by adopting the mathematical

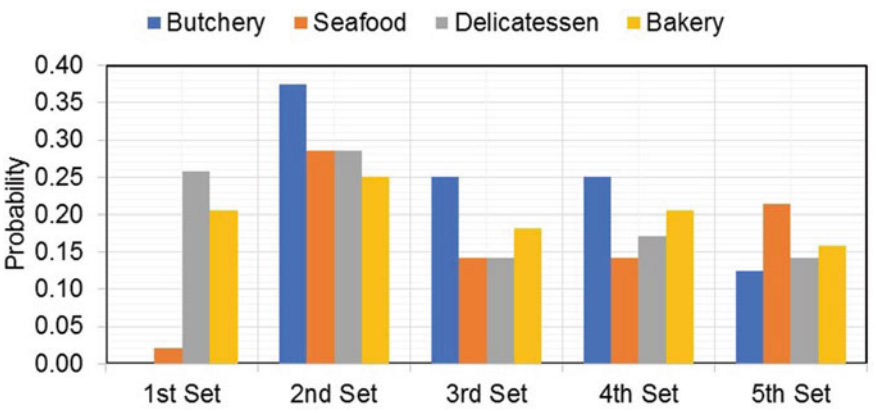


Fig. 3.5 Probability of Customers Visiting Various Service Sections

model developed from the Bee Algorithm for the probability of the best food source, different interpretations were made based on results shown in Table 3.2. Table 3.2 shows the probability of the best food source by onlookers. It was observed from the chart that the most highly rated section by the onlookers using the Bee Algorithm is the Section 3 due to the fully observed cluster at that section out of the 16 different Sections and the Bakery Unit at the serving Unit out of the other four (4) Serving unit due to its optimal values over other units. Summary of the solution is presented mathematically:

Section 1	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9th Set
(Detergents, Brush Ware, Air Fresheners, All Purpose Cleaners)	11	3	7	5	5	3	2	5	2

Total No. of Sets, NS = 9.
No. of visit per Set fi, For 1st Set = 11, Probability, $\text{Pi} = \frac{11}{\sum_{i=1}^{9} 11} = 0.2558$.
For 2nd Set; Probability, $\text{Pi} = \frac{3}{\sum_{i=1}^{9} 3} = 0.0697$.
For 3rd Set; Probability, $\text{Pi} = \frac{7}{\sum_{i=1}^{9} 7} = 0.1627$.
For 4th Set; Probability, $\text{Pi} = \frac{5}{\sum_{i=1}^{9} 5} = 0.1162$.
For 5th Set; Probability, $\text{Pi} = \frac{5}{\sum_{i=1}^{9} 5} = 0.1162$.
For 6th Set; Probability, $\text{Pi} = \frac{3}{\sum_{i=1}^{9} 3} = 0.0697$.
For 7th Set; Probability, $\text{Pi} = \frac{2}{\sum_{i=1}^{9} 2} = 0.0465$.
For 8th Set; Probability, $\text{Pi} = \frac{5}{\sum_{i=1}^{9} 5} = 0.1162$.
For 9th Set; Probability, $\text{Pi} = \frac{2}{\sum_{i=1}^{9} 2} = 0.0465$.

Section 2	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9th Set
(Baby Foods, Toilet Paper, Baby Toiletries, Manchester Goods)	3	3	0	7	5	3	2	3	2

Total No. of Sets, NS = 9.
No. of visit per set fi, for 1st Set = 3.
Therefore, Probability, $\text{Pi} = \frac{3}{\sum_{i=1}^{9} 3} = 0.1071$.
For 2nd Set; Probability, $\text{Pi} = \frac{3}{\sum_{i=1}^{9} 3} = 0.1071$.
For 3rd Set; Probability, $\text{Pi} = \frac{0}{\sum_{i=1}^{9} 0} = 0$.
For 4th Set; Probability, $\text{Pi} = \frac{7}{\sum_{i=1}^{9} 7} = 0.2500$.

For 5th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1785$.
For 6th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.1071$.
For 7th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{9} 2} = 0.0714$.
For 8th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.1071$.
For 9th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0714$.

Section 3	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9^{th} Set
(Bath Soap, Men Deodorant, Women Deodorant, Toiletries)	12	11	9	8	16	7	5	6	3

Total No. of sets, NS = 9.
No. of Visit per set fi, for 1st Set = 12.
Therefore, Probability, Pi $= \frac{12}{\sum_{i=1}^{9} 12} = 0.1558$.
For 2nd Set; Probability, Pi $= \frac{11}{\sum_{i=1}^{9} 11} = 0.1429$.
For 3rd Set; Probability, Pi $= \frac{9}{\sum_{i=1}^{9} 9} = 0.1169$.
For 4th Set; Probability, Pi $= \frac{8}{\sum_{i=1}^{9} 8} = 0.1039$.
For 5th Set; Probability, Pi $= \frac{16}{\sum_{i=1}^{9} 16} = 0.2078$.
For 6thSet; Probability, Pi $= \frac{7}{\sum_{i=1}^{9} 7} = 0.0909$.
For 7th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.0649$.
For 8th Set; Probability, Pi $= \frac{6}{\sum_{i=1}^{9} 6} = 0.0779$.
For 9th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0389$.

Section 4	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9th Set
(Eggs, Pasta, Bakery Aids, Canned Fruits)	5	6	0	3	7	5	3	3	2

Total No. of Sets, NS = 9.
No. of Visit per Set fi, for 1st Set = 12.
Therefore, Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1470$.
For 2nd Set; Probability, Pi $= \frac{6}{\sum_{i=1}^{9} 6} = 0.1764$.
For 3rd Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0$.
For 4th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0882$.
For 5th Set; Probability, Pi $= \frac{7}{\sum_{i=1}^{9} 7} = 0.2058$.

For 6th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1470$.
For 7th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0882$.
For 8th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0882$.
For 9th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{9} 2} = 0.0588$.

Section 5	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9th Set
(Spices, Crockery, Glassware, Condiments)	7	0	0	5	6	0	3	2	3

Total No. of Sets, NS = 9.
No. of Visits per Set fi, for 1stSet = 7.
Therefore, Probability, Pi $= \frac{7}{\sum_{i=1}^{9} 7} = 0.2692$.
For 2nd Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0$.
For 3rd Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0$.
For 4th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1923$.
For 5th Set; Probability, Pi $= \frac{6}{\sum_{i=1}^{9} 6} = 0.2307$.
For 6th Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0$.
For 7th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.1153$.
For 8th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{9} 2} = 0.0769$.
For 9th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.1153$.

Section 6	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9th Set
(Cooking Oil, Kitchen Ware, Kitchen Utensil, Canned Vegetables)	1	3	1	0	1	0	2	4	3

Total No. of Sets, NS = 9.
No. of Visits per Set fi, for 1st Set = 1.
Therefore, Probability, Pi $= \frac{1}{\sum_{i=1}^{9} 1} = 0.0666$.
For 2nd Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.2$
For 3rd Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{9} 1} = 0.0666$.
For 4th Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0$.
For 5th Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{9} 1} = 0.0666$.

For 6th Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0.$
For 7th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{9} 2} = 0.1333.$
For 8th Set; Probability, Pi $= \frac{4}{\sum_{i=1}^{9} 4} = 0.2666.$
For 9th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.2$

Section 7	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9th Set
(Tea, Coffee, Creamers, Small Appliance)	2	0	8	4	2	5	3	4	5

Total No. of Sets, NS $= 9.$
No. of Visits per Set fi, for 1stSet $= 2.$
Therefore, Probability, Pi $= \frac{2}{\sum_{i=1}^{9} 2} = 0.0606.$
For 2nd Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{9} 0} = 0.$
For 3rd Set; Probability, Pi $= \frac{8}{\sum_{i=1}^{9} 8} = 0.2424.$
For 4th Set; Probability, Pi $= \frac{4}{\sum_{i=1}^{9} 4} = 0.1212.$
For 5th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{9} 2} = 0.0606.$
For 6th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1515.$
For 7th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0909.$
For 8th Set; Probability, Pi $= \frac{4}{\sum_{i=1}^{9} 4} = 0.1212.$
For 9th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1515.$

Section 8	1st Set	2nd Set	3rd Set	4th Set	5th Set	6th Set	7th Set	8th Set	9th Set
(Yogurts, Breakfast Cereals, Cereals Bars)	5	3	1	3	5	9	8	5	7

Total No. of Sets, NS $= 9.$
No. of Visits per Set fi, for 1stSet $= 5.$
Therefore, Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1086.$
For 2nd Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0652.$
For 3rd Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{9} 1} = 0.0217.$
For 4th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{9} 3} = 0.0652.$
For 5th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1086.$
For 6th Set; Probability, Pi $= \frac{9}{\sum_{i=1}^{9} 9} = 0.1956.$
For 7th Set; Probability, Pi $= \frac{8}{\sum_{i=1}^{9} 8} = 0.1739.$

For 8th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{9} 5} = 0.1086.$
For 9th Set; Probability, Pi $= \frac{7}{\sum_{i=1}^{9} 7} = 0.1521.$

Section 9	1st Set	2nd Set	3rd Set	4th Set
(Juices, Biscuits, Fruit Juices)	1	3	3	5

Total No. of Sets, NS $= 4.$
No. of Visit per Set fi, for 1stSet $= 1.$
Therefore, Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.0833.$
For 2nd Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{4} 3} = 0.25.$
For 3rd Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{4} 3} = 0.25.$
For 4th Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{4} 5} = 0.4166.$

Section	1st Set	2nd Set	3rd Set	4th Set
10 (Sweets, Chocolates)	0	1	3	1

Total No. of Sets, NS $= 4.$
No. of Visit per Set fi, for 1stSet $= 0.$
Therefore, Probability, Pi $= \frac{0}{\sum_{i=1}^{4} 0} = 0.$
For 2nd Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.2$
For 3rd Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{4} 3} = 0.6$
For 4th Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{9} 1} = 0.2$

Section 11	1st Set	2nd Set	3rd Set	4th Set
(Chips, Snack, Outdoors Toys, Puzzles)	3	2	2	6

Total No. of Sets, NS $= 4.$
No. of Visit per Set fi, for 1stSet $= 3.$
Therefore, Probability, Pi $= \frac{3}{\sum_{i=1}^{4} 3} = 0.2307.$
For 2nd Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{4} 2} = 0.1538.$
For 3rd Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{4} 2} = 0.1538.$
For 4th Set; Probability, Pi $= \frac{6}{\sum_{i=1}^{4} 6} = 0.4615.$

Section 12	1st Set	2nd Set	3rd Set	4th Set
(Girl Toys, Boy Toys)	1	2	1	2

Total No. of Sets, NS = 4.
No. of Visit per Set fi, for 1stSet = 1.
Therefore, Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.1666.$
For 2nd Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{4} 2} = 0.3333.$
For 3rd Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.1666.$
For 4th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{4} 2} = 0.3333.$

Section 13	1st Set	2nd Set	3rd Set	4th Set
(Stationery, School Bag, School Stationery, Party Accessories)	1	0	3	2

Total No. of Sets, NS = 4.
No. of Visit per Set fi, for 1stSet = 1.
Therefore, Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.1666.$
For 2nd Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{4} 0} = 0.$
For 3rd Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{4} 3} = 0.5$
For 4th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{4} 2} = 0.3333.$

Section 14	1st Set	2nd Set	3rd Set	4th Set
(Luggage, Electric Ware, Motor Accessories, Bathroom)	1	1	2	1

Total No. of Sets, NS = 4.
No. of Visit per Set fi, for 1st Set = 1.
Therefore, Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.2$
For 2nd Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.2$
For 3rd Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{4} 2} = 0.4$
For 4th Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.2$

Section 15	1st Set	2nd Set	3rd Set	4th Set
(D.I.Y, Pet food, Pool care, Pet accessories)	1	0	0	1

Total No. of Sets, NS = 4.
No. Of Visit per Set fi, for 1st Set = 1.
Therefore, Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.5$

For 2nd Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{4} 0} = 0.$
For 3rd Set; Probability, Pi $= \frac{0}{\sum_{i=1}^{4} 1} = 0.$
For 4th Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{4} 1} = 0.5$
For Table 3.2.

Serving units	1st Set	2nd Set	3rd Set	4th Set	5th Set
Butchery	0	3	2	2	1

Total No. of Sets, NS $= 5.$
No. Of Visit per Set fi, for 1st Set $= 0.$
Therefore, Probability, Pi $= \frac{0}{\sum_{i=1}^{5} 0} = 0.$
For 2nd Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{5} 3} = 0.375.$
For 3rd Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{5} 2} = 0.25.$
For 4th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{5} 2} = 0.25.$
For 5th Set; Probability, Pi $= \frac{1}{\sum_{i=1}^{5} 1} = 0.125.$

Serving units	1st Set	2nd Set	3rd Set	4th Set	5th Set
Seafood	3	4	2	2	3

Total No. of Sets, NS $= 5.$
No. Of Visit per Set fi, for 1st Set $= 3.$
Therefore, Probability, Pi $= \frac{3}{\sum_{i=1}^{5} 3} = 0.0214.$
For 2nd Set; Probability, Pi $= \frac{4}{\sum_{i=1}^{5} 4} = 0.2857.$
For 3rd Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{5} 2} = 0.1428.$
For 4th Set; Probability, Pi $= \frac{2}{\sum_{i=1}^{5} 2} = 0.1428.$
For 5th Set; Probability, Pi $= \frac{3}{\sum_{i=1}^{5} 3} = 0.2142.$

Serving units	1st Set	2nd Set	3rd Set	4th Set	5th Set
Delicatessen	9	10	5	6	5

Total No. of Sets, NS $= 5.$
No. Of Visit per Set fi, for 1st Set $= 9.$
Therefore, Probability, Pi $= \frac{9}{\sum_{i=1}^{5} 9} = 0.2571.$
For 2nd Set; Probability, Pi $= \frac{10}{\sum_{i=1}^{5} 10} = 0.2857.$
For 3rd Set; Probability, Pi $= \frac{5}{\sum_{i=1}^{5} 5} = 0.1428.$
For 4th Set; Probability, Pi $= \frac{6}{\sum_{i=1}^{5} 6} = 0.1714.$

For 5th Set;

Serving Units	1st Set	2nd Set	3rd Set	4th Set	5th Set
Bakery	9	11	8	9	7

Probability, Pi $= \frac{5}{\sum_{i=1}^{5} 5} = 0.1428$.
Total No. of Sets, NS $= 5$.
No. Of Visit per Set fi, for 1st Set $= 9$.
Therefore, Probability, Pi $= \frac{9}{\sum_{i=1}^{5} 9} = 0.2045$.
For 2nd Set; Probability, Pi $= \frac{11}{\sum_{i=1}^{5} 11} = 0.25$.
For 3rd Set; Probability, Pi $= \frac{8}{\sum_{i=1}^{5} 8} = 0.1818$.
For 4th Set; Probability, Pi $= \frac{9}{\sum_{i=1}^{5} 9} = 0.2045$.
For 5th Set; Probability, Pi $= \frac{7}{\sum_{i=1}^{5} 7} = 0.1590$.

Bee algorithm plays a great role in real-life activities. Datasets obtained a retail store were analyzed using the modeled equations from the bee algorithm to optimize the flow of customers in and out of the retail store and to observe the most frequently visited sections in the system. Out of sixteen (16) sections selected for study, it was observed that the most highly rated section in the system by the onlookers using the Bee Algorithm model was Section three (3). The products available at the section are the most frequently consumed. They include bath soap, toilet paper, baby toiletries, and Manchester goods. For the four (4) service sections, the most highly rated section was the bakery unit. Just like the waggle dance observed by honeybees, it was interesting to note that same exists at the bakery unit when a customer gives a signal that new breads are out of the oven. The sound attracts customers to that point and there could be rush in most cases till the hot breads are completely exhausted. Conclusively, the metaheuristic ABC model is highly effective in system modeling and prediction.

References

1. Koc, E. 2010. *The bees algorithm: theory, improvements and application.* Cardiff: Cardiff University.
2. Panigrahi, B.K., M.H. Lim, and Y. Shi. 2011. *Handbook of swar, intelligence-concepts, principles and applications.* Berlin: Springer.
3. Tereshko, C.V., A. Loengarov. 2005. Collective decision-making in honey bee foraging dynamics. *Computing and Information Systems Journal* 9(3) ISSN 1352-9404
4. Pham, D.T., M. Castellani, and J. Chen. 2015. A comparative study of the bees algorithm as a tool for function optimisation. *Cogent Engineering.* https://doi.org/10.1080/23311916.2015.1091540.
5. Karagoba, D. 2005. An idea based on honey bee swarm for numerical optimization. Technical Report, Erciyes University, Engineering Faculty, Computer Engineering Department.
6. Camazine, S., and J. Sneyed. 1991. A model of collective nectar source selection by honey bees: self-organization through simple rules. *Journal of Theoretical Biology* 149: 547–571.

7. Yuce, B., M.S.P. Packianather, E. Mastrocinque, D.T. Pham, and A. Lambiase. 2013. Honeybees inspired optimization method: the Bees algorithm. *Insects* 4(4): 646–662.
8. Talbi, E.G. 2009. *Metaheuristics: from design to implementation*, vol. 74. New York: Wiley.

Chapter 4
Ant Colony Algorithm

4.1 Introduction

Ant algorithm or ant colony optimization algorithm is a Meta-heuristic optimization process that is probabilistic. It models the real-life movement or behavior of ants to solve the optimization problem. Since ants are blind, they move randomly from one place to another i.e. they choose a path based on probability. When ants move from their nest in search of food, they seek a path that has a shorter distance. During ants foraging, they deposit pheromones along the path followed until they get to their destination i.e. food source, and return to their nest along the same path. When other ants seek for food, a greater percentage of ants follow the path having a greater pheromone trail until all the ants eventually follow the shortest path to the food source as foraging continues.

Pheromone trail is the main reason ants are capable to form or maintain a line. While traveling, ants release a certain amount of pheromone which provides a path for other ants to follow. This basic instinct of ants explains why they generally seek the shortest path even in the presence of an obstacle that hinders an initial path. By default, ants travel in a straight line from the nest (N) to the food source (S). In case there is an obstacle as shown in Fig. 4.1, leading ants would have to choose between the left or right path. It can be assumed from investigation that half of the ants turn left and the remaining turn right. The ants that followed the shorter path would reconstitute the pheromone trail more rapidly as compared to the ones that followed a longer path. A larger amount of pheromone will be deposited on the shorter path per time unit resulting in a higher likelihood for the ants to choose this shorter path (Fig. 4.2) [1].

Ants can locate food sources around their nest without any spatial information and find the shortest path leading to it. As an ant finds foods, it returns to its nest leaving behind pheromone trails as shown in Fig. 4.3. This pheromone trail will be followed by the other ants who are less susceptible to travel randomly. As more ants find food, the trail will be reinforced with more pheromone as shown in Fig. 4.3. However, due

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_4

Fig. 4.1 Ants typical path around an obstacle

Fig. 4.2 Amount of pheromone trail based on ants path

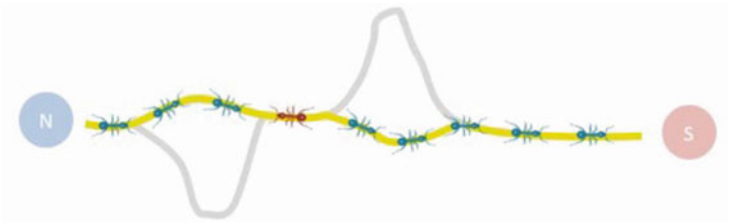


Fig. 4.3 Illustration of ants path and pheromone trails

to the stochastic behavior of some ants, they will not follow the pheromone trail lay down by previous ants and instead uncover more possible paths as shown in Fig. 4.3. Over time, the pheromone left on the trails that are not followed will evaporate which reduces the attractiveness strength. Ultimately, the shortest path would be identified and maintained as shown in Fig. 4.3.

The longer it takes for ants to travel back and forth along a specific path, the lower the pheromone density as a result of evaporation. In the context of optimization using ACO, the pheromone evaporation is useful because it avoids locally optimal solution and reduce the attractiveness of existing potential locally optimal solution.

4.2 Mathematical Formulation

The principles described in the previous sections are translated into a mathematical formulation and illustrated with an example in the following section. The probability for an ant k at node r to select a destination node s at a later stage is given as follows [2]:

$$p_k(r, s) = \begin{cases} \tau(r,s)^{\alpha}\eta(r,s)^{\beta} / \sum_{uM_k} \tau(r,u)^{\alpha}\eta(r,u)^{\beta}, & \text{for s } M_k \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

In this equation, α is the weight of pheromone τ, β is the degree of visibility, M_k represents the working memory and $\eta(r, s)$ is the visibility of cities r and s. The pheromone train on edge (r, u) is $\tau(r, u)$ while a heuristic function is represented by $\eta(r, u)$. The pheromone trail is subjected to change both locally and globally. If $\Delta\tau(r, s)$ represent the amount of pheromone deposited by the best ant on each visited edge, this value is inversely proportional to the length of the tour. The equation describing the global trail updating is given as follows [2]:

$$\tau(r, s) \leftarrow (1 - \rho).\tau(r, s) + \nu\tau(r, s) \tag{4.2}$$

In this equation, ρ is the evaporation rate parameter, $\Delta\tau(r, s) = \text{shortesttour}^{-1}$. While global updating reward edges of the shorter tour, local updating aims at avoiding edge selected by all ants. The local trail updating equation is given by the following equation [2]:

$$\tau(r, s) \leftarrow (1 - \rho).\tau(r, s) + \tau_0 \tag{4.3}$$

where τ_0 is a parameter. To illustrate the implementation of the ACO algorithm, a Traveling Salesman Problem (TSP) has been described in the next section. The ACO algorithm steps can be summarized as follows [2]:

1. Distribute the ants randomly between cities;
2. Considering each ant:
   a. Select a city that has not been visited as yet until a tour is completed;
   b. Perform optimize the tour;
   c. Update pheromone
3. Evaporate pheromone

Figure 4.4 shows the flowchart pointing out the working procedure of ACO [1].

Ant colony optimization algorithm has been applied in routing in telecommunication networks, Traveling Salesmen Problem (TSP), and scheduling.

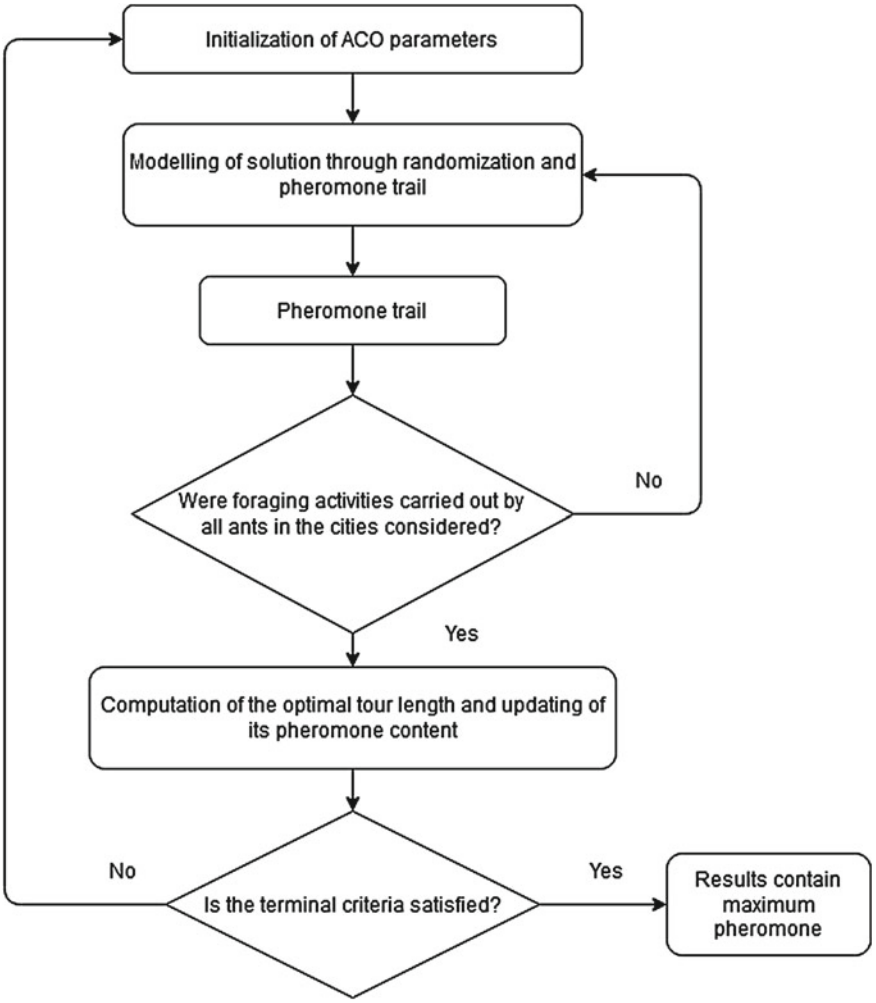


Fig. 4.4 Flowchart of ACO algorithm [3]

4.3 Application of ACO to Solve TSP Problem

This problem consists of determining the best route with minimum total distance supposing that a salesman has to visit 5 cities. The distance between cities is given in tabulated format as follows:

	City 1	City 2	City 3	City 4	City 5
City 1	0	10	12	11	14
City 2	10	0	13	15	8

(continued)

(continued)

	City 1	City 2	City 3	City 4	City 5
City 3	12	13	0	9	14
City 4	11	15	9	0	16
City 5	14	8	14	16	0

Step 1

We assume that the number of ants N = 3. The visibility (η) between cities can be calculated as the inverse of the distance between cities (1/d) as follows:

$\eta =$					
	0	1/10 = 0.1000	1/12 = 0.0833	1/11 = 0.0909	1/14 = 0.0714
	1/10 = 0.1000	0	1/13 = 0.0769	1/15 = 0.0667	1/8 = 0.1250
	1/12 = 0.0833	1/13 = 0.0769	0	1/9 = 0.1111	1/14 = 0.0714
	1/11 = 0.0909	1/15 = 0.0667	1/9 = 0.1111	0	1/16 = 0.0625
	1/14 = 0.0714	1/8 = 0.1250	1/14 = 0.0714	1/16 = 0.0625	0

Considering each segment between cities, we assume that the value τ_0 which represents the initial pheromone is equal to 1 for all segments.

τ_0 (or tauinit) =					
	1	1	1	1	1
	1	1	1	1	1
	1	1	1	1	1
	1	1	1	1	1
	1	1	1	1	1

Step 2

Calculate the probability of traveling from city 1 to a different city. The salesman departure from city 1. Because the trip is taking place from the last city visited city 1, the level of visibility of city 1 is reduced to zero since city 1 cannot be visited again.

$\eta =$					
	0	0.1000	0.0833	0.0909	0.0714
	0	0	0.0769	0.0667	0.1250
	0	0.0769	0	0.1111	0.0714

(continued)

(continued)

$\eta =$					
	0	0.0667	0.1111	0	0.0625
	0	0.1250	0.0714	0.0625	0

Using Eq. (4.1), we can calculate the probability to visit other cities from city 1. We assume that $\alpha = 1$ and $\beta = 2$.

Assuming that ant 1 from city 1 travels to the next city, we can calculate the value $\tau(1,s)^1\eta(1,s)^2$

$\tau(1,s)^1$	$\eta(1,s)^2$	$\tau(1,s)^1\eta(1,s)^2$	
1	0.1000^2	0.01	From city 1 to city 2
1	0.0833^2	0.0069	From city 1 to city 3
1	0.0909^2	0.0083	From city 1 to city 4
1	0.0714^2	0.0051	From city 1 to city 5
		$\Sigma = 0.0303$	

The probability for an ant 1 to travel from city 1 to all the other cities can be obtained as follows:

$\tau(1,s)^1\eta(1,s)^2$	$\Sigma\tau(1,s)^1\eta(1,s)^2$	$\tau(1,s)^1\eta(1,s)^2/\Sigma\tau(1,s)^1\eta(1,s)^2$	
0.01	0.0303	0.3299	From city 1 to city 2
0.0069	0.0303	0.2291	From city 1 to city 3
0.0083	0.0303	0.2727	From city 1 to city 4
0.0051	0.0303	0.1683	From city 1 to city 5

The cumulative values of these probabilities are 0.3299, 0.5590 (or 0.2291 + 0.3299), 0.8317, and 1 corresponding respectively to city 1, city 2, city 3, and city 4.

Step 3

A random number r is generated to determine the next city to visit. This value will be compared to the cumulative probability value. Assuming for instance that r = 0.6841, this value is less than the cumulative probability value of 0.8317. Subsequently, the next city to visit in the city 4. Selecting city 4 means that the 4th column of the matrix can be set to zero.

$\eta =$					
	0	0.1000	0.0833	0	0.0714
	0	0	0.0769	0	0.1250

(continued)

(continued)

$\eta =$					
	0	0.0769	0	0	0.0714
	0	0.0667	0.1111	0	0.0625
	0	0.1250	0.0714	0	0

We can calculate the value $\tau(4, s)^1 \eta(4, s)^2$ from city 4 to all the other cities (except city 1).

$\tau(4, s)^1$	$\eta(4, s)^2$	$\tau(4, s)^1 \eta(4, s)^2$	
1	0.0667^2	0.0044	From city 4 to city 2
1	0.1111^2	0.0123	From city 4 to city 3
1	0^2	0.0	From city 4 to city 4
1	0.0714^2	0.0039	From city 4 to city 5
		$\Sigma = 0.0207$	

The probability for an ant 1 to travel from city 4 to all the other cities can be obtained as follows:

$\tau(4, s)^1 \eta(4, s)^2$	$\Sigma\tau(4, s)^1 \eta(4, s)^2$	$\tau(4, s)^1 \eta(4, s)^2 / \Sigma\tau(4, s)^1 \eta(4, s)^2$	
0.0044	0.0207	0.2147	From city 1 to city 2
0.0123	0.0207	0.5965	From city 1 to city 3
0	0.0207	0	From city 1 to city 4
0.0039	0.0207	0.1887	From city 1 to city 5

The cumulative values of these probabilities are 0.2147 and 0.8113 (or 0.1887 + 0.5965) corresponding respectively to city 2 and city 3.

Assuming a random number r = 0.4024, this value is less than the cumulative probability value of 0.8113. Subsequently, the next city to visit is the city 3. Ant 1 route sequence will be 1-4-3. After repeating this step for all ants, we can determine the sequences and total distance of each route.

	Route sequence	Total distance
Ants 1	1-4-3-5-2-1	52
Ants 2	1-4-2-5-3-1	60
Ants 3	1-4-5-2-3-1	60

The pheromone level can be updated using Eq. (4.4).

$$\tau(r, s) \leftarrow (1 - \rho).\tau(r, s) + \Sigma\nu\tau(r, s) \tag{4.4}$$

Step 4

Additional pheromone $\Delta\tau(r, s)$ will be provided by ant 1. This additional pheromone is proportional to 1/total distance or $1/52 = 0.0192$. The evaporation coefficient ρ is equal to 0.5. Considering that ant 1 trajectory is 1-4-3-5-2-1. The pheromone matrix is given as follows:

0.5000	0.5000	0.5000	(0.5000 + 0.0192) = 0.5192	0.5000
(0.5000 + 0.0192) = 0.5192	0.5000	0.5000	0.5000	0.5000
0.5000	0.5000	0.5000	0.5000	(0.5000 + 0.0192) = 0.5192
0.5000	0.5000	(0.5000 + 0.0192) = 0.5192	0.5000	0.5000
0.5000	(0.5000 + 0.0192) = 0.5192	0.5000	0.5000	0.5000

Additional pheromone will be provided by ant 2. This additional pheromone is proportional to $1/60 = 0.0167$. Considering that ant 2 trajectory is 1-4-2-5-3-1. The pheromone matrix is given as follows:

0.5000	0.5000	0.5000	(0.5000 + 0.0192) + 0.0167 = 0.5192	0.5000
(0.5000 + 0.0192) = 0.5192	0.5000	0.5000	0.5000	(0.5000 + 0.0167) = 0.5167
(0.5000 + 0.0167) = 0.5167	0.5000	0.5000	0.5000	(0.5000 + 0.0192) = 0.5192
0.5000	(0.5000 + 0.0167) = 0.5167	(0.5000 + 0.0192) = 0.5192	0.5000	0.5000
0.5000	(0.5000 + 0.0192) = 0.5192	(0.5000 + 0.0167) = 0.5167	0.5000	0.5000

Additional pheromone will be provided by ant 3. This additional pheromone is proportional to $1/60 = 0.0167$. Considering that ant 3 trajectory is 1-4-5-2-3-1. The pheromone matrix is given as follows:

0.5000	0.5000	0.5000	(0.5000 + 0.0192) + 0.0167 + 0.0167 = 0.5192	0.5000
(0.5000 + 0.0192) = 0.5192	0.5000	(0.5000 + 0.0167) = 0.5167	0.5000	(0.5000 + 0.0167) = 0.5167

(continued)

(continued)

0.5000	0.5000	0.5000	(0.5000 + 0.0192) + 0.0167 + 0.0167 = 0.5192	0.5000
(0.5000 + 0.0167) + 0.0167 = 0.5	0.5000	0.5000	0.5000	(0.5000 + 0.0192) = 0.5192
0.5000	(0.5000 + 0.0167) = 0.5167	(0.5000 + 0.0192) = 0.5192	0.5000	(0.5000 + 0.0167) = 0.5167
0.5000	(0.5000 + 0.0192) + 0.0167 = 0.5359	(0.5000 + 0.0167) = 0.5167	0.5000	0.5000

Iteration will be repeated from step 2 until the maximum number of iterations is reached.

The MATLAB code shown in Appendix B was run. The following script can be used to find the best route and the minimum cost:

```
>> d=[0 10 12 11 14
10 0 13 15 8
12 13 0 9 14
11 15 9 0 16
14 8 14 16 0]
>> [bestroute,mincost]=ACO_code(d,20,10)
```

The best route (bestroute) was [1 2 5 3 4 1] corresponding to a minimum cost of 52.

References

1. Dorigo, M., M. Birattari, and T. Stutzle. 2006. Ant colony optimization. *IEEE Computational Intelligence Magazine* 1 (4): 28–39.
2. Dorigo, M., and G. Di Caro. 1999. Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol. 2, pp. 1470–1477. IEEE.
3. Yun, H.-Y., S.-J. Jeong, and K.-S. Kim. 2013. Advanced harmony search with ant colony optimization for solving travelling salesman problem. *Journal of applied mathematics* 1–8.

Chapter 5
Grey Wolf Optimizer

5.1 Introduction

The Grey Wolf Optimization (GWO) algorithm is a swarm or population-based meta-heuristics technique. The algorithm was developed taking into consideration motivation from the hunting pattern of the Grey Wolves (GW) or the rock-hard pyramid of the GW [1]. The algorithm of the GWO is such that individual assigned in a pack of wolf is allocated in one out of four diverse classified echelons (pyramid or hierarchy), usually from the highest to lowest category such as Alpha(α), Beta (β), Delta (δ), and Omega (ω) (Fig. 5.1). Following the hierarchy, Alpha signifies the foremost individual in charge of the park and is characterized by orders and verdicts. On the other hand, Beta signifies individual in the system who supports Alpha while deciding on the park. Such an individual is also involved in other activities. Beta commands low-level individuals while obeying the orders of the Alpha. Beta can be positioned as the counselor for Alpha. Beta equally represents the park's discipline. Omega on the rock-hard pyramid represent individual with the lowest ranking on the echelon system. Omega is pictured on the pyramid as the individual who is loyal to other individuals at the top echelon. In the absence of Omega, actions of the wolf pack like cub-carting, fighting, and hunting could be susceptible. Looking closely at wolves on the echelon system of the pyramid, the Delta is quite different from the other three packs of wolves (Omega, Alpha, and Beta). It is also important to note that the wolves park referred to as Deltas are very much submissive and loyal to the first two wolves park "Betas and Alphas", while they try to override the Omegas. They usually take the position as a detective, lookout, watchman, hoarier, predator, and curator [2].

