Neural Computing and Applications
https://doi.org/10.1007/s00521-020-05145-6

ORIGINAL ARTICLE

A novel meta-heuristic search algorithm for solving optimization problems: capuchin search algorithm

Malik Braik[1] · Alaa Sheta[2] · Heba Al-Hiary[1]

Received: 2 December 2019 / Accepted: 17 June 2020

Abstract
Meta-heuristic search algorithms were successfully used to solve a variety of problems in engineering, science, business, and finance. Meta-heuristic algorithms share common features since they are population-based approaches that use a set of tuning parameters to evolve new solutions based on the natural behavior of creatures. In this paper, we present a novel nature-inspired search optimization algorithm called the capuchin search algorithm (CapSA) for solving constrained and global optimization problems. The key inspiration of CapSA is the dynamic behavior of capuchin monkeys. The basic optimization characteristics of this new algorithm are designed by modeling the social actions of capuchins during wandering and foraging over trees and riverbanks in forests while searching for food sources. Some of the common behaviors of capuchins during foraging that are implemented in this algorithm are leaping, swinging, and climbing. Jumping is an effective mechanism used by capuchins to jump from tree to tree. The other foraging mechanisms exercised by capuchins, known as swinging and climbing, allow the capuchins to move small distances over trees, tree branches, and the extremities of the tree branches. These locomotion mechanisms eventually lead to feasible solutions of global optimization problems. The proposed algorithm is benchmarked on 23 well-known benchmark functions, as well as solving several challenging and computationally costly engineering problems. A broad comparative study is conducted to demonstrate the efficacy of CapSA over several prominent meta-heuristic algorithms in terms of optimization precision and statistical test analysis. Overall results show that CapSA renders more precise solutions with a high convergence rate compared to competitive meta-heuristic methods.

Keywords Optimization · Meta-heuristics · Bio-inspired algorithms · Capuchin search algorithm · Optimization techniques

1 Introduction

Over the past few decades, a wide variety of traditional algorithms based on linear and nonlinear programming methods have been used to address a variety of real-world problems in engineering [32, 85], science [16, 33], finance [31, 57], business [79], and economics [69, 104]. These traditional methods may require large gradient information and usually demand to improve the solution in the neighborhood of a starting point. In many cases, these methods have delivered a beneficial strategy for getting the global or near-global optimum solution for problems of modest complexity [69, 85]. Often, many of the real-world engineering problems are highly nonlinear, complicated in nature, include complex objective functions with a large number of various design variables that are customarily subject to many hard constraints [19, 24]. The objective functions of nonlinear engineering design optimization problems often contain many local optima, while the designer is always concerned with locating the global optimum solution.

Typically, traditional optimization methods find it difficult to efficiently solve complex engineering design

✉ Malik Braik
mbraik@bau.edu.jo

Alaa Sheta
shetaa1@southernct.edu

Heba Al-Hiary
hhiary@bau.edu.jo

[1] Al-Balqa Applied University, Salt, Jordan

[2] Computer Science Department, Southern Connecticut State University, 501 Crescent St., New Haven, CT 06515, USA

Published online: 05 July 2020

problems or provide an appropriate solution for them. These methods can only find local optima, and there is no guarantee to find the global optimum solution [59, 82]. In some cases, traditional methods are fragile and fail when the problem design is subject to a large number of constraints or when the problem has a large number of decision variables [2, 4]. This poses a substantial challenge in accurately finding the global optimality of interest. To overcome such issues and identify the optimal global solution for complex engineering problems, researchers rely on meta-heuristic algorithms [3, 29].

Meta-heuristic algorithms are global optimization methods that are designed based on simulations and nature-inspired methods [72, 92]. These nature-inspired algorithms have been known for a few decades. Understanding more of the social swarm behavior in fish, birds, ants, and many others have helped researchers to create optimization search algorithms that have been successfully used to tackle a variety of real-world problems in engineering [17, 21]. Surprisingly, these creatures can search to find optimal solutions and accomplish missions competently in groups. Therefore, it is rationale that we stimulate their behaviors to optimization problems. This field of research is defined as swarm intelligence (SI), which was coined in [12]. In the past few decades, many swarm-based meta-heuristic algorithms have emerged, such as the particle swarm optimization (PSO) [21], ant colony optimization (ACO) [17], artificial bee colony (ABC) algorithm [43], firefly algorithm (FA) [95], cuckoo search (CS) algorithm [99], bat algorithm (BA) [97], fruit fly optimization algorithm (FOA) [77], krill herd (KH) algorithm [23], grey wolf optimizer (GWO) [88], dragonfly algorithm (DA) [64], whale optimization algorithm (WOA) [67], moth search (MS) algorithm [92] and many more [47, 89, 98]. Social swarms always work cooperatively to achieve a goal or set of goals, such as searching for food, hunting, protecting a nest, or navigating.

Remarkably, all nature-inspired algorithms share some common characteristics regardless of their nature. For example, they mimic some natural phenomena, do not require gradient information, have several control parameters that must be appropriately defined to address a particular problem, and they use random operators. In such a context, meta-heuristic algorithms avail from random generators because they are a member of a family related to stochastic optimization algorithms [54, 100]. This feature of using random operators supports meta-heuristic algorithms to eschew local solutions in addressing optimization problems, which usually have a large amount of local optima [55, 66].

Meta-heuristic algorithms have achieved promising performance in addressing problems in several areas such as biomedical signal processing [74], process control [75], image processing [6, 45], text clustering [83], classification problems [18], and many other contemporaneous problems [70, 101, 103]. This broad popularity and extensive use of nature-inspired algorithms in solving a broad range of real-world optimization problems are attributed to many wonderful characteristics of meta-heuristic algorithms such as (1) their adaptability and flexibility, (2) they are gradient-free techniques, and (3) they can avoid local optimum solutions [66, 92, 100]. The first two key features stem from the fact that meta-heuristic methods address optimization problems by considering only the input and output parameters. It is merited to mention that optimization is a process that quests for the optimal potential solution for a particular optimization problem given a range of constraints according to the complexity of the problem and its type. Strictly speaking, meta-heuristic algorithms presume optimization problems as black boxes. Thus, there is no need to compute the derivative of the search domain. This makes meta-heuristic algorithms flexible in solving a wide variety of optimization problems [54, 98].

Meanwhile, it is not guaranteed that every nature-inspired algorithm reported in the literature can obtain global optimum solutions for all kinds of optimization problems. It is hoped that these algorithms work efficiently and reliably in most types of optimization problems, but not in all types of optimization problems. Some algorithms proffer a better solution for some particular problems compared with others. It is worth mentioning that advances in technology have led to the emergence of many challenging real-world problems [80] in different areas of engineering and science [8, 36, 66]. So, researchers in the scientific community are still developing new optimization techniques inspired by nature to solve new and more real-world optimization problems under the ideology of continuous improvement in order to arrive at a better solution. However, it turns out, based on the "no-free-lunch" (NFL) theorem [51], that there is no generic nature-inspired optimization algorithm that can optimally solve all real-world optimization problems [94]. This means that an optimization algorithm is suitable to solve a specific set of optimization problems, but it might not be effective to solve other sets of optimization problems [63, 66]. Obviously, the NFL theorem certainly causes this field of research to remain open and allows researchers to enhance the performance of existing meta-heuristic algorithms or suggest new meta-heuristic algorithms for better optimization and further accuracy [91, 102]. Therefore, developing new optimization techniques is an open problem [94]. Also, to date, researchers have only used limited characteristics of biological behavior or natural phenomena found in nature, and there is room for developing new nature-inspired algorithms that can be used to solve real-world problems that are constantly emerging in the world. A short review of some of

the well-known nature-inspired meta-heuristic algorithms is presented in Table 1.

In this work, a novel nature-inspired algorithm, called the capuchin search algorithm (CapSA), for solving constrained and global optimization problems are proposed. This algorithm simulates the foraging behavior of capuchin monkeys and their effective methods of locomotion, widely known as leaping, swinging, and climbing. As a group, capuchins show wonderful examples of mobility and often score a high degree of creativity in locomotion while roaming around and searching for food sources. They can jump craftily on trees, swing on tree branches, and climb

Table 1 A brief literature review on meta-heuristic search algorithms

Algorithm	Inspiration	Year
Genetic algorithm [38]	Evolutionary concepts	1975
Simulated annealing [50]	Annealing process in metallurgy	1983
Particle swarm optimization [21]	Collective social behavior of bird swarms	1995
Artificial fish-swarm algorithm [55]	Intelligent social behavior of fish swarms	2003
Termite algorithm [58]	Termite colony	2006
Ant colony optimization algorithm [20]	Ant colony	2006
Artificial bee colony algorithm [10]	Honey bee	2006
Imperialist competitive algorithm [9]	Imperialistic competition	2007
Monkey search [71]	Monkey behavior on trees during foraging	2007
Group search optimizer [36]	Foraging behavior of animals	2009
Firefly algorithm [95]	Social behavior of fireflies	2009
Gravitational search algorithm [84]	Gravity law and mass interactions	2009
Bat algorithm [97]	Echolocation behavior of bats	2010
Flower pollination algorithm [98]	Pollination process of flower species	2012
Fruit fly optimization algorithm [77]	Fruit foraging behavior of fruit fly	2012
Krill herd [23]	Herding behavior of krill individuals	2012
Mine blast algorithm [86]	Mine bomb explosion	2013
Dolphin echolocation [47]	Echolocation ability of dolphins	2013
Lightning search algorithm [89]	Natural phenomenon of lightning	2015
Dragonfly algorithm [64]	Swarming behavior of dragonflies	2015
Artificial algae algorithm [91]	Living behavior of micro-algae	2015
Ant lion optimizer [62]	Hunting method of ant lions	2015
Shark smell optimization [1]	Sharks mechanism to locate their prey by smell sense	2016
Dolphin swarm optimization algorithm [102]	Dolphins mechanisms in chasing and catching sardine swarms	2016
Virus colony search algorithm [54]	Virus infection and diffusion strategies	2016
Whale optimization algorithm [67]	Intelligent Social behavior of humpback whales	2016
Multi-verse optimizer [68]	Multi-verse theory	2016
Crow search algorithm [8]	Intelligent behavior of crows to locate hidden food	2016
Salp swarm algorithm [66]	Intelligent behavior of salps in foraging in oceans	2017
Grasshopper optimization algorithm [87]	Intelligent swarming behavior of grasshoppers	2017
Selfish herd optimizer [22]	Hamilton's selfish herd theory	2017
Electro-search algorithm [90]	Orbital movement of electrons around the atomic nucleus	2017
Thermal exchange optimization [46]	Newton's law of cooling	2017
Mouth brooding fish algorithm [40]	Life cycle of mouth brooding fish	2017
Weighted superposition attraction [11]	Superposition precept	2017
Spotted hyena optimizer [19]	Social behavior of spotted hyenas	2017
Butterfly-inspired algorithm [81]	Mate searching process of butterfly	2017
Lightning attachment procedure optimization [73]	Process of lightning attachment	2017
Squirrel search algorithm [41]	Simulation of the locomotion behaviors of squirrels	2019
Coral reefs optimization [26, 27]	Simulation of coral reefs	2019

on trees. In a capuchin flock, this behavior has many similarities with the optimization process, where food is represented by coveted solutions in the global optimization problem. The main contributions of the proposed algorithm are as follows:

1. A novel nature-inspired algorithm that thoroughly simulates the methods of capuchins in searching for food sources is proposed.
2. The performance of the proposed CapSA was assessed on 23 publicly available test problems, where a comprehensive comparative study was conducted with other existing nature-inspired algorithms. The importance of the experimental results is supported by statistical test analysis.
3. Further evaluation of CapSA is tested on four engineering design problems, and the obtained results are compared with other optimization algorithms.

The paper is organized as follows. In Sect. 2, we present the inspiration concepts of the proposed algorithm. In Sect. 3, we provide the basic concepts behind the proposed algorithm. Section 4 describes the mathematical model and implementation of the proposed algorithm. Section 5 presents an analysis of the proposed algorithm. The optimization results of the proposed method and other optimization methods on 23 test functions are presented and discussed in Sect. 6. Statistical test results are performed in Sect. 7. In Sect. 8, the performance of the proposed method is validated on different engineering problems and compared with other optimization methods. Finally, concluding comments of the main findings of the proposed algorithm and the most promising paths for future directions are presented in Sect. 9.

2 Inspiration

Capuchin monkeys live in a wide area of Brazil and other parts of Central and South America as far south as northern Argentina. They are native to tropical climates as well as dry forests from Nicaragua to Paraguay.

Social structure Capuchin monkeys are arboreal, highly social, diurnal, and territorial animals, where they spend most of their time in groups composing of adult males and females as well as small apes. They leap together on the top of tall trees while searching for food and engaging as a family. Meanwhile, they roam from the bottom of the forest to the canopy over the vertical range of their habitats. A herd's range of capuchins covers between 50 and 100 hectares, and the individuals wander approximately 1.9 miles a day within the range. In fact, in the wild, capuchins are extremely energetic throughout the day and often feed on a large variety of food types [15].

Communication Capuchin members share their observations and intentions within groups using postures, barking, calling, and many other physical activities. They interact within and outside the group according to the availability of food sources as well as attack postures. During foraging, group members interact with each other and with the leader over prolonged distances through a specific call that sounds like a horse's squeak. Each individual in the capuchins' group has its distinctive personal sound so that other members of the group can recognize the caller. This far-reaching communication allows capuchins to get-together, stay away from any attack, and share food sources and gossip.

Intelligence Capuchin monkeys are famed for their high level of intelligence and curiosity and are deemed the most intelligent new world monkeys [5]. This is due to that capuchins have probably large brain sizes compared to their body sizes [5], and in general, perform cognitive tasks [5].

Organization and foraging behavior Capuchins typically subsist in groups consisting of 10 to 35 individuals per group, where the group's members devote most of their time searching for food. Capuchins, in groups, move together and collect food throughout the day within a core area of the home range. Typically, a single male dominates a group, referred to as the alpha male. Nevertheless, the white-headed capuchin groups are commanded by an alpha female and also an alpha male [76]. A male-dominance hierarchy customarily distinguishes a cohabiting capuchin breeding group. Typically, a single alpha-male and many sub-alpha females and males live together. Each group covers a relatively large area in the forest, where members of each group search for the best areas within the forest that contain food sources.

Capuchins attractively find their food sources: An alpha male leads the group and is responsible for locating food sources for the group's members. In some species of capuchins, a single alpha male and a single alpha female lead the group to find food sources within the forest. In cases where they do not find enough food sources for the capuchins in the group, the group is split into smaller subgroups that forage independently [76].

Research on social learning mechanisms has found that capuchins share similar propensities in foraging skills and produced evidence for traditions in wild populations, in which they have developed social conventions, apparently to explore social bonds [15, 76]. The most interesting facts about capuchins are that they use three fantastic methods to

navigate while searching for food sources: leaping, swinging, and climbing.

Capuchins can leap up to three meters, and they practice this way of movement to move from one tree to another. They can move (1) from a branch to another branch on the same tree, (2) one place to another place on the ground or (3) from a riverbank to the other bank of the river. Moreover, capuchins swing and climb on trees and their branches to search locally for food sources on tree branches.

Capuchins ordinarily climb trees again and again and swing on tree branches, identifying the not visited branches with a substantial probability toward the branches with better values, until they reach undiscovered frontiers of the trees. The leaping movement allows capuchins to stretch to a global-wide search space, which would be beneficial to be used to address complex optimization problems. In this context, capuchins roam the trees and the extremities of trees' branches until they approach a particular area of the tree where food sources (i.e., feasible solutions) may be identified. Capuchins are not confined to their search on branches of a single tree; the search space may be instead considered as a forest. Each time an alpha capuchin locates a food source (i.e., best solution) with a better objective value; this solution is stored as the best solution.

The followers update their locations following their current locations and the leader's location. Capuchins reach the tops of trees and all their branches during the search for food. Whenever food sources (i.e., global solutions) are found, the positions of all capuchins are updated, and the search process (using leaping, swinging, and climbing) is reiterated, branch by branch, tree by tree and riverbanks, which may lead to food sources (i.e., better solutions). This process allows the exploration and exploitation of all positions in the search area. There are three strategies followed by the leader of the capuchins and the followers during foraging:

- *Strategy A*: A male alpha (global leader) commands the group and is liable for finding food sources, where members of the group follow the leader to get food sources.
- *Strategy B*: In certain types of capuchins, both male and female alphas (global leaders) command the group and are liable for identifying food, where the group's members follow the leaders.
- *Strategy C*: Individuals seek for food sources independently, but they follow group members who find food sources.