The flowchart representing the GWO optimization technique or procedure is shown in Fig. 5.1 [3]. As shown in the flowchart, the first step to the optimization process is the random initialization of the grey wolves. Then the selection of

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_5

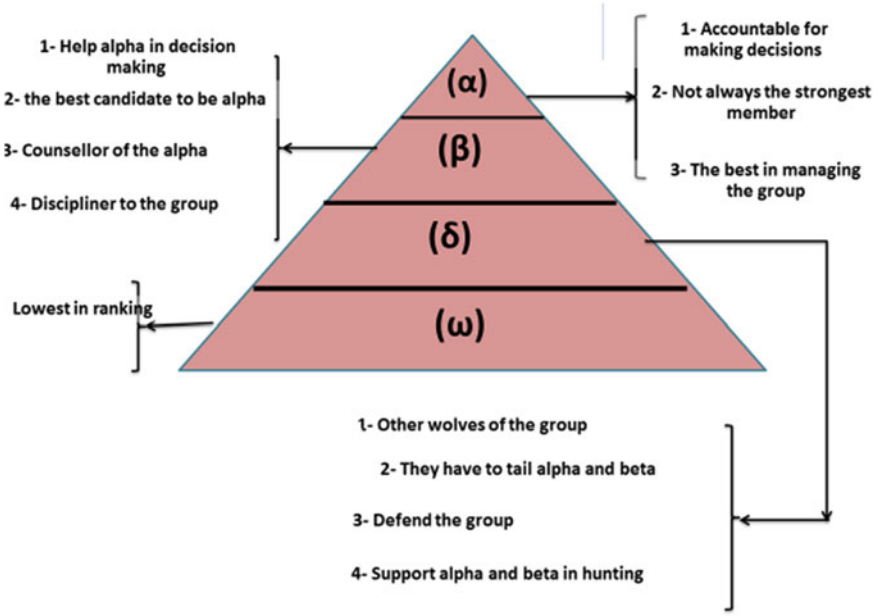


Fig. 5.1 Grey wolf hierarchy

chaotic input grey wolf optimization parameter and mapping of the chaotic membership values with the respective algorithm laterally with initialization process variable and disordered number [4]. Afterward, Chaotic Grey Wolf Optimization (CGWO) parameters of the process are adjusted and tuned. All grey wolves fitness adjusted in the quest space is estimated using several standard index functions which is arranged by their fitness. It is presumed that the first wolf obtained after the arrangement is Alpha and thus the second and third wolves are believed to be Beta and Delta correspondingly. Afterward, the fitness evaluation process takes place to identify the best three wolves. Then repetition of chaotic sequence with an update of the chaotic map, along with the position of all GWs takes place. This is followed by a fitness evaluation and replacement of the worst fit grey with the best fit grey. The next step is to decide on the criteria met before the best output solution (Fig. 5.2).

5.2 Fundamental Theory of Gray Wolf Optimizer (GWO)

The leadership and the hunting mechanism of grey wolves are the inspiration behind the Grey Wolf Optimization algorithm. The main steps of Grey wolf hunting can be summarized as follows [1]:

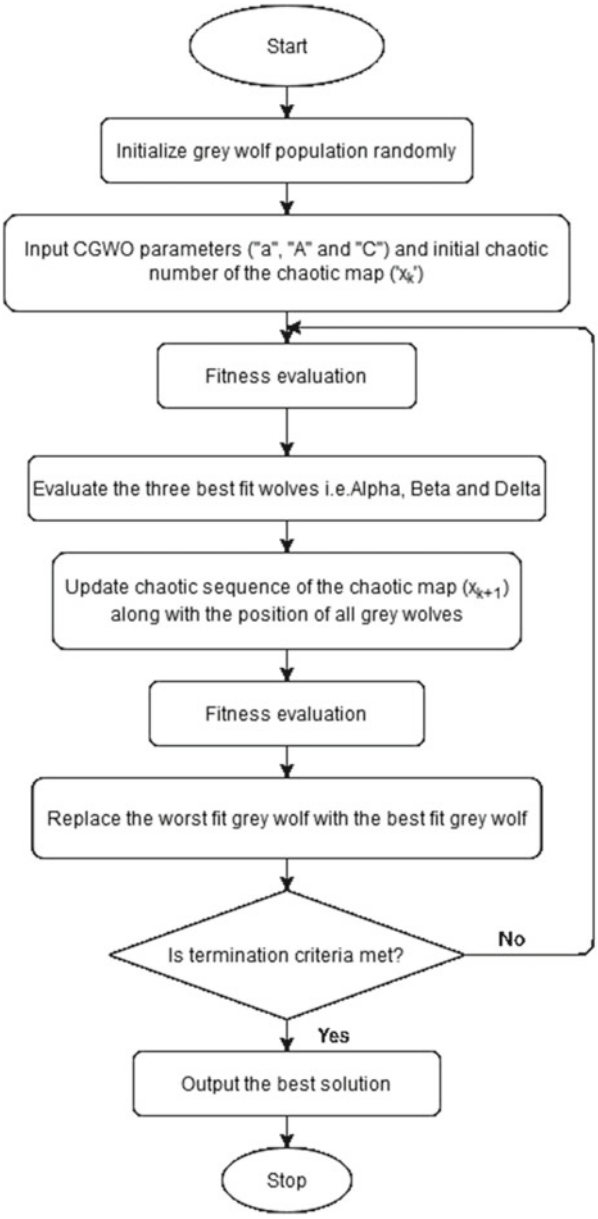


Fig. 5.2 Optimization procedure flowchart of GWO [1]

1. Searching of the prey;
2. Tracking, chasing and approaching the prey;
3. Engaging pursue, encircling and harassing the prey until it stops moving;
4. Attacking the prey.

The steps consist of searching, pursuing, encircling and attacking.

To convert this into a mathematical model, we will consider the fittest solution Alpha. The second and third fittest solution is Beta and delta wolves respectively. Omega follows these three wolves.

Mathematical model of encircling the prey can be described as follows:

$$\vec{D} = \left|\vec{C}.\overrightarrow{X_p} - \vec{X}(t)\right| \tag{5.1}$$

$$\vec{X}(t+1) = \left|\overrightarrow{X_p}(t) - \vec{A}.\vec{D}\right| \tag{5.2}$$

where t = current iterations.
$\overrightarrow{X_p}$ = position of the prey;
$\overrightarrow{X}$ = position of the grey wolf;
$\overrightarrow{A}$, $\overrightarrow{C}$ = coefficient vectors;
$\overrightarrow{A}$, $\overrightarrow{C}$ vectors are calculated as

$$\overrightarrow{A} = 2.\overrightarrow{a}.\overrightarrow{r_1} - \overrightarrow{a}; \tag{5.3}$$

$$\overrightarrow{C} = 2.\overrightarrow{r_2}; \tag{5.4}$$

$\overrightarrow{r_1}$, $\overrightarrow{r_2}$ = random vectors range [0,1];
Component $\overrightarrow{a}$ linearly decrease from 2 to 0 throughout iterations.

These equations can be used to update the grey wolf position according to the position of the prey. Several positions around the best search agents can be reached concerning the current position by adjusting the values of $\overrightarrow{A}$ and $\overrightarrow{C}$. The random vectors $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ allows the wolf to reach a position between 2 specific points.

Mathematical model of hunting can be described as follow.

In general, the hunting process is guided by Alpha. This process assumes that Alpha, Beta, and Delta have better knowledge about the location of the prey (or the optimal solution). The other wolves will update their positions based on the position of Alpha, Beta, and Delta [1, 5].

$$\overrightarrow{D_\alpha} = \left|\vec{C}_1.\overrightarrow{X_\alpha} - \vec{X}(t)\right| \tag{5.5}$$

$$\overrightarrow{D_\beta} = \left|\vec{C}_2.\overrightarrow{X_\beta} - \vec{X}(t)\right| \tag{5.6}$$

$$\vec{D}_\delta = \left|\vec{C}_3.\vec{X}_\delta - \vec{X}(t)\right| \tag{5.7}$$

$$\vec{X}_1 = \left|\vec{X}_\alpha - \vec{A}_1.\vec{D}_\alpha\right| \tag{5.8}$$

$$\vec{X}_2 = \left|\vec{X}_\beta - \vec{A}_2.\vec{D}_\beta\right| \tag{5.9}$$

$$\vec{X}_3 = \left|\vec{X}_\delta - \vec{A}_3.\vec{D}_\delta\right| \tag{5.10}$$

The position of the grey wolves will be updated using the following equation:

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{5.11}$$

$\vec{X}_\alpha, \vec{X}_\beta, \vec{X}_\delta$ are the position vector of α, β and δ. $\vec{A}_1$, $\vec{A}_2$ and $\vec{A}_3$ are the coefficient vectors.

Mathematical model for attacking the prey can be described as follows:

As the prey stops moving, wolves attack it to finalize the process of hunting. Mathematically, this is expressed by the decrease of $\vec{a}$ from 2 to 0 during iterations. As $\vec{a}$ decrease, $\vec{A}$ decreases as well. Value of $A < 1$ forces the wolf to launch the attack toward the prey. However, for $|A| > 1$ make the wolf searching for better prey (Fig. 5.3) [1]. $\vec{C}$ vectors are random values in the interval [0, 2]. $\vec{C}$ vectors add

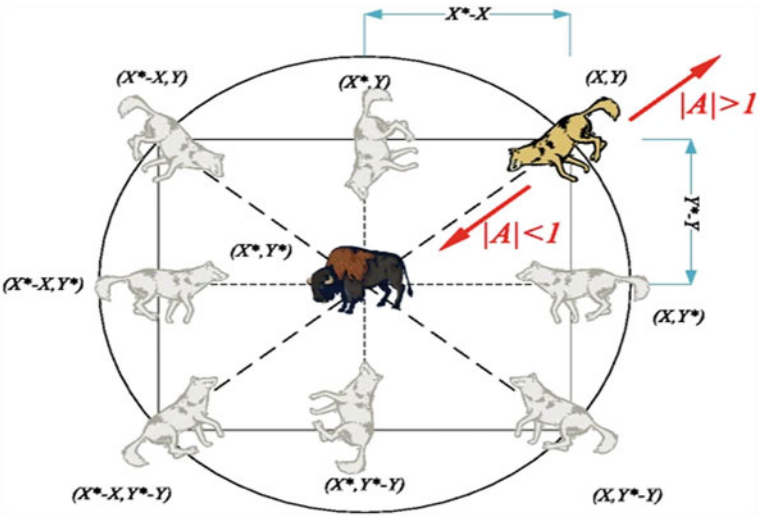


Fig. 5.3 Grey wolf positioning for search agent [1]

weight to the prey and make it difficult for the wolves to locate it. When C > 1, the prey importance is emphasized and for C < 1, the prey importance is reduced.

5.3 Application of Gray Wolf Optimization with a Numerical Example

Minimization of Korn function.

$$f(x_1, x_2) = \min\left[(x_1 - 5)^2 + (x_2 - 2)^2\right]$$

Initialize grey wolf population: n = 12

$$\overrightarrow{A} = 2.\overrightarrow{a}.\overrightarrow{r_1} - \overrightarrow{a}$$

$$\overrightarrow{C} = 2.\overrightarrow{r_2}$$

$$\overrightarrow{a} = 2 - 2\left(\frac{\text{iter}}{\max \text{iter}}\right)$$

We assume that Alpha, Beta, and Delta have better knowledge about the location of the prey (or the optimal solution). Hence, we assume:

	x_1	x_2	$f(x_1,x_2)$
1	6.16	4.41	7.1537
2	6.21	4.09	5.8322
3	7.42	8.38	46.5608
4	2.89	0.87	5.729
5	6.1	3.72	4.1684
6	6.34	3.21	3.2597
7	7.56	6.14	23.6932
8	6.24	4.04	5.6992
9	6.99	4.58	10.6165
10	4.73	3.3	1.7629
11	4.81	3.49	2.2562
12	5.94	3.44	2.9572

And the corresponding (best) values of X_α, X_β and X_δ taken from the previous table are:

	$f(x_1,x_2)$
X_α= [4.73, 3.30]	1.7629
X_β= [4.81, 3.49]	2.2562
X_δ= [5.94, 3.44]	2.9572

We can now update the position of the grey wolves:

$$\vec{a} = 2 - 2.\left(\frac{1}{3}\right) = 1.3333$$

$\vec{D_\alpha} = \left|\vec{C}_1.\vec{X_\alpha} - \vec{X}(t)\right| = |(2.\vec{r_2}) \times [4.73, 3.30] - [6.16, 4.41]|$

$\vec{D_\beta} = \left|\vec{C}_2.\vec{X_\beta} - \vec{X}(t)\right| = |(2.\vec{r_2}) \times [4.81, 3.49] - [6.16, 4.41]|$

$\vec{D_\delta} = \left|\vec{C}_3.\vec{X_\delta} - \vec{X}(t)\right| = |(2.\vec{r_2}) \times [5.94, 3.44] - [6.16, 4.41]|$

$\vec{X_1} = \left|\vec{X_\alpha} - \vec{A_1}.\vec{D_\alpha}\right| = |[4.7372, 3.3048] - (2 \times 1.3333 \times \vec{r_1} - \vec{a}) \times \vec{D_\alpha}|$

$\vec{X_2} = \left|\vec{X_\beta} - \vec{A_2}.\vec{D_\beta}\right| = |[4.8148, 3.4931] - (2 \times 1.3333 \times \vec{r_1} - \vec{a}) \times \vec{D_\beta}|$

$\vec{X_3} = \left|\vec{X_\delta} - \vec{A_3}.\vec{D_\delta}\right| = |[5.9444, 3.4433] - (2 \times 1.3333 \times \vec{r_1} - \vec{a}) \times \vec{D_\delta}|$

$\vec{r_1}$ and $\vec{r_2}$ are random numbers between [0, 1].

The new position of the grey wolves are

$$\vec{X}(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3} = [4.04, 2.60]$$

The new updated positions and corresponding fitness values are:

	x_1	x_2	$f(x_1,x_2)$
1	4.04	2.60	1.2816
2	4.64	3.04	1.2112
3	5.46	3.66	2.9672
4	5.60	3.59	2.8881
5	4.65	3.03	1.1834
6	4.74	3.33	1.8365
7	4.24	2.66	1.0132
8	4.90	3.24	1.5476
9	4.52	2.95	1.1329
10	5.39	3.54	2.5237

(continued)

(continued)

	x_1	x_2	$f(x_1,x_2)$
11	4.11	2.53	1.073
12	5.09	3.15	1.3306

And the corresponding (best) values of X_α, X_β and X_δ taken from the previous table are:

	$f(x_1,x_2)$
X_α= [4.24, 2.66]	1.0132
X_β= [4.11, 2.53]	1.073
X_δ= [4.65, 3.03]	1.1834

We can now update the position of the grey wolves:

$$\vec{a} = 2 - 2.\left(\frac{2}{3}\right) = 0.667$$

$$\vec{D_\alpha} = \left|\vec{C}_1.\vec{X_\alpha} - \vec{X}(t)\right| = |(2.\vec{r_2}) \times [4.24, 2.66] - [4.04, 2.60]|$$
$$\vec{D_\beta} = \left|\vec{C}_2.\vec{X_\beta} - \vec{X}(t)\right| = |(2.\vec{r_2}) \times [4.11, 2.53] - [4.04, 2.60]|$$
$$\vec{D_\delta} = \left|\vec{C}_3.\vec{X_\delta} - \vec{X}(t)\right| = |(2.\vec{r_2}) \times [4.65, 3.03] - [4.04, 2.60]|$$
$$\vec{X_1} = \left|\vec{X_\alpha} - \vec{A_1}.\vec{D_\alpha}\right| = |[4.24, 2.66] - (2 \times 0.667 \times \vec{r_1} - \vec{a}) \times \vec{D_\alpha}|$$
$$\vec{X_2} = \left|\vec{X_\beta} - \vec{A_2}.\vec{D_\beta}\right| = |[4.11, 2.53] - (2 \times 0.667 \times \vec{r_1} - \vec{a}) \times \vec{D_\beta}|$$
$$\vec{X_3} = \left|\vec{X_\delta} - \vec{A_3}.\vec{D_\delta}\right| = |[4.65, 3.03] - (2 \times 0.667 \times \vec{r_1} - \vec{a}) \times \vec{D_\delta}|$$

$\vec{r_1}$ & $\vec{r_2}$ are random numbers between [0, 1].

The new position of the grey wolves are

$$\vec{X}(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3} = [4.48, 2.78]$$

The new updated positions and corresponding fitness values are:

	x_1	x_2	$f(x_1,x_2)$
1	4.48	2.78	0.8788
2	4.56	2.82	0.866
3	4.58	2.83	0.8653
4	4.74	2.94	0.9512
5	4.63	2.86	0.8765

(continued)

(continued)

	x_1	x_2	$f(x_1,x_2)$
6	4.59	2.84	0.8737
7	4.58	2.84	0.882
8	4.57	2.83	0.8738
9	4.57	2.83	0.8738
10	4.57	2.83	0.8738
11	4.57	2.82	0.8573
12	4.56	2.82	0.866

And the corresponding (best) values of X_α, X_β and X_δ taken from the previous table are:

	$f(x_1,x_2)$
$X_\alpha = [4.57, 2.82]$	0.8573
$X_\beta = [4.56, 2.82]$	0.866
$X_\delta = [4.56, 2.82]$	0.866

In the previous sections, the fitness values have been calculated. The fitness function (or the evaluation function) is used to evaluate how close a given solution is to the optimum of a desired problem. This function depends on a given problem [5]. The fitness of each wolf namely Alpha, Beta, and Delta, has been calculated in turn. The fitness of the Alpha wolf will be considered as the most optimal solution to the problem after the last iteration.

This model was implemented in MATLAB using the **Grey Wolf Optimizer (GWO)** as described by Mirjalili et al. [1]. To illustrate the approach, 30 search agents were considered ($n = 30$) and the maximum number of iterations was set to 1000. The best solution obtained by GWO is: [4.9996 2.0003] and the best optimal value of the objective function found by GWO is 2.4264e-07. The parameter space and the objective space are shown in Figs. 5.4 and 5.5.

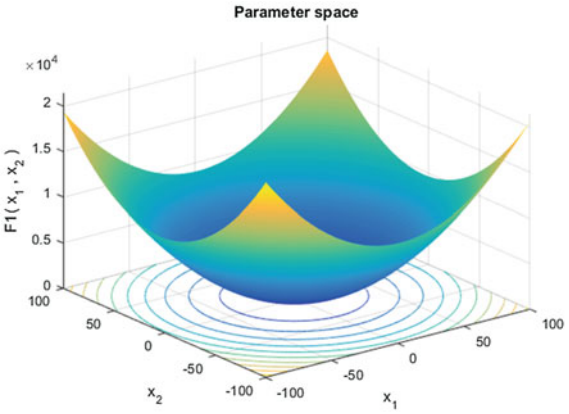


Fig. 5.4 Parameter space

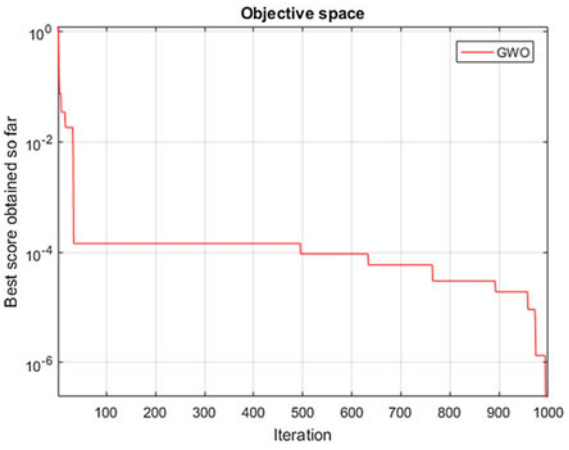


Fig. 5.5 Objective space

References

1. Mirjalili, S., S.M. Mirjalili, and A. Lewis. 2014. Grey wolf optimizer. *Advances in Engineering Software* 69: 46–61.
2. Shuyu, D., D. Niu, and Y. Li. 2018. Daily peak load forecasting based on complete ensemble empirical mode decomposition with adaptive noise and support vector machine optimized by modified greywolf optimization algorithm. *Energies* 11 (1): 163–175. https://doi.org/10.3390/en11010163.
3. Kohli, Mehak. 2018. Sankalap arora chaotic grey wolf optimization algorithm for constrained optimization problems. *Journal of Computational Design and Engineering* 5: 458–472.
4. Gandomi, A.H., and X.-S. Yang. 2014. Chaotic bat algorithm. *Journal of Computational Science* 5(2): 224–232.
5. Mirjalili, S. 2015. How effective is the grey wolf optimizer in training multi-layer perceptrons. *Applied Intelligence* 43 (1): 150–161.
6. Kohli, M., and S. Arora. 2018. Chaotic grey wolf optimization algorithm for constrained optimization problems. *Journal of Computational Design and Engineering* 5 (4): 458–472.

Chapter 6
Whale Optimization Algorithm (WOA)

6.1 Introduction

It is commonly recognized that Algorithms inspired by nature or nature-inspired optimization algorithms, which include swarm-based algorithms (SAs), and evolutionary algorithms (EAs) make up the computer intelligence discipline which has been existing in the past decades. Nature-inspired optimization (NIO) algorithm is said to be popular because of its usefulness in solving complex problems in different fields. NIO algorithms, particularly SAs, are versatile, simple to use, and this has greatly helped to solve problems in science and engineering field. This algorithm is proposed by [2] and is considered highly significant in problem-solving [4]. It is equally important to note the basic rules of the algorithms which are primarily inspired by nature. This will help to carefully watch as the current solutions are constantly updated to boost their cost function values. Conducting a search space efficiently and thoroughly, and reducing its normal size, a good solution is obtained by implementing the WOA. Nevertheless, algorithms are very different in the way they build a balance between discovery or global search competence and the local search competence in terms of exploitation around the optimal or near-optimal solution.

Furthermore, an appropriate balance between investigation, exploration and exploitation is seen as a very complex task in nature-inspired or meta-heuristic algorithms, especially when it has to do with the development process as a result of the random or haphazard nature of the process of optimization. The discovery process allows the optimizer to dig further into the search space in a global methodology or pattern. Also, the changes encountered in this process is quite unexpected as a result of the challenging population. The investigation stage or process has to do with refinement of solutions which are promising at that level or phase. Hof and Van der Gucht [1] believe that the whales possess certain cells that are just like that of the brain of a human being. These areas are referred to as the spindle cells. These cells are responsible for social behaviors, emotions, and judgment in humans. In other words, humans are distinguished from other creatures with the presence of the spindle cells.

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_6

The major reason for whale's smartness is as a result of the number of these cells in them which are more than those in humans. Whales can learn, think, judge, interact with their environment and get emotional as a human does, but one area of deficiency of the whale is in the area of smartness which is of lower degree. Further observation shows that whales especially the killer type of whales can develop their kind of language or dialect for effective communication and interaction among one another.

Another outstanding feature of whales is typically observed in their behavior socially [2]. A good number of whales live alone while most of the group to form communities. Some species of whales particularly the killer whales can live in a family over their complete life span. The humpback whales popularly known as Megaptera novaeangliae is one of the biggest baleen whales ever known. The fully-grown humpback whale is almost of the same size as the school bus. They generally love chasing for krill or small-sized fish herds [2]. The fascinating attribute of the humpback whales is their special hunting strategy. The well-known hunting behavior peculiar to them is their feeding pattern which can be described as the bubble-net pattern of feeding [2, 5]. The Humpback whales have a unique strategy of hunting for small fishes at the surface of the water. They look out for school of fishes or krill [2]. Studies conducted concerning the behavior of these species of whales shows how intelligent they are, as the actions and behavior of the humpback whales were as a result of the surface of outward observation of the categories of whales [6].

Three hundred tag-derived bubble-net feeding events were captured considering nine distinct humpback whales. Two bubble-related movements were established and called 'upward-spirals' and 'double-loops.' For most situations, the humpback whales dive down 12 m in the previous movement and then start the bubbling process around the prey in a spiral formation and swim upwards to the surface. There are three major stages associated with the process namely the capture loop, coral loop, and lob-tail. It is also necessary to comprehend the fact that the feeding pattern of the whale known as the bubble-net pattern of feeding is a special feeding activity common with the humpback whales. Figure 6.1 shows the navigation pattern of the whale.

6.2 Feeding Pattern of Whale—Encircling of Prey

It is important to state the pattern in which the whales encircle prey and the algorithm formation process. The humpback whale will be considered in this context. These set of whales can recognize and surround the location of the prey strategically [2]. Identifying the exact position in the a priori search space in whale is usually not imaginable. It is assumed by the WOA that the target prey is the best candidate solution or is near the optimum. If the best search pattern is created and established, the next aspect could be the attempt by the other search agents to shift their positions against the best search waiter.

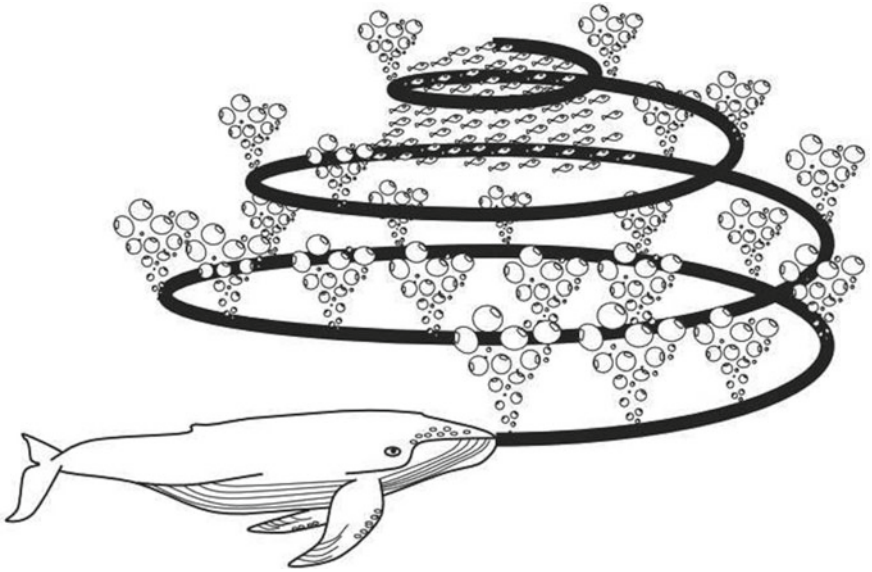


Fig. 6.1 Navigational pattern of a whale using a bubble net in search for food [3]

The following equations presented by [2, 4] translate mathematically the behaviors of whales:

$$\overrightarrow{\mathrm{D}} = |\overrightarrow{\mathrm{C}}.\overrightarrow{\mathrm{X}^{*}_{(\mathrm{t})}} - \overrightarrow{\mathrm{X}_{(\mathrm{t})}}| \tag{6.1}$$

$$\overrightarrow{X_{(t+1)}} = X^{*}_{(t)} - \overrightarrow{A}.\overrightarrow{D} \tag{6.2}$$

In the equations, t represents the iteration in progress; $\overrightarrow{\mathrm{A}}$ and $\overrightarrow{\mathrm{C}}$ represent the coefficient vectors; X^{*} represent the best solution in terms of position vector hitherto came across; $\overrightarrow{\mathrm{X}}$ denotes the existing location vector; || represents the absolute value.

The parameter $\overrightarrow{\mathrm{A}}$ and $\overrightarrow{\mathrm{C}}$ can be further expressed as:

$$\overrightarrow{A} = 2\overrightarrow{a}.\overrightarrow{r_1} - \overrightarrow{a} \tag{6.3}$$

$$\overrightarrow{C} = 2\overrightarrow{\mathrm{r}_2} \tag{6.4}$$

In these equations, constituents of $\overrightarrow{\mathrm{a}}$ are decreased from 2 to 0 in a linear manner in successive iterations and r_1 and r_2 are random vectors in [0,1]. The change of $\overrightarrow{\mathrm{a}}$ is shown as:

$$a = 2 - 2\frac{t}{t_{max}} \tag{6.5}$$

where t_{max} is the maximum number of iterations.

X^* must be restructured and updated after each iteration process. The purpose is to check if a better solution is obtained during the iteration process. The same idea is extended to an n-dimensional exploration space, and therefore the search agents can move in hypercube round the best answer achieved to this point.

6.3 Bubble-net Attacking

This behavior of humpback whales is modeled by the following two techniques:

(a) **Shrinking encircling mechanism**: In this behavior, the value of $\overrightarrow{a}$ is reduced in Eq. (6.3). With this the variation range of $\overrightarrow{A}$ is also reduced with the reduction in $\overrightarrow{a}$. Using the arbitrary values for $\overrightarrow{A}$ in the range [−1,1], the recent location of a search agent is located somewhere in the middle of the initial location of the search agent and the location of the existing best search agent.

(b) **Spiral updating location**: This location method is defined in such a way that, the location amongst the whale positioned at (X, Y) and prey positioned at (X ∗ , Y ∗) is computed. Then, the spiral-shaped motion of the humpback whales is simulated by the equation amongst the location of the whale and position of the prey as shown in Eq. (6.5).

$$\overrightarrow{X_{(t+1)}} = \overrightarrow{D'}.e^{bl}.\cos(2\pi l) + \overrightarrow{X_t^*} \tag{6.6}$$

In Eq. (6.6), $\overrightarrow{D'} = = |\overrightarrow{X_{(t)}^*} - \overrightarrow{X_{(t)}}|$ symbolizes the distances between the ith whale and the prey which is a function of the best solution obtained; b represents a constant which defines the outline and curve of the logarithmic ring-like or twisting process; l is the subjective figure in the interval [−1, 1] and symbolizes a component by component increase.

When the humpback whales are attacking their prey, they travel along a spiral path as they surround their prey. To simulate this behavior, the shrinking encompassing mechanism and formation pattern during navigation as well as the mechanism for the spiral updating location are presumed to have an equivalent probability value of 0.5.

This is demonstrated in the system of Eq. 6.6 as presented thus:

$$\overrightarrow{X_{(t+1)}} = \left\{ \begin{array}{c} \overrightarrow{X^*_{(t)}} - \overrightarrow{A} . \overrightarrow{D} if p < 0.5 \\ \overrightarrow{D'} . e^{bl} . \cos(2\pi l) + \overrightarrow{X^*_{(t)}} \text{ if } p \geq 0.5 \end{array} \right\} \tag{6.7}$$

In this equation p is an arbitrary number [0,1]

6.4 Pattern Adopted by Whale While Searching for Prey and Flowchart of WOA

The whale, particularly the humpback whales, have unique ways of hunting for prey. During the predation process, specific whales move randomly during the search process following a defined location. This can be represented by a coefficient variable of the form $|A| \geq 1$. The separate whales are enforced away from the locus point of the whales to enable them to find more and suitable prey. This action will further facilitate the global search strategy creating global search scope for the whale optimization algorithm (WOA). The equation of the model is presented in the system of Eq. (6.7):

$$\overrightarrow{D} = |\overrightarrow{C} . \overrightarrow{X_{rand}} - \overrightarrow{X_{(t)}}| \tag{6.8}$$

$$\overrightarrow{X_{(t+1)}} = \overrightarrow{X_{rand}} - \overrightarrow{A} . \overrightarrow{D} \tag{6.9}$$

Where $\overrightarrow{X_{rand}}$ represent the whale position vector arbitrarily selected from the specific population of the whale. The flowchart of WOA is shown in Fig. 6.2.

6.5 Application of Whale Optimization Algorithm with a Numerical Example

Minimization of Himmelblau's function.

$$f(x_1, x_2) = \min \left[\left(x_1^2 + x_2 - 11 \right)^2 + \left(x_1 + x_2^2 - 7 \right)^2 \right]$$

This model was implemented in MATLAB using the **Whale Optimization Algorithm (WOA)** as described in reference [2]. To illustrate the approach, 30 search agents were considered (n = 30) and the maximum number of iterations was set to 1000. The best solution obtained by WOA is: [3.0015 1.9969] and the best optimal value of the objective function found by WOA is 0.00015167. The parameter space and the objective space are shown in Figs. 6.3 and 6.4.

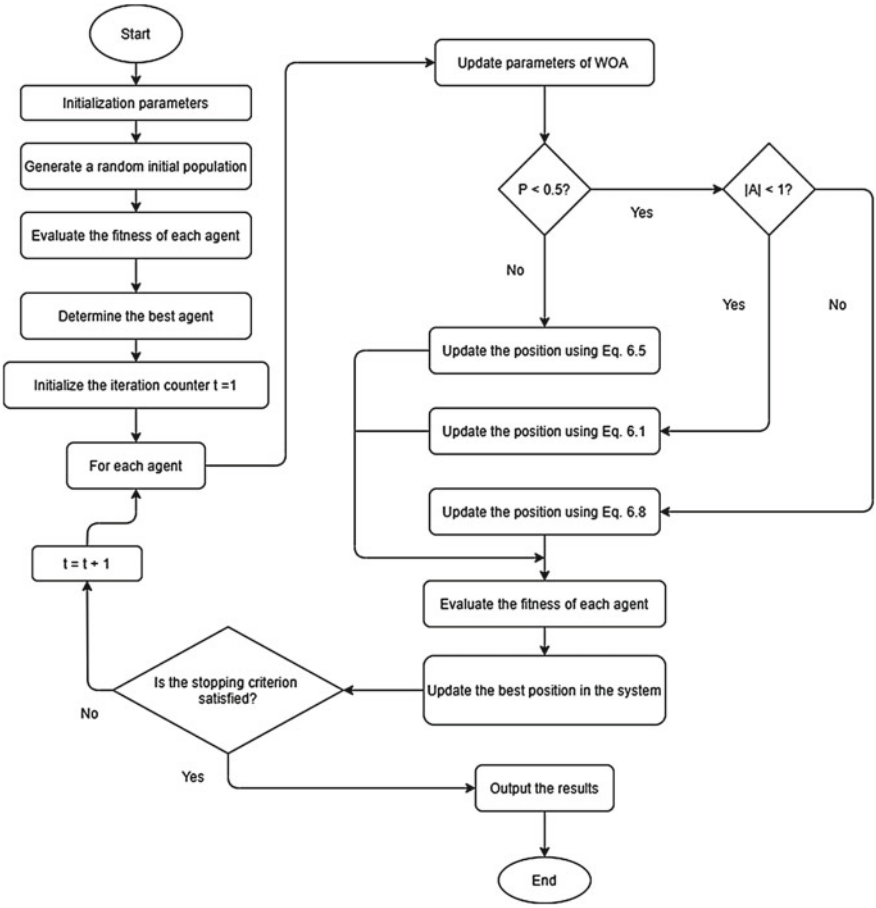


Fig. 6.2 Optimization Procedure Flowchart of Whale Algorithm

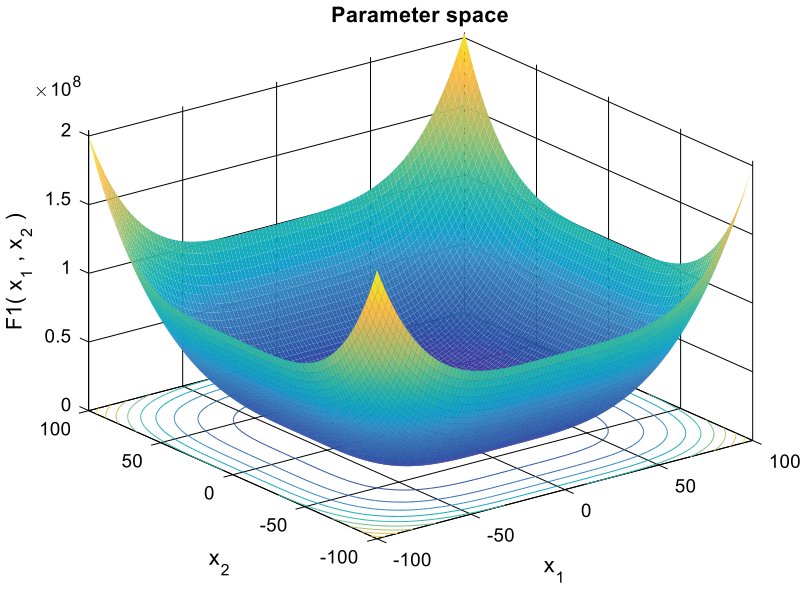


Fig. 6.3 Parameter space

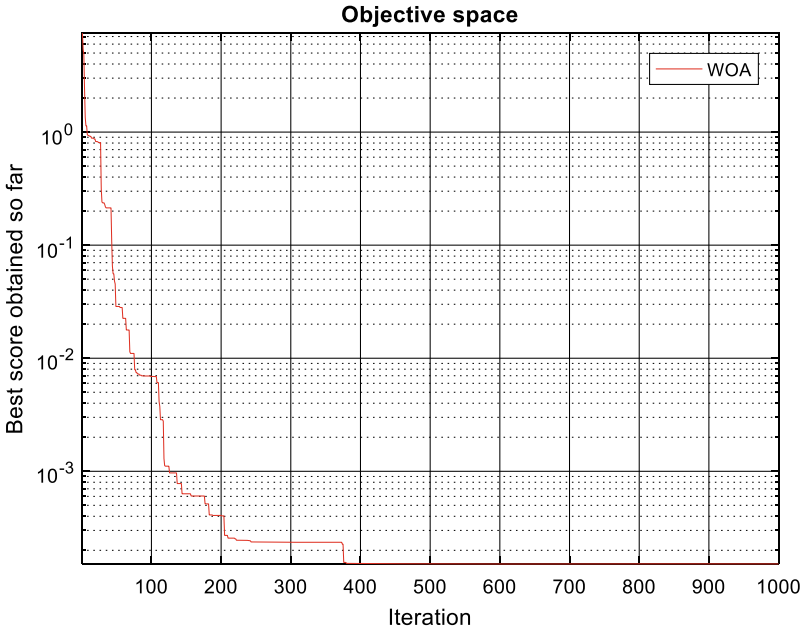


Fig. 6.4 Objective space

References

1. Hof, P.R., and E. Van der Gucht. 2007. Structure of the cerebral cortex of the humpback whale, Megaptera novaeangliae (Cetacea, Mysticeti, Balaenopteridae). *The Anatomical Record* 290 (1): 1–31.
2. Mirjalili, S., and A. Lewis. 2016. The whale optimization algorithm. *Advances in Engineering Software* 95: 51–67.
3. Bentouati, B., L. Chaib, and S. Chettih. 2016. A hybrid whale algorithm and pattern search technique for optimal power flow problem. In *2016 8th international conference on modelling, identification and control (ICMIC),* pp. 1048–1053. IEEE.
4. Adhirai, S., R.P. Mahapatra, and P. Singh. 2018. The whale optimization algorithm and its implementation in MATLAB. *International Journal of Computer and Information Engineering* 12 (10): 815–822.
5. Hu, H., Y. Bai, and T. Xu. 2016. A whale optimization algorithm with inertia weight. *WSEAS Transactions Comput* 15: 319–326.
6. Goldbogen, J.A., A.S. Friedlaender, J. Calambokidis, M.F. Mckenna, M. Simon, and D.P. Nowacek. 2013. Integrative approaches to the study of baleen whale diving behavior, feeding performance, and foraging ecology. *BioScience* 63 (2): 90–100.