The foraging behavior of capuchins within and outside the group is featured into five cases:

1. In the first case, the leaders within the group commence to search for food and assess their position from food sources.
2. In the second case, depending on the leaders' position and the identification of food sources, the group members update their positions toward the food sources.
3. In the third case, the followers update their positions within the group by following the leaders.
4. In the fourth case, the leaders update their best ever positions to locate the food sources.
5. In the fifth case, if the positions of the leaders and followers are not updated for several iterations, the group members start to search for food sources in different directions.

The above five cases are carried out continuously until the food source (i.e., the desired solution) is reached. Based upon the above description, we can vouch that the intelligent foraging behavior of capuchins is the primary source of motivation that encouraged us to develop the meta-heuristic algorithm described below.

3 Capuchin search algorithm

The broad objective of this research is to evolve a new meta-heuristic algorithm that focuses on capuchin monkeys as a concerted primate model of social behavior of animals questing for food. To do so, we have investigated the behavioral strategies of capuchins in contexts related to their effective skills of traveling around and searching for food sources on trees, ground, and riverbanks. To develop such an algorithm, we have developed a mathematical model consisting of global and local search strategies as described below.

3.1 Global search: leaping motion

Capuchins typically move about large distances to search for food sources on trees by jumping from one tree to another or may glide to the ground to search for food on the banks of a river. So, the jumping mechanism is analogous to global search. These foraging strategies are illustrated in Fig. 1.

It is perceived from Fig. 1 that the realistic movement of a capuchin during leaping from a tree to another tree is similar to the motion of projectiles as pictured in the projectile curved path and the curved motion of the capuchin in Fig. 2a, b, respectively.

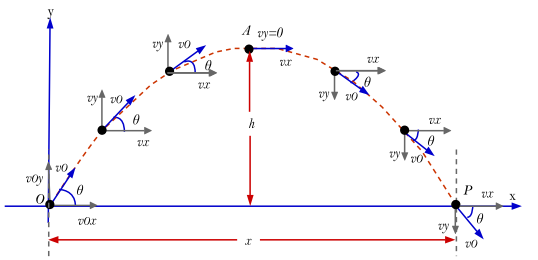
As clearly discerned from Fig. 2b, the real distance moved by the capuchin implements the horizontal distance, x, between the source tree branch and the target tree

(a)

(b)

Fig. 1 **a** Different stages of capuchin movement during leaping from a tree to another tree and **b** different stages of capuchin movement during leaping from a riverside to another riverside

(a)



(b)

Fig. 2 **a** A realistic model of capuchin hop using an approximate curved motion and **b** a curved model simulating the leaping motion of the capuchin in (**a**)

branch. We can use the third law of motion to express x as shown below:

$$x = x_0 + v_0 t + \frac{1}{2} a t^2 \tag{1}$$

where x stands for the new position of the capuchin, x_0 represents the initial position, v_0 represents the initial velocity, a represents the acceleration and t identifies the time instance.

The velocity of the capuchin, v, during the jumping movement can be identified using the first law of motion as shown in Eq. 2.

$$v = v_0 + at \tag{2}$$

The x and y components of the initial velocity of the capuchin in the motion model displayed in Fig. 2b can be defined as:

$$\begin{aligned} v_{0x} &= v_0 \cos(\theta_0) \\ v_{0y} &= v_0 \sin(\theta_0) \end{aligned} \tag{3}$$

where v_{0x} and v_{0y} are the initial velocities in the x and y-directions, respectively, and θ_0 is the angle measured from the positive x-direction. The horizontal velocity, v_x, can be derived from Eqs. 2 and 3 as follows:

$$\begin{aligned} v_x &= v_{0x} + a_x t \\ &= v_0 \cos(\theta_0) \end{aligned} \tag{4}$$

where a_x is the horizontal acceleration that is set to 0. The distance, x, of the capuchin can then be simplified as:

$$x = x_0 + v_0 \cos(\theta_0) t. \tag{5}$$

The vertical displacement, h, in Fig. 2b, can then be derived as:

$$y = y_0 + v_0 \sin(\theta_0) t + \frac{1}{2} a_y t^2 \tag{6}$$

where y_0 is the initial position and a_y is the vertical acceleration of the capuchin, which represents a downward acceleration due to gravity that is equal to $-g$. Equation 6 can be solved to find t when the height h is equal to the launch height y_0 as follows:

$$t = 2 v_0 \sin(\theta_0)/g \tag{7}$$

Substitute Eq. 7 into Eq. 5 and use $\sin(2\theta_0) = 2\sin(\theta_0)\cos(\theta_0)$ to get:

$$x = x_0 + v_0^2 \sin(2\theta_0)/g \tag{8}$$

where x is the new position of the capuchin, x_0 is the initial position, v_0 is the initial velocity, θ_0 is the leaping angle and g is the gravitational acceleration.

3.2 Local search: swinging motion

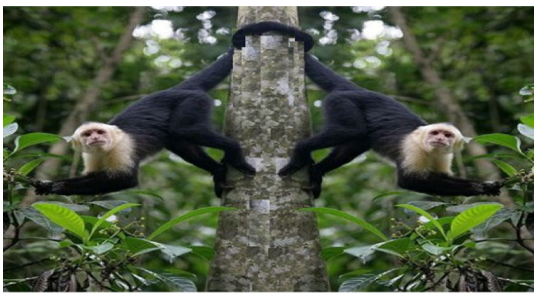
Capuchins use their tails, legs, or hands to wobble on tree branches during foraging in order to meet their daily food needs. This motion strategy simulates a local search process and is illustrated by two models. Figure 3 shows that the realistic motion of a capuchin during swinging on trees is similar to that of a pendulum as pictured in the swinging motion of the capuchin and the path of the pendulum motion in Fig. 4a, b, respectively.

In Fig. 4b, the blue ball simulates a capuchin of mass, m, swinging on a tree branch using its tail of length L. The ball is displaced from its equilibrium position by an angle θ with a distance x, along an arc in which the capuchin sways. There are two forces acting on the capuchin at position P, which are:

- the weight mg that acts downward vertically, and
- the tension T of the tail that acts along the direction of the tail.

The weight mg can be resolved into:

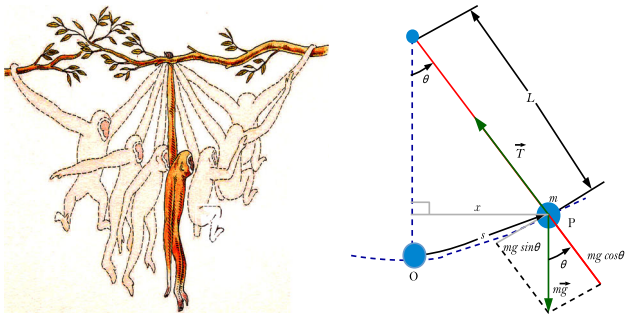
- $mg\cos\theta$ which identifies the weight along the tail direction, and
- $mg\sin\theta$ that identifies the weight along the arc in the direction of the capuchin movement.



(a)

(b)

Fig. 3 **a** A swinging motion for a capuchin on a tree branch using its tail and **b** a swinging motion for a capuchin on a tree branch using its hands



(a) **(b)**

Fig. 4 **a** A conceptual model simulating the swinging motion of a capuchin on a tree branch using its hands and **b** a pendulum model representing the motion of an object as a pendulum motion

According to trigonometry rules, the position, x, during the swinging motion of the capuchin in Fig. 4a can be defined as:

$$x = L\sin\theta \tag{9}$$

where θ is the angle at which the capuchin is moved from position O.

3.3 Local search: climbing motion

Capuchins climb trees and tree branches and their tips to intensify the search for food in nearby areas to meet their daily needs. This strategy represents a local search mechanism as evidenced by two models of capuchins climbing up and down trees as shown in Fig. 5. This climbing movement could be described by a representative model and a conceptual model as shown in Fig. 6a, b, respectively.

The position of the capuchin during climbing or sliding a tree can be obtained using a collaboration of Eqs. 1 and 2, and the result is shown in Eq. 10.

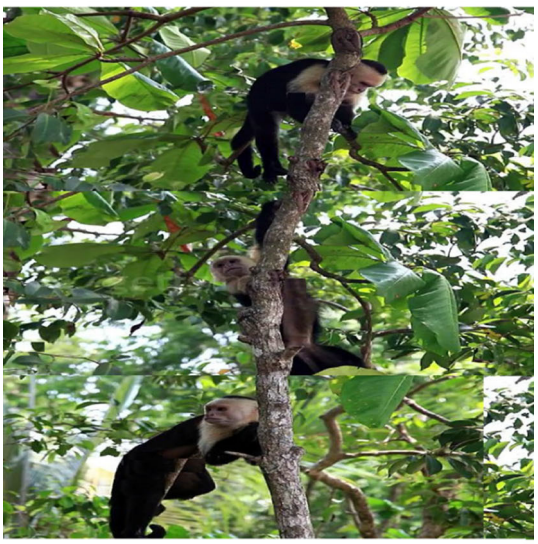
$$x = x_0 + v_0 t + \frac{1}{2}(v_f - v_0)t^2 \tag{10}$$

where t represents the iteration in optimization that identifies the discrepancy between iterations, which is always equal to 1.

4 Mathematical model of CapSA

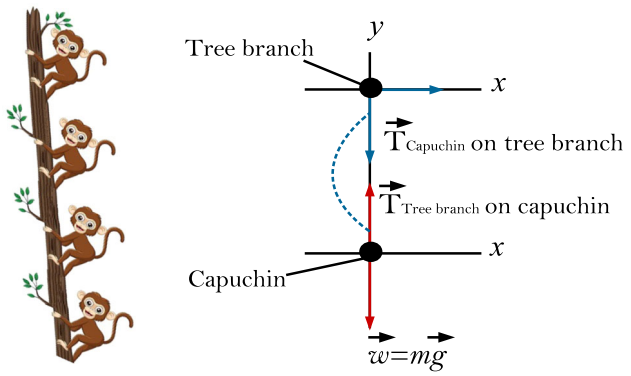
In this section, we present the underlying mathematical model of capuchin monkeys' behavior in locomotion and wandering on trees and ground during a search for food sources.

(a)



(b)

Fig. 5 **a** A capuchin climbs a tree and **b** a capuchin comes down from a tree



(a) (b)

Fig. 6 **a** A conceptual model of a capuchin climbing up a tree and **b** an approximate model of the climbing behavior of a capuchin

4.1 Initialization of CapSA

Like other SI-based algorithms, CapSA can be described as a population-based algorithm that commences by initializing a predetermined number of individuals (i.e., particles) at random. Each of these individuals constitutes a candidate solution to the target problem. There are two varieties of individuals in capuchins' swarms: alpha (i.e., leader) and followers. The leader is an alpha male or female or both in some capuchins' families. The leader typically guides the followers in their locomotion. The followers are the rest of the capuchins in the group.

A swarm of n capuchins in an d-dimensional search domain can be represented by a matrix, x of d dimension (see Eq. 11).

$$x = \begin{bmatrix} x_1^1 & x_2^1 & \dots & \dots & x_d^1 \\ x_1^2 & x_2^2 & \dots & \dots & x_d^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^n & x_2^n & \dots & \dots & x_d^n \end{bmatrix} \tag{11}$$

where x represents the positions of capuchins, d is the number of variables of a test problem, n stands for the number of capuchins and x_d^i represents the dth dimension of the position of the ith capuchin.

Equation 12 can be used to allocate the initial location of each capuchin in the swarm.

$$x^i = ub_j + r \times (ub_j - lb_j) \tag{12}$$

where lb_j and ub_j stand for the lower and upper bounds of the ith capuchin in the jth dimension, respectively, and r is a random number that is uniformly generated in the range from 0 to 1.

4.2 Evolutionary process of CapSA

In any meta-heuristic algorithm, there is a population of individuals who change their locations according to some synthesis parameters, so they can better search the space for all possible solutions to locate optimal or sub-optimal solutions. The population of CapSA consists of two types; (1) the leader of the group, referred to as alpha leader, and (2) the followers. Other capuchins may also accompany the leader in foraging behavior and pursue similar locomotion behavior.

The evolutionary process of CapSA relies on the current and best positions of the capuchins as well as the food source called F. The food source, F, is the capuchins' target in the d-dimensional search domain that needs to be iteratively updated. The mathematical model that describes the dynamic behavior of the alpha and the accompanying capuchins of the leader in relation to F and their new positions is formulated as described below.

Jumping on trees Alpha capuchins may jump from tree to tree or from a tree branch to another branch of the same

tree. In this case, the position of alpha capuchins can be given as follows:

$$x_j^i = F_j + \frac{P_{bf}(v_j^i)^2 \sin(2\theta)}{g} \quad i < n/2;\ 0.1 < \epsilon \leq 0.20. \tag{13}$$

The main attributes of Eq. 13 are defined as follows:

- x_j^i identifies the position of the alpha capuchins and the other escorting capuchins in the jth dimension,
- F_j is the position of the food in the jth dimension,
- ϵ is a random number generated uniformly in the interval [0, 1],
- P_{bf} is the probability of the balance provided by the tail of the capuchins during the leaping movement,
- g is the gravitational force which is equal to 9.81,
- θ is the jumping angle of the capuchins,
- τ is a lifetime parameter that is systematically decreased throughout iterations, and
- v_j^i is the velocity of the ith capuchin in the jth dimension.

The jumping angle of the capuchins θ can be given as shown in Eq. 14.

$$\theta = \frac{3}{2} r \tag{14}$$

where r is a random number generated uniformly in the interval [0, 1].

The performance level of any meta-heuristic algorithm is judged by its ability to strike a balance between exploration and exploitation. At the onset of the search, the algorithm requires exploring the search space with a predefined number of solutions in the capuchin's population. Then, while searching for food sources, the need for exploration diminishes while the need for exploitation grows.

Life time of CapSA In this scope, a lifetime exponential function, τ, was proposed in CapSA to strike a balance between exploration and exploitation during global and local search processes as shown in Eq. 15.

$$\tau = \beta_0 e^{-\beta_1 (\frac{k}{K})^{\beta_2}} \tag{15}$$

where k and K denote the current and maximum iteration values, respectively. The parameters β_0, β_1 and β_2 are arbitrarily selected as 2, 21 and 2.

The proposed algorithm uses the exponential function τ to explore the search space with a reasonable number of solutions, where the individuals exploit each search area to find the best solutions. The values produced by this function have a large impact on exploration and exploitation, as the proper values of this parametric formula reinforce the exploration and exploitation capabilities of the proposed algorithm. This balance is retained by adaptively locating the parameters β_0, β_1, and β_2 during iterations. The values of these parameters were obtained by rigorous analysis and experimentation on a large number of benchmark functions to achieve acceptable and satisfactory performance of the proposed algorithm. Obviously, they determine if the following position in the jth dimension should be in the direction of $-\infty$ or $+\infty$ as well as the step size. In more specific words, τ helps CapSA to effectively update the positions of the capuchins to rapidly locate the food source by exploring and exploiting the surrounding area, and its value greatly impacts the accuracy of CapSA.

The velocity of the ith capuchin in the jth dimension can be computed as given below.

$$v_j^i = \rho v_j^i + \tau a_1 \left(x_{best_j}^i - x_j^i\right) r_1 + \tau a_2 \left(F_j - x_j^i\right) r_2. \tag{16}$$

Given that:

- $i = 1, 2, \ldots, n$, represents the capuchin's index from a population of size n,
- $j = 1, 2, \ldots, d$, represents the problem dimension,
- v_j^i is the current velocity of the ith capuchin in the jth dimension,
- x_j^i is the current position of the jth element of the ith capuchin,
- $x_{best_j}^i$ denotes the best position of the ith capuchin in the jth dimension,
- F_j is the position of the food in the jth dimension,
- a_1 and a_2 are two positive constants that control the influence of $x_{best_j}^i$ and F_j on the velocity of the capuchin which was arbitrarily selected as 1.0 in the current work,
- r_1 and r_2 are two random numbers generated uniformly in the interval [0, 1],
- the inertia coefficient ρ controls the influence of the previous velocity on the movement that takes a value of 0.7 in this work.

Jumping on the ground Alpha capuchins may leap on the ground from one place to another, from a side of a river to the other side or may use normal walking to search for food sources. This behavior is used by capuchins to roam large distances, particularly at a time when food is scarce on trees. In this case, the new position of the leader and accompanying capuchins can be identified as follows:

$$x_j^i = F_j + \frac{P_{ef} P_{bf}(v_j^i)^2 \sin(2\theta)}{g} \quad i < n/2;\quad 0.2 < \epsilon \leq 0.30 \tag{17}$$

where P_{ef} represents the elasticity probability of the capuchin movement on the ground and θ is defined in

Eq. 14. On the other side, the new position of alpha capuchins when they use normal walking can be calculated as follows:

$$x_j^i = x_j^i + v_j^i \quad i < n/2 \;\; 0.3 < \epsilon \leq 0.50 \tag{18}$$

As can be deduced from the above explanation, there are two fundamental parameters to involve in the locomotion mechanisms that capuchins use to help them move to food sources.

1. The first parameter is the balance factor that the capuchin tail provides. This parameter is valuable to sustain capuchin movement during global and local searches for food sources. Also, it is used to obviate stagnation as well as to control and balance the movement of capuchins.
2. The second parameter is the elasticity coefficient, which supports the alpha male and alpha female to perform a global search on the ground. This control parameter assists the leaders in increasing the leaping distance on the ground and over riverbanks. This parameter is analogous to the spring elasticity coefficient of Hooke's law.

Both of the balance factor and elasticity coefficient, or referred to as P_{bf} and P_{ef}, respectively, provide efficient exploitation and exploration, as these parameters enhance the efficiency of both the local and global search methods. The values of these parameters are 0.7 and 9, which are obtained after intensive analysis.

Swinging Some alpha capuchins and other accompanying capuchins may use local search to search for food on tree branches and all tips of tree branches by wandering over small distances. They use their tails to grasp on a tree branch and use the swing operation to find food sources on either side of the branch. In this case, the position of capuchins can be obtained as follows:

$$x_j^i = F_j + \tau P_{bf} \times \sin(2\theta) \quad i < n/2; \;\; 0.5 < \epsilon \leq 0.75 \tag{19}$$

where θ is defined in Eq. 14.

Climbing Some alpha capuchins and other accompanying capuchins may climb trees and their branches and descend trees several times during foraging in a process similar to local search. In this case, the position of the capuchins can be tracked as follows:

$$x_j^i = F_j + \tau P_{bf}(v_j^i - v_{j-1}^i) \quad i < n/2; \;\; 0.75 < \epsilon \leq 1.0 \tag{20}$$

where v_j^i is the current velocity of the ith capuchin in the jth dimension and v_{j-1}^i is the previous velocity of the ith capuchin in the jth dimension.

Random relocation of the capuchins In some situations, capuchins may randomly search in several new directions in order to find a better food source. This behavior is used by capuchins to allow them to efficiently explore the forest to obtain new areas for food sources. This random relocation of capuchins during foraging can be mathematically formulated as follows:

$$x_j^i = \tau \times \left[lb_j + \epsilon \times (ub_j - lb_j)\right] \quad i < n/2; \;\; \epsilon \leq Pr \tag{21}$$

Pr is a positive constant of 0.1 to represent the probability of random search of the capuchins, ub_j and lb_j indicate the upper and lower bounds of the search space in the jth dimension.

The random search of capuchins as formulated in Eq. 21 can enhance the global search ability of CapSA with the randomization feature and capuchins swarming behavior. It may also enhance the local search ability and avoid local optima solutions.

In sum, as given in Eqs. 13 to 20, capuchins change their current positions according to the availability of food sources, where food is the target of the search or objective that capuchins are searching for in the search space. These cases specifically occur when $r > 0.1$.

On the other hand, in Eq. 21, the capuchins in CapSA randomly change their position in the search space during foraging. This is to explore different new areas in order to find any nearby food items that represent their optimum goals. This specifically occurs when $r \leq 0.1$. In this case, the parameter τ was suggested to reinforce the balance between exploration and exploitation of the search space.

The position of the followers is updated according to the alpha capuchins' positions through the utilization of the third law of motion as defined in Eq. 22:

$$x_f = x_i + v_0 t + \frac{1}{2} a t^2 \tag{22}$$

where x_f and x_i are the final and initial displacements, t is the time instance, v_0 denotes the initial velocity and a represents the acceleration given in Eq. 23.

$$a = \frac{\Delta v}{\Delta t} = \frac{v_f - v_0}{t_1 - t_0} \tag{23}$$

where t_1 and t_0 represent the final and initial time instances and the parameter v_f stands for the final velocity, given as presented below:

$$v_f = \frac{\Delta x}{\Delta t} = \frac{x_f - x_0}{t_1 - t_0}. \tag{24}$$

Substitute Eq. 24 into Eq. 23 and consider $v_0 = 0$, then a can be expressed as given below:

$$a = \frac{x_f - x_0}{(t_1 - t_0)^2}. \tag{25}$$

As the time in optimization denotes an iteration, the difference between successive iterations, $t_i - t_{i-1}$, is equal to a value of 1. Based upon Eqs. 22 and 25, we can devise a formula that can be used to simulate the behavior of the followers when they follow the leaders as given in Eq. 26:

$$x_j^i = \frac{1}{2}\left(\acute{x}_j^i + x_j^{i-1}\right) \quad n/2 \leq i \leq n. \tag{26}$$

Given:

- x_j^i is the current position of the followers in the jth dimension,
- x_j^{i-1} identifies the previous position of the followers in the $(j-1)$th dimension and
- $\acute{x}_j^i$ is the current position of the leaders in the jth dimension.

4.3 Fitness evaluation

The fitness function of each capuchin is assessed by setting the values of decision variables (i.e., the solution vector) in a user-defined fitness function. The corresponding values are stored in a matrix, f, as given in the following form:

$$f = \begin{bmatrix} f_1([x_1^1, & x_2^1, & \ldots, & \ldots, & x_d^1]) \\ f_2([x_1^2, & x_2^2, & \ldots, & \ldots, & x_d^2]) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_n([x_1^n, & x_2^n, & \ldots, & \ldots, & x_d^n]) \end{bmatrix}. \tag{27}$$

5 Analysis of CapSA

Understandably, the proposed algorithm is inspired by the natural behavior and practices of capuchins during foraging activity in real life. The mathematical models presented for the proposed optimization algorithm are similar to the capuchin's activities when roaming on trees, ground, and riverbanks during searching for food sources. The mathematical model of CapSA that mimics the behavior of capuchins in wandering and foraging has taken advantage of random number generators to make it applicable to solve real-world optimization problems. The eventual target of a single-objective optimizer is to identify the optimal global solution, which is the food source in the proposed algorithm. Here, in the proposed swarm model, the alpha capuchin moves toward the food source (i.e., global solution), and the followers pursue the alpha capuchins iteratively throughout the allocation of optimal solutions (i.e., global solution). Typically, the problem is that the global optimal of optimization problems is unknown. In such a case, it is presumed that the best solution obtained so far is the best global solution and is supposed to be the source of food chased by capuchins. Further, when the followers update their positions by learning from the leader and self-experience in the first, second, third, fourth, and fifth cases of the proposed algorithm, the followers provide favorable feedback mechanisms for self-organization. The random relocation case of capuchins is liable for fluctuations in the search for food sources where stagnant capuchin members are redirected to other locations. The pseudo-code of CapSA can be summed up by the iterative procedural steps, as described in Algorithm 1.

Algorithm 1 Pseudo code of the capuchin search algorithm.

$\epsilon \leftarrow$ A random number generated between 0 and 1
$P \leftarrow$ A constant number equals to 0.5
Initialize the positions of the n capuchins using Equation 12.
Evaluate the fitness of each capuchin's position
Initialize the velocity of the capuchins
Randomly select some capuchins ($< n/2$) to represent the leader and accompanying capuchins, where the remaining capuchins (followers) follow the leaders.
while (termination condition is not satisfied) **do**
  Update the parameter τ using Equation 15
  for k=1 to n (n= total number of capuchins (leaders and followers)) **do**
    if ($k < n/2$) ($n/2$= the leader and the accompanying capuchins) **then**
      Update the velocity of the leaders using Equation 16
      if ($\epsilon \geq 0.1$) **then**
        if ($\epsilon \leq P$) **then**
          if ($\epsilon \leq 0.2$) **then**
            Update the position of the leaders that leap on the trees using Equation 13
          else if ($0.2 < \epsilon \leq 0.30$) **then**
            Update the position of the leaders that leap over riverbanks using Equation 17
          else
            Update the position of the leaders that walk on the ground using Equation 18
          end if
        else if ($0.5 < \epsilon \leq 0.75$) **then**
          Update the position of the leaders that swing on tree branches using Equation 19
        else if ($0.75 < \epsilon \leq 1.0$) **then**
          Update the position of the leaders that climb on trees using Equation 20
        end if
      else
        Update the position of the leaders that relocate randomly using Equation 21
      end if
    else
      Update the position of the followers using Equation 26
    end if
  end for
  Calculate the fitness value of each capuchin
end while
The location of capuchins is the final optimal solution

Algorithm 1 shows that CapSA initiates the optimization process by randomly generating a population of solutions (capuchins). After that, the created solution is rated using a fitness function, where the fitness of each capuchin is recalculated inside each iteration loop in order to identify the capuchin with the optimum solution. In CapSA, the fittest solution is referred to as the food source, F, which will be pursued by the alpha and the accompanying capuchins and pursued by other capuchins (i.e., followers). At each iteration loop, the exponential function, τ, is updated using Eq. 15, and for each dimension, the positions of the alpha leaders (best capuchins) are updated using Eqs. 13 to 21, whilst the positions of the followers are updated using Eq. 26.

The leaders are likely able to identify a better solution by exploring and exploiting the surrounding region. The leaders have the potential to go ahead to the global optimal (i.e., global solution) over the course of iterations. This capability is implemented based on the current and best positions of the capuchins in addition to the position of the food source. Therefore, alpha capuchins are constantly able to explore and exploit the space near the food source. If any of the capuchins moves out of the search area, it will be returned to the boundary based on the simulated steps proposed for CapSA. All the procedural steps of CapSA, except the initialization step, are repeatedly carried out at each iteration until the termination condition is reached. To find out how the proposed capuchin swarm model and CapSA can be relied upon to solve optimization problems, some commentaries are introduced as follows:

- CapSA stores the best solution (i.e., best capuchins) obtained so far and designated it to the variable of the food source (F), so there is no way to lose it even if the entire population deteriorates.
- CapSA updates the position of the leader of the capuchins from the food source (F) at each iteration, so the leader always explores and exploits the surrounding search space during searching for the globally optimal solution (i.e., finding the food source).
- CapSA updates the position of the followers regarding their current positions and the position of the leaders iteratively so that they move gradually toward the leaders that determine the best solution obtained so far.
- CapSA has one main control parameter, τ, to balance exploration and exploitation. This parameter is adaptively reduced over the course of iterations, which allows CapSA to explore most of the search space at the beginning of the search process and then exploit the promising areas in the search space.

The above theoretical remarks confirm the potentiality of CapSA in reliably exploring and exploiting the surrounding area of food sources. These claims make CapSA capable of solving optimization problems with unknown search spaces. Moreover, the adaptive strategy of CapSA allows it to bypass local solutions and locate an accurate estimation of the best global solution obtained during optimization. Hence, this proposed algorithm can be applied to unimodal and multimodal functions in addition to many real-world optimization problems. The aforementioned features of CapSA are likely to allow it to perform better than many state-of-the-art algorithms such as ABC [43], CS [99], KH [23], FOA [77], GWO [88]. However, this cannot be

guaranteed for all optimization problems as per the NFL theory.

As described above, the proposed algorithm solves optimization problems with given dimensions by finding the global solutions that represent the food sources by iteratively updating the positions of the leaders and followers until they find the food sources (i.e., global solutions). Therefore, the proposed algorithm has a computational complexity defined as:

$$\mathcal{O}(v(K(pd + pc))) \tag{28}$$

where

- v is the number of evaluation experiments,
- K is the maximum number of iterations,
- p is the number of solutions (i.e., number of search agents),
- d is the number of variables of the optimization problem (i.e., dimension of the problem) and
- c denotes the cost of the objective function of the optimization problem.

The number of search agents is typically several times that of the cost of the objective function, which usually depends on the complexity of the problem. The number of variables in the problem is typically less than the number of search agents, but the number of variables is approximately similar in range to the cost of the objective function. Thus, the factor d is similar in magnitude to c, and p does form a significant factor of the complexity assessment. So, the order of the complexity issue of CapSA computation can be reduced to:

$$\mathcal{O}(vKpd). \tag{29}$$

The above claims and assumptions are investigated experimentally on both standard benchmark functions and engineering design problems as shown in Sects. 6 and 8, respectively.

6 Experimental results and discussion

The overall efficacy of the proposed CapSA was demonstrated by evaluating it on 23 widely known benchmark functions and comparing its accuracy with other influential optimization algorithms. These basic benchmark functions are minimization problems and can be categorized into three main groups: unimodal [19], multimodal [63], and fixed-dimension multimodal functions [19]. The details of these functions, including dimension, range, and $f_{\min}$, are mathematically described in Appendix A, where Dim refers to the function's dimension, range indicates the search space limits for the function and $f_{\min}$ identifies the optimal reported value. Tables 19, 20, and 21 show the characteristics of the unimodal (F_1–F_7), multimodal (F_8–F_{13}), and fixed-dimension multimodal (F_{14}–F_{23}) benchmark functions, respectively.

Figures 7, 8, and 9 illustrate the search landscape of the cost functions for the unimodal, multimodal, and fixed-dimension multimodal functions, respectively. The unimodal functions in Fig. 7 have only one global optima with no local optima, which are sufficiently adequate to explore the convergence characteristic behavior and exploitation power of the proposed algorithm. The multimodal and fixed-dimension multimodal functions in Figs. 8 and 9, respectively, encounter the presence of multiple local optima in addition to more than one global optima. These functions are valuable for verifying local optimum avoidance and the exploration capability of the proposed algorithm.

6.1 Experimental setup

Extensive experiments were conducted to demonstrate the theoretical assertions of CapSA. These experiments can be characterized as follows:

- First, a set of quantitative measures is presented to clarify the degree of efficiency of the proposed CapSA and demonstrate its efficiency compared to other meta-heuristic algorithms.
- Second, qualitative results are presented to assess the adequacy of CapSA in optimizing a common set of benchmark functions.
- Third, convergence curves and average fitness values are plotted to illustrate the efficiency of CapSA in optimizing a selected set of benchmark functions.

To manifest the general efficiency of the proposed CapSA in each experimental study, its results are compared with a collection of seven popular nature-inspired meta-heuristic algorithms on the benchmark test functions presented in Appendix A. These meta-heuristic algorithms are PSO [49], moth-flame optimization (MFO) [63], multi-verse optimizer (MVO) [68], sine cosine algorithm (SCA) [65], gravitational search algorithm (GSA) [84], genetic algorithm (GA) [13], and harmony search (HS) [30]. The parameter setting of CapSA and other comparative meta-heuristic algorithms are listed in Table 2. The parameter setting presented in Table 2 is set according to the reported literature. In the proposed CapSA, we used the same initialization technique as the other comparative algorithms to provide a fair comparison between CapSA and those meta-heuristic algorithms. The number of search agents and the maximum number of iterations for each algorithm is set to 30 and 1000, respectively, as reported in the literature.