Chapter 7
Firefly Algorithm

7.1 Introduction

Fireflies are tiny winged beetles with high capability of producing natural light. The light produced by these little beetles is referred to as the cold light since the light radiated by fireflies produces little or no heat. Some characteristics of the firefly include the ability to discharge energy in form of light; the ability to flashlight to attract a mate; they have a capacitor-like mechanism, which gradually discharges until the threshold is reached.

Firefly Algorithm (FA) is a metaheuristic algorithm which is categorized as one of the fast-growing swarm intelligence algorithms. Based on the flashing pattern of light and their intelligent behavior, FA can solve problems in all fields of optimization and is very useful for solving complex mathematical and engineering problems. The algorithm was developed by [1] sometimes in 2007 and early 2008 while he was researching at Cambridge University. This nature-inspired algorithm has demonstrated a high level of superiority over several other algorithms. This metaheuristic algorithm is based on the blinking nature and actions of the firefly, especially during navigation or direction-finding. FA can be implemented to solve hard combinatorial and optimization problems.

Firefly can produce rapid and recurring sparkles of light through the process of bioluminescence. The blinking light produced from the firefly has a unique function which is basically to attract associates and for communication. It also serves as a substantial tool to attract possible prey to the firefly and finally, the light can serve as a defensive or protective caution while advancing towards the predator. Therefore, the intensity of light radiated by the firefly is a determinant factor as it determines if other fireflies should move towards or away from available firefly radiating light source.

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_7

The intensity of light radiated by the firefly is speckled at the distance from the eyes of the onlooker. From every observation, the light intensity radiated by firefly decreases as the distance increases and vice versa. The light intensity equally has a great impact on the air absorbed by the surroundings, consequently, the intensity becomes less attractive as the distance increases. The attractiveness variation β of firefly and the distance covered r, can be well-defined by observing thoroughly at the attractiveness of firefly as well as its proportionality to the intensity of light source perceived by end-to-end fireflies.

According to [2], FA has two principal advantages compared to other algorithms due to the instinctive or reflex subdivision of the algorithm and the capacity of FA to handle multimodality. The foremost and the interesting part of this algorithm has to do with the attraction, and it is a known fact that the attractiveness declines with distance. This explains the fact that the entire population under investigation or study can automatically be sectioned into subcategories, with the concept of individual categories swarming around the local optimum. With careful investigation of available modes, it is easy to select the global best solution. The second advantage recorded for FA over other algorithms has to do with the subcategories which enables the fireflies to find all targets or goals instantaneously, especially where the population size is adequately complex and more than the required number of modes. Pictorial view of the firefly convergence is shown in Figs. 7.1 and 7.2 present the life cycle of the firefly.

Additionally, FA parameters can be modified to control randomness in the process of iterations. The essence of this is to accelerate convergence which is achievable by changing the parameters as well. Such advantages make the algorithm robust and flexible as well as capable of solving NP-hard problems and complex combinatorial problems.

Fig. 7.1 Firefly convergence

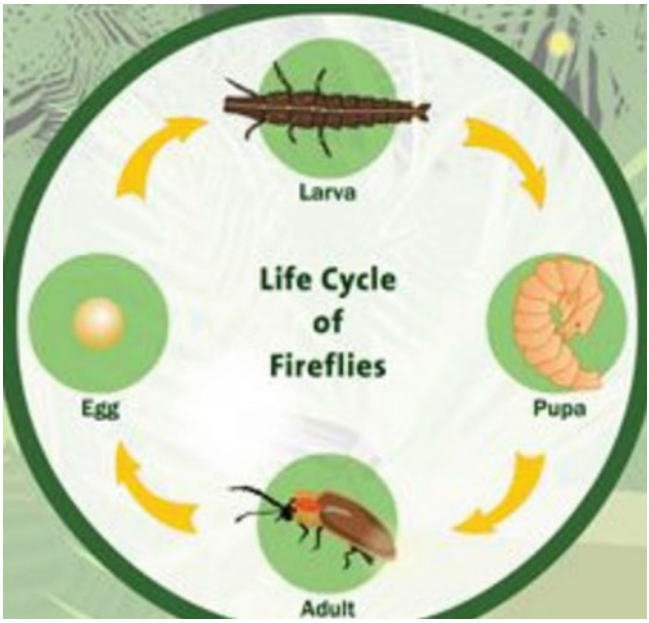


Fig. 7.2 Life cycle of a firefly

The firefly algorithm has three rules [1, 2]:

(i) All fireflies are unisex, implying that all the elements of a population can attract each other regardless of their sex, and the flashlight to attract their mate.

(ii) The attractiveness between fireflies is based on the brightness of light. Meaning that as the light intensity increases, there is a tendency of high level of attraction between fireflies. In other words, attractiveness amongst fireflies is proportional to their brightness. The firefly with less bright will move towards the brighter one. If no one is brighter than a firefly, it moves randomly. Attractiveness is proportional to the brightness which decreases with increasing distance between fireflies.

(iii) The light intensity or brightness of a firefly is based on the distance and highly determined by the objective function. In practice, the brightness of each firefly can be directly proportional to the value of the objective function. As a firefly's attractiveness is proportional to the light intensity seen by adjacent fireflies.

Advantages of Firefly Algorithm

- Firefly algorithm is capable of handling highly multi-modal and non-linear optimization problems logically and competently.
- Firefly algorithm does not require velocities.
- The convergence speed of the firefly algorithm is quite high in probability while handling global optimization problems.
- Firefly algorithm can be used as a hybrid technique due to its high level of flexibility, as it can easily integrate with other optimization techniques.
- A good initial solution is not required while starting the iteration process.
- With the right and required input variables in a dataset, the optimum solution can be obtained within the shortest possible time, using the Firefly algorithm.
- Firefly algorithm has a good knowledge-based sharing mechanism that can facilitate the algorithm to cluster beyond certain circumstances and the lower likelihood of entrapment into local optima.

Disadvantages of Firefly Algorithm

- The algorithm has additional control variables
- The algorithm requires a huge number of iterations
- The algorithm requires a proper setting of dependent parameters

Areas of Application of Firefly Algorithm

- Due to the stochastic nature of FA, the algorithm can be used in diverse areas
- The algorithm has additional control variables
- Optimization of process variables
- Solving routing problems
- Solving clustering related problems
- Scheduling of jobs in industrial, manufacturing, and service systems.

7.2 Stages of Implementation of Firefly Algorithm

(i) Measurement of the flashing light intensity of each firefly.
(ii) Compute distance between any two fireflies. Say x_i or y_i and x_j or y_j, at i and j respectively.
(iii) Calculate the attractiveness of the firefly corresponding to the distance between two fireflies. Say x_i or y_i and x_j or y_j.
(iv) The movement of firefly x_i or y_i which is more attracted to a firefly with higher intensity or firefly with brighter light x_j or y_j with current position and attractiveness function between adjacent fireflies.

First, each firefly generates an initial solution randomly; parameters like light intensity (I), Initial Attractiveness (β_o), and light absorption coefficient γ are defined. Then for each firefly, find the brightest firefly. If there is a brighter firefly then less bright firefly will move towards the brighter one. When a firefly moves, its light intensity decreases, and its attractiveness will change. Then best firefly will be chosen based on an objective function for the next iteration. This condition will continue until the max iteration is reached.

Key Parameters

- Light intensity and attractiveness β;
- Fireflies at horizontal location x_i;
- Fireflies at vertical location y_i;
- r_{ij} is the distance between firefly i and j;
- Where I_S is the intensity at the source.

Considering the firefly algorithm, there are two important issues to be taken into consideration: attractiveness formation of the firefly and the intensity of light variation. For straightforwardness and easy formation of the mathematical model, it will be presumed that the firefly attractiveness is a function of its intensity of brightness which in turn relates to the programmed objective function of the firefly model.

The brightness of the firefly (denoted by I) at a determined position or location x can be defined as presented in the system of Eq. (7.1). This can help address the simplest situation while considering optimization issues at the maximum level.

$$I(x)\alpha f(x) \tag{7.1}$$

Nevertheless, to effectively develop a mathematical model for the firefly algorithm, the attractiveness β is considered relative. The judgment based on attractiveness is a function of other fireflies. Consequently, this is likely to vary with distance or location with parameter r_{ij} which represents the difference in location between two fireflies, say i and j. The fact that the distance affect the light intensity means that the attractiveness must be adjusted as a function of the degree of absorption. Summarily, the intensity of light I(r) varies following the inverse square law.

$$I(r) = \frac{I_S}{r^2} \tag{7.2}$$

where I_S is the light intensity at the source.

Also, considering a medium with a static light absorption coefficient γ, the intensity of light I varies with the location or distance r. Therefore:

$$I = I_O e^{-\gamma r} \tag{7.3}$$

where I_O represent the actual intensity of light.

The firefly's attractiveness β can be approximated as follows:

$$\beta = \frac{\beta_O}{\left(1 + \gamma r^2\right)} \tag{7.4}$$

where β_O is the level of attractiveness at $r = 0$.

The distance (or separation) between two fireflies can be calculated. Assume that firefly i and firefly j are x_i and x_j from the vertical axis and y_i and y_j form the horizontal axis, the distance in-between them can be calculated as follows:

$$r_{i,j} = \sqrt{\left(x_i - x_j\right)^2 - \left(y_i - y_j\right)^2} \tag{7.5}$$

As stated earlier, the navigation of a firefly i is fascinated by another highly striking firefly with a brighter light say j. The movement of firefly i toward firefly j is represented mathematically as follows:

$$x_i = x_i + \beta_O e^{-\gamma r_{i,j}^2}\left(x_j - x_i\right) + \alpha\left(\text{rand} - \frac{1}{2}\right) \tag{7.6}$$

where rand is a random number in the interval [0,1], α is the coefficient of the random displacement vector, γ is the light absorption coefficient for the environment, and $r_{i,j}$ is the Euclidean distance between two fireflies.

7.3 Pseudocode of the Firefly Algorithim and Flowchart

The FA pseudo-code can be summarized as follows [3, 4]:

Define target function which is presented as: f(x): x = (x_1, x_2, ……. x_d)

Generate or develop preliminary or pilot population of fireflies: x_i, (i = 1…. n)

Define expression for intensity of light (1) so that it is linked with 1 = F(x)

Define the light adsorption, represented by y

While (t < maximum generation of light)

For i = I: n(for all fireflies in the sample space)

For j = I: n(for all fireflies in the sample space)

If (Ij > Ii)

Firefly i moves towards firefly j

End if

Express attractiveness of firefly based on the separation point (r) distance $\exp(-yr^2)$

Estimate original values and present the final value in terms of light intensity

End for

End for

Find the best possible firefly

End while

The optimization procedure flowchart of the firefly algorithm is shown in Fig. 7.3.

7.4 Application of FA Optimization Algorithm with a Numerical Example

A MATLAB code has been provided in the Appendix to illustrate the implementation of the FA algorithm for a typical optimization problem considering a numerical equation the following function:

$F(x) = ({x_1}^6 + {x_2}^4 - 17)^2 + (2x_1 + x_2 - 4)^2$ describing the Wayburen function.

The model considers 20 fireflies. The best results fbest = 8.9936e-13 corresponding to nbest = [1.0000 2.0000].

The FA MATLAB code used to solve this Equation has been provided in Appendix E (as described in References [2] and [3]) and a screenshot showing the results is shown in Fig. 7.4.

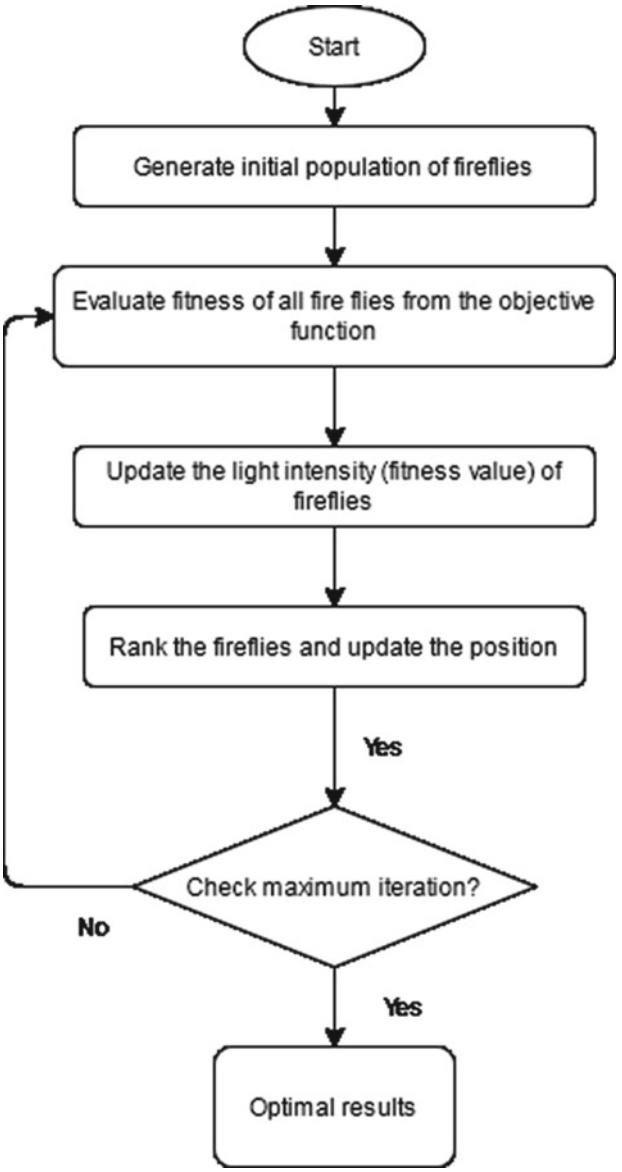


Fig. 7.3 Optimization procedure flowchart of firefly algorithm [1]

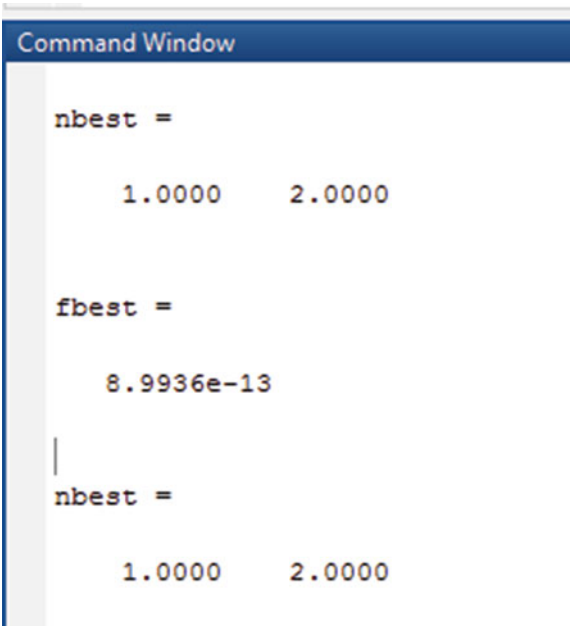


Fig. 7.4 Screenshot of the results

References

1. Yang, X. S. (2008). *Nature-Inspired metaheuristic algorithms.* Luniver Press.
2. Yang, X.S., and X. He. 2013. Firefly algorithm: Recent advances and applications. *International Journal of Swarm Intelligence* 1 (1): 36–50.
3. Yang, X.S. 2010. Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation* 2 (2): 78–84.
4. Arora, S., and S. Singh. 2013. The firefly optimization algorithm: convergence analysis and parameter selection. *International Journal of Computer Applications* 69(3).

Chapter 8
Bat Algorithm

8.1 Introduction

Bat algorithm is an innovative or population-based technique which belongs to the swarm intelligence. It is also referred to as a metaheuristic algorithm developed by Yang [1]. The bat algorithm as a unique algorithm provides a suitable solution technique than numerous and prevalent classical and heuristic algorithms. The algorithm is used for quick decision making and for solving complex problems in diverse fields of operations ranging from engineering, business, transportation, and other fields of human endeavor. It is important to understand the communication and navigational pattern of bats while defining the algorithm since the algorithm is based on the micro-bats echolocation (EL) [1]. EL is an enchanting and captivating sonar propensity produced bats. The appealing wave of sound made by the bats is a great strength they often exhibit while searching for prey. The wave of sound is a formula they adopt not only in food search but serves other purposes [2]. For instance, while searching for food in a completely dark environment, there might be obstacles or dangers on their way to the food source. This can easily be detected in some magical manner as they can sense and discern possible danger on their way to the food source as shown in Fig. 8.1 [3]. The structure and flying position of a bat is shown in Fig. 8.1. BA is therefore a very powerful algorithm as it uses the frequency-tuning methodology for range intensification of the solutions in the population. At the same time, the BA system implements the instinctive skyrocketing or zooming procedure to maintain stability during the food search or exploration process. In developing the BA, the pattern of direction finding, manipulation, and exploration during prey hunting is taken into consideration by mimicking the disparities in terms of emission rates of pulse and intensity of sound released by the bats while hunting or searching for prey. The algorithm demonstrates a high level of efficiency with a distinctive swift start process (Fig. 8.2).

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_8

Fig. 8.1 Structure and flying position of a bat

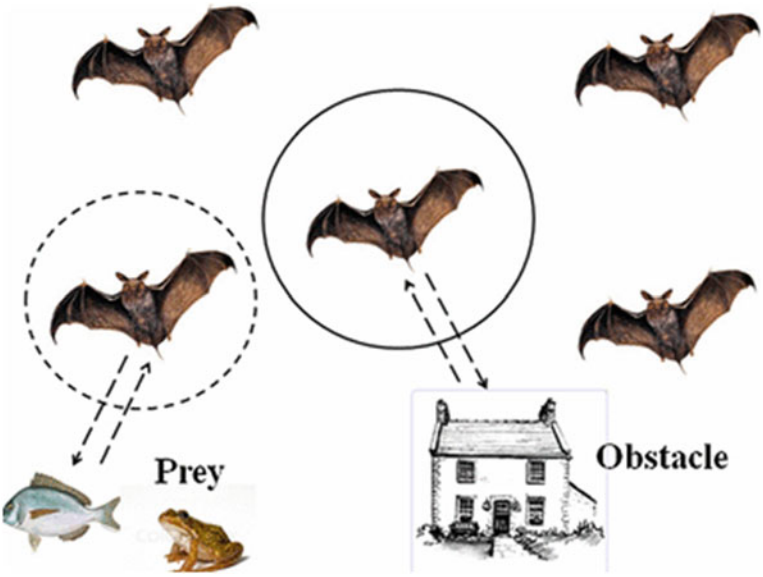


Fig. 8.2 Behavior and navigation pattern of bat

8.2 Idealized Rules of the Bat Algorithm

It is very important to note the rules associated with bats. Since bats emit loud ultrasonic sound waves and listen to the echo that reflects from the surrounding objects, the bat algorithm uses some idealized rules for uncomplicatedness. The following section points out some common rules of bat algorithm [1]:

- All bats employ the echolocation pattern to perceive sense or discern prey, barrier, distance, predator, or any form of obstacle during direction-finding, routing, or path navigation. This navigation pattern is performed in a very magical way, strange to man.
- Bats, in the course of direction-finding or path navigation fly haphazardly with a supposed velocity, v_i while maintaining position, x_i. They maintain a fixed frequency 'f' with varying wavelengths and loudness, A_i to reach their prey. They can adjust the frequency of pulse emission, r_i.
- As they get close to the prey, pulse increases, and loudness decreases.

The flow diagram describing the bat algorithm is shown in Fig. 8.3.

8.3 Example of Implementation of BAT Algorithm

Let's consider 3 bats, the Bat population = 3 ($i = 1, 2.3$), x_1, x_2 and x_3 represent respectively the position of the three bats.

Steps of the BAT algorithm implementation.

- Initialization of the frequency (f_i), the bat population (x_i), the loudness (A_i), the Pulse (r_i) and the velocity (v_i) with ($i = 1, 2, 3, \ldots n$).
  Initially ($t = 0$), we will assign random values to the frequency (f_i), position (x_i), loudness (A_i), Pulse (r_i) and velocity (v_i) as follows (Table 8.1):
- While t < Max number of iterations (t = 1, 2, 3, 4, 5, …), new solutions will be generated by adjusting frequencies, velocities, and locations.

BAT 1: Suppose that we consider 20 iterations for this example. Hence for the first iteration ($t = 1$), the condition is true. Therefore, new solutions will be generated by adjusting the frequency and updating velocity and position. The following equation will be used to adjust the frequency (NOTE: BAT can automatically adjust frequency).

$$f_i = f_{min} + \beta(f_{max} - f_{min}) \tag{8.1}$$

with the domain size of the problem $[f_{min}, f_{max}] = [0, 10]$ and $\beta \in [0, 1]$

$$v_i^t = v_i^{t-1} + \left(x_i^t - x_{gbest}^t\right) f_i \tag{8.2}$$

where x_{gbest}^t is the current best (nearest) position.

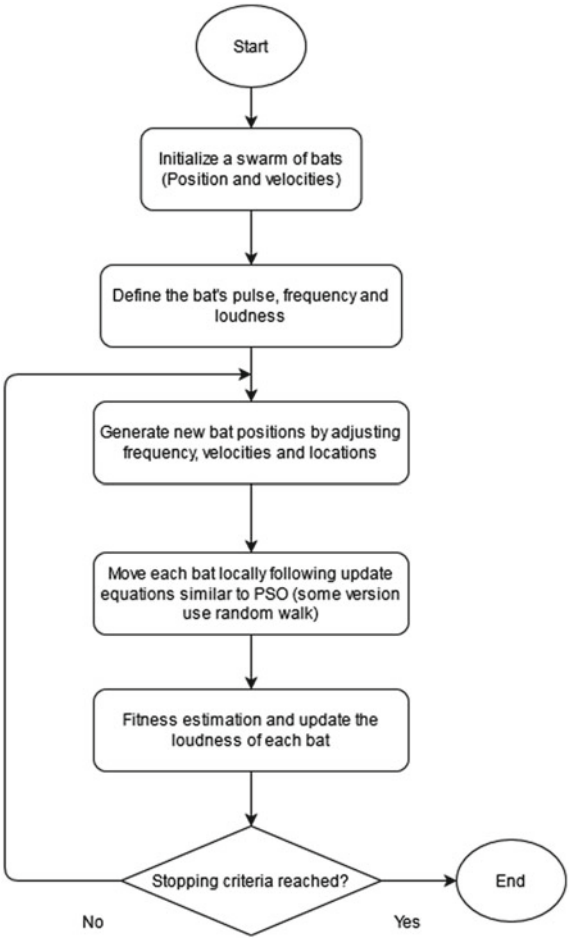


Fig. 8.3 Flow diagram of bat algorithm [1]

Table 8.1 Initial BAT parameters

BAT 1	BAT 2	BAT 3
$f_0 = 2$	$f_0 = 6$	$f_0 = 3$
$x_0 = 4$	$x_0 = 8$	$x_0 = 5$
$v_0 = 3$	$v_0 = 7$	$v_0 = 4$
$A_0 = 1$	$A_0 = 0.8$	$A_0 = 0.9$
$r_0 = 0$	$r_0 = 0.2$	$r_0 = 0.1$

$$x_i^t = x_i^{t-1} + v_i^t \tag{8.3}$$

- If the random value rand > r_i, selection of a solution among the best solutions can be done. Local solution around the best solution can be generated.

Table 8.2 Updated BAT parameters

BAT 1	BAT2	BAT3
$f_1 = 2 + 1(8-2) = 8$	$f_2 = 3 + 1(7-3) = 7$	$f_3 = 1+(5-1) = 5$
$x_1 = 4 + 35 = 39$	$x_2 = 8 + 63 = 71$	$x_3 = 5 + 29 = 34$
$v_1 = 3 + (4-0)8 = 35$	$v_2 = 7 + (8-0)7 = 63$	$v_3 = 4 + (5-0)5 = 29$
$A_1 = 1$	$A_2 = 0.8$	$A_3 = 0.9$
$r_1 = 0$	$r_2 = 0.2$	$r_3 = 0.1$

For the three bats, r_1, r_2 and r_3 are respectively equal to 0, 0.2 and 0.1. We will choose the random value between [0, 1], let's say 0.5. For BAT 1, 0.5 > 0: TRUE, for BAT 2, 0.5 > 0.2: TRUE and for BAT 3, 0.5 > 0.1: TRUE. If the condition is true, we will select a solution among the best solution.
How to select the best solution? According to the frequency: Best solution = if the prey is in the scope of frequency increase. The frequency of the waves increases if the BAT finds the prey (i.e. best solution). Referring to Table 8.2, the highest frequency corresponds to BAT 1, which is the best solution in this case.
We can generate a local solution around the best solution. We will update the position of the first BAT using the following equation:

$$x_{new} = x_{old} + \varepsilon A^t \tag{8.4}$$

with $\varepsilon \in [-1, 1]$

$$x_{new} = 35 + 1.1 = 36$$

- If rand < A_i and $f(x_i) < f(x_*)$ accept the solution and increase r_i and reduce A_i (as the BAT moves closer to the prey, the loudness decreases and pulse rate increases). $f(x_i)$ and $f(x_*)$ depend on the new BAT location and the best solution depending on the function. If the solution is TRUE, we will increase r_i and reduce A_i using the following equations:

$$r_i^{t+1} = r_i^0\left[1 - e^{\gamma t}\right] \tag{8.5}$$

$$A_i^{t+1} = \propto A_i^t \tag{8.6}$$

where the two constant $\propto$ and γ are in the range [0, 1].
Let's assume $\propto = \gamma = 0.9$, the updated values are:

BAT 1	BAT2	BAT3
$f_1 = 8$	$f_2 = 7$	$f_3 = 5$
$x_1 = 36$	$x_2 = 71$	$x_3 = 34$
$v_1 = 39$	$v_2 = 63$	$v_3 = 29$
$A_1 = 1$	$A_2 = 0.8$	$A_3 = 0.9$
$r_1 = 0.4$	$r_2 = 0.2$	$r_3 = 0.1$

- We can now rank the BAT and find the current best. We will check the loudness first and the frequency. Based on that, the first BAT has the highest loudness and frequency. It is the current best solution.

8.4 Application of BAT Optimization Algorithm with a Numerical Example

A MATLAB code has been provided in the Appendix to illustrate the implementation of the BAT algorithm for a typical optimization problem considering a numerical equation $F(x) = ({x_1}^6 + {x_2}^4 - 17)^2 + (2x_1 + x_2 - 4)^2$ describing the Wayburen function.

The model considers 10 bats and the maximum number of iterations was 1000. The best solution obtained by BAT is: [1.3138 1.8528 0.261 0.83905 0.34859 1.1436 0.73859 1.7623 0.16537 0.28717]. The best optimal value of the objective function found by BAT is: 0.23604. The parameter space and objective space are shown respectively in Figs. 8.4 and 8.5. The MATLAB code is provided in Appendix F. A detailed description of the CODE is provided in Ref. [1].

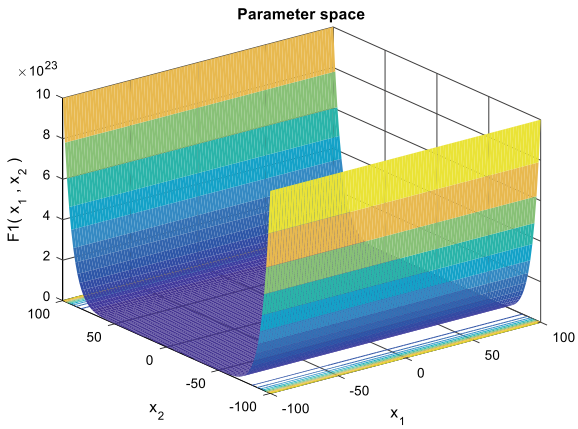


Fig. 8.4 Parameter space

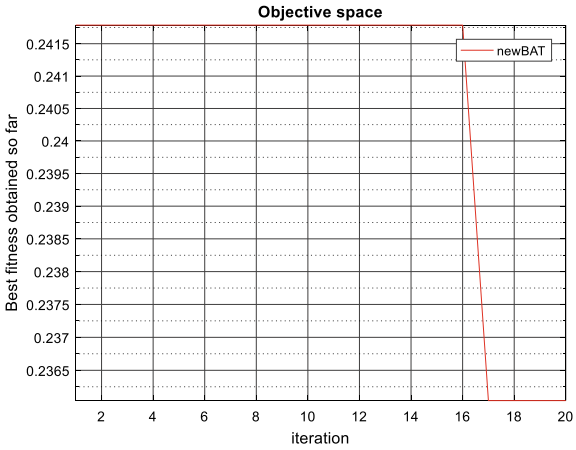


Fig. 8.5 Objective space

8.5 Application of BAT Optimization Algorithm for Traffic Congestion Problem

Case Scenario

A number of problems can be caused by congested traffic. Congestion is mainly caused by intensive use of vehicles. Traffic congestion in Delta State is a big challenge especially in the morning and night hours. Individuals who navigate to workplace in the morning hours (8 a.m.) encounter this recurring traffic congestion, leading to delay, high fuel consumption, environmental pollution and other operating costs. Also, returning home after work (4 p.m.) could be challenging too. This issue is common at certain locations in the state (PP junction in DSC and Mofor junction). Another location spotted with a high level of traffic congestion is the Udu road to Enerhen junction. From observation, the roads are very narrow to allow uninterrupted traffic flow.

Observation

It was observed that the transportation system in the state was poorly planned, as alternative routes were not mapped out for easy access, and to further congest the road, marketers and roadside sellers occupy the vast extension of the road. All these led to traffic congestion in a section of the state under un-signalized circumstances.

Network Architecture

To address the problem, road network can be taken as a directed graph G = (N, a), where 'N' is the set of nodes; i.e., the road junctions 'a' is the links connecting the junctions as shown in Fig. 8.6.

For each pair of origin (O) and destination (D), (O-D) signify a nonnegative travel demand, d_{rs}. The road network can represent a highly connected graph, where each node "i" is accessible by another node "j" following the directed path of the network

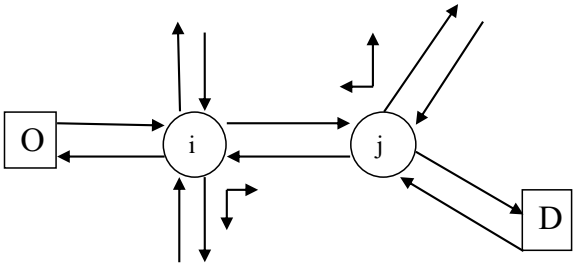


Fig. 8.6 An intersection of the network showing an O-D pair connected by a 3-way and a 4-way junction

'N'. It is assumed that the links connecting the nodes have a travel time function t_a, for the assigned rate of flow x_a.

This study aims to mimic the bat algorithm for solving traffic congestions issues in the state.

The major objectives include to:

(i) choose possible shortest route to travel from source to destination while maintaining uninterrupted traffic flow;
(ii) reduce the traffic delay at each junction under un-signalized circumstances.

The continuous network design model is chosen with budget constraints for the link capacity expansion. The objectives are interdependent and can be formulated as a bi-level problem. The upper level is responsible for reducing the travel time of the assigned road users or travellers. The lower level is the traffic assignment model which estimates the travelers flow. The model is presented clearly in Fig. 8.7.

This model can be formulated mathematically as shown below for both the layers.

Upper-Level Function

$$\min T(y) = \sum_{a \in A} \sum_{a \in A(y)} x_a t_a(x_a) + d_a y_a \tag{8.7}$$

$$\text{Such that, } \sum c_a y_a \leq B$$

where A is the set of all links 'a' in the network 'N'.

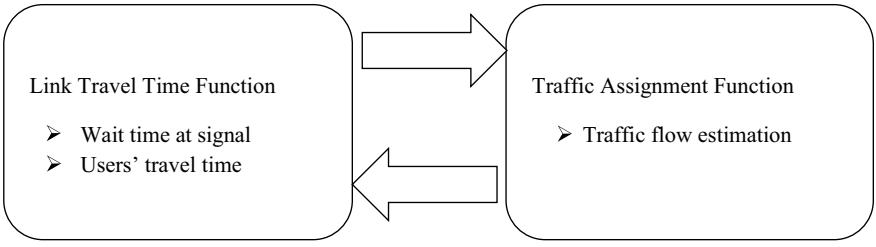


Fig. 8.7 Bi-level model for traffic network determination problem

X(a) gives the user equilibrium flow, which is estimated from the lower level of the model for the assigned value of link capacity 'y'. c_a is construction cost for link 'a' and B is the budget.