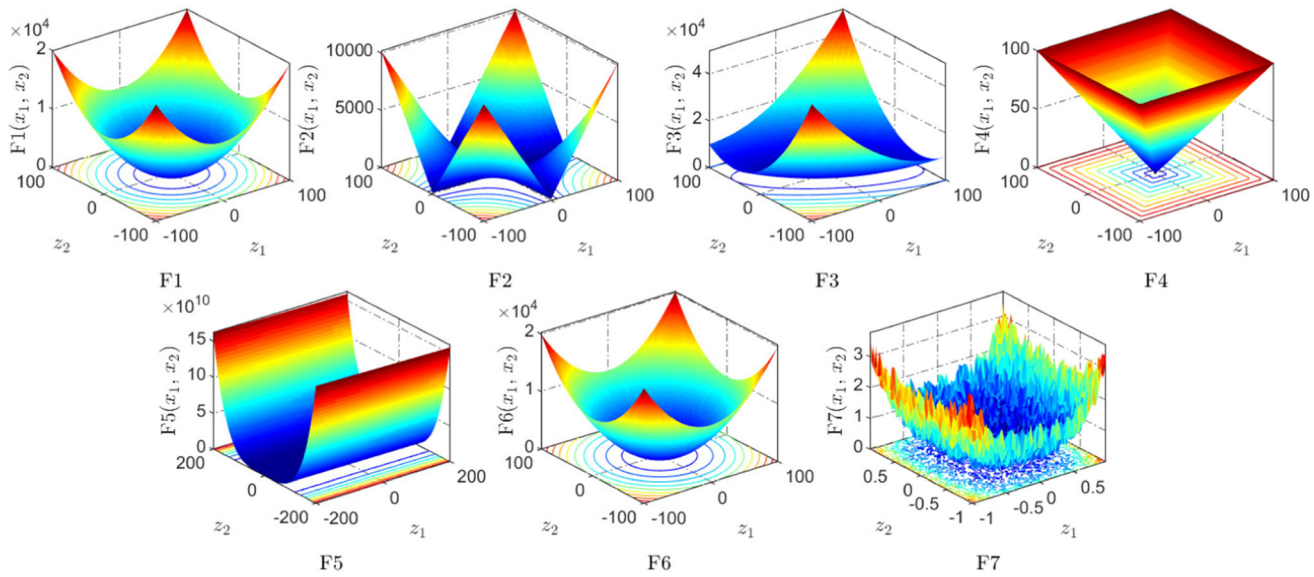


Fig. 7 Parameter space of the 2-D versions of unimodal benchmark functions

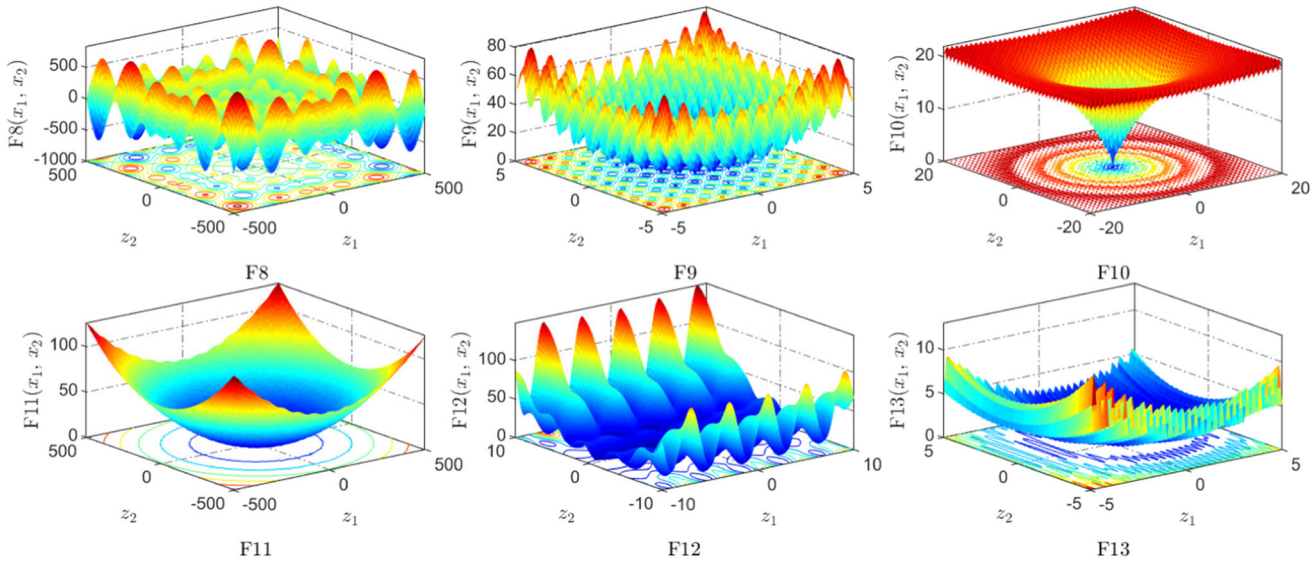


Fig. 8 Parameter space of the 2-D versions of multimodal benchmark functions

6.2 Performance comparison

The performance level of CapSA was compared with other meta-heuristic methods in terms of the average (mean) and standard deviation (STD) of the objective function values on 23 benchmark test functions. The mean and STD of the obtained solutions are computed at the last iteration to present a compelling assessment of the performance of the proposed algorithm. The stop evaluation criterion is set to the maximum number of predefined iterations. The mean and STD results of the function evaluation values were generated for each benchmark function, where each algorithm was executed for 30 independent runs where each run consisted of 1000 iterations. The analysis of STD results is to assure the stable performance of the proposed CapSA during the independent runs, as well as the decrease in the minimum solution. The mean and STD results for the other comparative algorithms were taken from references [19, 52, 63, 88]. The best results are shown in bold throughout the paper.

6.2.1 Evaluation of functions F_1–F_7 (exploitation)

The proposed algorithm was tested on functions F_1–F_7 to evaluate its exploitation efficiency. The statistical results obtained by CapSA and other comparative algorithms on

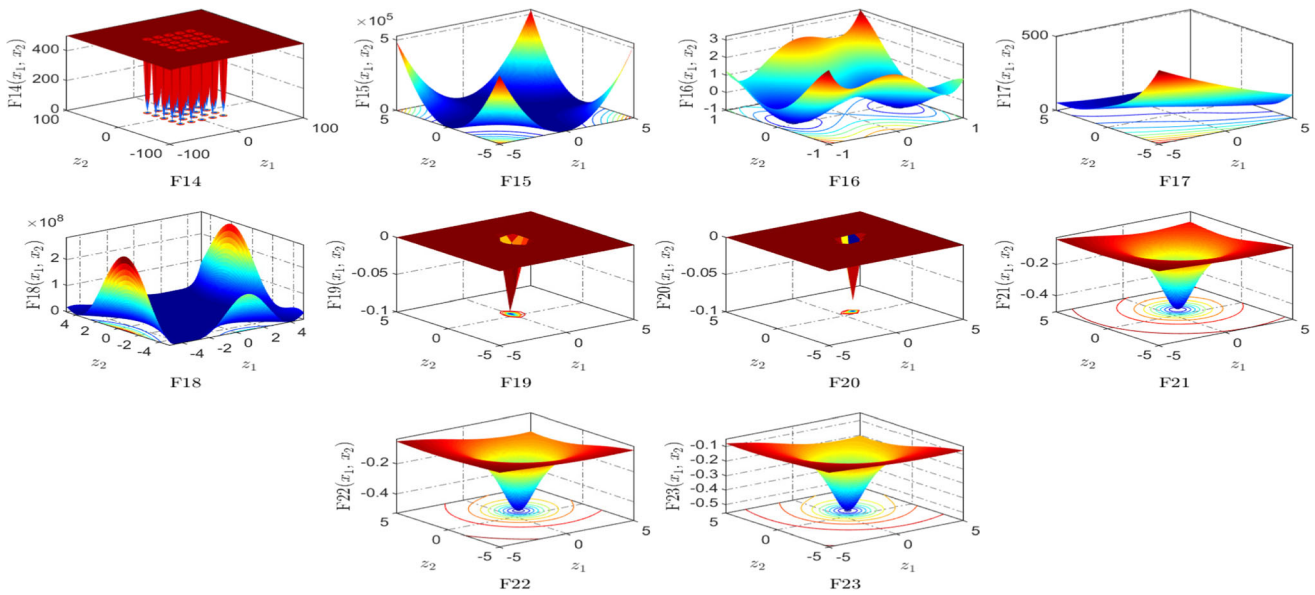


Fig. 9 Parameter space of the 2-D versions of fixed-dimension multimodal benchmark functions

the unimodal functions F_1–F_7, over 30 independent runs, are presented in Table 3.

Table 3 shows that CapSA is able to deliver competitive results compared to other well-known cogent meta-heuristic algorithms. In particular, CapSA was the most proficient algorithm for functions F_1, F_2–F_5, and F_7, where it found the best global optimal solution in comparison to other meta-heuristic algorithms. In sum, the results of CapSA on unimodal functions show that CapSA has a reliable exploitation capability and outstrips other comparative algorithms in the majority of test functions and that the STD results reveal that this superiority is stable.

6.2.2 Evaluation of functions F_8–F_{13} (exploration)

The proposed algorithm was tested on the functions F_8–F_{13} to evaluate its exploration adequateness. Table 4 shows the mean and STD results of the multimodal test functions over 30 independent runs.

As per the results reported in Table 4, CapSA is the most effective method in three test functions (i.e., F_9, F_{11}, and F_{12}), and also competitive to other meta-heuristic algorithms in the other test functions (i.e., F_8, F_{10}, and F_{13}). These findings affirm that CapSA has good worth in terms of exploration capability.

6.2.3 Evaluation of functions F_{14}–F_{23} (exploration)

The objective of this experimental test is to verify the exploration skillfulness of CapSA because these functions have fixed-dimension multimodal properties. The statistical results of CapSA and other meta-heuristic algorithms on the fixed-dimension multimodal functions F_{14} to F_{23}, over 30 independent runs of each algorithm, are presented in Table 5.

It is evident from the results displayed in Table 5 that CapSA is superior on F_{14}–F_{20} and F_{23}. Also, its degree of performance is comparable to other competitive algorithms on F_{21} and F_{22}. Further, CapSA is characterized to have the smallest STD value in most functions in comparison to other optimization algorithms. In sum, the average and STD results of the best solutions obtained during the 30 independent runs testify that the proposed CapSA exhibits a superior and steady performance level on average.

6.3 Qualitative analysis results of CapSA

Qualitative results are mostly derived from various visualization tools. The most widespread qualitative results of single-objective optimization in the literature are convergence curves. Researchers generally explore the best solutions obtained so far at each iteration loop and draw them as a line to be eligible to observe how well an algorithm heightens the approximation of the global optimum over a number of predetermined iterations. Figures 10 and 11 illustrate the qualitative convergence results of CapSA through solving the functions, F_1–F_4, F_6 and F_9–F_{11}, respectively.

The convergence curves, in Figs. 10 and 11, are drawn up to observe and confirm that CapSA performs exploration and exploitation effectively while solving standard benchmark functions. Remarkably, the graphical results of the convergence analysis disclose that CapSA offers satisfactory convergence rates and has a promising

Table 2 Parameter setting of CapSA and other meta-heuristic algorithms

Algorithm	Parameter	Value
CapSA	Velocity control constants	1
	Inertia parameter	0.7
	Balance and elasticity factors	0.7, 9
	Number of generations	1000
	Search agents	30
PSO [49]	Inertia coefficient	0.75
	Social coefficient	1.8
	Cognitive coefficient	2
	Number of generations	1000
	Search agents	30
MFO [63]	Logarithmic spiral	0.75
	Convergence constant	[− 1, − 2]
	Number of generations	1000
	Search agents	30
MVO [68]	Traveling distance rate	[0.6, 1]
	Wormhole existence prob.	[0.2, 1]
	Number of generations	1000
	Search agents	30
SCA [65]	Number of elites	2
	Number of generations	1000
	Search agents	30
GSA [84]	Alpha coefficient	20
	Gravitational constant	100
	Number of generations	1000
	Crossover and mutation	0.9, 0.05
GA [13]	Population size	30
	Number of generations	1000
	Harmony memory and rate	30, 0.95
HS [30]	Discrete set and fret width	17,700, 1
	Neighboring value rate	0.30
	Number of generations	1000

convergence behavior in all benchmark test functions. In other words, CapSA exhibits a rapid convergence in all test problems. This is credited to the robust global search mechanism in the initial stage of CapSA, as well as the local search on the best-saved positions in the search space. However, these convergence curves face abrupt changes at the initial steps of the optimization process. Then, they gradually converge after the initial steps to exploit the optimal global solutions or near-global optimal solutions to achieve outstanding behavior in the final iteration steps. This attests that capuchins move suddenly at the outset and resort to fluctuate increasingly proportional to the number of iterations. As per this observation, CapSA first requires that capuchins go around the search space and cause exploration of the search space. CapSA exploits the search space by inspiring the capuchins to a global optimal and stimulating them to move locally instead of globally. In more specific detail, CapSA exhibits three distinct convergence behaviors when optimizing various benchmark test functions.

- In the initial iteration steps, CapSA converges more swiftly to the propitious areas of the search space due to its adaptive mechanism. This behavior is evident in functions F_2, F_3, F_9, F_{10}, and F_{11}.
- In the second behavior, CapSA converges progressively toward the optimum values only in final iterations, which is apparent in function F_3.
- The last behavior of CapSA is the explicit convergence from the main phase of iterations as discerned in F_4 and F_6.

These outcomes show that CapSA preserves a balance of exploration and exploitation to locate the global optimum solution. Overall, the convergence analysis results expose a different trait of CapSA and it seems that the success rate of CapSA in solving optimization functions is highly reasonable. Further, CapSA offers a fast convergence response for F_6 and F_9, has a sensible convergence response for F_3 and F_{10}, and finds optimal solutions with fabulous convergence response for F_1 and F_2. In particular, it is perceived from Figs. 10 and 11 that capuchins first explored the search space, and most of the capuchins moved toward the optimal food source in the first 100 iterations. Thus, CapSA reported a promising performance even for complex multimodal functions.

6.4 Time complexity

This subsection provides an approximation of the time complexity of the algorithms in Tables 3, 4, and 5. The time complexity of the basic steps of these algorithms while solving the unimodal, multimodal, and fixed-dimension multimodal functions in Tables 19, 20, and 21, respectively, can be described as follows:

1. Generating the initial population of each algorithm requires a complexity of $\mathcal{O}(pd)$.
2. The next step in each algorithm is to evaluate the initial fitness value for each search agent, where this step requires an order of complexity of $\mathcal{O}(pc)$.
3. The evolutionary process of each algorithm requires a computational complexity of $\mathcal{O}(Kpd)$. In this process, the fitness function is assessed for each search agent at each iteration loop, which requires a complexity of $\mathcal{O}(Kpc)$.
4. The boundary limits of each test function are checked at each iteration to ensure that the search agents fall

Table 3 Computed results for the unimodal functions of the proposed algorithm and other meta-heuristic algorithms

Functions	CapSA		PSO		MFO		MVO	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
F_1	**1.11E−16**	**1.34E−14**	4.98E−09	1.40E−08	3.15E−04	5.99E−04	2.81E−01	1.11E−01
F_2	1.06E−09	1.28E−10	7.29E−04	1.84E−03	3.71E+01	2.16E+01	3.96E−01	1.41E−01
F_3	**1.08E−12**	**2.97 E−11**	1.40E+01	7.13	4.42E+03	3.71E+03	4.31E+01	8.97
F_4	**1.91E−08**	**2.38E−08**	6.00E−01	1.72E−01	6.70E+01	1.06E+01	8.80E−01	2.50E−01
F_5	**1.7574E−9**	**7.45E−11**	4.93E+01	3.89E+01	3.50E+03	3.98E+03	1.18E+02	1.43E+02
F_6	1.007E−13	7.435E−11	9.23E−09	1.78E−08	1.66E−04	2.01E−04	3.15E−01	9.98E−02
F_7	**2.22E−08**	**1.17E−06**	6.92E−02	2.87E−02	3.22E−01	2.93E−01	2.02E−02	7.43E−−03
Functions	SCA		GSA		GA		HS	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
F_1	3.55E−02	1.06E−01	1.16E−16	6.10E−17	1.95E−12	2.01E−11	7.86E−10	8.11E−09
F_2	3.23E−05	8.57E−05	1.70E−01	9.29E−01	6.53E−18	5.10E−17	**5.99E−20**	**1.11E−17**
F_3	4.91E+03	3.89E+03	4.16E+02	1.56E+02	7.70E−10	7.36E−09	9.19E−05	6.16E−04
F_4	1.87E+01	8.21	1.12	9.89E−01	9.17E+01	5.67E+01	8.73E−01	1.19E−01
F_5	7.37E+02	1.98E+03	3.85E+01	3.47E+01	5.57E+02	4.16E+01	8.91E+02	2.97E+02
F_6	4.88	9.75E−01	**1.08E−16**	**4.00E−17**	3.15E−01	9.98E−02	8.18E−17	1.70E−18
F_7	3.88E−02	5.79E−02	7.68E−01	2.77	6.79E−04	3.29E−03	5.37E−01	1.89E−01

Table 4 Computed results for the multimodal functions of the proposed algorithm and other meta-heuristic algorithms

Functions	CapSA		PSO		MFO		MVO	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
F_8	−1218.07	762	−6010	1300	−8040	880	−6920	919
F_9	**1.12E−13**	**1.53E−14**	4.72E+01	1.03E+01	1.63E+02	3.74E+01	1.01E+02	1.89E+01
F_{10}	1.18E−07	1.64E−08	3.86E−02	2.11E−01	1.60E+01	6.18	1.15	7.87E−01
F_{11}	**8.33E−13**	**9.67E−14**	5.50E−03	7.39E−03	5.03E−02	1.74E−01	5.74E−01	1.12E−01
F_{12}	**4.11E−12**	**5.42E−11**	1.05E−10	2.06E−10	1.26	1.83	1.27	1.02
F_{13}	1.51E−12	5.47E−10	4.03E−03	5.39E−03	7.24E−01	1.48	6.60E−02	4.33E−02
Functions	SCA		GSA		GA		HS	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
F_8	−3810	283	−2750	572	−5110	437	**−469**	**394**
F_9	2.23E+01	3.25E+01	3.35E+01	1.19E+01	1.23E−01	4.11E+01	4.85E−02	3.91E+01
F_{10}	1.55E+01	8.11	8.25E−09	1.90E−09	**5.31E−11**	**1.11E−10**	2.83E−08	4.34E−07
F_{11}	3.01E−01	2.89E−01	8.19	3.70	3.31E−06	4.23E−05	2.49E−05	1.34E−04
F_{12}	5.21E+01	2.47E+02	2.65E−01	3.14E−01	9.16E−08	4.88E−07	1.34E−05	6.23E−04
F_{13}	2.81E+02	8.63E+02	**5.73E−32**	**8.95E−32**	6.39E−02	4.49E−02	9.94E−08	2.61E−07

within the lower and upper bounds of the function. These limits are essential to verify and adjust the solutions that go beyond the boundary of the search area. This step has a computational complexity of $\mathcal{O}(Kpd)$.