Lower Level Function

$$\min \sum_{a \in A} \int_0^{x_a} t_a(u)du \tag{8.8}$$

$$\text{Such that,} \sum f_k = d_{ij} \quad f_k \geq 0$$

$$\sum_{rs} \sum f_k \delta_{ij} = x_a \forall \delta_{ij} = \{_{2,}^{1;} \text{ifa} \in k$$

where f is the flow on path k and r-s are the nodes on the path connecting O-D pair.

The flowchart for the traffic signal optimization from a BA perspective is shown in Fig. 8.8.

Solution Procedure

Using Delta State, Udu road as a case study from the Enerhen Junction to Orhuwhorun Junction. The nodes and links are presented in Fig. 8.9.

Model Notation and Formulations

- No of links, $a = 2$
- No of nodes/junctions, $N = 3$
- The possible number of networks in this case is given as 2^n
  $2^3 = 8$ possible number of networks
- Budget constraint, B = 500,000 NGN
- Upper-level function is considered
- First, according to the flow chart, the necessary parameters are considered
- Distance by car is 22 km
- Time taken is to navigate—33 min
- Assuming the following:
  - $X_a = 0.67$ km/min
  - $T_a = 33$ min
  - Y_a (link capacity) = 50
  - d_a (demand) = 300
  - c_a = 50,000 NGN per link

  Notice that:

$$\sum c_a y_a \leq B$$

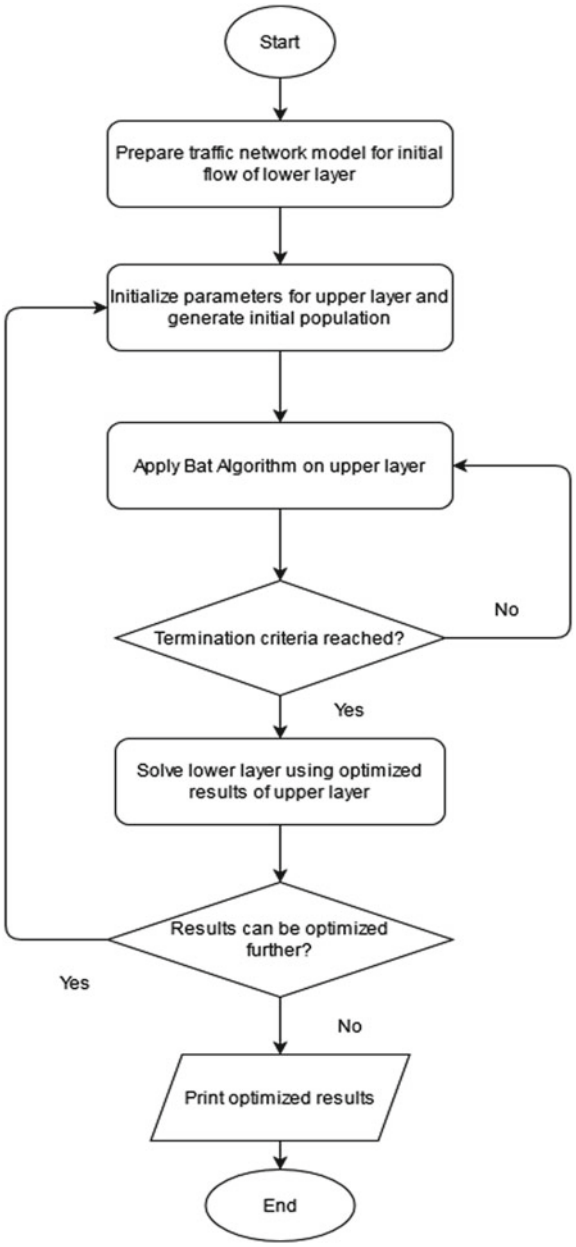


Fig. 8.8 Flow chart for the traffic signal optimization problem using bat algorithm

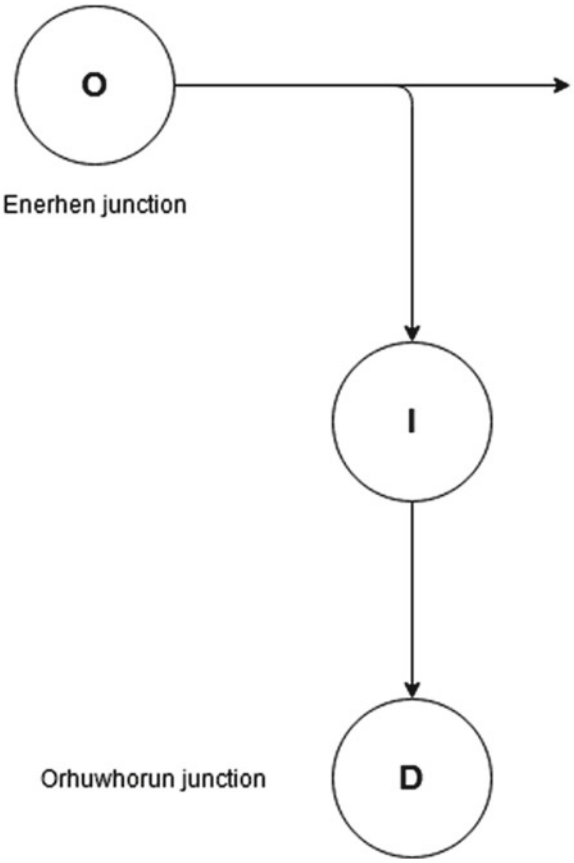


Fig. 8.9 Road network for three nodes and two links

$$\sum ₦50000 \times 50 = ₦2500000 \leq ₦4000000$$

Since the above condition is satisfied, we proceed;

$$\min T(y) = \sum_{a \in A} \sum_{a \in A(y)} x_a t_a(x_a) + d_a y_a$$

$$\sum_{a \in A} \sum_{a \in A(y)} 0.67 \times 33 \times 0.67 + (300 \times 50)$$

This gives min T(y) = 15,014.8137.
Bat algorithm is applied at the upper-level function.
Defining objective function f(x) and assigning the parameters.

- Bat population (demand), $x_i = 300$

- Velocity, $v_i = 0.67$ km/min
- Loudness, $a_i = 50{,}000$ (cost of link construction)
- Pulse rate, $r_i = 50$ (link capacity)
- Frequency, $f_i = 0.030$.

Solution Method

- Let the bats stationed for direction-finding represent the available vehicles in search space or road network. The map reading or course-plotting is made easy.
- Since the number of vehicles passing the road has been estimated to be roughly 300 periodically and the link capacity is 50 per link, then the total capacity for both links is $50 \times 2 = 100$.
- The roads mapped for the study is such that 100 vehicles can be allowed for easy navigation.
- 300 vehicles use the road frequently, a random value is assigned to the pulse rate or link capacity.
- For the first iteration, it is assumed that the link capacity or pulse rate is 70 per link. This makes a total of 140 vehicles for both links.
- According to the initially defined condition, Rand $> r_i$; $70 > 50$ satisfies the condition.
- It is important to generate a new solution around the defined condition.
- To achieve this, assume the link capacity (pulse rate) to be 75, making the demand for both links equal to 150 vehicles.
- Since the link capacity has been increased, the time rate of flow will reduce. So that t_a equal 31 min; f_i equal 0.032 and velocity, v_i equal 0.71 km/min. The cost of construction (loudness) increases to 90,000 NGN.
- For the second iteration, assume the link capacity is 100, therefore, demand for both links equal 200 vehicles.
- Recall the established condition, Rand $> r_i$; $100 > 50$ satisfies the condition, it is important to store and generate a new solution around the stated condition.
- Assume the link capacity to be 105, making the demand for both links equal to 210.
- Again, the time rate of flow will automatically increase as the link capacity has reduced.
- So, t_a equals 29 min; f_i equals 0.034 and v_i equals 0.76 km/min.
- The cost of construction (loudness) increases to 100,000 NGN.
- For the third iteration, let Rand equal 140, making the demand for both links equal to 280 vehicles.
- Following the established condition, Rand $> r_i$; $140 > 50$ satisfies the stated condition. It is important to store and generate a new solution around the stated condition.
- Considering the link capacity of 145. The demand for both links becomes 290. With increase in the link capacity, the time rate of flow will increase to $t_a = 27$ min; $f_i = 0.037$ and $v_i = 0.81$ km/min. The construction cost (loudness) equals to 150,000 NGN.

- Finally, for the last iteration, Rand equals 155, and the demand for both links equals 310. Since Rand > r_i; 155 > 50. Since the condition is satisfied, a new solution is generated.
- With a pulse rate of 160, the demand for both links becomes 320 vehicles. The time flow rate increases to 25 min, at f_i of 0.040, and v_i equal 0.88 km/min. The construction cost (loudness) increases to 200,000 NGN.

Calculating the fitness of generated population, the table is presented thus:

Population/demand	Capacity links
100	50
150	75
210	105
290	145
320	155

Comparing obtained information using the condition established thus:

- Rand < a_i and $f(x_i) < f(x)$
- $a_i = 50{,}000$ and $f(x_i) = 0.030$.

For the first iteration, the random loudness was ₦90,000 which does not satisfy the condition since it is greater ₦50,000. But, $f(x_i)$, 0.030 is less than f(x) which are, 0.032, 0.034, 0.037 and 0.040, so the results are shown and ranked in the following table.

Population/demand	Capacity per link	Velocity (v_i) (km/min)	Time (t_a) (min)	Frequency (f_i)
100	50	0.67	33	0.030
150	75	0.71	31	0.032
210	105	0.76	29	0.034
290	145	0.81	27	0.037
320	155	0.88	25	0.040

At this stage, the termination criteria is reached. Optimal result is generated by creating a road with the link capacity of 155 vehicles so that a total of 320 vehicles can freely navigate through, from the origin to destination at a faster rate, 0.88 km/min and for a lesser time, 25 min. The construction of this road will only cost 200,000 NGN per link summing up to a total amount of ₦400,000 which is clearly within the assigned budget.

So having solved the problem of time, the lower-level function can be obtained by using optimized results

$$\sum_{a \in A} \int_0^{X_a} t_a(u)du$$

where, $t_a = 25$ min, $x_a = 0.88$ km/min.

Substituting in the above equation,

$$\sum \int_0^{0.88} 25u \cdot du$$

This results in total value of 9.68 which represent the traveller's flow. The result cannot be optimized further so that the construction budget is not exceeded.

By mimicking the bat algorithm and following the echolocation pattern of bats for easy location of prey and avoiding obstacles, a transportation network problem has been solved in Delta State, Udu road. The following parameters have been considered: population of bat (of the vehicles that passes the roads), velocity, loudness (construction cost for link), and pulse rate (link capacity), corresponding to the bat behavior. The problem of travel time has been optimized to allow uninterrupted traffic flow under un-signalized situation at Udu road. After four iterations, best solution was obtained with maximum link capacity of 160 to accommodate vehicles travelling that route on a daily basis.

References

1. Yang, X.S. 2010. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, 65–74. Berlin, Heidelberg: Springer.
2. Yang, X.S. 2011. Bat algorithm for multi-objective optimisation. *International Journal of Bio-Inspired Computation* 3 (5): 267–274.
3. Rizk-Allah, R.M., and A.E. Hassanien. 2018. New binary bat algorithm for solving 0–1 knapsack problem. *Complex and Intelligent Systems* 4 (1): 31–53.

Chapter 9
Ant Lion Optimization Algorithm

9.1 Introduction

Ant lion optimization (ALO) is an innovative optimization technique inspired by nature and was first proposed by [1]. The ALO algorithm as a biomimicry replicates the hunting pattern of the ant lions in a real-life scenario. Following the hunting nature of an ant lion to develop the algorithm is a five-step process. Ant lion has five main strategies of hunting prey. The first process often implemented by an ant lion is the random walk technique of the ants. The second has to do with the process of building traps for prey. The third process is the technique of stinging and entrapment of ants in traps. The fourth process has to do with holding down or catching prey and the last is the re-building of traps to hunt down prey. Strong and robust mathematical equations of the ALO has been developed by mimicking the behavior of the ALO in real life. This is very useful for solving problems in a diverse area of human life. The life cycle of the ant lion is shown in Fig. 9.1. Ant Lion Optimizer (ALO) can be used to solve NP-hard problems, combinatorial optimization problems like scheduling problems, transportation and transshipment problems, and other path planning problems. Ant lion algorithm is very useful in combinatorial and stochastic operations. Especially for: parallel machine scheduling, maintenance, cost-effective integrated maintenance scheduling in power systems, optimal sizing and siting of distributed generation, gear train design problem, ship propeller design, cantilever beam design problem, optimal scheduling of thermal units for small scale electrical economic power dispatch problem and many more areas of operations.

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_9

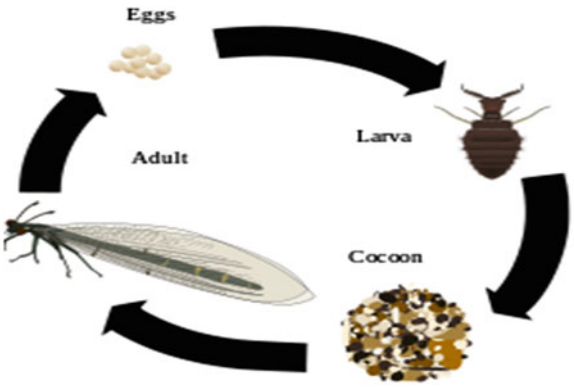


Fig. 9.1 Life cycle of the ant lion

9.2 Hunting Nature of Ant Lion

Antlions are a group of species of insects in the family Myrmeleontid, known for their dangerous hunting technique. They dig holes in a cone-like shape and this insect goes down to the bottom of the hole and places its jaw at that end to ensure proper eating of prey falling into the trap [2, 3]. When ants come across the hole and tend to walk into the hole, it is a sign that the trap was going to work. In another case, the ant notices the trap and would want to walk back up but at this point, the antlion would throw sand up to the top of the hole to slip the ant back downward towards the antlion. This system of feeding/hunting has been known to be very effective in their feeding and in choosing their chain of command. The fully-grown ant lion insects are not too popular since they frequently fly at dusk or subsequently in dark areas. They are often time identified wrongly as other flies like the damselflies or dragonflies. They are sometimes known as antlion lacewings. The larvae of these interesting insects are occasionally referred to as doodlebugs in countries like North America because of the strange marks they leave in the sand.

Antlions are found in almost every part of the globe. They are known to have wide-reaching circulation. They are very common in the tropics, but a few of the classes of ant lion are available in cold-temperate regions of the world. They most commonly occur in dry and sandy habitats. Such habitats are usually very soft to allow the larvae to excavate their pits easily. In cases the sands are so hard, the larvae prefer to hide beneath or underneath fragments and debris while other sets of species prefer to ensnare their prey beneath chaotic, muddled and messy leaves. The hunting strategy of the antlion is shown in Fig. 9.2.

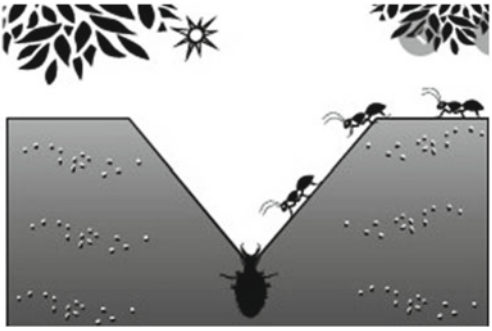


Fig. 9.2 Hunting strategy of the ant lion by creating a cone-shaped trap [1]

9.3 Model Equation of the Antlion Algorithm

In a research conducted by [4], equations mimicking the haphazard or random movement of ants while sourcing for food was developed as described in the following equations:

$$X(t) = [0, \text{Csum}(2r(t_1) - 1), \text{Csum}(2r(t_2) - 1), \ldots, \text{Csum}(2r(t_n) - 1)] \quad (9.1)$$

where

Csum denote the cumulative sum,
n represents the maximum number of iterations,
t denotes the step of random walk,
r(t) represent the stochastic function defined by Eq. (9.2)

$$r(t) = \begin{cases} 1 \text{ if rand} > 0.5 \\ 0 \text{ if rand} \leq 0.5 \end{cases} \quad (9.2)$$

rand is a generated random number following unchanging distribution in the range [0, 1].

Ant position are saved in matrix format as follows:

$$M_{Ant} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,d} \end{bmatrix} \quad (9.3)$$

M_{Ant} represent each ant position matrix,

A_{ij} represent the value of j-th variable of the i-th ant,

n represent the number of ants,

d represents the number of variables.

Each ant is evaluated using a fitness function. M_{OA} and $M_{Antlion}$ show respectively the fitness functions of all ants and the matrix describing the position of each antlion.

$$M_{OA} = \begin{bmatrix} f([A_{1,1}, A_{1,2}, \ldots, A_{1,d}]) \\ f([A_{2,1}, A_{2,2}, \ldots, A_{2,d}]) \\ \vdots \\ f([A_{n,1}, A_{n,2}, \ldots, A_{n,d}]) \end{bmatrix} \tag{9.4}$$

where n is the number of antlions and d is the number of variables.

$$M_{Antlion} = \begin{bmatrix} AL_{1,1} & AL_{1,2} & \ldots & AL_{1,n} \\ AL_{2,1} & AL_{2,2} & \ldots & AL_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ AL_{n,1} & AL_{n,2} & \cdots & AL_{n,d} \end{bmatrix} \tag{9.5}$$

$AL_{i,j}$ represent the value of j-th variable of the i-th antlion.

The matrix M_{OAL} is made of fitness of each antlion. It is represented as follows:

$$M_{OAL} = \begin{bmatrix} f([AL_{1,1}, AL_{1,2}, \ldots, AL_{1,d}]) \\ f([AL_{2,1}, AL_{2,2}, \ldots, AL_{2,d}]) \\ \vdots \\ f([AL_{n,1}, AL_{n,2}, \ldots, AL_{n,d}]) \end{bmatrix} \tag{9.6}$$

The random walks of ants is represented mathematically with the following normalized Equation:

$$X_i^t = \frac{\left(X_i^t - a_i\right) \times (d_i - c_i^t)}{(d_i - a_i^t)} + c_i \tag{9.7}$$

where

$$c_i^t = \text{Antlion}_j^t + c^t \tag{9.8}$$

$$d_i^t = \text{Antlion}_j^t + d^t \tag{9.9}$$

and

c^t is the smallest value of all variables of t-th iteration,

d^t is the largest value of all variables at t-th iteration,

Antlion_j^t is the position of the selected j-th antlion at t-th iteration.

The hunting capability of Antlions can be modeled by adopting the fitness comparative roulette wheel selection process. The model which describes the pattern in which

ants are trapped by the antlion and how the slide down during towards the antlion during the trapping process is presented mathematically in system of Eqs. 9.10 and 9.11.

$$c^t = \frac{c^t}{I} \tag{9.10}$$

$$d^t = \frac{d^t}{I} \tag{9.11}$$

where I is a ratio calculated as:

$$I = 10^w \cdot \frac{t}{T} \tag{9.12}$$

where t represent the present reiteration process,
T is the highest value or number of iterations,
w is the constant that is subject to existing iterations as follows.
And

$$w = \begin{cases} 2 \text{ if } t > 0.1T \\ 3 \text{ if } t > 0.5T \\ 4 \text{ if } t > 0.75T \\ 5 \text{ if } t > 0.9T \\ 6 \text{ if } t > 0.95T \end{cases} \tag{9.13}$$

Elitism has been applied to model the ALO algorithm. Elite is considered as the best antlion in each process of iteration. This means that every ant in the system walks randomly from one location to another within the selected antlion space and has a position as presented in the system of Eq. 9.14:

$$Ant_i^t = \frac{R_A^t + R_E^t}{2} \tag{9.14}$$

where R_A^t is the haphazard movement of ant around the antlion carefully chosen by the roulette wheel at t-th iteration, and R_E^t is the haphazard movement of ant from point to point haphazardly around the elite antlion at t-th iteration.

9.4 Pseudo-Code of Antlion

The step by step approach is described as follows:

Step 1. The first step in implantation of the antlion algorithm is to initialize the preliminary ant population as well as that of antlions randomly

Step 2. The second step is to calculate the fitness value of ants and antlions respectively
Step 3. The next process is to find the best or feasible antlions which is assumed to be the elite (selected as optimum)
No satisfaction for the end criterion
for all ant represented in the system
Step 4. An Antlion is carefully selected using Roulette wheel
Step 5. c and d are updated using well-defined equations
Step 6. A random walk is created and normalized
Step 7. The position of ant is updated
end for
Step 8. The fitness value of all ants is calculated
Step 9. The next step is to substitute an antlion using its equivalent ant to obtain a perfect fit
Step 10. The next step is to update the elite for a fitted antlion. The antlion must fit properly compared to the elite
end while
Return elite
Step 11. The fitness of all ant is calculated
Step 12. Ant lion is replaced with its equivalent ant if and only if it becomes fitter
Step 13. Finally elite is updated when antlion becomes fitter than the elite
end while.
Return elite.

9.5 Application of Ant Lion Optimization Algorithm with a Numerical Example

Minimization of Booth function

$$f(x_1, x_2) = \min\left[(x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2\right]$$

This model was implemented in MATLAB using the **Ant Lion Optimization (ALO) Algorithm** described by Mirjalili [1]. To illustrate the approach, 40 search agents were considered (n = 30) and the maximum number of iterations was set to 1000. The best solution obtained by ALO is: [1, 2] and the best optimal value of the objective function found by ALO is 2.393e−14. The test function and the convergence curve are shown in Figs. 9.3 and 9.4. The MATLAB code is provided in Appendix G.

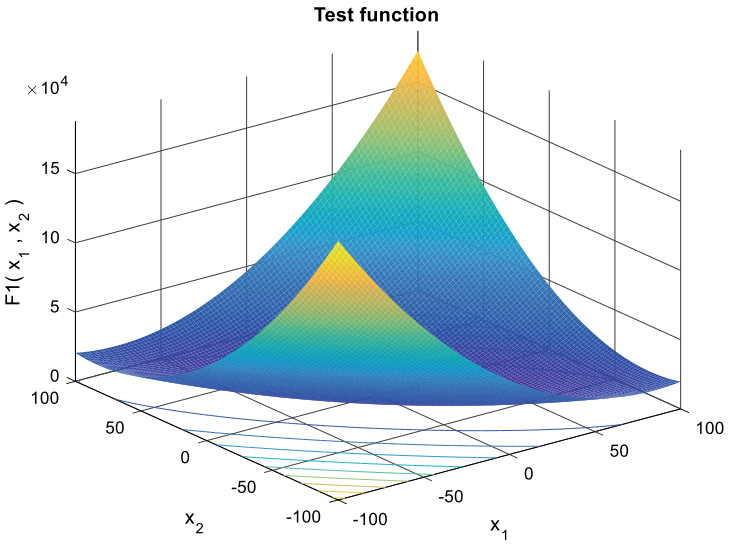
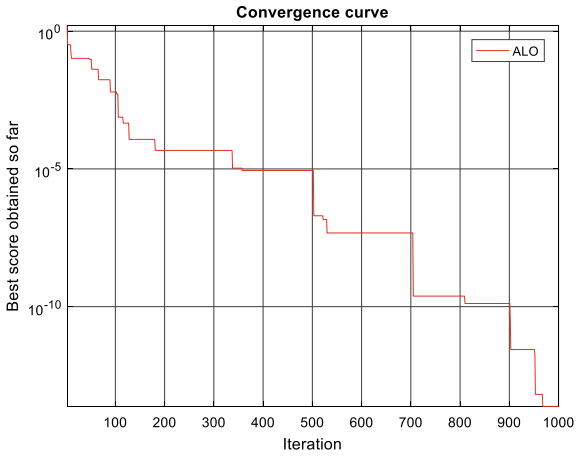


Fig. 9.3 Objective space

Fig. 9.4 Convergence curve



9.6 Application in Heat Recovery Steam Generator (HRSG)

Figure 9.5 shows the flow distribution through the evaporator and the economized together with the T-S diagram.

Optimization of heat recovery steam generator

The entropy generation for the HRSG model can be expressed as

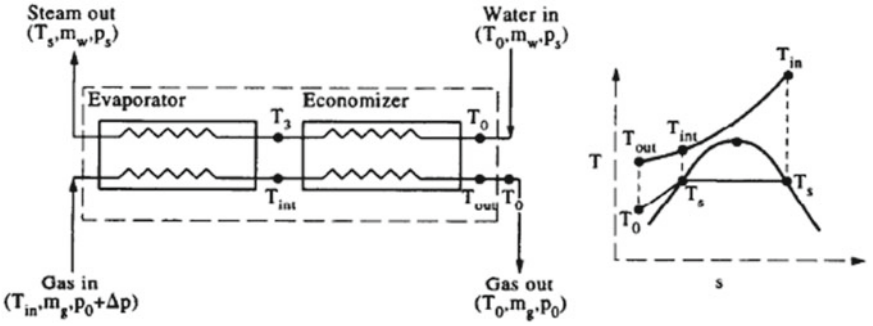


Fig. 9.5 The model's flow and T-S diagrams [5]

$$S = mg \cdot Cg \cdot \ln(To/Tin) + mg \cdot Rg \cdot \ln(1 + \nu p/p) + \left[Mg \cdot Cg \cdot (Tout - Tin)\right]/To + mg \cdot Cw \cdot \ln(Ts/To) + (mw \cdot hfg)Ts$$

where,

- S = Entropy (kJ/kg K)
- Rg = Specific Gas Constant (kJ/Kg K)
- C = Specific heat (kJ/kg K)
- mg = Mass Flow Rate of exhaust gas (kg/s)
- mw = Mass Flow Rate of water (kg/s)
- Ts = Steam Temperature (K)
- To = inlet Temperature to Economizer (K)
- Tin = Inlet Temperature of Exhaust Gas (K)
- Tout = Outlet Temperature of Exhaust Gas (K)
- hfg = Enthalpy of Evaporation (kJ/K)

Assumptions

According to [5], the following assumptions are considered

(1) The system is in steady-state
(2) There is no pressure drop in the Steam/Waterside
(3) There is no extraneous heat or mass transfer
(4) The specific heats of exhaust gas and water are constant.

Plant Design Parameters

Using the design parameters of the HRSG Plant of a typical Refining and Petrochemical company, the following was obtained as the minimum entropy zone.

Input parameters used

- Rg = 0.265 kJ/kg K
- Cg = 1.996 kJ/kg K

- Mg = 0.44 kg/s
- Mw = 10.72 kg/s
- To = 393 K
- hfg = 2260 kJ/K.

Antlion Optimization Output

- At iteration 5 the elite fitness is 58.8361
- At iteration 10 the elite fitness is 58.8361
- At iteration 15 the elite fitness is 58.7828
- At iteration 20 the elite fitness is 58.7617
- At iteration 25 the elite fitness is 58.7608
- At iteration 30 the elite fitness is 58.7565
- At iteration 35 the elite fitness is 58.7565
- At iteration 40 the elite fitness is 58.746
- At iteration 45 the elite fitness is 58.7394
- At iteration 50 the elite fitness is 58.7384
- At iteration 55 the elite fitness is 58.7384
- At iteration 60 the elite fitness is 58.7383
- At iteration 65 the elite fitness is 58.7383
- At iteration 70 the elite fitness is 58.7383
- At iteration 75 the elite fitness is 58.738
- At iteration 80 the elite fitness is 58.738
- At iteration 85 the elite fitness is 58.738
- At iteration 90 the elite fitness is 58.738
- At iteration 95 the elite fitness is 58.738
- At iteration 100 the elite fitness is 58.738
- The best solution obtained by ALO is: 723.15 353.15 539.413
- The best optimal value of the objective function found by ALO is: 58.738.

Hence, the results from ALO are:

- Tout = 353.15 K
- Tin = 723.15 K
- Ts = 539.413 K
- Min. Entropy S = **58.738** kJ/Kg K.

The screenshot of the convergence curve is shown in Fig. 9.6.

The Ant Lion Algorithm Optimization showed that it is theoretically possible to obtain an outlet steam temperature of 539.413 K (266.413 °C) and an Entropy of 58.738 kJ/kg K when the Flue gas temperature from the Gas Turbine into the WRPC's Heat Recovery Steam Generator (HRSG) is 723.15 K (450.15 °C) and the temperature of Feed Water is 393 K (120 °C).

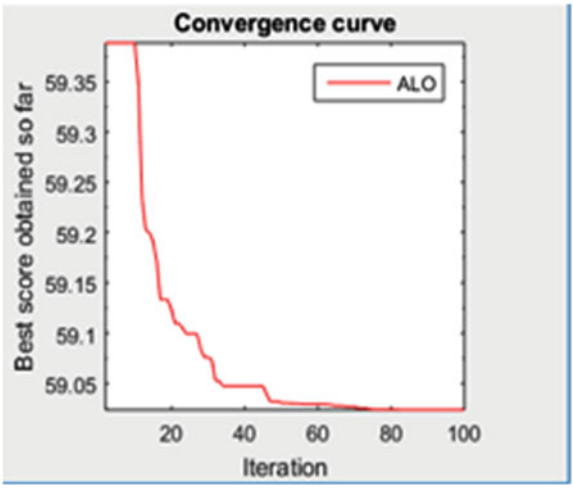


Fig. 9.6 Convergence curve

References

1. Mirjalili, S. 2015. The ant lion optimizer. *Advances in Engineering Software* 83: 80–98.
2. Scharf, I., A. Subach, and O. Ovadia. 2008. Foraging behaviour and habitat selection in pit-building antlion larvae in constant light or dark conditions. *Animal Behaviour* 76 (6): 2049–2057.
3. Griffiths, D. 1986. Pit construction by ant-lion larvae: A cost-benefit analysis. *The Journal of Animal Ecology* 39–57.
4. Petrović, M., J. Petronijević, M. Mitić, N. Vuković, A. Plemić, Z. Miljković, and B. Babić. 2015. The ant lion optimization algorithm for flexible process planning. *Journal of Production Engineering* 18–18 (2).
5. Nag, P.K., and S. De. 1997. Design and operation of a heat recovery steam generator with minimum irreversibility. *Applied Thermal Engineering* 17 (4): 385–391.

Chapter 10
Grasshopper Optimisation Algorithm (GOA)

10.1 Introduction

Grasshopper Optimization algorithm is one of the newly introduced algorithms by Saremi et al. which is gaining so much attention in the literature of computational intelligence. Grasshoppers are usually found on farmland and grasses and are usually categorized as one of the chomping herbivorous creatures or insects. They are very flexible and possess drawn-out hind tiny legs for skipping and jumping from one place to another on farmland and grasses. In some cases, since they feed on grasses, they are usually seen as pests since they damage crops on farmland. Figure 10.1 is a pictorial view of the grasshopper and Fig. 10.2 represents the incomplete metamorphosis of the lifecycle of grasshopper. They have a way of communicating and can converge together. The swarming ability of the grasshopper makes them unique herbivorous insects. When the group on farmland, they can destroy crops on farmland in a terrible manner. According to Rogers et al. [1], grasshoppers have a unique feature of swarming which is common in the nymph and adulthood. Millions of grasshoppers at nymphs' leap and move like spinning cylinders. They feed on nearly every vegetation on their route. After that action, they form a swarm in the air when they become adults. That is how big-distance grasshoppers move. Slow-motion and quick movements of the grasshoppers are the key characteristics of the swarm in the larval process. Saremi et al. [2] discussed some of the basic characteristics of the swarm in adulthood which include the long-range and abrupt motion of the insects. Another important feature of grasshoppers swarming is the quest for food source. Just like other nature-inspired algorithms, the search process of the grasshopper is usually in two categories which include the exploration and exploitation process. As grasshopper navigate to attain their target, the two categories of movement is performed. To develop the grasshopper algorithm for solving a real-life problem, it

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_10

Fig. 10.1 Typical real-life grasshopper

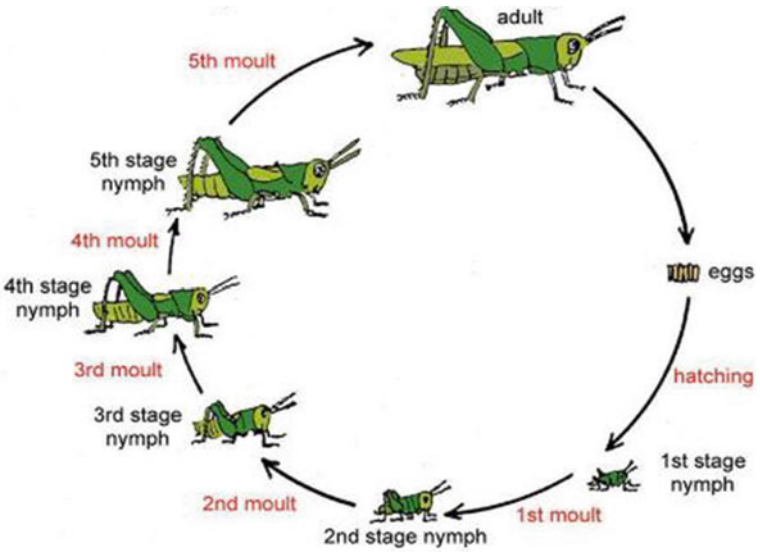


Fig. 10.2 Lifecycle of grasshoppers-incomplete metamorphosis

is important to carefully observe and mimic the navigational pattern of the herbivorous insect. A detailed description of the step by step mathematical model of the grasshopper algorithm are available in Refs. [2, 3].

10.2 Grasshopper Algorithm, Model Notation and Formulation

The grasshopper pictorial view and the metamorphosis cycle are shown in Figs. 10.1 and 10.2 respectively. It is important to showcase the formulation of the algorithm by mimicking the swarming nature of grasshoppers in a real-life situation. The formulation of the model as well as notations adopted are presented. Also, the mathematical equations for the grasshopper algorithm developed by Saremi et al. [2] are included. The model notations and formulations are described in the following section.

- X_i represents the location or position of the i-th grasshopper
- S_i represents the interaction of the social insects
- G_i represents the force of gravity on i-th grasshopper during navigation
- A_i represent the wind advection
- d_{ij} represents the Euclidean location or distance between two given grasshoppers
- f represents the intensity of attraction
- l represents the attractive length scale
- ub_d represent the upper bound in the d-th dimension
- lb_d represent the lower bound in the d-th dimension
- c represents the decreasing coefficient to shrink the zones or neighborhood in terms of comfort, repulsion and attraction
- c_{max} represents the maximum value of c
- c_{min} represent the minimum value of c
- l_c represent the current iteration
- L represents the maximum number of iterations.

$$X_i = S_i + G_i + A_i \tag{10.1}$$

As presented in Eq. 10.1, the social interaction (S_i) is simulated by taking into consideration the aforementioned components of the system (gravitational force impact, and wind advection). This is presented in Eq. 10.2.

$$S_i = \sum_{j=1, j\neq i}^{N} s\left(d_{ij}\right)\widehat{d_{ij}} \tag{10.2}$$

where the distance d_{ij} between the i-th and j-th grasshopper can be calculated using $d_{ij} = |x_j - x_i|$, the unit vector $\widehat{d_{ij}} = \frac{x_j - x_i}{d_{ij}}$ and the strength of social forces s(r) is represented as follows [2, 4]:

$$s(r) = fe^{\frac{-r}{l}} - e^{-r} \tag{10.3}$$

where f and l are respectively the strength of attraction and the attractive length scale.

To illustrate Eq. 10.3, Saremi et al. use a cyclical figure to demonstrate forces of attraction and repulsion between converging grasshoppers [2]. Considering the analysis of the grasshopper convergence or swarm, the analysis of forces due to the location of grasshoppers was demonstrated by comparison with closer grasshoppers in the swarm. The forces considered are neutral, forces of repulsion, and forces. A clearer illustration is presented in Fig. 10.4, which shows a conceptual model of the zone of comfort, attraction zone, and repulsion zone for converging or swarming grasshoppers. As shown in Fig. 10.3, the interplanetary system is sectioned into a neighborhood of comfort, the three zones are "before, inside, and after".

From the analysis further conducted in Ref. [2], the same figure has been presented to highlight the occurrence of repulsion at a defined functional range of value from zero (0) to 2.079. The range of value defined can be controlled or adjusted when unraveling or solving the problem using the grasshopper optimization technique. Further explanation of the process as defined in the figure is that the herbivorous insect (grasshopper) will maintain a position at a neutral state without the application of any force while maintaining a distance 2.079. A gradual shift in the movement of the grasshopper towards a point of benefit, the insect is faced with additional attractive force until it maintains a position of about 5.0. At this level, there is a decrease in the magnitude of forces as a result of an increase in distance. In the grasshopper optimization algorithm (GOA), it is supposed that the functional range(s) is usually between 0 and 3.3. The defined function as presented in Eq. 10.3 has been designed to function with two key parameters which serve as the control, namely l and f. Figure 10.4 presents a graphical illustration of the parameters in terms of control and functional changes in the system. It presents the forces (attractive, repulsive

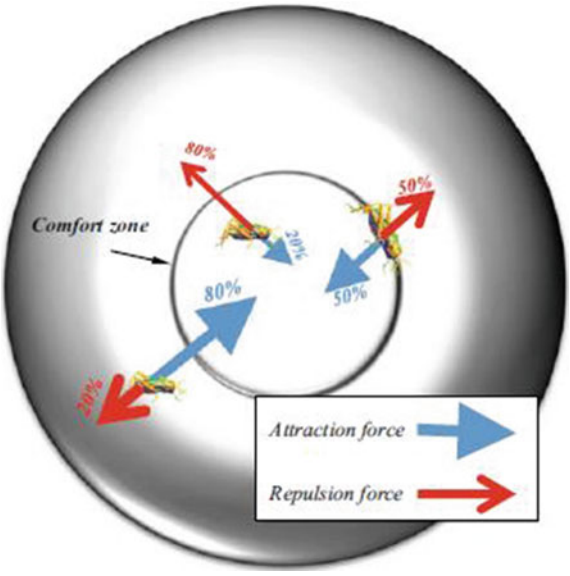


Fig. 10.3 Comfort zone, attraction, and repulsion in the swarm of grasshoppers [2]

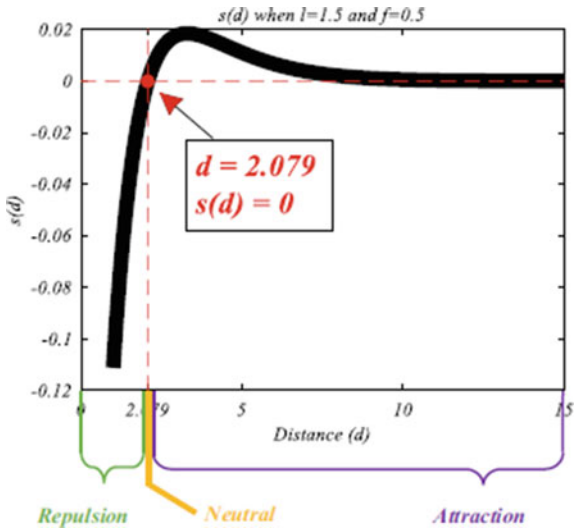


Fig. 10.4 Curve related to Eq. 10.3 [2]

comfort) at different neighborhoods or zones and how they can be meaningfully changed by fine-tuning the parameters under control. The functional range(s) has a certain level of limitation as the forces are not strong enough in situations where grasshoppers are separated by large distances. The easiest way to overcome such limitation is to normalize the distance between the corresponding grasshoppers [1–3]. The graphical illustration presented in Fig. 10.4 shows the curve obtained from the system of Eq. 10.3. The curve explains what happens when the grasshopper is at a neutral position or comfort zone at distance 2.079. It also shows that within a range of $1 \leq d < 2.0790$, there is a tendency of repulsion. Also, at a range of value $2.0790 \leq d$, there is a possibility of great attraction. Figure 10.5 shows the curves obtained when changing the parameters for the function(s).

Saremi et al. defined the parameters of the model showing the grasshoppers' exploration process and the exploitation process to obtain the global optimum using the metaheuristic algorithm [2]. They developed a mathematical model describing the updating position of the grasshopper.

$$X_i^d = c\left(\sum_{j=1, j\neq i}^{N} c\frac{ub_d - lb_d}{2} s\left(\left|x_j^d - x_i^d\right|\right)\frac{x_j - x_i}{d_{ij}}\right) + \widehat{T_d} \tag{10.4}$$

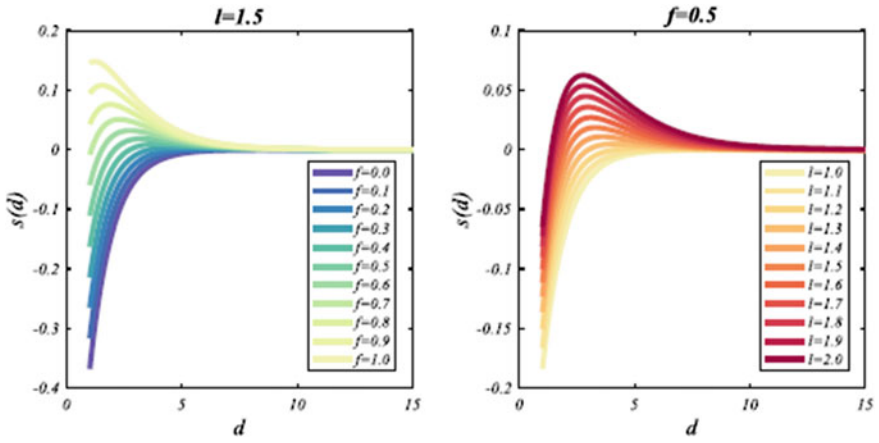


Fig. 10.5 Curves obtained for the function s when changing its parameters [2]

The iteration counter intensifies as the outer or external c decreases the exploration coverage round the target ($\widehat{T_d}$). The central or inner c contributes to the reduction in the magnitude of forces (attraction or repulsion) between converging grasshoppers. It is relatively proportional to the number of iterations. The solution of the model is expected to converge and diverge towards the best global. The system parameter (c) should be restructured or modified to enable an increase in exploitation and reduction in exploration with a proportional increase in the set of iteration. Hence:

$$c = c_{Max} - l\frac{c_{Max} - c_{Min}}{L} \tag{10.5}$$

Further details about the grasshopper optimization algorithm can be found in Ref. [2].

10.3 Area of Application of Grasshopper Algorithm

The solution strength and robustness of the Grasshopper optimization algorithm has been demonstrated in solving real-life combinatorial problems. The metaheuristic algorithm can be used in diverse areas of operation for scheduling, estimation, simulation, and control.

Examples of Possible Field of Operation of GOA.

- For optimization in transportation problems with a single objective like cost minimization.
- Layout design and optimization.
- Problems associated with scheduling and inventory.
- Structural optimization.
- Optimizing power flow.
- Machine sequencing problem.
- Optimizing relay nodes in wireless sensor networks.
- Load frequency control in energy systems and many more.

10.4 Application of Grasshopper Algorithm with a Numerical Example

Minimization of Bukin function

$$f(x, y) = 100\sqrt{|y - 0.01x^2|} + 0.01|x + 10|$$

This model was implemented in MATLAB using the **Grasshopper Optimization Algorithm** (GOA) proposed by Mirjalili (2015). To illustrate the approach, 100 search agents were considered ($n = 100$) and the maximum number of iterations was set to 1000. The best solution obtained by GOA is: [10 1] and the best optimal value of the objective function found by GOA is 0.2361. The parameter space, the test history, the trajectory of the first grasshopper, and the convergence test are shown in Figs. 10.6, 10.7, 10.8 and 10.9 respectively. The MATLAB code described in Ref. [2] has been provided in Appendix H.

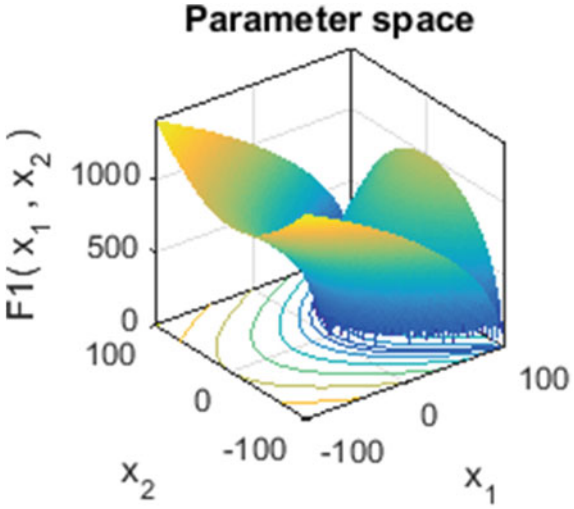


Fig. 10.6 Parameter space

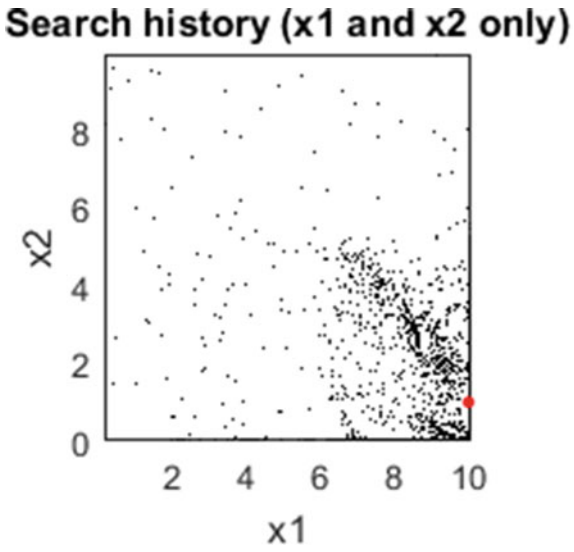


Fig. 10.7 Illustration of search history

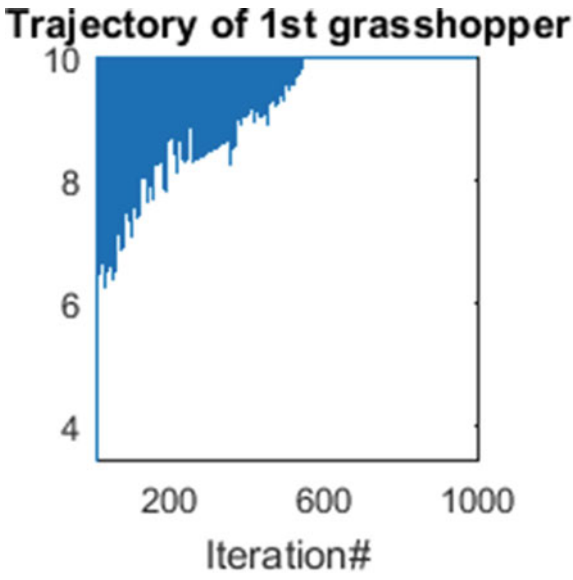


Fig. 10.8 Illustration of the trajectory of 1st grasshopper

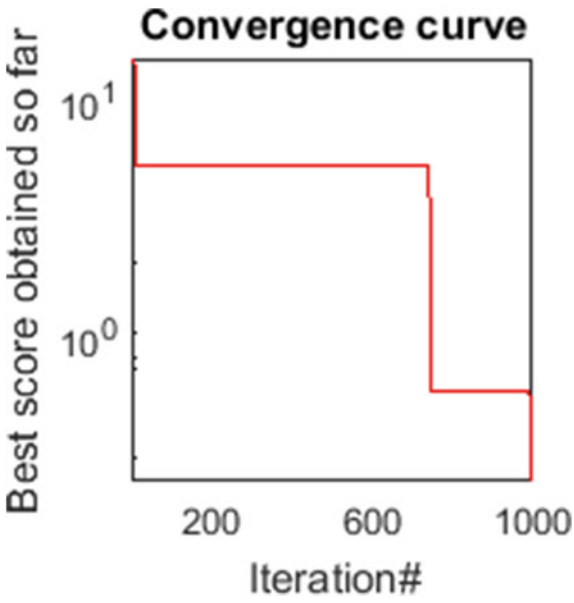


Fig. 10.9 Convergence curve

References

1. Rogers, S.M., T. Matheson, E. Despland, T. Dodgson, M. Burrows, and S.J. Simpson. 2003. Mechanosensory-induced behavioural gregarization in the desert locust Schistocerca gregaria. *Journal of Experimental Biology* 206 (22): 3991–4002.
2. Saremi, S., S. Mirjalili, and A. Lewis. 2017. Grasshopper optimisation algorithm: Theory and application. *Advances in Engineering Software* 105: 30–47.
3. Topaz, C.M., A.J. Bernoff, S. Logan, and W. Toolson. 2008. A model for rolling swarms of locusts. *The European Physical Journal Special Topics* 157 (1): 93–109.
4. Saremi, S., S. Mirjalili, S. Mirjalili, and J.S. Dong. 2020. Grasshopper optimization algorithm: Theory, literature review, and application in hand posture estimation. In *Nature-inspired optimizers*, 107–122. Cham: Springer.

Chapter 11
Butterfly Optimization Algorithm

11.1 Introduction

Butterfly Optimization Algorithm is a unique algorithm inspired by nature and for solving global problems [1]. The algorithm replicates the behavior of the butterfly which can be described to perform a cooperative movement while navigating towards its food source and position. The butterflies are well known for having the power to analyze and receive smell in the air thereby tracing and discovering the direction of their partner and food source [1]. In contemporary studies and literature of soft computing and computational intelligence, the BOA demonstrates the robustness and better results, in comparison with other population-based optimization algorithms. Butterflies are in the class of Lepidoptera in the Linnaean system of Animal Kingdom. There are countless species of butterflies all over the world with separate families. Their sense of smell plays a great role in the survival of species of butterflies that exist in the universe. Butterflies make effective use of their sense of touch, taste, smell, sight, and that of hearing to search for mating partner and food. These senses help them to migrate from one place to another and escape from predators. When it is time for butterflies to lay eggs, they use the power of smell to discover the most suitable, convenient, and safe environment to lay eggs. In all the sensory systems, the sense of smell is the most important to butterflies. It enables the herbivores species of butterflies to search and discover the possible location of food [1]. Nectar is the frequently sourced food, others are interesting species of flowers and leaves which are sensed from far distances in most cases. Biologically, the butterflies have sense receptors as their source of smell. Chemoreceptors in the sense receptors which make smelling of important food source easy for butterflies and they are distributed like antennae all over the body of the butterfly as well as take the form of palps, legs, etc. These chemoreceptors, which are available around the butterfly, lead the herbivorous insect to search for the unsurpassed mating partner which enables them to continue to multiply in number with the formation of a strong genetic line. It is also important to note that the male and female butterflies are similar with the same structure, making

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_11

it very difficult for humans to separate the male butterfly from the female. But it is not the same for the species as it is very easy for a male butterfly to identify the female ones. This is possible with the help of the pheromone [2]. The female butterfly release scent secretions (pheromone) which react around the surrounding, making it very easy for the male butterfly to locate their female counterparts.

11.2 Power of Fragrance and Route-Finding Pattern Butterflies

In BOA, every fragrance has its personal touch and unique scent. Research has shown that butterflies have an extremely distinct feeling in finding the fountain of scent. They can identify various scents and sense their forces. A butterfly by nature can generate fragrance with some power which relates to its comfort. Often, they gather and can swarm from one location to another while exercising their strength which often fluctuates. The fragrance released will stimulate and spread easily for other available butterflies around the neighborhood to perceive. The butterflies, through this means, can effortlessly share perceived information with other butterflies in the neighborhood to profile a collective social information. During this point, when a butterfly senses the scent from any other butterfly around the neighborhood, it will gradually navigate or shift towards the direction of the scent and this singular act, motivate the path planning algorithm of the defined nature inspired insect. There are cases where the butterfly cannot perceive the fragrance discharged from the surrounding area. When this occurs, it will shift randomly at that moment. This singular act helps to properly formulate the butterfly optimization algorithm in the neighborhood search [1]. The pictorial view of the butterfly is shown in Fig. 11.1 and the lifecycle of a butterfly is clearly shown in Fig. 11.2.

11.3 Movement of Butterflies

The butterflies have the following common features [1]:

- Fragrance is released by virtually all butterflies, which is very important to the herbivorous insects for easy motion and attraction to one another.
- Random movement is possible for butterflies while looking out for partners or in search of food. The search process requires moving towards the direction of the butterfly with the best fragrance.
- The intensity or concentration of every butterfly in a zone is determined by the neighborhood of the functional objectives.

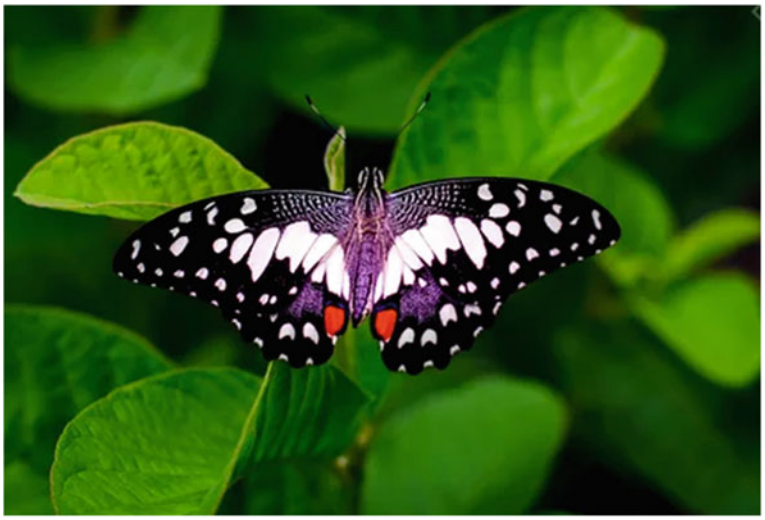


Fig. 11.1 Typical butterfly

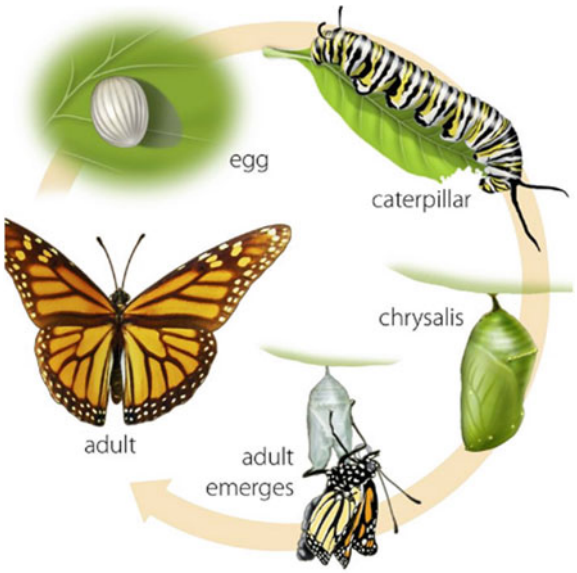


Fig. 11.2 The life cycle of a butterfly

11.4 Stages in Butterfly Optimization Algorithm (BOA)

BOA can be categorized into three phases [1]: the first is the Initialization phase, the second is the iteration phase, and the final phase of the BOA. The process of initialization is demonstrated first in all phases of BOA. This is followed by iteratively searching method of the BOA and then the final step of the algorithm. At the

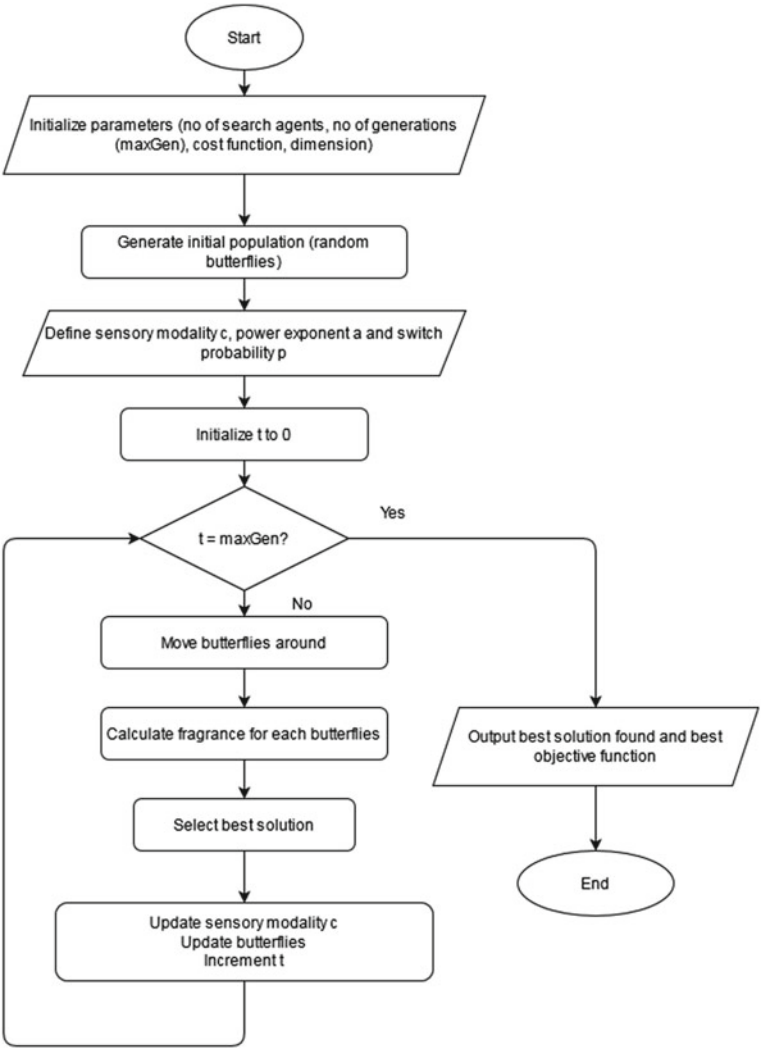


Fig. 11.3 Butterfly optimization algorithm flowchart [1]

final stage of the algorithm, the search process is stopped when the best solution is obtained. What the algorithm does at the initialization stage is to impeccably describe the objective function and the solution space of the model. It also appropriates the principles for the mode of operation used in BOA. The algorithm moves after this to generate the initial butterfly population for effective optimization. As the total number of butterflies remain constant, there is no form of modification during BOA simulation. They are allocated a succinct memory for the storage of information. In the search space, butterflies maintain haphazard positions. The value of strength

maintained by butterflies based on navigational patterns towards high fragrance or area of high concentration of scent can be summed up and stored. This finalizes the phase of initialization to enable the algorithm to commence the process of iteration, and the performance search for the creative butterfly models created.

In the second phase of the iteration process, the algorithm performs a good number of iterations. The solution space shows that all the butterflies in that space navigate to occupy new positions in each of their iteration, and the values of their fitness are estimated at this stage. In the solution space, the algorithm calculates the values of fitness for all the butterflies at all positions first. Each butterfly uses the two main step process (local and global search process) in the algorithm. This is for the emission of fragrance at the different positions. The butterfly searches for the appropriate or suitable butterfly solution in the global quest stage.

The quest for mating, search for food source and nourishment during swarming can take place in a neighborhood swarm process, with physical closeness taken into consideration. The strength of navigation can be rated effective based on external factors like rain, sunlight, and wind. The quest for nutrition may have very systematic divisions in the butterfly's general mating and sustenance, with full consideration given to the aforementioned factors.

11.5 Mathematical Model

Butterfly phenomenon can be traced to two significant subject matter [1]: the first is the terminology I which represent the variation and the notation f which represents the fragrance formulation process. For convenience, the programmed objective function is positioned as I of a butterfly, f is proportional, meaning that it can be sensed by accompanying butterflies in the neighborhood. The formulation and development of the butterfly algorithm by mimicking the pattern of navigation of the butterfly towards the intensity of fragrance using Steven's power law. The power law is available in a research conducted in Ref. [3]. Having known the variation and formulation process of the butterfly, developing the algorithm requires looking at the pattern or swarming nature of the butterfly. The model is developed such that in a neighborhood, the connectivity of butterflies is based on variation and formulation process. The design process increases more quickly than the variation based on the navigation of the butterfly. That is, as a butterfly with connecting strength move towards butterfly with high connecting strength leading to an increase in the design or process formulation. The formulation process of the fragrance is shown in Eq. 11.1.

11.6 Notation and Formulation of Model

f	represents the degree of formulation of fragrance by the butterfly
	represents the power exponent parameter of the butterfly
c	represents the modality or pattern of sensory
I	represent the butterfly stimulus strength or level of concentration
xi t	represents the solution vector *xi* for *i*th butterfly in iteration number t
$g^* \leftarrow$	represents the current best solution found among all the solutions in the current iteration
f_i and r	represent the scent or fragrance of ith butterfly and a random numbers in [0, 1].

$$\mathrm{f} = \mathrm{cI}^{\mathrm{a}} \tag{11.1}$$

a and c are parameters in the range [0, 1]. The equation shall concentrate on the strength or level of concentration of the fragrance emitted by the butterfly and the relationship, level of absorption, or swarm pattern of butterfly. Considering a case scenario with the value of $\mathrm{a} = 1$, it presents the fact that the fragrance emitted by the butterfly is at a central pattern and not absorbed in any way. This can be further explained that the amount of scent or fragrance discharged by a given butterfly at a particular location can be perceived or sensed by one other butterfly or group of butterflies at another location at the same level of concentration. This shows that the fragrance emitted by the butterfly can be perceived at the same level within the space and the same level of concentration within a defined domain. It can help attain a global optimum with ease.

In a situation where $\mathrm{a} = 0$, this takes another connotation entirely. It denotes the fact that as butterflies emit fragrance, no other butterfly around the defined neighborhood can sense it in any way. This shows the effective power of the parameter a, as it is the control power in terms of formulation and sensitivity control of the novel algorithm. It is important to note one other very important parameter of the model c which determines the behavior of BOA in terms of navigational speed of the butterfly during the period of convergence especially when a butterfly is tracing another butterfly to a given location due to the level of concertation of fragrance detected at that zone. In theory, this can be explained further in the form $\mathrm{c} \in [0, \infty]$. Looking at it in practice, the value of c is evaluated based on the system characteristics subject to optimization. Based on the parameters c and a, there is no doubt that the two parameters have a very strong effect on the BOA algorithm in terms of speed of convergence.

The next phase of BOA is the iteration phase, where the algorithm performs several iterations. Throughout the iteration performed, all butterflies navigate to occupy new positions where their fitness values estimated. In the solution space, the fitness values of the algorithm are evaluated at different locations and positions in the neighborhood. Fragrance is released by the butterflies at the defined position as presented in Eq. 11.1. During the navigation process, the butterfly strategizes to take position at the best location favorable by the high level of fragrance noted as the phase determined by global search strategy where the best solution is possible as represented in Eq. 11.2. Equation 11.3 represents the local search space of the BOA.

$$x_i^{t+1} = x_i^t + \left(r^2 \times g^* - x_i^t\right) \times f_i \tag{11.2}$$

$$x_i^{t+1} = x_i^t + \left(r^2 \times x_j^t - x_k^t\right) \times f_i \tag{11.3}$$

The parameter x_k^t and x_j^t are the kth and jth butterflies within the solution space. In cases where x_k^t and x_j^t are found in the same neighborhood with a well presented random number r in the range [0, 1]. Equation 11.3 is presented as a local solution model.

Hunting and mating process in butterflies by navigation following the fragrance discharged by other butterflies take place in global and local space. Some factors may limit the navigation process in search of food or mate, which include rain, wind, temperature, and other factors. A large fraction p is possible in the quest for food or partner seeking actions of the interesting insects. In BOA, therefore, an evolution probability p is used in the time of transition from global to local position in neighborhood search. The iteration process continues in cases where the search criteria does not match, till a definite or suitable process is attained. As soon as the iteration process is finalized or determined, the best solution, having the best fitness is obtained at the output section of the algorithm. The pseudocode of the Butterfly Optimization Algorithm is given as follows:

1: Objective function f(x), x = (x_1 … xd) where d is number of dimensions
2: Generate initial population of butterflies xi = (i = 1, 2, …, n)
3: Find the best solutions g* in the initial population
4: Define switch probability p ∈ [0, 1]
5: **while** stopping criteria not met **do**
6: for each butterfly in population **do**
7: Draw rand from a uniform distribution in [0, 1]
8: Calculate fragrance of the butterfly
9: **if** rand < **p then**
10: Global search using Eq. (11.2)
11: **else**

12: Randomly choose j and k among all the solutions
13: Do Local search using Eq. (11.3)
14: end if
15: Evaluate new solutions
16: If new solutions are better, update them in population
17: **end for**
18: Find the current best solution g*
19: **end** while
20: Output the best solution obtained.

Areas of Application of BOA

Just like every other metaheuristic algorithm, the butterfly algorithm is considered very useful for solving complex stochastic problems like:

- Load frequency control in energy systems and many more.
- Signal processing in mechanical systems
- Signal detection for rotating machinery
- Layout design, planning and optimization
- Flow shop scheduling problem
- For optimization in transportation problems with a single objective like cost minimization
- Problems associated with scheduling and inventory
- Structural optimization
- Optimizing power flow
- Machine sequencing problem
- Optimizing relay nodes in wireless sensor networks and other fields of engineering and science (Fig. 11.3).

11.7 Application of Butterfly Optimization Algorithm with a Numerical Example

Minimization of Himmelblau's function

$$f(x, y) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2.$$

This model was implemented in MATLAB using the **Butterfly Optimization Algorithm** (BOA) described by Arora and Singh [1]. To illustrate the approach, 30 search agents were considered ($n = 30$) and the maximum number of iterations was

set to 1000. The best solution obtained by BOA is: [3.0175 1.9708] and the best optimal value of the objective function found by BOA is 0.015431. The parameter space and the convergence test are shown in Figs. 11.4 and 11.5 respectively. The MATLAB codes are provided in Appendix I.

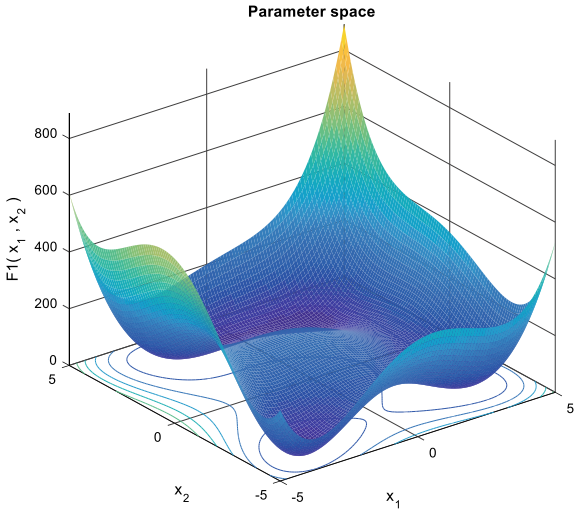


Fig. 11.4 Parameter space

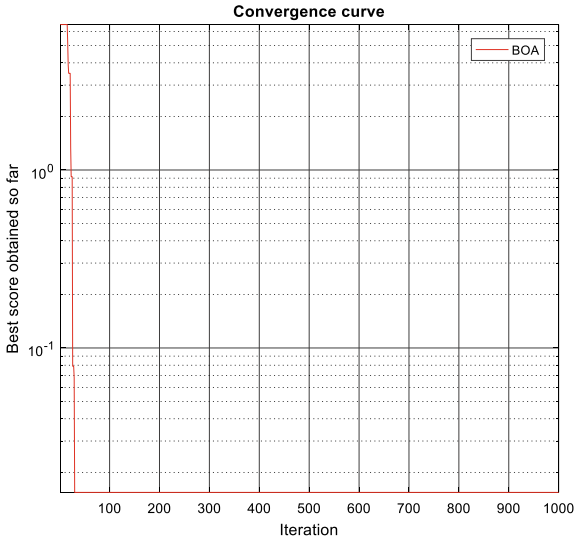


Fig. 11.5 Convergence curve

References

1. Arora, S., and S. Singh. 2019. Butterfly optimization algorithm: A novel approach for global optimization. *Soft Computing* 23 (3): 715–734.
2. Wyatt, Tristram D. 2003. *Pheromones and animal behavior: Communication by smell and taste.* Cambridge University Press.
3. Stevens, S.S. 2017. *Psychophysics: Introduction to its perceptual, neural and social prospects.* Routledge.

Chapter 12
Moths–Flame Optimization Algorithm

12.1 Introduction

Research has proved that there are more than 160,000 varieties of moth species that exist in nature and have been identified and confirmed in the literature. These moths are decorative insects that have a similar resemblance or appearance like the widespread butterflies. One interesting part of these insects is their life cycle. The life cycle of a moth is in two categories, the first is the larva stage and the second is the adult stage. This is a transformation process in the life of a moth, and it follows a sequence of processes to achieve the adult stage where the adult moth is formed. In the life of moths, the most fascinating fact is their unique night navigation method. They were designed to use the moonlight to fly through the dark neighborhood. One of the interesting navigation methods of the moth is the transverse orientation. This pattern of orientation allows the moth to fly by keeping a steady moon angle during the navigation process, a straight path is maintained by the moth when this process of movement along a path is maintained. This makes the pattern of navigation and communication a unique and powerful mechanism.

The pattern of navigation of a moth during transverse orientation is shown in Fig. 12.1. The insects are intelligent enough to observe the moon during navigation. The system ensures flying in a straight line, since the moon is far away from the moth. Humans can follow the same navigational pattern [1]. As shown in Fig. 12.2, the moths do not make their way forward, they are known for flying spirally around the source of light. This is as a result of the transverse orientation approach which is effective only because the light source is so far away. In the case of artificial illumination, as shown in Fig. 12.2a, the moths seek to retain the same angle with the source of light. In terms of problem-solving as stated by Shehab et al. [1], there are two major steps required using the moth flame optimization techniques. The first is the qualitative step process and the second is the quantitative process. Moth flame algorithm is a very reliable algorithm for solving problems, though it has its limitation which has to do with the first step process of defining the objective function

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_12

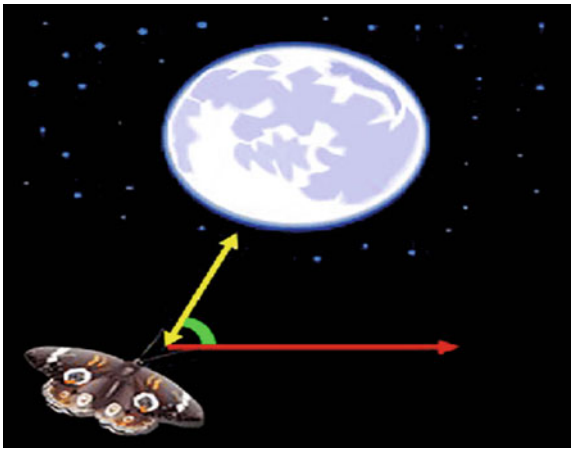


Fig. 12.1 Transverse orientation demonstrated by Moth during navigation [1]

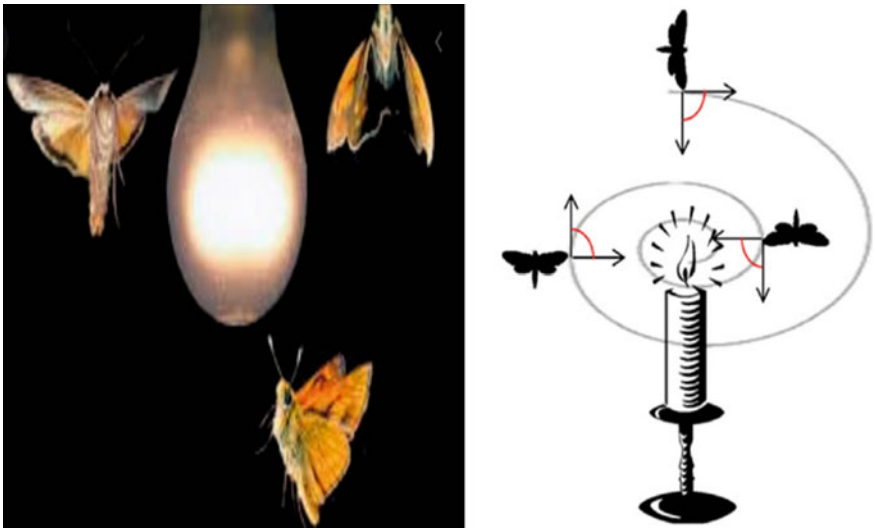


Fig. 12.2 **a** A flying moth round a light source; **b** Navigational path around light source [1]

and developing accurate constraints to fit into the defined objective(s). The balance in the equality and inequality sign while building up constraints is very important in the moth flame optimization process.