Table 5 Computed results for the fixed-dimension multimodal functions of the proposed algorithm and other meta-heuristic algorithms

Functions	CapSA		PSO		MFO		MVO	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
F_{14}	**0.998**	**1.43E−2**	2.77	2.32	2.21	1.80	0.998	9.14E−1
F_{15}	**3.062E−4**	**8.46E−09**	9.09E−04	2.38E−04	1.58E−03	3.50E−03	7.15E−03	1.26E−02
F_{16}	**− 1.0316**	**5.57E−12**	− 1.03	0.00	− 1.03	0.00	− 1.03	4.74E−08
F_{17}	**3.977E−01**	**1.92E−09**	3.97E−01	9.03E−16	3.98E−01	1.13E−16	3.98E−01	1.15E−07
F_{18}	**3.00**	**3.27E−08**	3.00	6.59E−05	3.00	4.25E−15	5.70	1.48E+01
F_{19}	**− 0.76768**	**1.35E−10**	3.90	3.37E−15	− 3.86	3.16E−15	− 3.86	3.53E−07
F_{20}	**− 0.51523**	**5.54E−02**	− 3.32	2.66E−01	− 3.23	6.65E−02	− 3.23	5.37E−02
F_{21}	− 3.2513	2.58E−01	− 7.54	2.77	− 6.20	3.52	− 7.38	2.91
F_{22}	− 4.2084	1.77E−01	− 8.55	3.08	− 7.95	3.20	− 8.50	3.02
F_{23}	**− 2.3144**	**4.28E−01**	− 9.19	2.52	− 7.50	3.68	− 8.41	3.13
Functions	SCA		GSA		GA		HS	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
F_{14}	1.26	6.86E−01	3.61	2.96	4.39	4.41E−02	6.79	1.12
F_{15}	1.01E−03	3.75E−04	6.84E−03	7.37E−03	7.36E−03	2.39E−04	5.15E−03	3.45E−04
F_{16}	− 1.03	3.23E−05	− 1.03	0.00	− 1.04	4.19E−07	− 1.03	3.64E−08
F_{17}	3.99E−01	7.61E−04	3.98E−01	1.13E−16	3.98E−01	3.71E−17	3.99E−01	9.45E−15
F_{18}	3.00	2.25E−05	3.01	3.24E−02	3.01	6.33E−07	3.00	1.94E−10
F_{19}	− 3.86	2.55E−03	− 3.22	4.15E−01	− 3.30	4.37E−10	− 3.29	9.69E−04
F_{20}	− 2.84	3.71E−01	− 1.47	5.32E−01	− 2.39	4.37E−01	− 2.17	1.64E−01
F_{21}	**− 2.28**	**1.80**	− 4.57	1.30	− 5.19	2.34	− 7.33	1.29
F_{22}	− 3.99	1.99	− 6.58	2.64	− 2.97	1.37E−02	**− 1.00**	**2.89E−04**
F_{23}	− 4.49	1.96	− 9.37	2.75	− 3.10	2.37	− 2.46	1.19

5. Steps 3 and 4 are iterated a number of v experiments until a maximum number of iterations is reached. This results in a complexity issue of:

$$\mathcal{O}(v(pd + pc + K(pd + pc + pd))) \tag{30}$$

Since:

$$\begin{aligned} &\mathcal{O}(vK(pd + pc + pd)) \gg \mathcal{O}(vpd) \\ &\mathcal{O}(vK(pd + pc + pd)) \gg \mathcal{O}(vpc) \end{aligned} \tag{31}$$

Accordingly, the complexity of the algorithms can be reduced to:

$$\mathcal{O}(v(K(pd + pc + pd))) \tag{32}$$

The factor d is nearly similar in magnitude to the factor c, where K and p are essential factors for assessing the complexity. Thus, the complexity issue in Eq. 32 can be reduced to:

$$\mathcal{O}(vKpd) \tag{33}$$

In fact, the computational complexities of the algorithms in Tables 3, 4, and 5 are approximately close together. However, the computational time per iteration level for each algorithm depends on the machine and program specifications used to perform the test.

In fact, in all of these algorithms, the search for the optimal global solution is stopped when the maximum number of iterations is reached. Hence, the difference between the computational loads of these algorithms is of low importance. However, some algorithms may converge more rapidly than others. In this case, the difference is significant in terms of finding the optimum solution at a small number of iterations.

Comparing the computational complexity of these algorithms in Eq. 33 with the computational complexity of the proposed algorithm in Eq. 29 affirms that the proposed algorithm has a computational complexity within the range of the computational burden of the other algorithms. This asserts that the proposed algorithm is a computationally efficient method of optimization and that the difference in computational load with these algorithms is attributed to the specifications of the machine and the compiler in

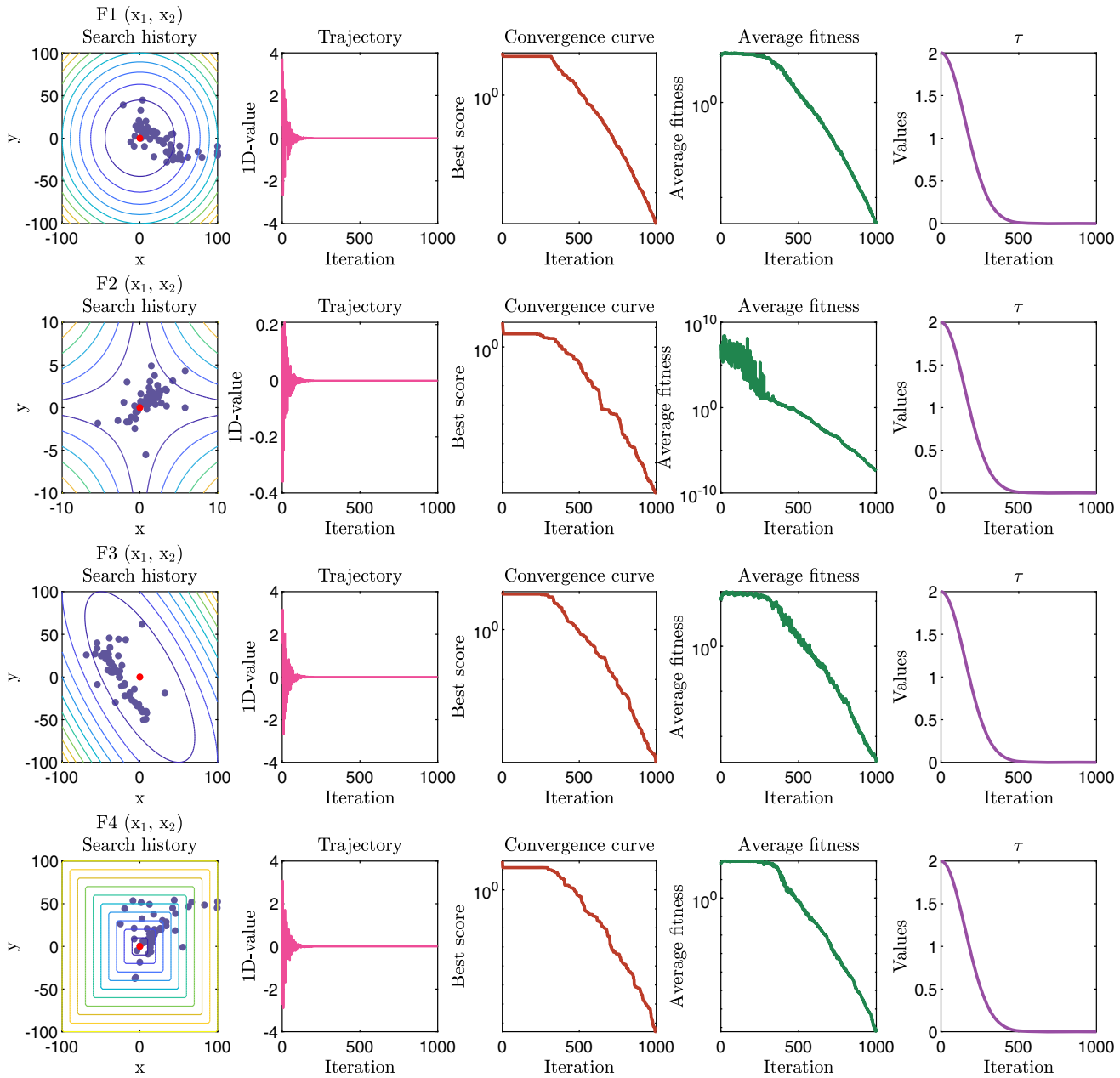


Fig. 10 Qualitative results of CapSA for F_1–F_4: search history, trajectory in the first dimension of the first capuchin, convergence curve, average objective function of all capuchins and the parameter τ

addition to any specific control parameters of the algorithms.

In a nutshell, each algorithm requires the identification of four parameters:

1. Number of experiments,
2. Number of iterations,
3. Number of search agents, and
4. The dimension of the problem.

These parameters have an impact on the speed of the algorithms in the evolutionary process while searching for the optimal solution. The search time to find the global optimum solution increases as the values of these parameters increase, where the stopping criterion of these algorithms is the maximum number of iterations. The number of iterations and search agents needed to satisfactorily find the best solution depends on the complexity of the test function. The values of these parameters should be specified craftily to solve the problem reliably. Too few numbers of iterations can result in a low level of performance. Too many numbers of iterations and search agents will increase the computational cost. Therefore, the values of

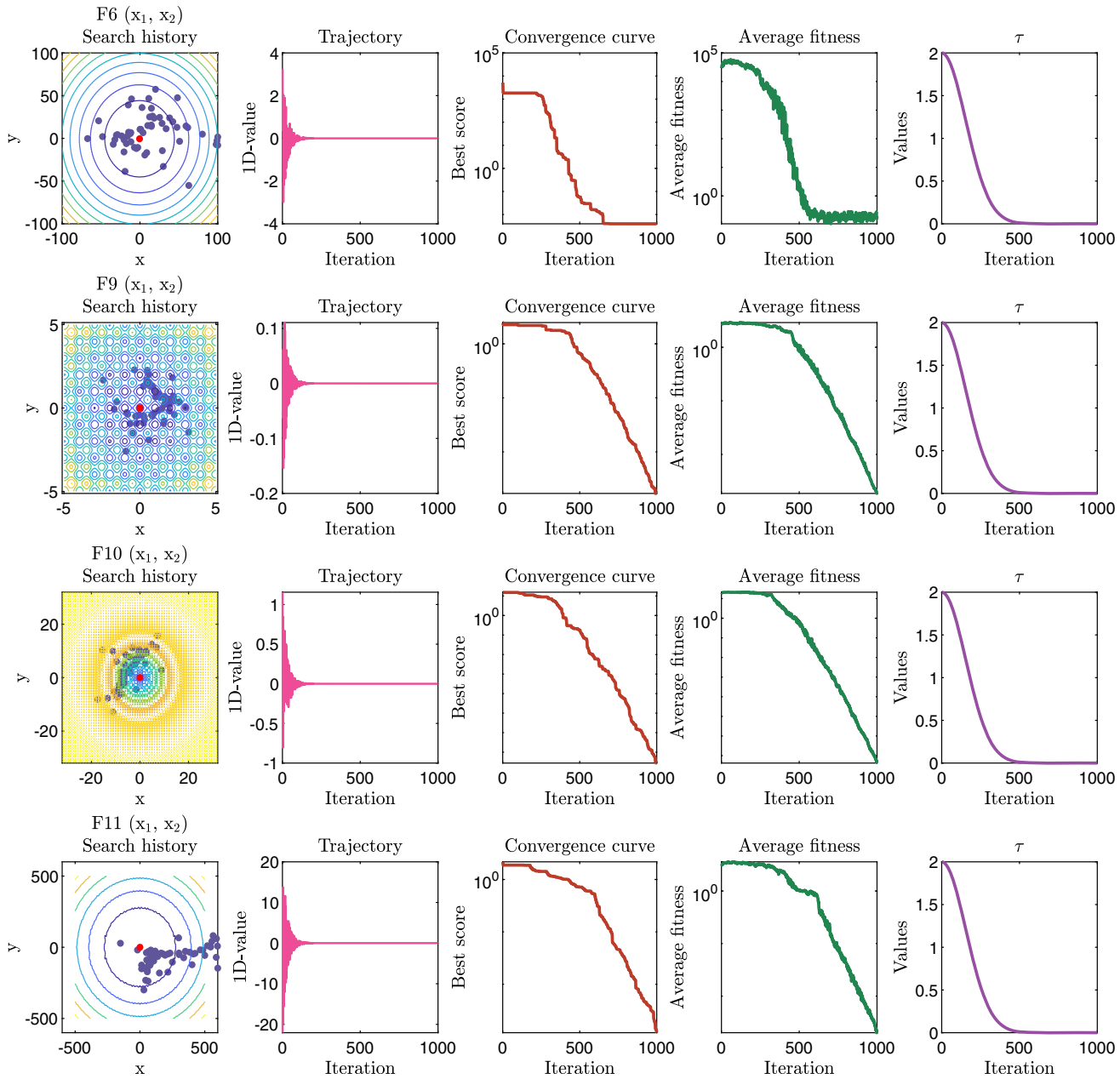


Fig. 11 Qualitative results of CapSA for F_6, F_9, F_{10}, and F_{11}: search history, trajectory in the first dimension of the first capuchin, convergence curve, average objective function of all capuchins and the parameter τ

these parameters should be chosen carefully to obtain a high degree of performance.

7 Statistical test results

The fundamental statistical analysis of the results such as average and STD imparts a general conception about the performance of CapSA. These statistical results have shown the reliable exploration and exploitation potentialities of CapSA, but are unable to show how good this proposed algorithm is. So, there is a need to conduct statistical tests to corroborate that the results are not created by chance. These tests are imperative to analyze the consistency and performance of CapSA for the mean results obtained in all of the 30 independent runs. In order to compare each of the runs and confirm the importance of the results, statistical test methods, such as Friedman's and Holm's tests [78], are performed to determine whether the results gained from the presented algorithm in this paper deviate from the results of other promising meta-heuristic algorithms in a statistically significant way.

Friedman's test is a common nonparametric statistical test method used to rate the performance level of algorithms. Friedman's test purports to pinpoint if there is a substantial difference between the outcomes of different algorithms. This statistical test is based on a null hypothesis that there is no discrepancy in the performance of all the comparative algorithms. The best acting algorithm obtains the lowest rank, while the worst acting algorithm gets the highest rank. In regard to Friedman's test, if the p value revealed by this statistical test is equal or less than the degree of significance, here it is 0.05, a null hypothesis is rejected, suggesting that there are considerable differences between the performance of the evaluated algorithms. This test is then followed by a post hoc analysis method to explore the pairwise comparison of the algorithms using Holm's test. The lowest-ranked algorithm by Friedman's statistical test is generally employed as a control method for post hoc analysis. A summary of the statistical test results of Friedman's test on unimodal, multimodal, and fixed-dimension multimodal test functions is displayed in Table 6 on the basis of the results tabulated in Tables 3, 4, and 5, respectively.

As per the statistical results in Table 6, CapSA is statistically significant and the best performing algorithm among all other competitors' algorithms. Considering the 5% significance level, CapSA scored the lowest rank at 1.57 in unimodal test functions, 2.66 in multimodal test functions, and 2.05 in fixed-dimension multimodal functions. HS was the second-best performing algorithm on unimodal test functions that outperformed CapSA in some cases. GA, which was one of the best algorithms in the literature, was a senior competitor to CapSA in multimodal test functions, and its performance was better than CapSA in only two functions, worse than CapSA in 16 functions and provided nonsignificant results in only five functions. SCA, which was one of the best meta-heuristic algorithms in the literature, was a major competitor to CapSA in fixed-dimension multimodal functions and performed better than CapSA in 2 functions, worse than CapSA in 16 functions and presented reasonable results in 5 functions. In summary, it is clear that the proposed CapSA is ranked first in the unimodal test functions, followed successively by HS, PSO, GA, GSA, MVO, SCA, and MFO in the last rank. In regard to the multimodal test functions, the order produced by Friedman's test is CapSA, GA, HS, PSO, GSA, both of MFO and MVO in the same rank and SCA in the last rank. Finally, the average ranking in optimizing fixed-dimension multimodal functions is as follows: CapSA, PSO, MFO, MVO, SCA, GSA, GA, and finally, HS. CapSA has proven to be a robustly competitive optimization algorithm and can be utilized to address challenges and very complex optimization issues.

Table 6 A summary of statistical results using Friedman's test on unimodal, multimodal, and fixed-dimension multimodal benchmark functions

Algorithm	Unimodal Rank	Multimodal Rank	Fixed-dimension Rank
CapSA	**1.57**	**2.66**	**2.05**
PSO	4.00	4.00	2.95
MFO	6.71	5.83	3.74
MVO	5.35	5.83	3.90
SCA	6.14	6.33	4.79
GSA	4.42	4.5	5.35
GA	4.07	3.00	5.50
HS	3.71	3.83	5.89

The p values computed by Friedman's test based on the statistical findings of unimodal, multimodal, and fixed-dimension multimodal functions are presented in Table 7. The null hypothesis of equivalent accuracy is rejected to confirm that there are statistically significant differences between the accuracy of the evaluated algorithms.

Holm's procedure is applied as a post-test method to identify whether there are statistically significant differences between the control algorithm that has the lowest rank and the other evaluated algorithms. The statistical results obtained using Holm's method based on unimodal, multimodal, and fixed-dimension multimodal benchmark functions are illustrated in Tables 8, 9, and 10, respectively.

It was found that Holm's procedure findings in Table 8 rejects those hypotheses that have a p value ≤ 0.0125, where the results of the p values show that the superiority of CapSA is statistically significant.

It was found that Holm's procedure in Table 9 rejects those hypotheses that have a p value ≤ 0.007142. The outcomes of Holm's method in Table 9 reveal that CapSA is statistically better than SCA, MVO, MFO, GSA, PSO, HS, and GA.

It is clearly observed from Table 10 that Holm's procedure rejects those hypotheses that have a p value ≤ 0.0125. As can be seen from the results in Table 10, CapSA is proficient at delivering very competitive results in these case studies, such as those of other convincing meta-heuristic algorithms reported in the literature. The p values also support the degree of performance of CapSA on fixed-dimension multimodal test functions and illustrate how important this algorithm is. These outcomes show that CapSA has succeeded in evading local optimum solutions by adequately balancing exploration and exploitation capabilities.

In short, the results of Friedman's and Holm's tests in Tables 8, 9, and 10 illustrate the satisfactory performance, robustness and reliability of the proposed algorithm. It has

Table 7 Friedman's test results obtained on the basis of the statistical results of the unimodal, multimodal, and fixed-dimension multimodal functions

Benchmark functions	Statistical value	p value	Hypothesis
Unimodal	20.96428	0.003823	Rejected
Multimodal	13.22222	0.066875	Rejected
Fixed-dimension multimodal	16.26666	0.022788	Rejected

Table 8 Results of Holm's method based on the statistical results of the unimodal benchmark functions

i	Method	z	p value	$\alpha \div i$	Hypothesis
7	MFO	3.927922	8.568298E−5	0.00714	Rejected
6	SCA	3.491486	4.803411E−4	0.008333	Rejected
5	MVO	2.891387	0.003835	0.01	Rejected
4	GSA	2.182178	0.029096	0.0125	Rejected
3	GA	1.909406	0.056209	0.01666	Not rejected
2	PSO	1.854852	0.063617	0.025	Not rejected
1	HS	1.636634	0.101706	0.05	Not rejected

Table 9 Results of Holm's method based on the statistical results of the multimodal benchmark functions

i	Method	z	p value	$\alpha \div i$	Hypothesis
7	SCA	22.592724	0.009521	0.00714	Rejected
6	MVO	2.239174	0.025144	0.00833	Rejected
5	MFO	2.239171	0.025144	0.01	Rejected
4	GSA	1.296362	0.194850	0.0125	Not rejected
3	PSO	0.942809	0.345778	0.01666	Not rejected
2	HS	0.824957	0.409395	0.025	Not rejected
1	GA	0.235702	0.813663	0.05	Not rejected

outperformed other algorithms in most of the unimodal test functions with outstanding results in multimodal and fixed-dimension multimodal test functions. These findings exhibit that CapSA is adept at solving very challenging problems. It is worth mentioning that the exploitation capability of CapSA tends to be better than its exploration capability. This can be deduced from the results of this algorithm on unimodal functions compared to the results obtained on multimodal test functions. This is due to the swarm-based nature of this algorithm, where there are small sharp changes in the solutions. Despite this fact, the results showed that this is not a big concern because the exploratory behavior of CapSA is also sensible due to the updating mechanism used to explore new locations in the search space. Little exploration does not guarantee locating a global optimum solution, where there is a necessity to reach an adequate balance between exploration and exploitation.

Table 10 Results of Holm's method based on the statistical results of the fixed-dimension multimodal benchmark functions

i	Method	z	p value	$\alpha \div i$	Hypothesis
7	HS	3.331978	8.623079E−4	0.00714	Rejected
6	GA	2.966830	0.003008	0.00833	Rejected
5	GSA	2.829899	0.004656	0.01	Rejected
4	SCA	2.327820 3	0.019921 1	0.0125	Rejected
3	MVO	2.099603	0.035763	0.01666	Rejected
2	MFO	1.506237	0.132006	0.025	Not rejected
1	PSO	1.369306	0.170903	0.05	Not rejected

8 CapSA for classical engineering problems

The adeptness of CapSA in addressing real-world problems, particularly nonlinear constrained optimization problems, is substantiated by assessing it on classical engineering design problems and comparing the results with other compelling meta-heuristic methods. In the following subsections, we used CapSA to solve four well-studied constrained engineering design problems: the welded beam design problem, the pressure vessel design problem, the tension/compression spring design problem, and the speed reducer design problem. These design problems have a variety of constraints, so to optimize these problems, there is a demand to use a constraint handling method. In this paper, CapSA is fitted with a death penalty function to deal with the constraints of these classical engineering design problems. This is imputed to the naive and low computational burden cost of this constraint

handling method. This handling method does not use the information of infeasible solutions, which are beneficial to solve the design problems with dominated infeasible areas. The maximum number of iterations and search agents for CapSA in these optimization design problems below are set to 1000 and 200, respectively. The parameter setting of CapSA in solving the engineering design problems below is the same as those given in Table 2, except the inertia parameter of the velocity, is set to 1.0 in solving the pressure vessel design problem.

8.1 Welded beam design

The welded beam design problem, a classical benchmark study, seeks to lessen the fabrication cost of the welded beam structure shown in Fig. 12 [93].

The welded beam structure presented in Fig. 12 consists of a beam, A and the welding desired to be attached to the part, B. This design problem is subject to four related optimization constraints defined as follows:

- Buckling load on the bar (P_c).
- Bending stress in the beam (θ).
- Shear stress (τ).
- End deflection of the beam (δ).

To optimize this design problem, there is a need to locate the feasible set of four structural optimization parameters of the welded beam design structure: the thickness of the weld (h), length of the clamped bar (l), height of the bar (t), and the thickness of the bar (b). These design parameters are measured in inches. The design variable vector can be written as: $\vec{x} = [x_1, x_2, x_3, x_4]$, where x_1, x_2, x_3 and x_4 represent h, l, t, and b, respectively. The mathematical formula of the fitness function of the welded beam design that needs to be minimized is defined as follows:

$$f(\vec{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$$

Subject to the following constraints,

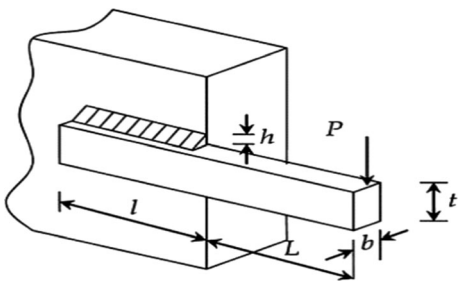


Fig. 12 A schematic structure of a welded beam design

$$g_1(\vec{x}) = \tau(\vec{x}) - \tau_{\max} \le 0$$
$$g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{\max} \le 0$$
$$g_3(\vec{x}) = x_1 - x_4 \le 0$$
$$g_4(\vec{x}) = 1.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \le 0$$
$$g_5(\vec{x}) = 0.125 - x_1 \le 0$$
$$g_6(\vec{x}) = \delta(\vec{x}) - \delta_{\max} \le 0$$
$$g_7(\vec{x}) = P - P_c(\vec{x}) \le 0$$

where the other parameters of the welded beam problem are defined as follows:

$$\tau(\vec{x}) = \sqrt{((\tau')^2 + (\tau'')^2) + \frac{2\tau'\tau''x_2}{2R}}, \tau' = \frac{p}{\sqrt{2}x_1x_2}$$
$$\tau'' = \frac{MR}{J}, M = P\left(L + \frac{x_2}{2}\right), R = \sqrt{\left(\frac{x_1 + x_3}{2}\right)^2 + \frac{x_2^2}{4}}$$
$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + (\frac{x_1 + x_3}{2})^2\right]\right\}, \sigma(\vec{x}) = \frac{6PL}{x_4x_3^2}$$
$$\delta(\vec{x}) = \frac{4PL^3}{Ex_4x_3^3}, P_c(\vec{x}) = \frac{4.013\sqrt{EGx_3^2x_4^6/36}}{L^2}\left(1 - \frac{x^3}{2L}\sqrt{\frac{E}{4G}}\right)$$

where $P = 6000\,\text{lb}, L = 14$ in., $\delta_{\max} = 0.25$ in., $E = 30 * 10^6$ psi, $G = 12 * 10^6$ psi, $\delta_{\max} = 13{,}600psi$, $\sigma_{\max} = 30{,}000$ psi. The variable ranges of this design problem are taken as: $0.1 \le x_i \le 2.0$ when $i = 1$ and 4 and $0.1 \le x_i \le 10.0$ when $i = 2$ and 3.

This engineering optimization problem was addressed by several algorithms such as MFO [63], MVO [68], SCA [65], GA [13], evolutionary strategy (ES) [60], simulated annealing (SA) [37], co-evolutionary PSO [34], GSA [84], improved PSO [14], differential evolution (DE) algorithm [61], CS algorithm [25], ABC [44], and ACO [48]. The comparative results of the best solution obtained by CapSA and other algorithms for the welded beam design problem are presented in Table 11.

It is evident from the results in Table 11 that the proposed CapSA produced an optimal design with the lowest cost of 1.72481904 and outperformed all other algorithms. A summary of the statistical results after 30 independent runs in terms of the best score, worst score, mean score, and standard deviation score obtained by CapSA and other algorithms for the welded beam problem is shown in Table 12.

The results of the welded design problem in Table 12 show that CapSA again performs better on average, worst, standard deviation, as well as achieving the best optimal design. This verifies the degree of reliability of CapSA in addressing this optimization problem.

Table 11 A comparison of the optimization results obtained by CapSA and other optimization algorithms for the welded beam design problem

Algorithm	Optimal values for variables				Optimum cost
	h	*l*	*t*	*b*	
CapSA	0.205723	3.470789	9.036622	0.205737	**1.72481904**
MFO [63]	0.203567	3.443025	9.230278	0.212359	1.732541
MVO [68]	0.205611	3.472103	9.040931	0.205709	1.725472
SCA [65]	0.204695	3.536291	9.004290	0.210025	1.759173
GA [13]	0.164171	4.032541	10.00000	0.223647	1.873971
ES [60]	0.199742	3.612060	9.037500	0.20682	1.73730
SA [37]	0.20564426	3.472578742	9.03662391	0.2057296	1.7250022
Co-evolutionary PSO [34]	0.20573	3.47049	9.03662	0.20573	1.72485084
GSA [84]	0.18219	3.856979	10.0000	0.202376	1.879952
Improved PSO [14]	0.205729	3.470488	9.036624	0.205729	1.724852
DE [61]	0.20573	3.470489	9.0336624	0.205730	1.724852
CS [25]	0.2015	3.562	9.0414	0.2057	1.73121
ABC [44]	0.205730	3.470489	9.036624	0.205730	1.724852
ACO [48]	0.205700	3.471131	9.036683	0.205731	1.724918

Table 12 Statistical results of different optimization algorithms for solving the welded beam design problem

Algorithm	Best score	Worst score	Mean score	STD
CapSA	**1.72481904**	**1.72723071**	**1.72541110**	**4.23761E−7**
MFO [63]	1.732541	1.802364	1.775231	0.012397
MVO [68]	1.725472	1.741651	1.729680	0.004866
SCA [65]	1.759173	1.873408	1.817657	0.027543
GA [13]	1.873971	2.320125	2.119240	0.034820
ES [60]	1.728226	1.993408	1.792654	0.07471
SA [37]	1.7250022	1.8843960	1.7564428	NA
Co-evolutionary PSO [34]	1.728024	1.782143	1.748831	0.012926
GSA [84]	2.172858	3.003657	2.544239	0.255859
Improved PSO [14]	1.724852	NA	2.0574	0.2154
DE [61]	1.724852	NA	1.725	1E−15
CS [25]	1.7312065	2.3455793	1.8786560	0.2677989
ABC [44]	1.724852	NA	1.741913	0.031
ACO [48]	1.72918	1.775961	1.729752	0.009200

8.2 Pressure vessel design problem

The pressure vessel design is one of the broadly used structural design benchmark problems [42]. The objective of this optimization problem is to lessen the overall cost of materials, formation, and welding of the cylindrical vessel, which are capped at both ends by hemispherical heads, as depicted in Fig. 13. The design variables of this problem are characterized as follows:

- Thickness of the shell (T_s)
- Thickness of the head (T_h)
- Inner radius (R)
- Length of the cylindrical section of the vessel without looking at the head (L)

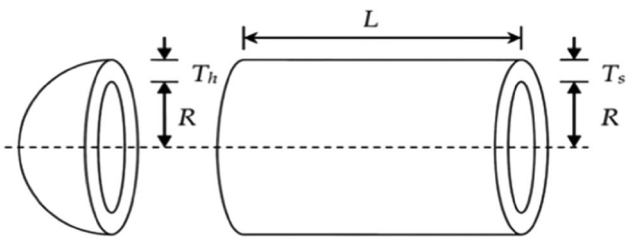


Fig. 13 A schematic structure of the cross section of a pressure vessel design

The variable vector of the pressure vessel design was formulated as: $\vec{x} = [x_1, x_2, x_3, x_4]$, where x_1, x_2, x_3 and x_4 represent T_s, T_h, R and L, respectively. These design variables are measured in inches. The mathematical

formula of the pressure vessel design problem can be formulated a minimization problem of the function:

$$f(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3.$$

This problem is subject to four constraints as given below,

$$g_1(\vec{x}) = -x_1 + 0.0193x_3 \leq 0$$
$$g_2(\vec{x}) = -x_2 + 0.00954x_3 \leq 0$$
$$g_3(\vec{x}) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1{,}296{,}000 \leq 0$$
$$g4(\vec{x}) = x_4 - 240 \leq 0$$

where $0 \leq x1 \leq 99$, $0 \leq x_2 \leq 99$, $10 \leq x_3 \leq 200$, and $10 \leq x_4 \leq 200$.