12.2 MFO Algorithm

Moth–flame optimization (MFO) algorithm is a simple and easy to implement, nature-inspired, metaheuristic algorithm. It was developed by Mirjalili [2]. The procedure for developing the algorithm is quite straightforward. This is presented in the MFO flowchart. The first step is to generate moths randomly within the neighborhood or solution space. Then the fitness value for each moth is calculated and the best position obtained is tagged by flame. Then the updating process takes place after which the process will be repeated until a point where criteria for termination of the process is attained. As proposed in Ref. [2], the candidate solution of the model was assumed to be the moth, and the variables of the problem to be addressed take position of moth in the neighborhood. It was also assumed that the moth can fly in any of the three dimensions 1–3. Moth flame algorithm is a population-based algorithm. The set of the moth is presented in matrix form using the matrix described by Eq. 12.1 [2].

$$M_{MFO} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,d} \end{bmatrix} \tag{12.1}$$

where n represents the number of moths in the search space.

d represents the dimension variables.

The equivalent fitness value can be arranged as presented in Eq. 12.2 [2].

$$OM_{MFO} = \begin{bmatrix} OM_{1,1} \\ OM_{2,1} \\ \vdots \\ OM_{n,1} \end{bmatrix} \tag{12.2}$$

Since the Moth matrix is presented, to balance the algorithm, it is equally important to present the flame matrix which is of the same number of row and column with the moths' matrix presented in Eq. 12.1. This is presented in Eq. 12.3 [2].

$$F_{MFO} = \begin{bmatrix} F_{1,1} & F_{1,2} & \cdots & F_{1,n} \\ F_{2,1} & F_{2,2} & \cdots & F_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ F_{n,1} & F_{n,2} & \cdots & F_{n,d} \end{bmatrix} \tag{12.3}$$

It is equally important to fit the dimension of moth and flame for equivalent fitness value which can be arranged as presented in Eq. 12.4 [2].

$$OF_{MFO} = \begin{bmatrix} OF_{1,1} \\ OF_{2,1} \\ \vdots \\ OF_{n,1} \end{bmatrix} \tag{12.4}$$

Solution to a problem using moth flame optimization can be obtained by planning the path for both moths and flames resolution. The difference between them is the way every iteration is handled and modified during the iteration process. In the neighborhood search, the moth represents search agents moving around the space during the search process while flames are the best moths' location so far obtained. Alternatively, during neighborhood search, flames can be regarded as pins or streamers which are released by moths. Each moth, therefore, searches for a streamer representing flame which is updated in case a better solution is found. One moth never loses its best solution with this method. This moth flame algorithm is considered as a three-tuple algorithm capable of approximating the optimal global solution in an optimization problem. The mathematical formulation is further expanded in the system of Eq. 12.5 [2].

$$MFO = (I, P, T) \tag{12.5}$$

where I represent a function capable of generating an arbitrary value in a population of moths in a neighborhood search as well as the equivalent fitness values.

The systematic model representing the function (I) is shown in Eq. 12.6 [2].

$$I : \Phi \rightarrow \{M, OM\} \tag{12.6}$$

Equally, the P as shown in Eq. 12.5 represent the key function of the mathematical formulation, mimicking the movement of moths during neighborhood search around the space. The moths are moved around the search space by unction P. This is represented in Equation showing matrix M as it proceeds after every update [2].

$$P : M \rightarrow M \tag{12.7}$$

Finally, from Eq. 12.5, T represents another useful function which checkmates the satisfactory level of the termination condition in the system. If the condition for termination is satisfied then T is true but when the termination condition is not satisfied, T is considered as false. This is illustrated in Eq. 12.8 [2].

$$T \rightarrow M\{true, false\} \tag{12.8}$$

It is important to note that the MFO general framework can be exploited using any of the three functions P, T, and I. The variables equally have matrixes of upper and lower bound which can be defined using the variables of arrays ub and lb as shown in Eqs. 12.9 and 12.10 [2].

$$ub = \left[ub_1, ub_2 \ldots ub_{n-1}, ub_n\right] \tag{12.9}$$

$$lb = \left[lb_1,\ lb_2, \ldots lb_{n-1}, ub_n\right] \tag{12.10}$$

Referring to the functions P, T, and I, on completion of the process of initialization, the key function (P) is simulated iteratively until a point where the function (T) returns true. At this point, the function (P) remains the definitive function which facilitates movement of the moths from one place to another in the search space. This algorithm is inspired by transverse orientation as earlier discussed in the introduction. To model the behaviour of this creative insect, the location of each moth is updated based on the flame as shown in Eq. 12.11 [2].

$$M_i = S\left(M_i, F_j\right) \tag{12.11}$$

where S represents the spiral function, M_i represents the position or location of i-th moth while j represents the j-th flame.

Considering certain conditions from the spiral function with a focus on the position or location of moth and flame in the space, the logarithmic spiral function of the MFO is shown in Eq. 12.12 [2].

$$S\left(M_i, F_j\right) = D_i \cdot E^{bt} \cdot \cos(2\pi t) + F_j \tag{12.12}$$

where b represents a constant which defines the shape of the logarithmic spiral in the system, t represents a random number in the range of -1 to $+\ 1$ and D_i represents the possible distance intended for the i-th moth to a j-th flame in the system. D is calculated as follows [2]:

$$D = \left|F_j - M_i\right| \tag{12.13}$$

The Equation describing the flame number (F_N) in a neighborhood search is given as follows [2]:

$$F_N = \text{round}[(N - I) * (N - 1/T)] \tag{12.14}$$

F_N has earlier been defined as flame number, T represents the maximum possible iteration number, I signify the present position of iteration. An illustration of the lifecycle of a moth is presented in Fig. 12.3. The flowchart of the algorithm is provided in Fig. 12.4.

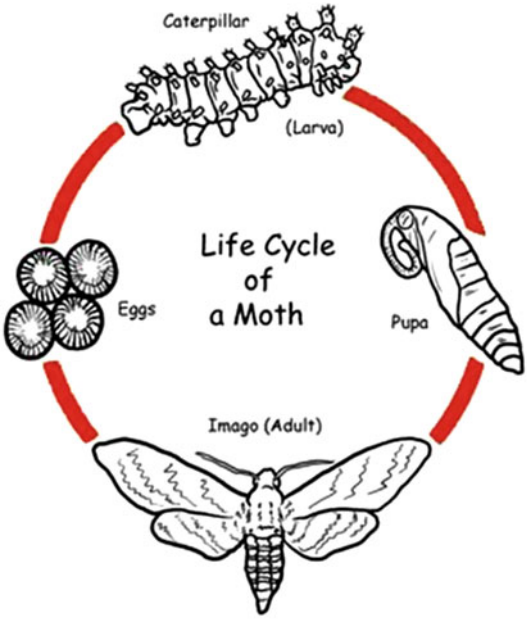


Fig. 12.3 Life cycle of a Moth

12.3 Application of Moth–Flame Optimization with a Numerical Example

Minimization of Goldstein–Price function

$$f(x, y) = \left[1 + (x + y + 1)^2\left(19 - 14x + 3x^2 - 14y + 6xy + 3y^2\right)\right]$$
$$\left[30 + (2x{-}3y)^2\left(18{-}32x + 12x^2 + 48y{-}36xy + 27y^2\right)\right]$$

This model was implemented in MATLAB using the **Moth–Flame Optimization** (MFO) proposed by Mirjalili (2015). To illustrate the approach, 30 search agents were considered ($n = 30$) and the maximum number of iterations were set to 100. The best solution obtained by MFO is: [1.7999 0.19996] and the best optimal value of the objective function found by MFO is 84. The parameter space and the convergence test are shon in Figs. 12.5 and 12.6 respectively. The MATLAB codes is provided in Appendix J and described in Ref. [2].

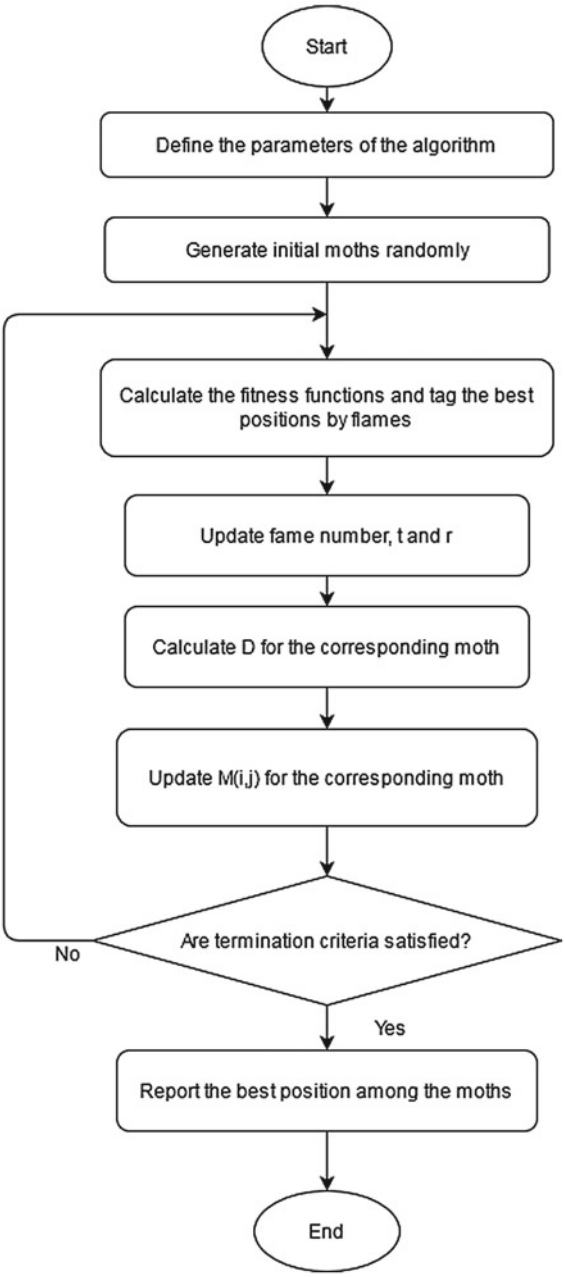


Fig. 12.4 Flowchart of the MFO algorithm [1]

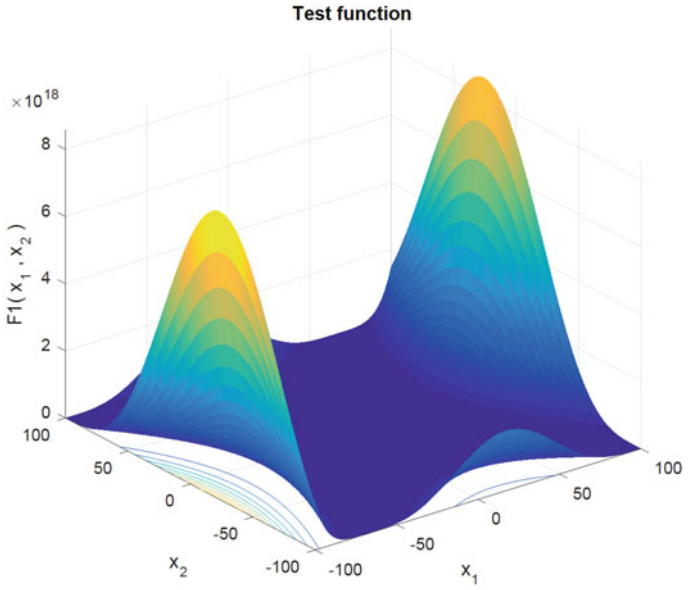


Fig. 12.5 Test function

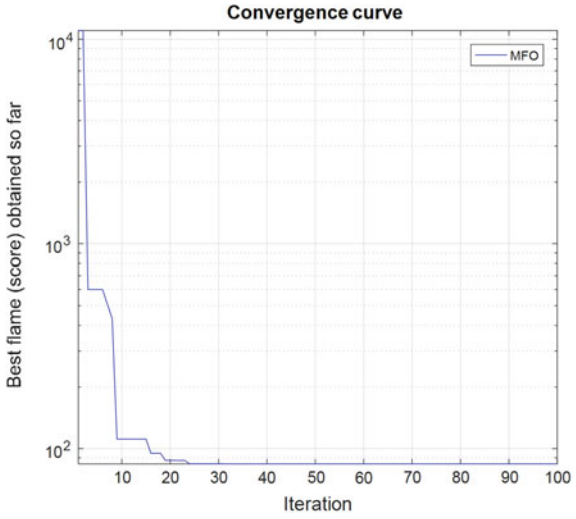


Fig. 12.6 Convergence curve

References

1. Shehab, M., L. Abualigah, H. Al Hamad, H. Alabool, M. Alshinwan, and A.M. Khasawneh. 2020. Moth–flame optimization algorithm: Variants and applications. *Neural Computing and Applications* 32 (14): 9859–9884.
2. Mirjalili, S. 2015. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems* 89: 228–249.

Chapter 13
Genetic Algorithm

13.1 Introduction

Genetic Algorithm (GA) was introduced by John Holland in 1975 with the help of his colleagues and students. GA is an optimization algorithm that is often categorized as a global search technique. Being a branch of the field of study called evolutionary computation; it is known to mimic the natural selection of biological processes of reproduction and to solve the 'fittest' solutions [1]. It is often known to be an optimizer of vector feature weights in either a linear or a nonlinear fashion. The selection and quality of each pattern feature influence the classification of subsequent success and this pattern classification requires that a set of measurable features objects be described [2].

This algorithm is used in minimizing or maximizing an objective function by mimicking the process of natural evolution. Genetic algorithm as an optimization technique takes its root from nature and the terminologies often used or associated with it are biological. The basic components of GA include; Fitness function used for optimization; the population of chromosomes; selection of chromosomes which reproduce; production of next generation of chromosomes by crossover; mutation of chromosomes in a new generation in a random manner. Fitness function is the function that is to be optimized by the solution produced by the GA, the fitness function is one of the most important parts of the algorithm hence it is done carefully. The selection operator selects based on the probability of the chromosomes to be used for reproduction. The fitter a chromosome, the more likely it is to be selected.

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_13

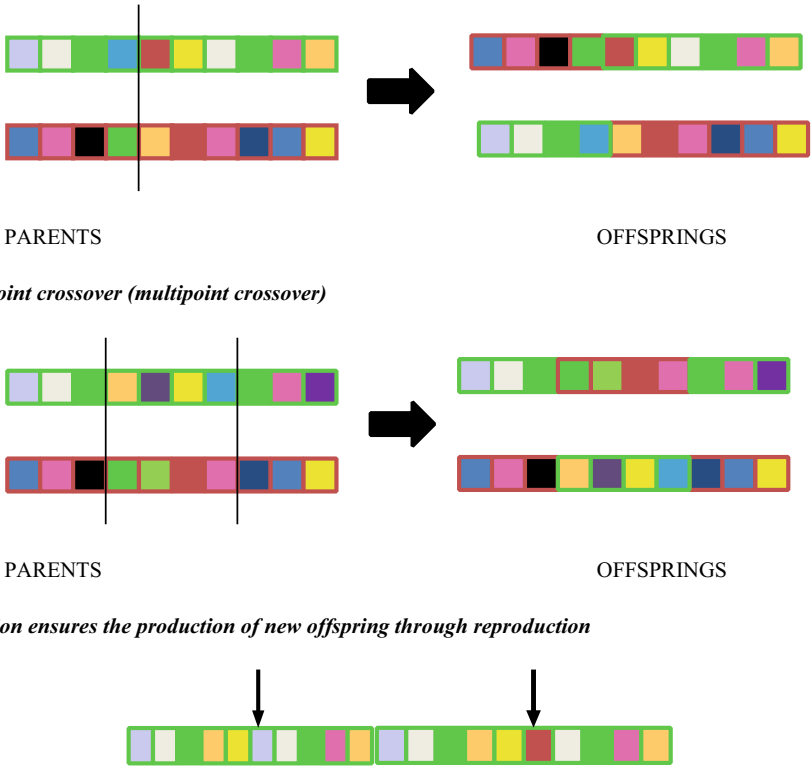


Fig. 13.1 Generating new offspring from a single parent

13.2 Cross-Over Operation in GA

The cross over operation is used to create offspring. It swaps a subsequence of two of the selected chromosomes to create two offspring in the crossover process. Several strategies can be employed. They include: single-point crossover, uniform crossover, multipoint crossover (Fig. 13.1).

13.3 Some Characteristics of GA that Makes It Unique Compared to PSO

(i) GA has the selection operator for easy optimization while there is no selection operator in particle swarm optimization algorithms;
(ii) GA implement survival of the fittest strategy unlike the particle swarm optimization algorithms;

(iii) Not all individuals are kept as members of the population in a genetic algorithm, unlike PSO where all individuals are kept as members of the population.

The genetic algorithm has vast applications in engineering and sciences. Some areas of application of GA include: Knapsack problem; aircraft design; communication network; gas pipeline; poker; encryption and decryption; traveling salesman problem and others. The application of a genetic algorithm in solving the Knapsack problem is considered in the next section. GA flowchart is presented in the figure (Fig. 13.2).

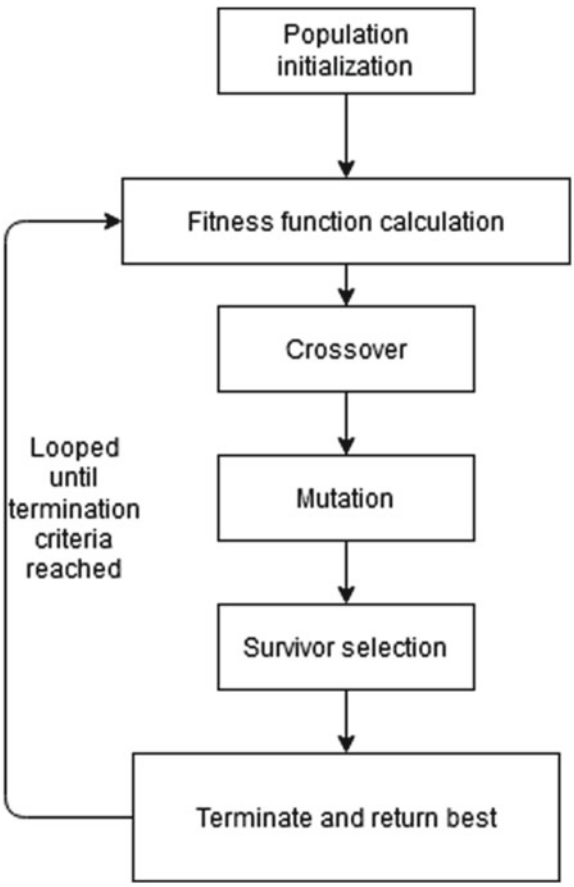


Fig. 13.2 Structure of the genetic algorithm

13.4 Application of Genetic Algorithm

Case Scenario 2. 0–1 Knapsack Problem Using Genetic Algorithm

The knapsack problem in this context is based on the determination of strength or capacity of bags used in conveying loads. It makes use of combinatorial optimization in search of a solution to a problem under uncertainty. When solving the knapsack's problem, two attributes are usually considered: Value or benefit upon which the importance of the item depends on and the volume or weight of items to fit into the bag.

Mathematically, the two attributes can be represented as a modeled equation:

$$\text{Maximize} \sum_{i=1}^{N} \text{BiXi} \tag{13.1}$$

Subject to the constraint:

$$\sum_{i=1}^{N} \text{ViXi and} \leq \text{V and } 0 \leq \text{Xi} \leq \text{Qi} \tag{13.2}$$

where

N = number of items that may potentially be placed in the knapsack
Vi = positive integer volume
Bi = positive integer benefit.

To solve the knapsack problem, an assumption is made for the capacity/weight the knapsack can carry. Hence, it is assumed that the capacity of the bag is 35 kg, signifying that the maximum allowable item the bag can contain is 35 kg. Considering an individual buying five items from a shopping mall with weight and benefit assigned as shown in Fig. 13.3. The products contained in the bag include: Tangerine (T), grape (G), mango (M), apple (A) and orange (O).

To effectively fit in items available into the knapsack, models are to be generated for the constraints. From the problem being considered, there are two constraints, weights and benefits. Equations (13.1) and (13.2) can be extended further considering N = 5:

$$\sum_{i=1}^{N=5} \text{BiXi} = X_1 + 2X_2 + 5X_3 + 6X_4 + 3X_5 \tag{13.3}$$

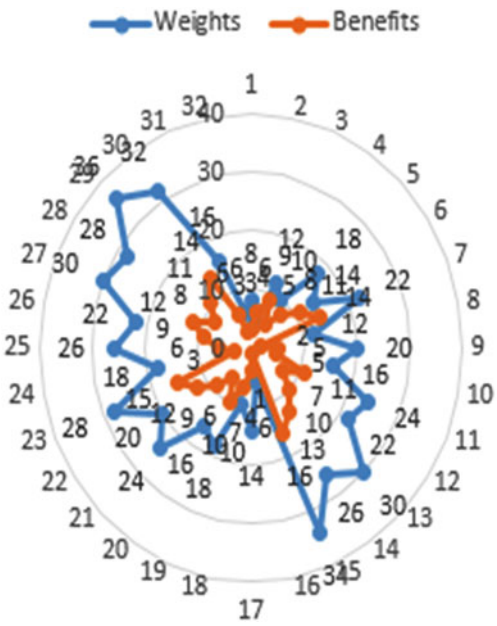


Fig. 13.3 Graphical representation of weights and benefits for Knapsack problem

Subject to the constraints:

$$\sum_{i=1}^{N=5} \mathrm{ViXi} = 6X_1 + 12X_2 + 10X_3 + 4X_4 + 8X_5 \leq 35 \tag{13.4}$$

and

Xi $\in \{0, 1\}$ for $1, 2, \ldots,$ since in GA, chromosomes are usually represented in bits of 0 and 1.

For this problem, the levels of iteration $2^n = 32$ considering that $n = 5$.

Model Solving

- For the benefits of items

$$\sum_{i=1}^{N=5} \mathrm{BiXi} = X_1 + 2X_2 + 5X_3 + 6X_4 + 3X_5$$

where X_1, X_2, X_3, X_4, X_5 are values of the benefit of the item when the chromosome representation is 1 or 0 with solution set space of 32. The model solution as shown in Table 13.1 are: 3, 6, 9, 5, 8, 11, 14, 2, 5, 5, 11, 7, 10, 13, 16, 1, 4, 7, 10, 6, 9, 12, 15, 3, 6, 9, 12, 8, 11, 14, 6, 3.

Table 13.1 Selected items, weight of items and benefits

Items	Weights (kg)	Benefits
T	6	1
G	12	2
M	10	5
A	4	6
O	8	3

- For the weights

$$\sum_{i=1}^{N=5} \mathrm{ViXi} = 6X_1 + 12X_2 + 10X_3 + 4X_4 + 8X_5 \leq 35$$

The model solution obtained from the above weighted equation with set space of 32 are: 8, 4, 12, 10, 18, 14, 22, 12, 20, 16, 24, 22, 30, 26, 34, 6, 14, 10, 18, 16, 24, 20, 28, 18, 26, 22, 30, 28, 36, 32, 16, 6 (Table 13.2).

Results and Discussion

The solution space is the chromosome encoding Ci. The 0–1 concept is initiated where Gene: 0-represent absence of an item in the Knapsack while 1-signify the presence of an item in the bag. 5 bits are requested to represent chromosome encoding.

The initial population is created, and chromosomes randomly created. The fitness function is used to evaluate how good a solution with 32 set space. The presence of item O is felt in the initial space set. The rating or benefit of the knapsack is 3 and the weight of knapsack is 8 kg. The second set space has the presence of item D. Value of knapsack is 6 and the weight of knapsack is 4 kg. The third set space in the bag, has the presence of items A and O. Combined value of knapsack is 9 and the weight of knapsack is 12 kg. Considering the fourth set space, item M is available with knapsack value of 5 and weight of 10 kg. Item M, A and O are available in the 7th space set, the value of knapsack is 14, with weight of 22 kg. M_{15}, $M_{29,}$ and M_{30} where there is a tendency of highest probability of getting selected in the next generation. A clearer observation of the fifteenth set space in Table 13.1 reveals the presence of items G, M, A, and O, at this point, the value of the bag is 16, with a weight of 34 kg. The solution of the model shows that no combination of any form would give an exact weight or capacity the bag can carry except set spaces 15 and 29 as shown in Table 13.1. where the weight of items is 34 and 36 respectively. Hence the feasible solution would be 34 at a value of 16. Any weight of material stored in the bag, which is above 36 will lead to the deformation of the bag.

Table 13.2 Solution set space for the knapsack problem

S/N	T	G	M	A	O	Weights	Benefits
1	0	0	0	0	1	8	3
2	0	0	0	1	0	4	6
3	0	0	0	1	1	12	9
4	0	0	1	0	0	10	5
5	0	0	1	0	1	18	8
6	0	0	1	1	0	14	11
7	0	0	1	1	1	22	14
8	0	1	0	0	0	12	2
9	0	1	0	0	1	20	5
10	0	1	0	1	0	16	5
11	0	1	0	1	1	24	11
12	0	1	1	0	0	22	7
13	0	1	1	0	1	30	10
14	0	1	1	1	0	26	13
15	0	1	1	1	1	34	16
16	1	0	0	0	0	6	1
17	1	0	0	0	1	14	4
18	1	0	0	1	0	10	7
19	1	0	0	1	1	18	10
20	1	0	1	0	0	16	6
21	1	0	1	0	1	24	9
22	1	0	1	1	0	20	12
23	1	0	1	1	1	28	15
24	1	1	0	0	0	18	3
25	1	1	0	0	1	26	6
26	1	1	0	1	0	22	9
27	1	1	0	1	1	30	12
28	1	1	1	0	0	28	8
29	1	1	1	0	1	36	11
30	1	1	1	1	0	32	14
31	1	0	1	0	0	16	6
32	1	0	0	0	0	6	3

References

1. Simon, D. 2013. *Evolutionary optimization algorithms: Biologically inspired and population-based approaches to computer intelligence*. Hoboken: Wiley.
2. Raymer, M.L., W.F. Punch, E.D. Goodman, L.A. Kuhn, and A.K. Jain. 2000. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation* 4 (2): 164–171.

Chapter 14
Artificial Neural Network

14.1 Introduction

Artificial Neural Networks (ANNs) are known to be one of the most prevalent intelligent systems whose architecture or structure is modeled by mimicking the human brain. The system is built in such a way that hundreds of Simple Processing Units (SPU) are contained in a set of complex network systems. The SPU is collectively reinforced, strengthened, and intertwined in a network of multifaceted communication settings. The architecture consists of nodes of which each set is unassumingly connected to neurons capable of sending information and signals in the form of input from one end of the node to another following the system of connections. According to Albert Einstein, real-life problems can be addressed by looking deep into nature. Most of the real-life problem that has been solved so far followed the dimension of natural systems be it aquatic or terrestrial. Summarily, intelligent and soft computational techniques are developed looking deep into nature and mimicry approach of natural systems.

14.2 The Natural Brain and Neural Network

By closely observing the natural brain, it is evident that the system is developed in such a way that information is received from the neurons which are usually from the external source, and the perceived information is encoded in the form of electric signals. The signals received from the system are coded as input which are accepted as they connect to the neurons and flow from one level to another.

As the electrical input crosses through a threshold of the neurons, the neuron will automatically send signals to the next available neurons. The inputs are received

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_14

by the perceptron. After accessing the input in the form of information, it sums the weights of all the input into the system. If the total of the information in terms of summed up weight reaches the threshold, it takes arbitrarily long time for the perceptron to learn and iterate the operation. The continuous iteration process is triggered to obtain the optimal output.

From its outset, research on artificial neural networks was driven by the awareness that the human brain computes in a radically different way from the traditional digital computer. The human brain is a highly dynamic organ that processes information in parallel. It can coordinate its structural constituents, known as neurons, to perform such computations several times faster than today's fastest available digital computer. The brain typically performs perceptual recognition tasks in around 100–200 ms, e.g. recognizing a familiar face embedded in an unfamiliar image, whereas tasks of much lesser complexity that take days on a traditional machine.

A neural network is an algorithm built into a system. It models the way the brain carries out a function. The network is applied using electronic mechanisms, or on a digital computer as simulated in software as network fitting tool to simulate input parameters and produce best output solutions. A neural network is a massively parallel distributed processor made up of simple processing units which have a natural tendency to store and make available solution to experimental information. The artificial neural network resembles the natural brain in two ways:

(i) The first observation and similarity have to do with the acquisition and storage of knowledge. The network can acquire knowledge through a learning process from its environment.
(ii) The second way in which the neural network resembles the natural brain has to do with the interconnection of neurons. Neurons are reinforced, strengthened, and intertwined together in a multifaceted pattern.

The technique used to conduct the learning process in a neural network is called a learning algorithm, which has the task of changing the network's synaptic weights in an orderly fashion to achieve a desired objective.

14.3 Description of the Neural Network

Artificial neural network is a component or type of artificial intelligence which mimics the human brain by simulating information flow from an external source through the neuron to other functional parts of the system. The connecting link and the neuron in the system are the two basic component form of the artificial neural network. A neuron represents the processing unit and the connecting link of the neuron is the connecting system for linking neurons in the brain with other available neurons. The weight of the links vary across the system of connections. A neuron receives information from external sources, processes it as input in the system to yield an output. Neurons are grouped into a layering series. The first is the input, followed by the middle or hidden layer and finally the output.

The first layer which is also the input layer represents the buffer in the system of connection that is known for presenting data to the network system. This layer is not for any form of computation but majorly for receiving information from an external source, since it has no activation function and input weights. The hidden layer, on the other hand, is known for performing a non-linear function in the system, by transforming information in the form of input to the network, it is located between the input and output section of the network layer. They represent the last layer of the network which is known for presenting information at the output level.

The information from an external source or other neurons coming into a neuron is multiplied by connecting weights on the links in different echelons, which is collectively summed with other inbound activations in the system of connections. The multilayer feed-forward network is defined as a neural network that propagates a high level of activation from the input layer only in the forward direction through a single or multiple hidden layer to the output layer. This category of neural network can give a good non-linear relationship where a set of data are presented. This class of neural network is unique and strikingly attractive as continuous functions of any kind can easily be approximated with a feed-forward network system. One of the most important factors to consider during implementation of the neural network is the structure of the neuron, it is very important to factor into consideration the number of layers in the system as well as the number of neurons available in the layers. The network of the neuron gets more complex as the hidden layer and node increases. There is a tendency of having training date overfitting, especially where the structure of the network is highly complicated. This means that data in the training set have a high chance of performing far better than the dataset presented for testing.

14.4 Neural Network Training Pattern

Neural network training is a complex process which is highly essential for supervised learning. MLP are artificial neural networks with a feed-forward network system and well recognized as a solution model in machine learning operations. MLPs are commonly used as classifiers, and this specialized kind of neural network is required to go through a process of training before classifying test data [1]. To effectively train ANN, it is important to adjust the connecting weights [2]. The backpropagation (BP) Algorithm is one of the popular algorithms for conducting such an analysis. According to [3], the algorithm has noticeable shortcomings, one of which includes getting trapped in local minima and relaxed convergence. Researchers are working hard to enhance and boost the performance of the algorithm.

The use of meta-heuristic techniques has been suggested as a good replacement for the BP algorithm at the training stage [4]. The inspiration of the neural network is cantered on the robustness, strength, and suppleness of the brain system. ANN is quite flexible, and the training process is quite easy since it can learn the relationships that exist between possible sets of data (input and output), this is done by changing the connecting weights [5]. In the literature of computational intelligence, a good

number of examples have been presented where metaheuristics were successfully and effectively used to improve different machine learning models. Some of the metaheuristics used in such operations include: Swarm Intelligence, Evolutionary algorithm and Stochastic Local Search [6]. In the literature of computational intelligence, soft computing technique is regarded as a robust technique for solving complex problems efficiently. These interesting and robust techniques have also enabled the expansion of more effective and useful models for solving complex tasks. ANN is no doubt one of the very effective soft computing techniques which has been used extensively in virtually all area of research [7–13].

14.5 Multi-Layer Perceptron (MLP)

As earlier mentioned, MLP has been regarded as one of the very common ANN models [1]. MLP is known to have the input layer (one), output layer (one), and a single or multiple hidden layers in each system of the network. Apart from the nodes in the input layer, all other nodes consist of neurons. There is a possible interconnection of layers across the network system from input to output.

The nature of the problem to be solved is a function of the nodes in each connecting layer. As the number of layers in the network increases, the more complex the structure or architecture of the network. Figure 14.1 represents a good example of an MLP.

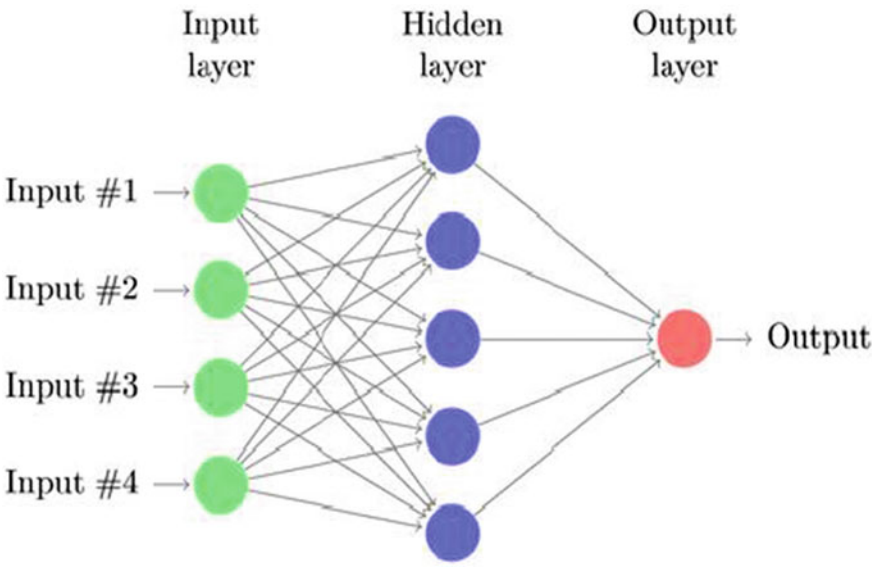


Fig. 14.1 Example MLP network

14.6 The Perceptron Learning Rule

The rule of perceptron learning works by figuring out the possible problem within the neural network and tracing possible solutions or minor corrections and systematically prevent the same errors from reoccurring. This is achieved by contrasting the real network with the goal performance within the training set. When the real performance and goal performance of the network do not match, it means there is a problem in the system, It is, therefore, important to keep updating the weights based on the amount of noticeable error in the system of operation. To solve a problem effectively in each iteration process minor changes are made to the weight. This is done with the application of a trivial learning rate (r). When the rate of learning is high, there is a high tendency that the perceptron can miss the target. A situation where the learning rate is low, can take arbitrarily long time to train the network.

14.7 General Mathematical Model of a Neuron

Notation and definition of terms

$w_{k,1}$: The neuron weight or connection strength of neuron which is from 1 to k
$w_{k,2}$: The neuron weight or connection strength of neuron which is from 2 to k
$w_{k,3}$: The neuron weight or connection strength of neuron which is from 3 to k
$w_{k,n}$: Weight of connection of neuron from n to k
x_1: Assigned input from 1 to neuron k
x_2: Assigned input from 2 to neuron k
x_n: Assigned input from n to neuron k
j: Represent the index classifying the number of input
n: Represent the Index classifying the source
k: Index classifying the neuron
U_k: Linear output combiner as a result of the input signal
V_k: Linear output combiner with bias as backup for input signal
Y_k: Total output.