This design problem was addressed by many researchers using different algorithms such as MFO [63], MVO [68], SCA [65], GA [13], HS [53], DA [64], Co-evolutionary PSO [34], ES [60], CS algorithm [25], ABC [3], improved PSO [35], penalty guided ABC [28], and DE [61]. Table 13 presents a comparison of the best solutions obtained so far by CapSA and other algorithms previously reported in the literature for the pressure vessel design problem.

According to Table 13, CapSA is able to find an optimal design for the pressure vessel problem at the lowest cost of about 5885.3328. Table 14 shows the statistical results in regard to the best, worst, mean and STD scores for the pressure vessel design problem after 30 independent runs obtained using CapSA and other meta-heuristic algorithms.

The statistical results in Table 14 show that the performance of the proposed CapSA in solving the pressure vessel design problem is superior and considerably better than the other evaluated algorithms. The standard deviation of the results obtained by CapSA after 30 independent runs is 1.023511, which is much lower than the other evaluated algorithms. This indicates that the proposed algorithm is beneficial and reliable in solving this optimization problem.

8.3 Tension–compression spring design problem

The third classical engineering problem is the design of the tension/compression spring given in Fig. 14 [7].

The objective of this optimization design problem is to minimize weight. This problem is subject to a range of certain constraints, including surge frequency, shear stress, and minimum deflection. The parameters in this design optimization problem are defined as follows: wire diameter (d), mean coil diameter (D), and the number of active coils (N). The variable vector of this optimization problem could be written as: $\vec{x} = [x_1, x_2, x_3]$, where x_1, x_2 and x_3 represent d, D, and N, respectively. The mathematical formula of this optimization problem is given as follows:

$$\text{Minimize: } f(\vec{x}) = (x_3 + 2)x_2x_1^2$$

This problem is subject to the following constraints:

$$g_1(\vec{x}) = 1 - \frac{x_2^3 x_3}{71785x_1^4} \leq 0$$
$$g_2(\vec{x}) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \leq 0$$
$$g_3(\vec{x}) = 1 - \frac{140.45x_1}{x_2^2x_3} \leq 0$$
$$g_4(\vec{x}) = \frac{x_1 + x_2}{1.5} - 1 \leq 0$$

where $0.05 \leq x_1 \leq 2.0$, $0.25 \leq x_2 \leq 1.3$, and $2 \leq x_3 \leq 15.0$.

Table 13 A comparison of the optimization results obtained by CapSA and other optimization algorithms for the pressure vessel design problem

Algorithm	Optimal values for variables				Optimum cost
	T_s	T_h	R	L	
CapSA	12.450710	6.154413	40.319636	199.999795	**5885.3328**
MFO [63]	0.835241	0.409854	43.578621	152.21520	6055.6378
MVO [68]	0.845719	0.418564	43.816270	156.38164	6011.5148
SCA [65]	0.817577	0.417932	41.74939	183.57270	6137.3724
GA [13]	0.752362	0.399540	40.452514	198.00268	5890.3279
HS [53]	1.099523	0.906579	44.456397	179.65887	6550.0230
DA [64]	0.782825	0.384649	40.3196	200	5923.11
Co-evolutionary PSO [34]	0.812500	0.437500	42.091266	176.746500	6061.077
ES [60]	0.812500	0.437500	42.098087	176.640518	6059.7456
CS [25]	0.812500	0.437500	42.0984456	176.6363595	6059.7143348
ABC [3]	0.812500	0.437500	42.098446	176.636596	6059.714339
Improved PSO [35]	0.812500	0.437500	42.098445	176.6365950	6059.7143
Penalty guided ABC [28]	0.7781686	0.3846491	40.3210545	199.9802367	5885.40322828
DE [61]	0.812500	0.437500	42.098446	176.6360470	6059.701660

Table 14 Statistical results of different optimization algorithms for solving the pressure vessel design problem

Algorithm	Best score	Worst score	Mean score	STD
CapSA	**5885.3328**	**5888.753255**	**5885.532117**	**1.023511**
MFO [63]	6055.6378	7023.8521	6360.6854	365.597
MVO [68]	6011.5148	7250.9170	6477.3050	327.007
SCA [65]	6137.3724	6512.3541	6326.7606	126.609
GA [13]	5890.3279	7005.7500	6264.0053	496.128
HS [53]	6550.0230	8005.4397	6643.9870	657.523
DA [64]	5923.11	222536	21342.2	47044.2
Co-evolutionary PSO [34]	6061.077	6363.8041	6147.1332	86.4545
ES [60]	6059.7456	7332.8798	6 850.004	9426.000
CS [25]	6059.714	6495.3470	6447.7360	502.693
ABC [3]	6059.714339	NA	6245.308144	205
Improved PSO [35]	6059.7143	NA	6289.92881	305.78
Penalty guided ABC [28]	5885.403282	5895.126804	5887.557024	2.745290

Fig. 14 A schematic diagram of a tension/compression spring design

This optimization problem was addressed in the literature by several algorithms such as spotted hyena optimizer (SHO) [19], GWO [64], Co-evolutionary PSO [34], MFO [63], MVO [68], SCA [65], GSA [84], GA [13], improved HS [56], salp swarm algorithm (SSA) [66], ES [60], and DE [39]. A comparison of the best optimization results arrived at by CapSA and those algorithms for the tension/compression spring design problem is presented in Table 15. To conduct an equitable comparison between the proposed algorithm and those algorithms, a penalty function was used in a manner similar to that described in [96].

The results in Table 15 emphasize that CapSA has identified the optimal design for this problem with a minimum cost of 0.12665, which is slightly better than other optimization methods. Table 16 shows the statistical results for the tension/compression spring design problem after 30 independent runs obtained using CapSA and other algorithms.

The results in Table 16 present that the performance of CapSA is outstanding and considerably better than other algorithms. The standard deviation of the results reached by CapSA for this design problem is 0.000001345, which is substantially lower than other algorithms. This substantiates that the proposed algorithm is highly effective in solving this design problem.

Table 15 A comparison of the optimization results obtained by CapSA and other optimization algorithms for the tension/compression spring design problem

Algorithm	Optimum variables			Optimum weight
	d	*D*	*N*	
CapSA	0.05177	0.3588	11.167	**0.1266640**
SHO [19]	0.051144	0.343751	12.0955	0.012674000
GWO [64]	0.05169	0.356737	11.28885	0.0126665
Co-evolutionary PSO [34]	0.051728	0.357644	11.244543	0.0126747
MFO [63]	0.05000	0.313501	14.03279	0.012753902
MVO [68]	0.05000	0.315956	14.22623	0.012816930
SCA [65]	0.050780	0.334779	12.72269	0.012709667
GSA [84]	0.050276	0.323680	13.525410	0.0127022
GA [13]	0.05010	0.310111	14.0000	0.013036251
Improved HS [56]	0.05025	0.316351	15.23960	0.012776352
SSA [66]	0.051207	0.345215	12.004032	0.0126763
ES [60]	0.051989	0.363965	10.890522	0.0126810
DE [39]	0.051609	0.354714	11.410831	0.0126702

Table 16 Statistical results of different optimization algorithms for solving the tension/compression spring design problem

Algorithm	Best score	Worst score	Mean score	STD
CapSA	**0.1266640**	**0.012689025**	**0.126667**	**0.000001345**
SHO [19]	0.012674000	0.012715185	0.012684106	0.000027
GWO [64]	0.012678321	0.012720757	0.012697116	0.000041
Co-evolutionary PSO [34]	0.013192580	0.01786250	0.0148171817	0.002272
MFO [63]	0.012753902	0.017236590	0.014023657	0.001390
MVO [68]	0.012816930	0.017839737	0.014464372	0.001622
SCA [65]	0.012709667	0.01299844	0.0128396378	0.000078
GSA [84]	0.012873881	0.014211731	0.013438871	0.000287
GA [13]	0.013036251	0.016251423	0.014036254	0.002073
Improved HS [56]	0.012776352	0.015214230	0.013069872	0.000375

8.4 Speed reducer design problem

The design of the speed reducer problem with the structural design shown in Fig. 15 is difficult, because this design is correlated with seven design variables [24].

The weight to be minimized in this problem is subject to four constraints [19], outlined as follows:

- Bending stress of the gear teeth.
- Surface stress.
- Transverse deflections of the shafts.
- Stresses in the shafts.

The seven design variables of this problem are set as: b, m, z, l_1, l_2, d_1, and d_2. These variables are defined, respectively, as the face width, module of teeth, number of teeth in the pinion, length of the first shaft between bearings, length of the second shaft between bearings, the diameter of the first shafts, and the diameter of second shafts. These variables were represented during solving this problem by a vector as $\vec{x} = [x_1 x_2 x_3 x_4 x_5 x_6 x_7]$. The mathematical formula of the speed reducer problem is formulated as follows:

$$f(\vec{x}) = 0.7854 x_1 x_2^2 (3.3333 x_3^2 + 14.9334 x_3 - 43.0934) - 1.508 x_1 (x_6^2 + x_7^2) + 7.4777 (x_6^3 + x_7^3) + 0.7854 (x_4 x_6^2 + x_5 x_7^2)$$

Subject to the following constraints,

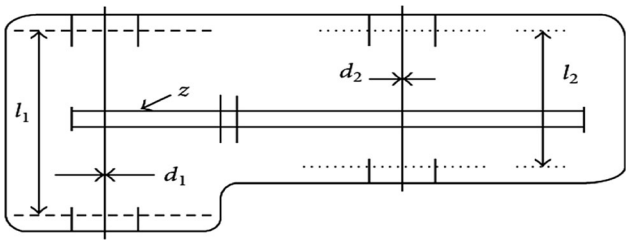


Fig. 15 A structural structure of a speed reducer design

$$g_1(\vec{x}) = \frac{27}{x_1 x_2^2 x_3} - 1 \le 0$$

$$g_2(\vec{x}) = \frac{397.5}{x_1 x_2^2 x_3^2} - 1 \le 0$$

$$g_3(\vec{x}) = \frac{1.9 x_4^3}{x_2 x_6^4 x_3} - 1 \le 0$$

$$g_4(\vec{x}) = \frac{1.93 x_5^3}{x_2 x_7^4 x_3} - 1 \le 0$$

$$g_5(\vec{x}) = \frac{\left[(745(x_4/x_2 x_3))^2 + 16.9 \times 10^6\right]^{1/2}}{110 x_6^3} - 1 \le 0$$

$$g_6(\vec{x}) = \frac{\left[(745(x_5/x_2 x_3))^2 + 157.5 \times 10^6\right]^{1/2}}{85 x_7^3} - 1 \le 0$$

$$g_7(\vec{x}) = \frac{x_2 x_3}{40} - 1 \le 0$$

$$g_8(\vec{x}) = \frac{5 x_2}{x_1} - 1 \le 0$$

$$g_9(\vec{x}) = \frac{x_1}{12 x_2} - 1 \le 0$$

$$g_{10}(\vec{x}) = \frac{1.5 x_6 + 1.9}{x_4} - 1 \le 0$$

$$g_{11}\vec{x}) = \frac{1.1 x_7 + 1.9}{x_5} - 1 \le 0$$

where the range of the design parameters b, m, z, l_1, l_2, d_1 and d_2 were given as $2.6 \le x_1 \le 3.6$, $0.7 \le x_2 \le 0.8$, $17 \le x_3 \le 28$, $7.3 \le x_4 \le 8.3$, $7.3 \le x_5 \le 8.3$, $2.9 \le x_6 \le 3.9$, and $5.0 \le x_4 \le 5.5$, respectively. This design problem was addressed in the literature by several algorithms such as SHO [19], GWO [64], MFO [63], MVO [68], SCA [65], GSA [84], GA [13], and improved HS [56]. A comparison of the best solutions obtained by CapSA and other optimization algorithms for the speed reducer design problem is presented in Table 17.

According to the obtained optimum costs in Table 17, CapSA is capable of finding an optimal design for the speed reducer design problem with a minimum cost of approximately 2994.4710. Table 18 shows the statistical

Table 17 A comparison of the optimization results obtained by CapSA and other optimization algorithms for the speed reducer design problem

Algorithm	Optimum variables							Optimum cost
	b	m	z	l_1	l_2	d_1	d_2	
CapSA	3.500	0.7	17.	7.30	7.715320	3.350215	5.286654	**2994.4710**
SHO [19]	3.50159	0.7	17	7.3	7.8	3.35127	5.28874	2998.5507
GWO [64]	3.506690	0.7	17	7.380933	7.815726	3.357847	5.286768	3001.288
MFO [63]	3.507524	0.7	17	7.302397	7.802364	3.323541	5.287524	3009.571
MVO [68]	3.508502	0.7	17	7.392843	7.816034	3.358073	5.286777	3002.928
SCA [65]	3.508755	0.7	17	7.3	7.8	3.461020	5.289213	3030.563
GSA [84]	3.600	0.7	17	8.3	7.8	3.369658	5.289224	3051.120
GA [13]	3.510253	0.7	17	8.35	7.8	3.362201	5.287723	3067.561
Improved HS [56]	3.520124	0.7	17	8.37	7.8	3.366970	5.288719	3029.002

Table 18 Statistical results of different optimization algorithms for solving the speed reducer design problem

Algorithm	Best score	Worst score	Mean score	STD
CapSA	**2994.47106**	**2998.09236**	**2995.12109**	**2.90121E−4**
SHO [19]	2998.5507	3003.889	2999.640	1.93193
GWO [64]	3001.288	3008.752	3005.845	5.83794
MFO [63]	3009.571	3054.524	3021.256	11.0235
MVO [68]	3002.928	3060.958	3028.841	13.0186
SCA [65]	3030.563	3104.779	3065.917	18.0742
GSA [84]	3051.120	3363.873	3170.334	92.5726
GA	3067.561	3313.199	3186.523	17.1186
Improved HS [56]	3029.002	3619.465	3295.329	57.0235

results in regard to the best, worst, mean, and STD scores for the speed reducer design problem after 30 independent runs obtained using CapSA and other meta-heuristic algorithms.

According to the comparison of the statistical results in Table 18, CSA has found the best optimal solution.

9 Conclusion and future work

This paper proposed a novel nature-inspired capuchin meta-heuristic search algorithm to solve real-world engineering optimization problems. The foraging behavior of capuchin monkeys presents critical inspiration for the proposed algorithm. This behavior is studied and modeled mathematically, including each feature of the search for food sources, to obtain the coveted optimization. The best feasible solution of CapSA, possessed so far, is deemed the primary food source to be chased by capuchins. The proposed algorithm uses an adaptive mechanism that represents a life scheme convergence strategy to achieve a balance between exploration and exploitation by starting the search at an early stage with a relatively sizeable exponential value, and this value decreases during the course of iterations. To explore the potential performance of the proposed algorithm in practice, it was tested on 23 test functions and four real-world engineering problems. We also provided a comparison between the proposed algorithm and a variety of widely known and familiar meta-heuristic algorithms. According to the simulation results, findings, analysis, and statistical tests, it can be perceived and may be asserted that CapSA has many merits over other compelling optimization algorithms and worth applying to address other benchmark and real-world problems with complex search spaces. The proposed optimization algorithm provides a basic framework for relatively low-dimension optimization issues, which may be expanded to solve large-scale and constrained optimization issues. Further research is to be undertaken to develop a binary version of CapSA, which could be a worthwhile contribution to solving real-world applications. Further, it may be possible to expand the applications of CapSA to single-objective problems in diverse areas as well as multi-objective problems.

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interest regarding the publication of this paper.

Appendix A. Objective test problems used in this work

Unimodal test functions

A description of the unimodal test functions (F_1–F_7) is shown in Table 19.