A typical model of a neuron consists of a set of synapses each of which is characterized by a weight or strength of its own. Other elements that make up a neuron include: adder, activation junction, and a bias. Mathematically, a neuron k can be described by Eq. (14.1).

$$U_k = \sum_{j=1}^{n} x_j w_{k,j} \tag{14.1}$$

Introducing an adder, an activation junction, and a bias the modified neuron with the strength of connection of the neuron as presented in Fig. 14.2. Introducing a bias

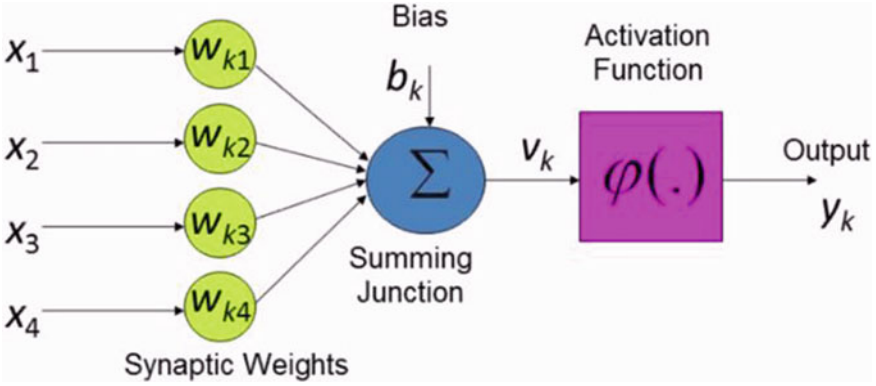


Fig. 14.2 Model of a neural network

(b_k) which is an external parameter of the artificial neuron as input to the system, then the combined input (V_k) can be expressed by Eq. (14.2).

$$V_k = U_k + b_k \tag{14.2}$$

Substituting the value of U_k from the Eq. (14.1) into Eq. (14.2), the combined input can be computed with Eq. (14.3).

$$V_k = \sum_{j=1}^{n} x_j w_{k,j} + b_k \tag{14.3}$$

Defining the threshold function as a sigmoid function, then the total output can be expressed in Eq. (14.4).

$$Y_k = S_g F_n(V_k) \tag{14.4}$$

$S_g F_n$ is substituted as expressed in Eq. (14.5).

$$Y_k = \varphi(.) V_k \tag{14.5}$$

Substitution the value of (V_k) in Eq. (14.3) transforms the total output as expressed in Eq. (14.6). The modified neuron structure is shown in Fig. 14.3.

$$Y_k = \varphi(.) \sum_{j=1}^{n} x_j w_{k,j} + b_k \tag{14.6}$$

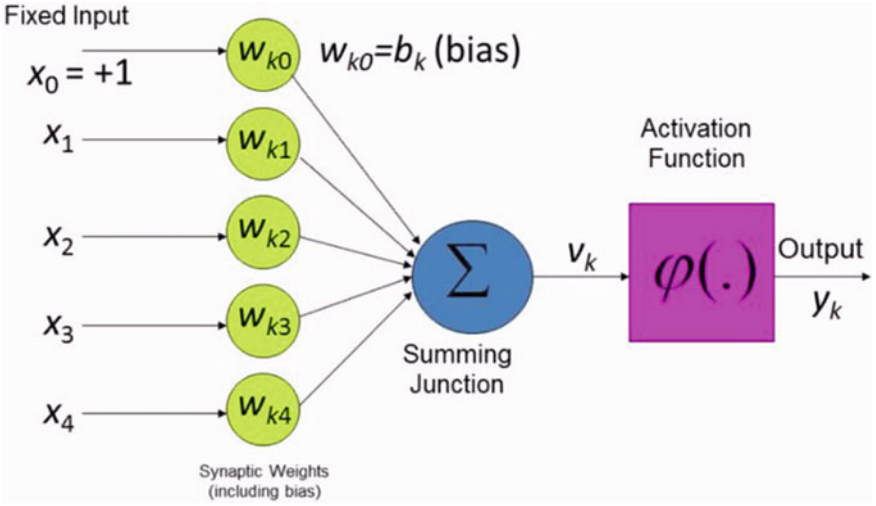


Fig. 14.3 Modified neuron structure

14.8 Application of ANN to Open Pit Mine Vibration Prediction

Blasting operations can damage mine plant and structure when the design has not been done properly particularly if the vibrations induced are sufficiently high. Therefore, prediction of the peak particle velocity (PPV) resulting from the blasting operations in an open-pit mine is necessary. Values of the distance between source and measurement points (D), the maximum charge per delay (WD), and scaled distance (SD) have been collected. To illustrate how ANN could be useful for the prediction of PPV, a model was developed in MATLAB as summarized in this section (Table 14.1).

The ANN model was developed in a MATLAB environment. The structure of the neural network is shown in Fig. 14.4. The model has three inputs, 10 hidden layers, and a single output. The network properties are shown in Fig. 14.5.

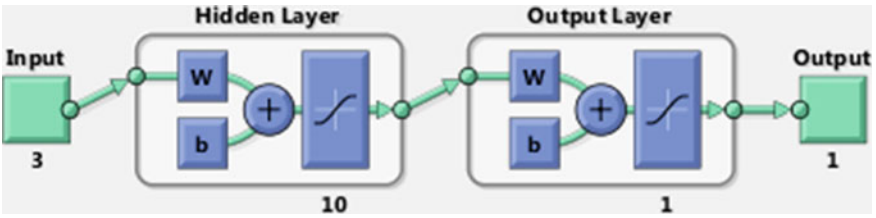


Fig. 14.4 Structure of the neural network

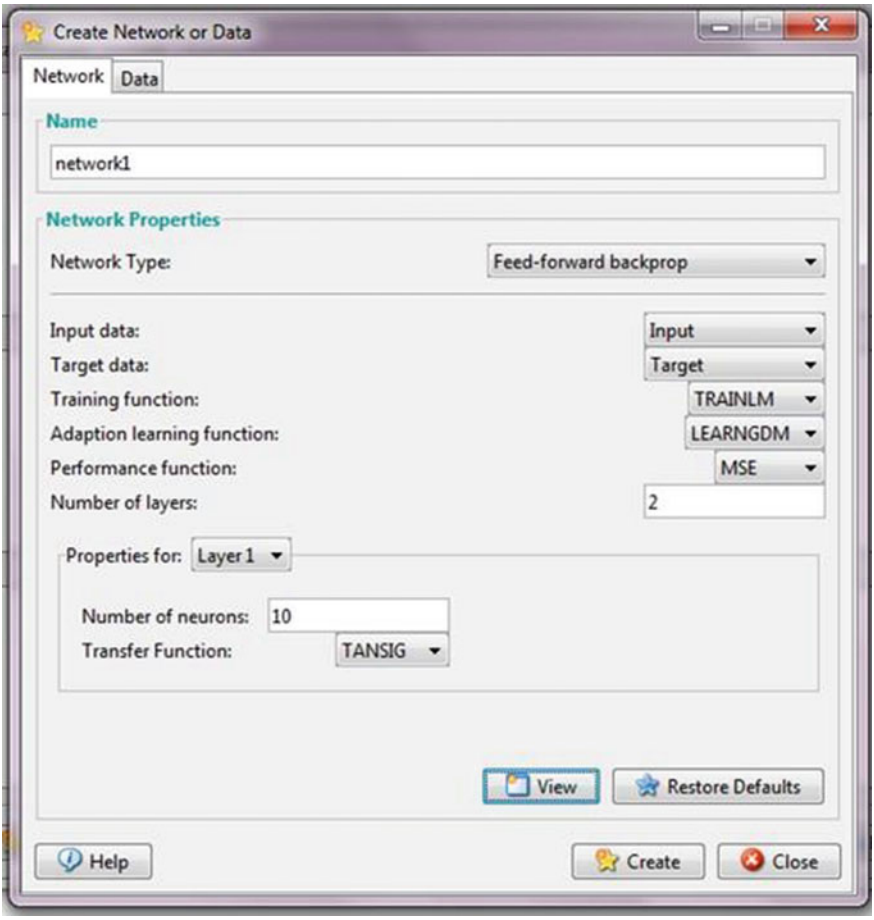


Fig. 14.5 Properties of the neural network

The ANN validation performance is shown in Fig. 14.6. The best validation performance happened at epoch 13 and the test stopped at epoch 18.

Figure 14.7 allows easy visual comparison between the predicted PPV (output) against field recorded values (target). The overall regression value R^2 of 0.95468 suggests that the output tracks the targets very well. This is evidenced by Fig. 14.8 showing field recorded data and output data on the same corresponding to each shot.

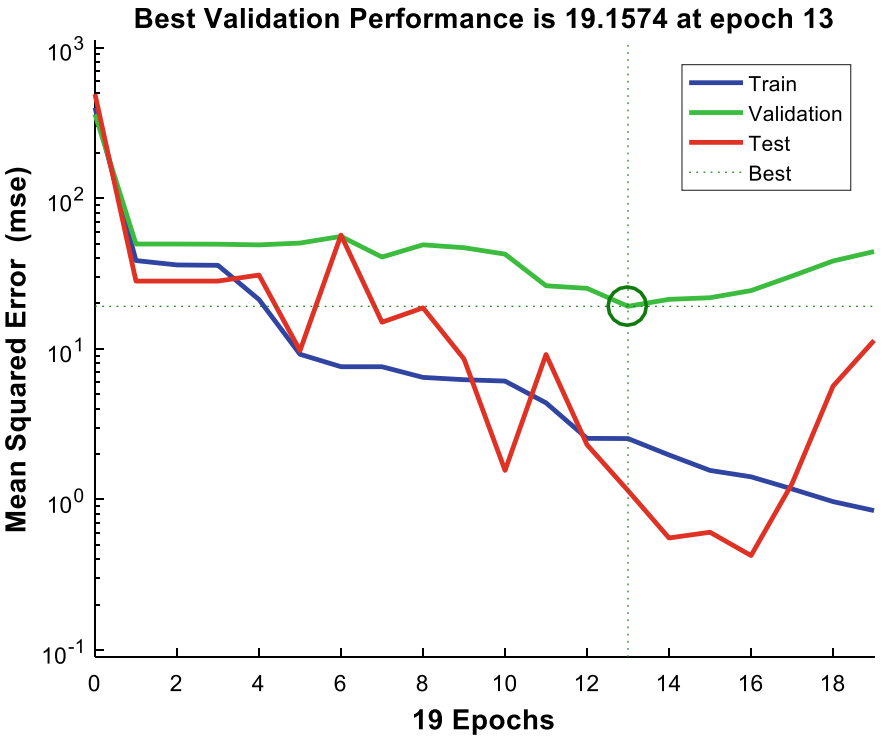


Fig. 14.6 ANN validation performance

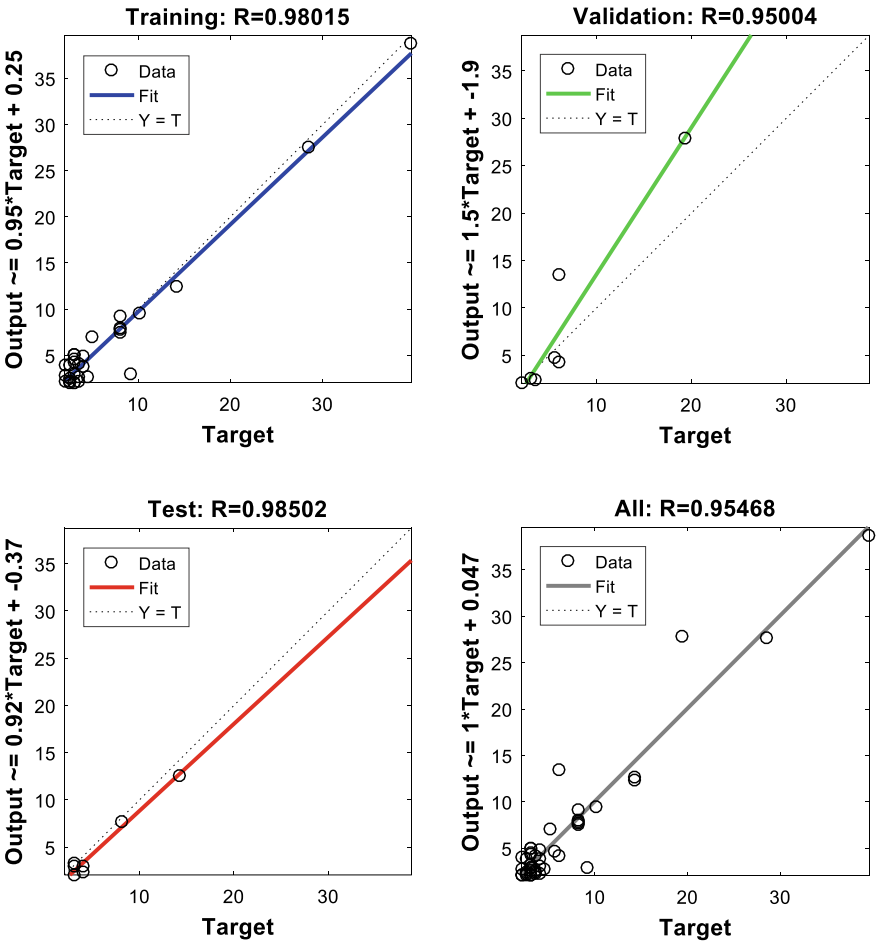


Fig. 14.7 Regression values pointing out the performance of ANN

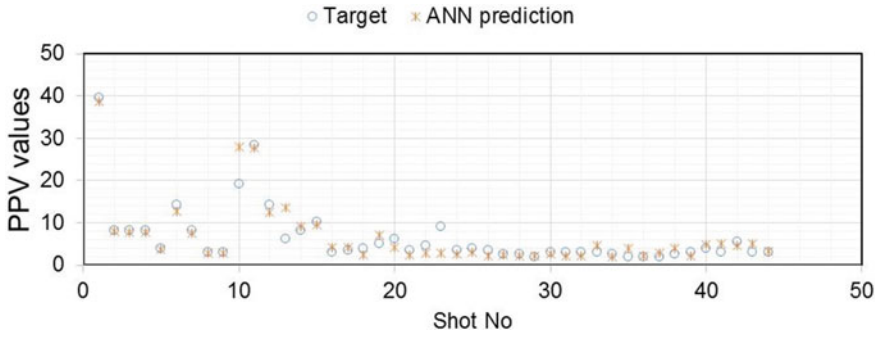


Fig. 14.8 Comparison between fields recorded data (target) and ANN prediction

Table 14.1 Inputs and corresponding output parameters

Input parameters				Output parameter
Shot no	Distance	Charge weight per delay	Scaled distance	Measured peak particle velocity
1	90	269.72	5.48	39.6
2	198	112.5	18.67	8.128
3	199	112.5	18.76	8.128
4	200	112.5	18.86	8.128
5	200	74.5	23.17	4.064
6	200	150	16.33	14.224
7	201	112.5	18.95	8.128
8	220	74.5	25.49	3.048
9	220	74.5	25.49	3.048
10	220	377.5	11.32	19.304
11	250	600	10.21	28.448
12	260	294.3	15.16	14.224
13	280	395.9	14.07	6.096
14	320	484	14.55	8.128
15	350	830.5	12.15	10.16
16	360	391	18.21	3.048
17	360	377.5	18.53	3.556
18	370	151	30.11	4.064
19	380	906	12.62	5.08
20	390	600	15.92	6.096
21	410	226.5	27.24	3.556
22	420	371.3	21.8	4.572
23	420	486	19.05	9.144
24	430	377.5	22.13	3.556
25	430	679.5	16.5	4.064
26	440	226.5	29.24	3.556
27	440	302	25.32	2.54
28	450	226.5	29.9	2.54
29	480	226.5	31.89	2.032
30	610	830.5	21.17	3.048
31	620	491	27.98	3.048
32	620	518.8	27.22	3.048
33	670	957.5	21.65	3.048
34	710	573.3	29.65	2.54

(continued)

Table 14.1 (continued)

Input parameters				Output parameter
Shot no	Distance	Charge weight per delay	Scaled distance	Measured peak particle velocity
35	710	947.6	23.06	2.032
36	730	535	31.56	2.032
37	730	755	26.57	2.032
38	740	981.5	23.62	2.54
39	760	453	35.71	3.048
40	780	906	25.91	4.064
41	790	953.3	25.59	3.048
42	790	1132.5	23.48	5.588
43	850	604	34.59	3.048
44	1280	1208	36.83	3.048

References

1. Heaton, J. 2008. *Introduction to neural networks for Java*, 2nd ed. Heaton Research, Inc.
2. Valian, E., S. Mohanna, and S. Tavakoli. 2011. Improved cuckoo search algorithm for feed-forward neural network training. *International Journal of Artificial Intelligence and Applications (IJAIA)* 2 (3).
3. Mingguang, L., and L. Gaoyang. 2009. Artificial neural network co-optimization algorithm based on differential evolution. In *Second International Symposium on Computational Intelligence and Design.*
4. Espinal, A., M. Sotelo-Figueroa, J.A. Soria-Alcaraz, M. Ornelas, H. Puga, M. Carpio, R. Baltazar, and J.L. Rico. 2011. Comparison of PSO and DE for training neural networks. In *10th Mexican International Conference on Artificial Intelligence.*
5. Popoola, L.T., G. Babagana, and A.A. Susu. 2013. A review of an expert system design for crude oil distillation column using the neural networks model and process optimization and control using genetic algorithm framework. *Advances in Chemical Engineering and Science* 3: 164–170.
6. Hoos, H.H., and T. Stützle. 2004. *Stochastic local search foundations and applications*. Morgan Kaufmann/Elsevier.
7. Kanakana, G.M., and A.O. Olanrewaju. 2011. Predicting student performance in engineering education using an artificial neural network at Tshwane University of Technology. In *Proceedings of the International Conference on Industrial Engineering, Systems Engineering and Engineering Management for Sustainable Global Development*, 1–7.
8. Lykourentzou, I., I. Giannoukos, G. Mpardis, V. Nikolopoulos, and V. Loumos. 2009. Early and dynamic student achievement prediction in E-learning courses using neural networks. *Journal of the American Society for Information Science and Technology* 60 (2): 372–380.
9. Okwu, M.O., and O. Adetunji. 2018. A comparative study of artificial neural network (ANN) and adaptive neuro-fuzzy inference system (ANFIS) models in distribution system with nondeterministic inputs. *International Journal of Engineering Business Management* 10: 1847979018768421.
10. Olayode, O.I., L.K. Tartibu, and M.O. Okwu. 2020. Application of artificial intelligence on the traffic congestion of autonomous vehicles at a signalized road intersection. In *30th CIRP Design Conference 2020*, Johannesburg.

11. Okwu, M.O., B.U. Oreko, S. Okiy, A.C. Uzorh, and O. Oguoma. 2018. Artificial neural network model for cost optimization in a dual-source multi-destination outbound system. *Cogent Engineering* 5 (1): 1447774.
12. Machesa, M.G.K., L.K. Tartibu, F.K. Tekweme, and M.O. Okwu. 2019. Evaluation of the Stirling heat engine performance prediction using ANN-PSO and ANFIS models. In *2019 6th International Conference on Soft Computing and Machine Intelligence (ISCMI)*, 217–222. IEEE.
13. Okwu, M.O., V.U. Chukwu, and O. Oguoma. 2019. Application of artificial neural network model for cost optimization in a single-source, multi-destination system with non-deterministic inputs. In *International Work-Conference on Artificial Neural Networks*, 539–554. Cham: Springer.

Chapter 15
Future of Nature Inspired Algorithm, Swarm and Computational Intelligence

15.1 Introduction

The nature-inspired algorithm has been discussed intensively in this book. This algorithm has a vast area of application in this new revolution. The first, second, and third industrial revolution, powered by a steam engine, electricity, and the world of computing and mobile communication respectively, led to radical transformation, migration, growth in economy, and immense gains mostly for the developed world. More jobs were created for countries that keyed into the new technologies. The Information Age opened the eyes of many as factories and large organizations shifted from classical techniques to the use of global techniques centered on internet and mobile communications and other soft computing technologies. This present revolution which represents the successive trend of digital technologies can be described as a blurred streak between idea and reality, biological and digital connections. Metaheuristic techniques are suggested for consideration in cases where: The use of classical techniques seem almost impossible or proves very difficult to obtain, a superiority measure which allows assessment or evaluation of countless answers to the problem exists, classical or traditional techniques cannot suitably model or solve the problem in question, nonlinear and complex problems encompassing many variables are presented, problems with quite a few objectives are presented.

M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*,
Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8_15

15.2 Other Areas of Application of Nature-Inspired Algorithms and Computational Intelligence

Nature-Inspired Algorithm and computational intelligence being a stochastic optimization technique is known for solving complex problems. Other areas where the algorithms have been effectively used to solve the problems include travelling scenario and routing problem [1]; transportation network and optimal highway complex problem [2]; Traffic control system [3]; Optimal biodiesel production [4, 5], Prediction of Oscillatory Heat Transfer Coefficient in Heat Exchangers of Thermo-Acoustic Systems [6]; Analysis of Stirling Heat Engine System [7]; Production–distribution planning problems; production–distribution planning problems [8]; optimization process in distribution system [9, 10]; Optimizing clustering in railroad track maintenance jobs [11]; optimizing power flow system [12]; Dynamic performance enhancement for wind energy conversion system [13].

15.3 Other Nature-Inspired Algorithms

As discussed in this text, Nature-Inspired Algorithms and swarm intelligence represent a class of meta-heuristic procedures that are inspired by nature. One generic feature of the algorithms is the ability to combine rules and randomness to mimic natural systems. Nature-inspired algorithms based on swarm intelligence has been discussed with numerical and real-world problems solved in this book. However, there are many more algorithms developed in recent times. Some of the newly evolved algorithms are biological-based while others are physics and chemistry-based [14]. Swarm intelligence is categorized under biological-based algorithms [15], quite a good number of the algorithms have been discussed in this text. Some of the interesting algorithms inspired by nature but not discussed in this text include:

Evolutionary Algorithms

- Cultural Algorithm (CA)
- Evolutionary Programming (EP)
- Evolution Strategies (ES)
- Genetic Programming (GP),
- Differential Evolution (DE) and others.

Bio-Inspired Algorithms

- Artificial Immune System (AIS)
- Atmosphere Cloud Model (ACM)
- Biogeography based optimization (BBO)
- Dolphin echolocation (DE)
- Egyptian vulture (EV)
- Flower pollination algorithm (FPA)

- Great salmon run (GSR)
- Invasive weed optimisation (IWO)
- Japanese tree frogs calling (JTFC)
- Paddy field algorithm (PFA)
- Roach infestation algorithm (RIA)
- Root and Tuber Algorithm
- Shuffle frog leaping algorithm (SFLA) and others.

Swarm Intelligence-Based Algorithms

- Bacteria Foraging Optimization algorithm (BFOA)
- Cat swarm optimisation algorithm (CSOA)
- Eagle strategy optimization algorithm (ESOA)
- Fish Algorithm (FA)
- Krill herd optimization algorithm (KHOA)
- Monkey search optimization algorithm (MSOA)
- Weightless Swarm Algorithms (WSA) and others.

Physics-Inspired Algorithms

- Artificial Physics-based Optimisation (APO)
- Central Force Optimisation (CFO)
- Gravitational Search Algorithm (GSA)
- Gravitational Interaction Optimisation (GIO)
- Harmony Search Algorithm (HSA)
- Integrated Radiation Search (IRS)
- River Formation Dynamics Algorithm (RFDA)
- Simulated Annealing (SA)
- Space Gravitation Optimisation (SGO)
- Water Drop Algorithm (WDA)
- Water Cycle Algorithm (WCA) and others.

15.4 Conclusion

Nature-inspired algorithms, swarm intelligence and other computational intelligence is quite broad. Scientists, engineers, and researchers continue to dig deep into other new areas of soft computing techniques. This book focused on nature-inspired algorithms—swarm intelligence-based, a part of the evolutionary algorithm, genetic algorithm, and very popular computational intelligence technique, artificial neural network. In recent times, the research dimension has focused on computational intelligence like the Artificial Neural Network, Fuzzy Logic techniques, and evolutionary computing. The interesting part is the paradigm shift from classical heuristics to metaheuristic techniques. Exploration of the hybrid algorithm is also an interesting part of the problem-solving technique. Hybrid algorithms—breeding of two or more

intelligent systems for solving Non-polynomial (NP) hard problems to obtain more effective solutions. There is a great interconnection between nature-inspired algorithms and computational intelligence. We, therefore, suggest further insight into the hybridization of the two-emerging field in future research.

References

1. Chow, J.Y.J. 2014. Activity-based travel scenario analysis with routing problem reoptimization. *Computer-Aided Civil and Infrastructure Engineering* 29 (2): 91–106.
2. Chen, X., L. Zhang, X. He, C. Xiong, and Z. Li. 2014. Surrogate-based optimization of expensive-to-evaluate objective for optimal highway toll charging in a large-scale transportation network. *Computer-Aided Civil and Infrastructure Engineering* 29 (5): 359–381.
3. Olayode, I.O., L.K. Tartibu, and M.O. Okwu. 2020. Application of artificial intelligence in traffic control system of non-autonomous vehicles at signalized road intersection. *Procedia CIRP* 91: 194–200.
4. Samuel, O.D., M.O. Okwu, O.J. Oyejide, E. Taghinezhad, A. Afzal, and M. Kaveh. 2020. Optimizing biodiesel production from abundant waste oils through empirical method and grey wolf optimizer. *Fuel* 281: 118701.
5. Samuel, O.D., and M.O. Okwu. 2018. Comparison of response surface methodology (RSM) and artificial neural network (ANN) in modelling of waste coconut oil ethyl esters production. *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*. https://doi.org/10.1080/15567036.2018.1539138.
6. Machesa, M.G.K., L.K. Tartibu, F.K. Tekweme, and M.O. Okwu. 2019. Prediction of oscillatory heat transfer coefficient in heat exchangers of thermo-acoustic systems. In *International Mechanical Engineering Congress and Exposition, Proceedings of the ASME 2019*, November 11–14, 2019, Salt Lake City, UT, USA.
7. Machesa, M.G.K., L.K. Tartibu, F.K. Tekweme, and M.O. Okwu. 2019. Evaluation of the stirling heat engine performance prediction using ANN-PSO and ANFIS models. In *6th International Conference on Soft Computing and Machine Intelligence (ISCMI)*, Johannesburg, South Africa, 211–216. ISBN: 978-1-7281-4576-1.
8. Jia, L., Y. Wang, and L. Fan. 2014. Multiobjective bilevel optimization for production-distribution planning problems using hybrid genetic algorithm. *Integrated Computer-Aided Engineering* 21 (1): 77–90.
9. Okwu, M.O., and A. Olufemi. 2018. A comparative study of artificial neural network (ANN) and adaptive neuro-fuzzy inference system (ANFIS) models in distribution system with non-deterministic inputs. *International Journal of Engineering Business Management* 10 (1–17). https://doi.org/10.1177/1847979018768421.
10. Abdel-mawgoud, H., S. Kamel, M. Tostado, J. Yu, and F. Jurado. 2018. Optimal installation of multiple DG using chaotic moth-flame algorithm and real power loss sensitivity factor in distribution system. In *2018 IEEE International Conference on Smart Energy Systems and Technologies (SEST)*, 1–5.
11. Peng, F., and Y. Ouyang. 2014. Optimal clustering of railroad track maintenance jobs. *Computer-Aided Civil and Infrastructure Engineering* 29 (4): 235–247.
12. Bentouati, B., L. Chaib, and S. Chettih. 2016. Optimal power flow using the moth flam optimizer: A case study of the Algerian power system. *Indonesian Journal of Electrical Engineering and Computer Science* 1 (3): 431–445.
13. Ebrahim, M., M. Becherif, and A.Y. Abdelaziz. 2018. Dynamic performance enhancement for wind energy conversion system using moth-flame optimization-based blade pitch controller. *Sustainable Energy Technologies and Assessments* 27: 206–212.

14. Szeto, W.Y., Y. Wang, and S.C. Wong. 2014. The chemical reaction optimization approach to solving the environmentally sustainable network design problem. *Computer-Aided Civil and Infrastructure Engineering* 29 (2): 140–158.
15. Siddique, N., and H. Adeli. 2015. Nature inspired computing: An overview and some future directions. *Cognitive Computation* 7: 706–714. https://doi.org/10.1007/s12559-015-9370-8.

Appendix A
PSO Matlab Codes

```
% Save the following codes in MATLAB script file (*.m) and save as ofun.m.
function f=ofun(x)
x1=x(:,1);x2=x(:,2);x3=x(:,3);
% objective function (minimization)
of=5*(x1-1)^2+10.*x1.*x2+(x3-3)^2;
% constraints
% no constraints is presented as follows:
c0=[];
c0(1)=x1+x2+x3-5;
c0(2)=x1^2+x2^2-x3;
% defining penalty for each constraint
for i=1:length(c0)
if c0(i)>0
c(i)=1;
else
c(i)=0;
end
end
penalty=10000; % penalty on each constraint violation
f=of % fitness function
```

```
% Save the following main program codes in MATLAB script file (*.m) as
run_pso.m and run
tic
clc
clear all
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
close all
rng default
LB=[-5 -5 -5]; %lower bounds of variables
UB=[5 5 5]; %upper bounds of variables
% pso parameters values
m=3; % number of variables
n=5; % population size
wmax=0.9; % inertia weight
wmin=0.4; % inertia weight
c1=1; % acceleration factor
c2=1; % acceleration factor
% pso main program----------------------------------------------------start
maxite=20; % set maximum number of iteration
maxrun=10; % set maximum number of runs need to be
for run=1:maxrun
run
% pso initialization--------------------------------------------------start
for i=1:n
for j=1:m
x0(i,j)=round(LB(j)+rand()*(UB(j)-LB(j)));
end
end
x=x0; % initial population
v=0.1*x0; % initial velocity
for i=1:n
f0(i,1)=ofun(x0(i,:));
end
[fmin0,index0]=min(f0);
pbest=x0; % initial pbest
gbest=x0(index0,:); % initial gbest
% pso initialization----------------------------------------------------end
% pso algorithm-------------------------------------------------------start
ite=1;
tolerance=1;
while ite<=maxite && tolerance>10^-12
w=wmax-(wmax-wmin)*ite/maxite; % update inertial weight
% pso velocity updates
for i=1:n
```

```
for j=1:m
v(i,j)=w*v(i,j)+c1*rand()*(pbest(i,j)-x(i,j))...
+c2*rand()*(gbest(1,j)-x(i,j));
end
end
% pso position update
for i=1:n
for j=1:m
x(i,j)=x(i,j)+v(i,j);
end
end
% handling boundary violations
for i=1:n
for j=1:m
if x(i,j)<LB(j)
x(i,j)=LB(j);
elseif x(i,j)>UB(j)
x(i,j)=UB(j);
end
end
end
% evaluating fitness
for i=1:n
f(i,1)=ofun(x(i,:));
end
% updating pbest and fitness
for i=1:n
if f(i,1)<f0(i,1)
    pbest(i,:)=x(i,:);
f0(i,1)=f(i,1);
end
end
[fmin,index]=min(f0); % finding out the best particle
ffmin(ite,run)=fmin; % storing best fitness
ffite(run)=ite; % storing iteration count
% updating gbest and best fitness
if fmin<fmin0
gbest=pbest(index,:);
```

```
fmin0=fmin;
end
% calculating tolerance
if ite>100;
tolerance=abs(ffmin(ite-100,run)-fmin0);
end
% displaying iterative results
if ite==1
disp(sprintf('Iteration Best particle Objective fun'));
end
disp(sprintf('%8g %8g %8.4f',ite,index,fmin0));
ite=ite+1;
end
% pso algorithm-----------------------------------------------------------end
gbest;
fvalue=10*(gbest(1)-1)^2;
fff(run)=fvalue;
rgbest(run,:)=gbest;
disp(sprintf('--------------------------------------'));
end
% pso main program-------------------------------------------------------------end
disp(sprintf('\n'));
disp(sprintf('********************************************************'));
disp(sprintf('Final Results-----------------------------'));
[bestfun,bestrun]=min(fff)
best_variables=rgbest(bestrun,:)
disp(sprintf('********************************************************'));
toc
% PSO convergence characteristic
plot(ffmin(1:ffite(bestrun),bestrun),'-k');
xlabel('Iteration');
ylabel('Fitness function value');
title('PSO convergence characteristic')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix B
ACO Matlab Codes

```
function [besttour,mincost]=ACO_code(d,iter,n_ants)
%iter=100;%number of cycles.
% d is n x n distance matrix
%xy is coordinates of cities m x 2 (n is number of cities)
m=n_ants;%200;%number of ants.
n=length(d);%number of cities.
e=.5;%evaporation coefficient.
alpha=1;%order of effect of ants' sight.
beta=2;%rank order of trace's effect.
% for i=1:n
% for j=1:n
% d(i,j)=sqrt((xy(i,1)-xy(j,1))^2+(xy(i,2)-xy(j,2))^2);
% calculate distance between cities
% end
% end
for i=1:n%generating sight matrix.
for j=1:n
if d(i,j)==0
h(i,j)=0;
else
h(i,j)=1/d(i,j);%inverse distance
end
end
end
tauinit=.1*ones(n);%initial tau.
%el=.96;%coefficient of common cost elimination.
for i=1:iter
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
for i=1:m
%ants initial placing.
route(i,1)=1;%ants start from city 1
end
%ant tour
for i=1:m %for all ants
mh=h;%inverse distance matrix
for j=1:n%next node
c=route(i,j);%choose next city to visit
mh(:,c)=0;%for column c,inv distance=0
tau=(tauinit(c,:).^beta).*(mh(c,:).^alpha);%update pheromone
s=(sum(tau));%number of tau
p=(1/s).*tau;%probability
% p
r=rand;
s=0;
for k=1:n %number of city
s=s+p(k);%use roulette wheel
if r<=s
route(i,j+1)=k;%the placement of ants i in the next node
break
end
end
end
end
%%%%% resulting tour/route
route_c=horzcat(route,route(:,1));%add first city to the tour/route
% %compute total distance of path traverse by each ant
for i=1:m %number of ant
    s=0;
for j=1:n-1
s=s+d(route_c(i,j),route_c(i,j+1));
end
f(i)=s;
end
distancetot=f;
[minf,idk]=min(f);
ter=route_c(idk,:);
```

```
%f=f-el*min(f);%elimination of common cost.
%update edge belonging to Lbest
tauinit=(1-e)*tauinit;
for i=1:m % for all ant
% for i=idk;
for j=1:n-1 % for all cities
dt=1/f(i);%inverse total distance, delta pheromone
%update pheromone
tauinit(route_c(i,j),route_c(i,j+1))=tauinit(route_c(i,j),route_c(i,j+1))+dt;
%updating traces.
% tauinit(route_c(i,j),route_c(i,j+1))=(1-
e)*tauinit(route_c(i,j),route_c(i,j+1))+dt;%updating traces.
end
end
%tauinit;
% costoa(i)=mean(cost);
%[mincost(i),number]=min(distancetot);
%besttour(i,:)=route_c(number,:);
%iteration(i)=i;
end
besttour=ter;
mincost=distsp(besttour,d);
function distance=distsp(x1,dx)
%function tsp
%input:
%x1= route tsp (example: 1 2 3 4 5 1, 1=hometown, 2345=destination city
%dx= distance matrix route tsp
%output:
%distance = distance total route tsp
[r,c]=size(x1);
k=c-1;%number of cities on the route
s=0; %distance/starting position in the first city
for j=1:k
s=s+dx(x1(j),x1(j+1)); %accumulating distance of route
end
distance=s;
```

Appendix C
GWO Matlab Codes

```
%%%%%%%GWO consist of five different source codes:
%%%%%%%func_plot(func_name)
%%%%%%%Get_Functions_details(F)
%%%%%%%GWO(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
%%%%%%%initialization(SearchAgents_no,dim,ub,lb)
%%%%%%%main.m
%%%%%%%These codes must be saved under the same directory and "run main.m"
%%%%%%%Adapted from:
%%%%%%%https://au.mathworks.com/matlabcentral/fileexchange/44974-grey-wolf-
optimizer-gwo
```

```
%Get_Functions_details
function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
case 'F1'
fobj = @F1;
lb=0;
ub=10;
dim=2;
end
end
% F1
function o = F1(x)
o=(x(1)-5)^2+(x(2)-2)^2;
end
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Main code
clear all
clc
SearchAgents_no=30; % Number of search agents
Function_name='F1'; % Name of the function
Max_iteration=1000; % Maximum numbef of iterations
% Load details of the selected benchmark function
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
[Best_score,Best_pos,GWO_cg_curve]=GWO(SearchAgents_no,Max_iteration,lb,ub,di
m,fobj);
figure('Position',[500 500 660 290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Parameter space')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
%Draw objective space
subplot(1,2,2);
semilogy(GWO_cg_curve,'Color','r')
title('Objective space')
xlabel('Iteration');
ylabel('Best score obtained so far');
axis tight
grid on
box on
legend('GWO')
display(['The best solution obtained by GWO is : ', num2str(Best_pos)]);
display(['The best optimal value of the objective funciton found by GWO is :
', num2str(Best_score)]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%GWO code
% Grey Wolf Optimizer
function
[Alpha_score,Alpha_pos,Convergence_curve]=GWO(SearchAgents_no,Max_iter,lb,ub,
dim,fobj)
% initialize alpha, beta, and delta_pos
Alpha_pos=zeros(1,dim);
Alpha_score=inf; %change this to -inf for maximization problems
Beta_pos=zeros(1,dim);
Beta_score=inf; %change this to -inf for maximization problems
Delta_pos=zeros(1,dim);
Delta_score=inf; %change this to -inf for maximization problems
%Initialize the positions of search agents
Positions=initialization(SearchAgents_no,dim,ub,lb);
Convergence_curve=zeros(1,Max_iter);
l=0;% Loop counter
% Main loop
while l<Max_iter
for i=1:size(Positions,1)
% Return back the search agents that go beyond the boundaries of the search
space
Flag4ub=Positions(i,:)>ub;
Flag4lb=Positions(i,:)<lb;
Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb
;
% Calculate objective function for each search agent
fitness=fobj(Positions(i,:));
% Update Alpha, Beta, and Delta
if fitness<Alpha_score
Alpha_score=fitness; % Update alpha
Alpha_pos=Positions(i,:);
end
if fitness>Alpha_score && fitness<Beta_score
Beta_score=fitness; % Update beta
Beta_pos=Positions(i,:);
end
if fitness>Alpha_score && fitness>Beta_score && fitness<Delta_score
```

```
Delta_score=fitness; % Update delta
Delta_pos=Positions(i,:);
end
end
a=2-l*((2)/Max_iter); % a decreases linearly fron 2 to 0
% Update the Position of search agents including omegas
for i=1:size(Positions,1)
for j=1:size(Positions,2)
r1=rand(); % r1 is a random number in [0,1]
r2=rand(); % r2 is a random number in [0,1]
A1=2*a*r1-a;
C1=2*r2;
D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j));
X1=Alpha_pos(j)-A1*D_alpha;
r1=rand();
r2=rand();
A2=2*a*r1-a;
C2=2*r2;
D_beta=abs(C2*Beta_pos(j)-Positions(i,j));
X2=Beta_pos(j)-A2*D_beta;
r1=rand();
r2=rand();
A3=2*a*r1-a;
C3=2*r2;
D_delta=abs(C3*Delta_pos(j)-Positions(i,j));
X3=Delta_pos(j)-A3*D_delta;
Positions(i,j)=(X1+X2+X3)/3;
end
end
l=l+1;
Convergence_curve(l)=Alpha_score;
end
```

Appendix D
WOA Matlab Codes

```
%%%%%%%WOA consist of five different source codes:
%%%%%%%func_plot(func_name)
%%%%%%%Get_Functions_details(F)
%%%%%%%WOA(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
%%%%%%%initialization(SearchAgents_no,dim,ub,lb)
%%%%%%%main.m
%%%%%%%These codes must be saved under the same directory and "run main.m"
%%%%%%%Adapted from:
%%%%%%%https://www.mathworks.com/matlabcentral/fileexchange/55667-the-whale-
optimization-algorithm
```

```
% Get_Functions_details
function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
case 'F1'
fobj = @F1;
lb=0;
ub=10;
dim=2;
end
end
% F1
function o = F1(x)
o=((x(1)^2+x(2)-11)^2)+(x(1)+(x(2)^2)-7)^2;
end
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Main code
clear all
clc
SearchAgents_no=30; % Number of search agents
Function_name='F1'; % Name of the test function
Max_iteration=1000; % Maximum number of iterations
% Details of function
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
[Best_score,Best_pos,WOA_cg_curve]=WOA(SearchAgents_no,Max_iteration,lb,ub,di
m,fobj);
figure('Position',[269   240   660   290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Parameter space')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
%Draw objective space
subplot(1,2,2);
semilogy(WOA_cg_curve,'Color','r')
title('Objective space')
xlabel('Iteration');
ylabel('Best score obtained so far');
axis tight
grid on
box on
legend('WOA')
display(['The best solution obtained by WOA is : ', num2str(Best_pos)]);
display(['The best optimal value of the objective function found by WOA is :
', num2str(Best_score)]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%WOA codes
function
[Leader_score,Leader_pos,Convergence_curve]=WOA(SearchAgents_no,Max_iter,lb,u
b,dim,fobj)
% initialize position vector and score for the leader
Leader_pos=zeros(1,dim);
Leader_score=inf; %change this to -inf for maximization problems
%Initialize the positions of search agents
Positions=initialization(SearchAgents_no,dim,ub,lb);
Convergence_curve=zeros(1,Max_iter);
t=0;% Loop counter
% Main loop
while t<Max_iter
for i=1:size(Positions,1)
% Return back the search agents that go beyond the boundaries of the search
space
Flag4ub=Positions(i,:)>ub;
Flag4lb=Positions(i,:)<lb;
Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb
;
% Calculate objective function for each search agent
fitness=fobj(Positions(i,:));
% Update the leader
if fitness<Leader_score % Change this to > for maximization problem
Leader_score=fitness; % Update alpha
Leader_pos=Positions(i,:);
end
end
a=2-t*((2)/Max_iter); % a decreases linearly fron 2 to 0
% a2 linearly dicreases from -1 to -2 to calculate t
a2=-1+t*((-1)/Max_iter);
% Update the Position of search agents
for i=1:size(Positions,1)
r1=rand(); % r1 is a random number in [0,1]
r2=rand(); % r2 is a random number in [0,1]
A=2*a*r1-a;
C=2*r2;
```

```
b=1;
l=(a2-1)*rand+1;
p = rand();
for j=1:size(Positions,2)
if p<0.5
if abs(A)>=1
rand_leader_index = floor(SearchAgents_no*rand()+1);
X_rand = Positions(rand_leader_index, :);
D_X_rand=abs(C*X_rand(j)-Positions(i,j));
Positions(i,j)=X_rand(j)-A*D_X_rand;
elseif abs(A)<1
D_Leader=abs(C*Leader_pos(j)-Positions(i,j));
Positions(i,j)=Leader_pos(j)-A*D_Leader;
end
elseif p>=0.5
distance2Leader=abs(Leader_pos(j)-Positions(i,j));
Positions(i,j)=distance2Leader*exp(b.*l).*cos(l.*2*pi)+Leader_pos(j);
end
end
end
t=t+1;
Convergence_curve(t)=Leader_score;
[t Leader_score]
end
```

Appendix E
FA Matlab Code

```
% -------- Start the Firefly Algorithm (FA) main loop ------------------ %
function fa_ndim_new
n=20;                    % Population size (number of fireflies)
alpha=1.0;               % Randomness strength 0--1 (highly random)
beta0=1.0;               % Attractiveness constant
gamma=0.01;              % Absorption coefficient
theta=0.97;              % Randomness reduction factor theta=10^(-5/tMax)
d=2;                    % Number of dimensions
tMax=500;                % Maximum number of iterations
Lb=-5*ones(1,d);        % Lower bounds/limits
Ub=5*ones(1,d);         % Upper bounds/limits
% Generating the initial locations of n fireflies
for i=1:n,
ns(i,:)=Lb+(Ub-Lb).*rand(1,d);          % Randomization
Lightn(i)=cost(ns(i,:));                % Evaluate objectives
end
%%%%%%%%%%%%%%%%% Start the iterations (main loop) %%%%%%%%%%%%%%%%%%%%%%
for k=1:tMax,
alpha=alpha*theta;      % Reduce alpha by a factor theta
scale=abs(Ub-Lb);       % Scale of the optimization problem
% Two loops over all the n fireflies
for i=1:n,
for j=1:n,
% Evaluate the objective values of current solutions
Lightn(i)=cost(ns(i,:));           % Call the objective
% Update moves
if Lightn(i)>=Lightn(j),           % Brighter/more attractive
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
r=sqrt(sum((ns(i,:)-ns(j,:)).^2));
beta=beta0*exp(-gamma*r.^2);    % Attractiveness
steps=alpha.*(rand(1,d)-0.5).*scale;
% The FA equation for updating position vectors
ns(i,:)=ns(i,:)+beta*(ns(j,:)-ns(i,:))+steps;
end
end % end for j
end % end for i
% Check if the new solutions/locations are within limits/bounds
ns=findlimits(n,ns,Lb,Ub);
%% Rank fireflies by their light intensity/objectives
[Lightn,Index]=sort(Lightn);
nsol_tmp=ns;
for i=1:n,
 ns(i,:)=nsol_tmp(Index(i),:);
end
%% Find the current best solution and display outputs
fbest=Lightn(1), nbest=ns(1,:)
end % End of the main FA loop (up to tMax)
% Make sure that new fireflies are within the bounds/limits
function [ns]=findlimits(n,ns,Lb,Ub)
for i=1:n,
nsol_tmp=ns(i,:);
% Apply the lower bound
I=nsol_tmp<Lb;  nsol_tmp(I)=Lb(I);
% Apply the upper bounds
J=nsol_tmp>Ub;  nsol_tmp(J)=Ub(J);
% Update this new move
ns(i,:)=nsol_tmp;
end
%% Define the objective function or cost function
function z=cost(x)
x1=x(:,1);x2=x(:,2);
% objective function (minimization) e.g. WAYBUREN FUNCTION
z=((x1)^6+(x2)^4-17)^2+(2.*(x1)+(x2)-4)^2;
% ------------------------------------------------------------------------%
```

Appendix F
Bat Algorithm (BA) Matlab Codes

```
%%%%%%%BAT consist of five different source codes:
%%%%%%%func_plot(func_name)
%%%%%%%Get_Functions_details(F)
%%%%%%%newBAT(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
%%%%%%%initialization(SearchAgents_no,dim,ub,lb)
%%%%%%%main.m
%%%%%%%These codes must be saved under the same directory and "run main.m"
%%%%%%%Adapted from:
%%%%%%%https://www.mathworks.com/matlabcentral/fileexchange/68981-bat-
optimization-algorithm
```

```
% Get_Functions_details
function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
case 'F1'
fobj = @F1;
lb=0;
ub=10;
dim=2;
end
end
% F1
function o = F1(x)
o=(((x(1)^6)+(x(2)^4)-17)^2)+((2.*x(1))+x(2)-4)^2;
end
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Main code
clc
clear all
close all
N=10; % Number of Bats
Function_name='F1'; % Name of the test function
Max_iter=1000; % Maximum number of iterations
% Load details of the selected benchmark function
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
[bestfit,BestPositions,fmax,BAT_Cg_curve]=newBAT(N,Max_iter,lb,ub,dim,fobj);
figure('position',[500 500 660 290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Parameter space')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
%Draw objective space
subplot(1,2,2);
semilogy(BAT_Cg_curve,'Color','r')
title('Objective space')
xlabel('iteration');
ylabel('Best fitness obtained so far');
axis tight
grid on
box on
legend('newBAT')
display(['The best solution obtained by BAT is : ', num2str(BestPositions)]);
display(['The best optimal value of the objective function found by BAT is :
', num2str(bestfit)]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%BAT code
function
[bestfit,BestPositions,fmin,Convergence_curve]=newBAT(N,Max_iter,lb,ub,dim,fo
bj)
Max_iter=20;            % maximum generations
N=10;                   %BAT numbers
dim=10;
lb=-2*zeros(1,dim);
ub=2*ones(1,dim);
Fmax=2;                 %maximum frequency
Fmin=0;                 %minimum frequency
A=rand(N,1);            %loudness for each BAT
r=rand(N,1);            %pulse emission rate for each BAT
alpha=0.5;              %constant for loudness update
gamma=0.5;              %constant for emission rate update
ro=0.001;                 %initial pulse emission rate
% Initializing arrays
F=zeros(N,1);           % Frequency
v=zeros(N,dim);           % Velocities
% Initialize the population
x=initializationb(N,Max_iter,dim,ub,lb);
Convergence_curve=zeros(1,Max_iter);
%calculate the initial solution for initial positions
for ii=1:N
fitness(ii)=fobj(x(ii,:));
end
[fmin,index]=min(fitness);          %find the initial best fitness value,
bestsol=x(index,:); %find the initial best solution for best fitness value
iter=1;             % start the loop counter
while iter<=Max_iter                        %start the loop for iterations
for ii=1:size(x)
F(ii)=Fmin+(Fmax-Fmin)*rand;        %randomly chose the frequency
v(ii,:)=v(ii,:)+(x(ii,:)-bestsol)*F(ii);  %update the velocity
x(ii,:)=x(ii,:)+v(ii,:);                  %update the BAT position
%         x(ii,:)=round(x(ii,:));
% Apply simple bounds/limits
Flag4up=x(ii,:)>ub;
```

```
Flag4low=x(ii,:)<lb;
x(ii,:)=(x(ii,:).*(~(Flag4up+Flag4low)))+ub.*Flag4up+lb.*Flag4low;
%check the condition with r
if rand>r(ii)
% The factor 0.001 limits the step sizes of random walks
%               x(ii,:)=bestsol+0.001*randn(1,dim);
eps=-1+(1-(-1))*rand;
x(ii,:)=bestsol+eps*mean(A);
end
fitnessnew=fobj(x(ii,:));  % calculate the objective function
% Update if the solution improves, or not too loud
if (fitnessnew<=fitness(ii)) && (rand<A(ii)) ,
fitness(ii)=fitnessnew;
A(ii)=alpha*A(ii);
r(ii)=ro*(1-exp(-gamma*iter));
end
if fitnessnew<=fmin,
bestsol=x(ii,:);
fmin=fitnessnew;
end
end
Convergence_curve(iter)=  fmin;
iter=iter+1;                              % update the while loop counter
end
[bestfit]=(fmin);
BestPositions=bestsol;
end
```

Appendix G
ALO Matlab Codes

```
%%%%%%%ALO consist of five different source codes:
%%%%%%%func_plot(func_name)
%%%%%%%Get_Functions_details(F)
%%%%%%%ALO(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
%%%%%%%initialization(SearchAgents_no,dim,ub,lb)
%%%%%%%main.m
%%%%%%%These codes must be saved under the same directory and "run main.m"
%%%%%%%Adapted from:
%%%%%%%https://www.mathworks.com/matlabcentral/fileexchange/49920-ant-lion-
optimizer-alo
```

```
%Get_Functions_details
function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
case 'F1'
fobj = @F1;
lb=0;
ub=10;
dim=2;
end
end
% F1
function o = F1(x)
o=((x(1)+2.*x(2)-7)^2)+((2.*x(1)+x(2)-5)^2);
end
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Main code
clear all
clc
SearchAgents_no=40; % Number of search agents
Function_name='F1'; % Name of the test function
Max_iteration=1000; % Maximum numbef of iterations
% Load details of the selected benchmark function
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
[Best_score,Best_pos,cg_curve]=ALO(SearchAgents_no,Max_iteration,lb,ub,dim,fo
bj);
figure('Position',[500 500 660 290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Test function')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
grid on
%Draw objective space
subplot(1,2,2);
semilogy(cg_curve,'Color','r')
title('Convergence curve')
xlabel('Iteration');
ylabel('Best score obtained so far');
axis tight
grid on
box on
legend('ALO')
display(['The best solution obtained by ALO is : ', num2str(Best_pos)]);
display(['The best optimal value of the objective function found by ALO is :
', num2str(Best_score)]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ALO code
function
[Elite_antlion_fitness,Elite_antlion_position,Convergence_curve]=ALO(N,Max_it
er,lb,ub,dim,fobj)
% Initialize the positions of antlions and ants
antlion_position=initialization(N,dim,ub,lb);
ant_position=initialization(N,dim,ub,lb);
% Initialize variables to save the position of elite, sorted antlions,
% convergence curve, antlions fitness, and ants fitness
Sorted_antlions=zeros(N,dim);
Elite_antlion_position=zeros(1,dim);
Elite_antlion_fitness=inf;
Convergence_curve=zeros(1,Max_iter);
antlions_fitness=zeros(1,N);
ants_fitness=zeros(1,N);
% Calculate the fitness of initial antlions and sort them
for i=1:size(antlion_position,1)
antlions_fitness(1,i)=fobj(antlion_position(i,:));
end
[sorted_antlion_fitness,sorted_indexes]=sort(antlions_fitness);
for newindex=1:N
Sorted_antlions(newindex,:)=antlion_position(sorted_indexes(newindex),:);
end
Elite_antlion_position=Sorted_antlions(1,:);
Elite_antlion_fitness=sorted_antlion_fitness(1);
% Main loop start from the second iteration since the first iteration
% was dedicated to calculating the fitness of antlions
Current_iter=2;
while Current_iter<Max_iter+1
% This for loop simulate random walks
for i=1:size(ant_position,1)
% Select ant lions based on their fitness
Rolette_index=RouletteWheelSelection(1./sorted_antlion_fitness);
if Rolette_index==-1
Rolette_index=1;
end
% RA is the random walk around the selected antlion by rolette wheel
```

```
RA=Random_walk_around_antlion(dim,Max_iter,lb,ub,
Sorted_antlions(Rolette_index,:),Current_iter);
% RA is the random walk around the elite (best antlion so far)
[RE]=Random_walk_around_antlion(dim,Max_iter,lb,ub,
Elite_antlion_position(1,:),Current_iter);
ant_position(i,:)= (RA(Current_iter,:)+RE(Current_iter,:))/2;
end
for i=1:size(ant_position,1)
% Boundary checking (bring back the antlions of ants inside search
% space if they go beyoud the boundaries
Flag4ub=ant_position(i,:)>ub;
Flag4lb=ant_position(i,:)<lb;
ant_position(i,:)=(ant_position(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*F
lag4lb;
ants_fitness(1,i)=fobj(ant_position(i,:));
end
% Update antlion positions and fitnesses based of the ants
double_population=[Sorted_antlions;ant_position];
double_fitness=[sorted_antlion_fitness ants_fitness];
[double_fitness_sorted I]=sort(double_fitness);
double_sorted_population=double_population(I,:);
antlions_fitness=double_fitness_sorted(1:N);
Sorted_antlions=double_sorted_population(1:N,:);
% Update the position of elite if any antlinons becomes fitter than it
if antlions_fitness(1)<Elite_antlion_fitness
Elite_antlion_position=Sorted_antlions(1,:);
Elite_antlion_fitness=antlions_fitness(1);
end
% Keep the elite in the population
Sorted_antlions(1,:)=Elite_antlion_position;
antlions_fitness(1)=Elite_antlion_fitness;
% Update the convergence curve
Convergence_curve(Current_iter)=Elite_antlion_fitness;
% Display the iteration and best optimum obtained so far
if mod(Current_iter,50)==0
display(['At iteration ', num2str(Current_iter), ' the elite fitness is ',
num2str(Elite_antlion_fitness)]);
end
Current_iter=Current_iter+1;
end
```

Appendix H
GOA Matlab Codes

```
%%%%%%%GOA consist of five different source codes:
%%%%%%%func_plot(func_name)
%%%%%%%Get_Functions_details(F)
%%%%%%%GOA(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
%%%%%%%initialization(SearchAgents_no,dim,ub,lb)
%%%%%%%main.m
%%%%%%%These codes must be saved under the same directory and "run main.m"
%%%%%%%Adapted from:
%%%%%%%https://www.mathworks.com/matlabcentral/fileexchange/61421-
grasshopper-optimisation-algorithm-goa
```

```
%Get_Functions_details
function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
case 'F1'
fobj = @F1;
lb=0;
ub=10;
dim=2;
end
end
% F1
function o = F1(x)
o=100.*sqrt(abs(x(2)-0.01.*(x(1)^2)))+0.01.*abs(x(1)+10);
end
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Main code
clear all
clc
SearchAgents_no=100; % Number of search agents
Function_name='F1'; % Name of test function
Max_iteration=1000; % Maximum numbef of iterations
% Load details of the selected benchmark function
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
[Target_score,Target_pos,GOA_cg_curve, Trajectories,fitness_history,
position_history]=GOA(SearchAgents_no,Max_iteration,lb,ub,dim,fobj);
figure('Position',[454   445   894   297])
%Draw search space
subplot(1,5,1);
func_plot(Function_name);
title('Parameter space')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
box on
axis tight
subplot(1,5,2);
hold on
for k1 = 1: size(position_history,1)
    for k2 = 1: size(position_history,2)
plot(position_history(k1,k2,1),position_history(k1,k2,2),'.','markersize',1,'
MarkerEdgeColor','k','markerfacecolor','k');
end
end
plot(Target_pos(1),Target_pos(2),'.','markersize',10,'MarkerEdgeColor','r','m
arkerfacecolor','r');
title('Search history (x1 and x2 only)')
xlabel('x1')
ylabel('x2')
box on
axis tight
subplot(1,5,3);
hold on
```

```
plot(Trajectories(1,:));
title('Trajectory of 1st grasshopper')
xlabel('Iteration#')
box on
axis tight
%subplot(1,5,4);
%hold on
%plot(mean(fitness_history));
%Average fitness of all grasshoppers')
%xlabel('Iteration#')
%box on
%axis tight
%Draw objective space
subplot(1,5,5);
semilogy(GOA_cg_curve,'Color','r')
title('Convergence curve')
xlabel('Iteration#');
ylabel('Best score obtained so far');
box on
axis tight
set(gcf, 'position' , [39         479        1727         267]);
display(['The best solution obtained by GOA is : ', num2str(Target_pos)]);
display(['The best optimal value of the objective funciton found by GOA is :
', num2str(Target_score)]);
```

%%

```
%GOA code
% The Grasshopper Optimization Algorithm
function
[TargetFitness,TargetPosition,Convergence_curve,Trajectories,fitness_history,
position_history]=GOA(N, Max_iter, lb,ub, dim, fobj)
disp('GOA is now estimating the global optimum for your problem....')
flag=0;
if size(ub,1)==1
ub=ones(dim,1)*ub;
lb=ones(dim,1)*lb;
end
if (rem(dim,2)~=0) % this algorithm should be run with a even number of
variables. This line is to handle odd number of variables
```

```
dim = dim+1;
ub = [ub; 100];
lb = [lb; -100];
flag=1;
end
%Initialize the population of grasshoppers
GrassHopperPositions=initialization(N,dim,ub,lb);
GrassHopperFitness = zeros(1,N);
fitness_history=zeros(N,Max_iter);
position_history=zeros(N,Max_iter,dim);
Convergence_curve=zeros(1,Max_iter);
Trajectories=zeros(N,Max_iter);
cMax=1;
cMin=0.00004;
%Calculate the fitness of initial grasshoppers
for i=1:size(GrassHopperPositions,1)
if flag == 1
GrassHopperFitness(1,i)=fobj(GrassHopperPositions(i,1:end-1));
else
GrassHopperFitness(1,i)=fobj(GrassHopperPositions(i,:));
end
fitness_history(i,1)=GrassHopperFitness(1,i);
position_history(i,1,:)=GrassHopperPositions(i,:);
Trajectories(:,1)=GrassHopperPositions(:,1);
end
[sorted_fitness,sorted_indexes]=sort(GrassHopperFitness);
% Find the best grasshopper (target) in the first population
for newindex=1:N
Sorted_grasshopper(newindex,:)=GrassHopperPositions(sorted_indexes(newindex),
:);
end
TargetPosition=Sorted_grasshopper(1,:);
TargetFitness=sorted_fitness(1);
% Main loop
l=2; % Start from the second iteration since the first iteration was
dedicated to calculating the fitness of antlions
while l<Max_iter+1
c=cMax-l*((cMax-cMin)/Max_iter);
```

```
for i=1:size(GrassHopperPositions,1)
temp= GrassHopperPositions';
for k=1:2:dim
S_i=zeros(2,1);
for j=1:N
if i~=j
Dist=distance(temp(k:k+1,j), temp(k:k+1,i)); % Calculate the distance between
two grasshoppers
r_ij_vec=(temp(k:k+1,j)-temp(k:k+1,i))/(Dist+eps); % xj-xi/dij
xj_xi=2+rem(Dist,2); % |xjd - xid|
s_ij=((ub(k:k+1) - lb(k:k+1))*c/2)*S_func(xj_xi).*r_ij_vec;
S_i=S_i+s_ij;
end
end
S_i_total(k:k+1, :) = S_i;
end
X_new = c * S_i_total'+ (TargetPosition);
GrassHopperPositions_temp(i,:)=X_new';
end
% GrassHopperPositions
GrassHopperPositions=GrassHopperPositions_temp;
for i=1:size(GrassHopperPositions,1)
% Relocate grasshoppers that go outside the search space
Tp=GrassHopperPositions(i,:)>ub';Tm=GrassHopperPositions(i,:)<lb';GrassHopper
Positions(i,:)=(GrassHopperPositions(i,:).*(~(Tp+Tm)))+ub'.*Tp+lb'.*Tm;
% Calculating the objective values for all grasshoppers
if flag == 1
GrassHopperFitness(1,i)=fobj(GrassHopperPositions(i,1:end-1));
else
GrassHopperFitness(1,i)=fobj(GrassHopperPositions(i,:));
end
fitness_history(i,l)=GrassHopperFitness(1,i);
position_history(i,l,:)=GrassHopperPositions(i,:);
Trajectories(:,l)=GrassHopperPositions(:,1);
% Update the target
if GrassHopperFitness(1,i)<TargetFitness
TargetPosition=GrassHopperPositions(i,:);
TargetFitness=GrassHopperFitness(1,i);
```

```
end
end
Convergence_curve(l)=TargetFitness;
disp(['In iteration #', num2str(l), ' , target''s objective = ',
num2str(TargetFitness)])
l = l + 1;
end
if (flag==1)
TargetPosition = TargetPosition(1:dim-1);
end
```

Appendix I
BOA Matlab Codes

```
%%%%%%%BOA consist of five different source codes:
%%%%%%%func_plot(func_name)
%%%%%%%Get_Functions_details(F)
%%%%%%%BOA(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
%%%%%%%initialization(SearchAgents_no,dim,ub,lb)
%%%%%%%main.m
%%%%%%%These codes must be saved under the same directory and "run main.m"
%%%%%%%Adapted from:
%%%%%%%https://www.mathworks.com/matlabcentral/mlc-
downloads/downloads/b4a529ac-c709-4752-8ae1-1d172b8968fc/67a434dc-8224-4f4e-
a835-bc92c4630a73/previews/BOA.m/index.html
```

```
%Get_Functions_details
function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
case 'F1'
fobj = @F1;
lb=0;
ub=10;
dim=2;
end
end
function o = F1(x)
o= (((x(1)^2)+(x(2))-11)^2)+(x(1)+(x(2)^2)-7)^2;
end
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Main code
clear all
clc
warning off all
SearchAgents_no=30; % Number of search agents
Max_iteration=1000; % Maximum number of iterations
Function_name='F1';
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
[Best_score,Best_pos,BOA_cg_curve]=BOA(SearchAgents_no,Max_iteration,lb,ub,di
m,fobj);
figure('Position',[500 500 660 290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Parameter space')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
%Draw objective space
subplot(1,2,2);
semilogy(BOA_cg_curve,'Color','r')
title('Objective space')
xlabel('Iteration');
ylabel('Best score obtained so far');
axis tight
grid on
box on
legend('BOA')
%Draw convergence curve
subplot(1,2,2);
semilogy(BOA_cg_curve,'Color','r')
title('Convergence curve')
xlabel('Iteration');
ylabel('Best score obtained so far');
axis tight
grid on
box on
```

```
legend('BOA')
display(['The best solution obtained by BOA is : ', num2str(Best_pos)]);
display(['The best optimal value of the objective function found by BOA is :
', num2str(Best_score)]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%BOA code
function [fmin,best_pos,Convergence_curve]=BOA(n,N_iter,Lb,Ub,dim,fobj)
% n is the population size
% N_iter represnets total number of iterations
p=0.8;                        % probabibility switch
power_exponent=0.1;
sensory_modality=0.01;
%Initialize the positions of search agents
Sol=initialization(n,dim,Ub,Lb);
for i=1:n,
Fitness(i)=fobj(Sol(i,:));
end
% Find the current best_pos
[fmin,I]=min(Fitness);
best_pos=Sol(I,:);
S=Sol;
% Start the iterations -- Butterfly Optimization Algorithm
for t=1:N_iter,
for i=1:n, % Loop over all butterflies/solutions
%Calculate fragrance of each butterfly which is correlated with objective
function
Fnew=fobj(S(i,:));
FP=(sensory_modality*(Fnew^power_exponent));
%Global or local search
if rand<p,
dis = rand * rand * best_pos - Sol(i,:);
S(i,:)=Sol(i,:)+dis*FP;
else
% Find random butterflies in the neighbourhood
epsilon=rand;
JK=randperm(n);
dis=epsilon*epsilon*Sol(JK(1),:)-Sol(JK(2),:);
S(i,:)=Sol(i,:)+dis*FP;
```

```
end
% Check if the simple limits/bounds are OK
S(i,:)=simplebounds(S(i,:),Lb,Ub);
% Evaluate new solutions
Fnew=fobj(S(i,:));  %Fnew represents new fitness values
% If fitness improves (better solutions found), update then
if (Fnew<=Fitness(i)),
Sol(i,:)=S(i,:);
Fitness(i)=Fnew;
end
% Update the current global best_pos
if Fnew<=fmin,
best_pos=S(i,:);
fmin=Fnew;
end
end
Convergence_curve(t,1)=fmin;
%Update sensory_modality
sensory_modality=sensory_modality_NEW(sensory_modality, N_iter);
end
% Boundary constraints
function s=simplebounds(s,Lb,Ub)
% Apply the lower bound
ns_tmp=s;
I=ns_tmp<Lb;
ns_tmp(I)=Lb;
% Apply the upper bounds
J=ns_tmp>Ub;
ns_tmp(J)=Ub;
% Update this new move
s=ns_tmp;
function y=sensory_modality_NEW(x,Ngen)
y=x+(0.025/(x*Ngen));
```

Appendix J
MFO Matlab Codes

```
%%%%%%%MFO consist of five different source codes:
%%%%%%%func_plot(func_name)
%%%%%%%Get_Functions_details(F)
%%%%%%%MFO(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
%%%%%%%initialization(SearchAgents_no,dim,ub,lb)
%%%%%%%main.m
%%%%%%%These codes must be saved under the same directory and "run main.m"
%%%%%%%Adapted from:
%%%%%%% https://www.mathworks.com/matlabcentral/fileexchange/52269-moth-flame-
optimization-mfo-algorithm
```

```
%Get_Functions_details
function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
case 'F1'
fobj = @F1;
lb=0;
ub=10;
dim=2;
end
end
% F1
function o = F1(x)
o=(1+((x(1)+x(2)+1)^2)*(19-(14.*x(1))+(3.*(x(1)^2)-
14.*x(2)+(6.*x(1)*x(2))+3.*(x(2)^2))))*(30+((2.*x(1)-3.*x(2))^2)*(18-
32.*x(1)+12.*x(1)^2+48.*x(2)-36.*x(1)*x(2)+27.*x(2)^2));
end
```


M. O. Okwu and L. K. Tartibu, *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications*, Studies in Computational Intelligence 927, https://doi.org/10.1007/978-3-030-61111-8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Main code
clear all
clc
SearchAgents_no=30; % Number of search agents
Function_name='F1'; % Name of the function
Max_iteration=100; % Maximum numbef of iterations
% Load details of the selected benchmark function
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
[Best_score,Best_pos,cg_curve]=MFO(SearchAgents_no,Max_iteration,lb,ub,dim,fo
bj);
figure('Position',[284   214   660   290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Test function')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
grid on
%Draw objective space
subplot(1,2,2);
semilogy(cg_curve,'Color','b')
title('Convergence curve')
xlabel('Iteration');
ylabel('Best flame (score) obtained so far');
axis tight
grid on
box on
legend('MFO')
display(['The best solution obtained by MFO is : ', num2str(Best_pos)]);
display(['The best optimal value of the objective funciton found by MFO is :
', num2str(Best_score)]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%MFO code
function
[Best_flame_score,Best_flame_pos,Convergence_curve]=MFO(N,Max_iteration,lb,ub
,dim,fobj)
display('MFO is optimizing your problem');
%Initialize the positions of moths
Moth_pos=initialization(N,dim,ub,lb);
Convergence_curve=zeros(1,Max_iteration);
Iteration=1;
% Main loop
while Iteration<Max_iteration+1
% Number of flames Equation
Flame_no=round(N-Iteration*((N-1)/Max_iteration));
for i=1:size(Moth_pos,1)
% Check if moths go out of the search spaceand bring it back
Flag4ub=Moth_pos(i,:)>ub;
Flag4lb=Moth_pos(i,:)<lb;
Moth_pos(i,:)=(Moth_pos(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb;
% Calculate the fitness of moths
Moth_fitness(1,i)=fobj(Moth_pos(i,:));
end
if Iteration==1
% Sort the first population of moths
[fitness_sorted I]=sort(Moth_fitness);
sorted_population=Moth_pos(I,:);
% Update the flames
best_flames=sorted_population;
best_flame_fitness=fitness_sorted;
else
% Sort the moths
double_population=[previous_population;best_flames];
double_fitness=[previous_fitness best_flame_fitness];
[double_fitness_sorted I]=sort(double_fitness);
double_sorted_population=double_population(I,:);
fitness_sorted=double_fitness_sorted(1:N);
sorted_population=double_sorted_population(1:N,:);
% Update the flames
```

```
best_flames=sorted_population;
best_flame_fitness=fitness_sorted;
end
% Update the position best flame obtained so far
Best_flame_score=fitness_sorted(1);
Best_flame_pos=sorted_population(1,:);
previous_population=Moth_pos;
previous_fitness=Moth_fitness;
% a linearly dicreases from -1 to -2 to calculate t
a=-1+Iteration*((-1)/Max_iteration);
for i=1:size(Moth_pos,1)
for j=1:size(Moth_pos,2)
if i<=Flame_no % Update the position of the moth with respect to its
corresponsing flame
distance_to_flame=abs(sorted_population(i,j)-Moth_pos(i,j));
b=1;
t=(a-1)*rand+1;
Moth_pos(i,j)=distance_to_flame*exp(b.*t).*cos(t.*2*pi)+sorted_population(i,j
);
end
if i>Flame_no % Upaate the position of the moth with respct to one flame
distance_to_flame=abs(sorted_population(i,j)-Moth_pos(i,j));
b=1;
t=(a-1)*rand+1;
Moth_pos(i,j)=distance_to_flame*exp(b.*t).*cos(t.*2*pi)+sorted_population(Fla
me_no,j);
end
end
end
Convergence_curve(Iteration)=Best_flame_score;
% Display the iteration and best optimum obtained so far
if mod(Iteration,50)==0
display(['At iteration ', num2str(Iteration), ' the best fitness is ',
num2str(Best_flame_score)]);
end
Iteration=Iteration+1;
end
```