Table 19 Unimodal benchmark functions

Function	Dim	Range	Shift position	$f_{\min}$
$F_1(x) = \sum_{i=1}^{n} x_i^2$	30	[− 100, 100]	[− 30, − 30, …, − 30]	0
$F_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	[− 10, 10]	[− 3, − 3, …, − 3]	0
$F_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	30	[− 100, 100]	[− 30, − 30, …, − 30]	0
$F_4(x) = \max_i \{\lvert x_i \rvert,\ 1 \le i \le n\}$	30	[− 100, 100]	[− 30, − 30, …, − 30]	0
$F_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	[− 30, 30]	[− 15, − 15, …, − 15]	0
$F_6(x) = \sum_{i=1}^{n} (\lvert x_i + 0.5 \rvert)^2$	30	[− 100, 100]	[− 750, − 750, …, − 750]	0
$F_7(x) = \sum_{i=1}^{n} i x_i^4 + \text{random}[0, 1)$	30	[− 1.28, 1.28]	[− 0.25, …, − 0.25]	0

Multimodal test functions

A description of the multimodal test functions (F8-F_{13}) is shown in Table 20.

Table 20 Multimodal benchmark functions

Function	Dim	Range	Shift position	$f_{\min}$
$F_8(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	30	[− 500, 500]	$[-300, \ldots, -300]$	-418.9829×5
$F_9(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	30	[− 5.12, 5.12]	[− 2, − 2, …, − 2]	0
$F_{10}(x) = -20exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) -$ $exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$	30	[− 32, 32]		0
$F_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{(i)}}) + 1$	30	[− 600, 600]	[− 400,…, − 400]	0
$F_{12}(x) = \frac{\pi}{n}\{10\sin(\pi y_i) + \sum_{i=1}^{n-1}(y_i - 1)^2[1+ 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{x_i + 1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	30	[− 50, 50]	$[-30, -30, \ldots, -30]$	0
$F_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{n1}(x_i - 1)^2[1 + \sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	[− 50, 50]	$[-100, \ldots, -100]$	0

Fixed-dimension multimodal test functions

A description of the fixed-dimension multimodal test functions (F_{14}–F_{23}) is shown in Table 21.

Table 21 Fixed-dimension multimodal benchmark functions

Function	Dim	Range	Shift position	$f_{\min}$
$F_{14}(x) = \left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}\right)^{-1}$	2	[− 65, 65]	[− 2, − 2, …, − 2]	1
$F_{15}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4	[− 5, 5]	[− 2, − 2, …, − 2]	0.00030
$F_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	[− 5, 5]	[− 2, − 2, …, − 2]	− 1.0316
$F_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right) + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	2	[− 5, 5]	[− 2, − 2, …, − 2]	0.398
$F_{18}(x) = \left[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \times \left[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right]$	2	[− 2, 2]	[− 2, − 2, …, − 2]	3
$F_{19}(x) = -\sum_{i=1}^{4} c_i exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$	3	[1, 3]	$[-2, -2, \ldots, -2]$	− 3.86
$F_{20}(x) = -\sum_{i=1}^{4} c_i exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$	6	[0, 1]	[− 2, − 2, …, − 2]	− 3.32
$F_{21}(x) = -\sum_{i=1}^{5} \left[(X - a_i)(X - a_i)^T + c_i\right]^{-1}$	4	[0, 10]	[− 2, − 2, …, − 2]	− 10.1532
$F_{22}(x) = -\sum_{i=1}^{7} \left[(X - a_i)(X - a_i)^T + c_i\right]^{-1}$	4	[0, 10]	[− 2, − 2, …, − 2]	− 10.4028
$F_{23}(x) = -\sum_{i=1}^{1} \left[(X - a_i)(X - a_i)^T + c_i\right]^{-1}$	4	[0, 10]	[− 2, − 2, …, − 2]	− 10.536

References

1. Abedinia O, Amjady N, Ghasemi A (2016) A new metaheuristic algorithm based on shark smell optimization. Complexity 21(5):97–116
2. Acevedo J, Pistikopoulos EN (1997) A multiparametric programming approach for linear process engineering problems under uncertainty. Ind Eng Chem Res 36(3):717–728
3. Akay B, Karaboga D (2012) Artificial bee colony algorithm for large-scale problems and engineering design optimization. J Intell Manuf 23(4):1001–1014
4. Aktemur C, Gusseinov I (2017) A comparison of sequential quadratic programming, genetic algorithm, simulated annealing, particle swarm optimization and hybrid algorithm for the design and optimization of golinski's speed reducer. Int J Energy Appl Technol 4(2):34–52
5. Alfaro JWL, Silva JDSE, Rylands AB (2012) How different are robust and gracile capuchin monkeys? An argument for the use of sapajus and cebus. Am J Primatol 74(4):273–286
6. Arnay R, Fumero F, Sigut J (2017) Ant colony optimization-based method for optic cup segmentation in retinal images. Appl Soft Comput 52:409–417
7. Arora JS (2004) Optimum design concepts: optimality conditions. In: Introduction to optimum design
8. Askarzadeh A (2016) A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput Struct 169:1–12
9. Atashpaz-Gargari E, Lucas C (2007) Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: IEEE congress on evolutionary computation. IEEE, pp 4661–4667
10. Basturk B (2006) An artificial bee colony (ABC) algorithm for numeric function optimization. In: IEEE swarm intelligence symposium, Indianapolis
11. Baykasoğlu A, Akpinar Ş (2015) Weighted superposition attraction (wsa): a swarm intelligence algorithm for optimization problems-part 2: constrained optimization. Appl Soft Comput 37:396–415
12. Beni G, Wang J (1993) Swarm intelligence in cellular robotic systems. In: Dario P, Sandini G, Aebischer P (eds) Robots and

biological systems: towards a new bionics?. Springer, Berlin, pp 703–712
13. Bonabeau E, de Recherches D, Marco DF, Dorigo M, Theraulaz G et al (1999) Swarm intelligence: from natural to artificial systems, vol 1. Oxford University Press, Oxford
14. Cagnina LC, Esquivel SC, Coello CAC (2008) Solving engineering optimization problems with the simple constrained particle swarm optimizer. Informatica 32:3
15. Chen MK, Lakshminarayanan V, Santos LR (2006) How basic are behavioral biases? Evidence from capuchin monkey trading behavior. J Polit Econ 114(3):517–537
16. Chumburidze M, Basheleishvili I, Khetsuriani A (2019) Dynamic programming and greedy algorithm strategy for solving several classes of graph optimization problems. Broad Res Artif Intell Neurosci 10(1):101–107
17. Colorni A, Dorigo M, Maniezzo V et al (1992) Distributed optimization by ant colonies. In: Proceedings of the first European conference on artificial life, Cambridge, vol 142, pp 134–142
18. Devi SG, Sabrigiriraj M (2019) A hybrid multi-objective firefly and simulated annealing based algorithm for big data classification. Concurr Comput Pract Exp 31(14):e4985
19. Dhiman G, Kumar V (2017) Spotted hyena optimizer: a novel bio-inspired based metaheuristic technique for engineering applications. Adv Eng Softw 114:48–70
20. Dorigo M, Birattari M (2010) Ant colony optimization. Springer, Berlin
21. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Micro machine and human science. In: Proceedings of the sixth international symposium on MHS'95. IEEE, pp 39–43
22. Fausto F, Cuevas E, Valdivia A, González A (2017) A global optimization algorithm inspired in the behavior of selfish herds. Biosystems 160:39–55
23. Gandomi AH, Alavi AH (2012) Krill herd: a new bio-inspired optimization algorithm. Commun Nonlinear Sci Numer Simul 17(12):4831–4845
24. Gandomi AH, Yang XS (2011) Benchmark problems in structural optimization. In: Computational optimization, methods and algorithms, vol 356. Springer, Berlin, Heidelberg, pp 259–281
25. Gandomi AH, Yang X-S, Alavi AH (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 29(1):17–35
26. García-Hernández L, Lorenzo Salas-Morera C, Carmona-Muñoz JAG-H, Salcedo-Sanz S (2020) A novel island model based on coral reefs optimization algorithm for solving the unequal area facility layout problem. Eng Appl Artif Intell 89:103445
27. García-Hernández L, Lorenzo Salas-Morera JA, Garcia-Hernandez SS-S, de Oliveira JV (2019) Applying the coral reefs optimization algorithm for solving unequal area facility layout problems. Expert Syst Appl 138:112819
28. Garg H (2014) Solving structural engineering design optimization problems using an artificial bee colony algorithm. J Ind Manag Optim 10(3):777–794
29. Garg H (2016) A hybrid pso-ga algorithm for constrained optimization problems. Appl Math Comput 274:292–305
30. Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: harmony search. Simulation 76(2):60–68
31. Gilli M, Maringer D, Schumann E (2019) Numerical methods and optimization in finance. Academic Press, Cambridge
32. Grossmann IE, Apap RM, Calfa BA, Garcia-Herreros P, Zhang Q (2017) Mathematical programming techniques for optimization under uncertainty and their application in process systems engineering. Theor Found Chem Eng 51(6):893–909
33. Harjunkoski I, Grossmann IE (2002) Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. Comput Chem Eng 26(11):1533–1552
34. He Q, Wang L (2007) An effective co-evolutionary particle swarm optimization for constrained engineering design problems. Eng Appl Artif Intell 20(1):89–99
35. He S, Prempain E, Wu QH (2004) An improved particle swarm optimizer for mechanical design optimization problems. Eng Opt 36(5):585–605
36. He S, Wu QH, Saunders JR (2009) Group search optimizer: an optimization algorithm inspired by animal searching behavior. IEEE Trans Evol Comput 13(5):973–990
37. Hedar A-R, Fukushima M (2006) Derivative-free filter simulated annealing method for constrained continuous global optimization. J Global Optim 35(4):521–549
38. Holland JH et al (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press, Cambridge
39. Huang F, Wang L, He Q (2007) An effective co-evolutionary differential evolution for constrained optimization. Appl Math Comput 186(1):340–356
40. Jahani E, Chizari M (2018) Tackling global optimization problems with a novel algorithm-mouth brooding fish algorithm. Appl Soft Comput 62:987–1002
41. Jain M, Singh V, Rani A (2019) A novel nature-inspired algorithm for optimization: squirrel search algorithm. Swarm Evolut Comput 44:148–175
42. Kannan BK, Kramer SN (1994) An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. J Mech Des 116:405–411
43. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459–471
44. Karaboga D, Basturk B (2008) On the performance of artificial bee colony (abc) algorithm. Appl Soft Comput 8(1):687–697
45. Karasulu B, Korukoglu S (2011) A simulated annealing-based optimal threshold determining method in edge-based segmentation of grayscale images. Appl Soft Comput 11(2):2246–2259
46. Kaveh A, Dadras A (2017) A novel meta-heuristic optimization algorithm: thermal exchange optimization. Adv Eng Softw 110:69–84
47. Kaveh A, Farhoudi N (2013) A new optimization method: dolphin echolocation. Adv Eng Softw 59:53–70
48. Kaveh A, Talatahari S (2010) An improved ant colony optimization for constrained engineering design problems. Eng Comput 27(1):155–182
49. Kennedy J, Eberhart R (1995) Particle swarm optimization (pso). In: Proceedings of the IEEE international conference on neural networks, Perth, Australia, pp 1942–1948
50. Kirkpatrick S, Gelatt CD, P Vecchi M (1983) Optimization by simulated annealing. Science 220(4598):671–680
51. Koppen M, Wolpert DH, Macready WG (2001) Remarks on a recent paper on the "no free lunch" theorems. IEEE Trans Evol Comput 5(3):295–296
52. KS SR, Murugan S (2017) Memory based hybrid dragonfly algorithm for numerical optimization problems. Expert Syst Appl 83:63–78
53. Lee KS, Geem ZW (2005) A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. Comput Methods Appl Mech Eng 194(36–38):3902–3933
54. Li MD, Zhao H, Weng XW, Han T (2016) A novel nature-inspired algorithm for optimization: virus colony search. Adv Eng Softw 92:65–88

55. Li X (2003) A new intelligent optimization method-artificial fish school algorithm. Doctor thesis of Zhejiang University
56. Mahdavi M, Fesanghary M, Damangir E (2007) An improved harmony search algorithm for solving optimization problems. Appl Math Comput 188(2):1567–1579
57. Mao X-B, Min W, Dong J-Y, Wan S-P, Jin Z (2019) A new method for probabilistic linguistic multi-attribute group decision making: application to the selection of financial technologies. Appl Soft Comput 77:155–175
58. Martin R, Stephen W (2006) Termite: a swarm intelligent routing algorithm for mobilewireless ad-hoc networks. In: Stigmergic optimization, vol 31. Springer, Berlin, Heidelberg, pp 155–184
59. Mehta VK, Dasgupta B (2012) A constrained optimization algorithm based on the simplex search method. Eng Optim 44(5):537–550
60. Mezura-Montes E, Coello CAC (2008) An empirical study about the usefulness of evolution strategies to solve constrained optimization problems. Int J Gen Syst 37(4):443–473
61. Mezura-Montes E, Coello CA, Velázquez-Reyes J, Muñoz-Dávila L (2007) Multiple trial vectors in differential evolution for engineering design. Eng Optim 39(5):567–589
62. Mirjalili S (2015) The ant lion optimizer. Adv Eng Softw 83:80–98
63. Mirjalili S (2015) Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. Knowl Based Syst 89:228–249
64. Mirjalili S (2016) Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput Appl 27(4):1053–1073
65. Mirjalili S (2016) Sca: a sine cosine algorithm for solving optimization problems. Knowl Based Syst 96:120–133
66. Mirjalili S, Gandomi AH, Mirjalili SZ, Saremi S, Faris H, Mirjalili SM (2017) Salp swarm algorithm: a bio-inspired optimizer for engineering design problems. Adv Eng Softw 114:163–191
67. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67
68. Mirjalili S, Mirjalili SM, Hatamlou A (2016) Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Comput Appl 27(2):495–513
69. Mlinarić D, Perić T, Matejaš J (2019) Multi-objective programming methodology for solving economic diplomacy resource allocation problem. Croat Oper Res Rev 8:165–174
70. Mosavi MR, Khishe M, Naseri MJ, Parvizi GR, Mehdi AYAT (2019) Multi-layer perceptron neural network utilizing adaptive best-mass gravitational search algorithm to classify sonar dataset. Arch Acoust 44(1):137–151
71. Mucherino A, Seref O (2007) Monkey search: a novel metaheuristic search for global optimization. In: AIP conference proceedings, vol 953. AIP, pp 162–173
72. Nabil E (2016) A modified flower pollination algorithm for global optimization. Expert Syst Appl 57:192–203
73. Foroughi Nematollahi A, Rahiminejad A, Vahidi B (2017) A novel physical based meta-heuristic optimization method known as lightning attachment procedure optimization. Appl Soft Comput 59:596–621
74. Nguyen P, Kim J-M (2016) Adaptive ecg denoising using genetic algorithm-based thresholding and ensemble empirical mode decomposition. Inf Sci 373:499–511
75. Noshadi A, Shi J, Lee WS, Shi P, Kalam A (2016) Optimal pid-type fuzzy logic controller for a multi-input multi-output active magnetic bearing system. Neural Comput Appl 27(7):2031–2046
76. Ottoni EB, Izar P (2008) Capuchin monkey tool use: overview and implications. Evolut Anthropol Issues News Rev Issues News Rev 17(4):171–178
77. Pan W-T (2012) A new fruit fly optimization algorithm: taking the financial distress model as an example. Knowl Based Syst 26:69–74
78. Pereira DG, Afonso A, Medeiros FM (2015) Overview of Friedman's test and post-hoc analysis. Commun Stat Simulat Comput 44(10):2636–2653
79. Perić T, Babić Z, Matejaš J (2018) Comparative analysis of application efficiency of two iterative multi objective linear programming methods (mp method and stem method). CEJOR 26(3):565–583
80. Pulgar-Rubio F, Rivera-Rivas AJ, Pérez-Godoy MD, González P, Carmona CJ, Mefasd-bd MJDJ (2017) multi-objective evolutionary fuzzy algorithm for subgroup discovery in big data environments-a mapreduce solution. Knowl Based Syst 117:70–78
81. Qi X, Zhu Y, Zhang H (2017) A new meta-heuristic butterfly-inspired algorithm. J Comput Sci 23:226–239
82. Qi Y, Jin L, Wang Y, Xiao L, Zhang J (2019) Complex-valued discrete-time neural dynamics for perturbed time-dependent complex quadratic programming with applications. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2019.2944992
83. Rashaideh H, Sawaie A, Al-Betar MA, Abualigah LM, Al-Laham MM, Ra'ed M, Braik M (2018) A grey wolf optimizer for text document clustering. J Intell Syst 29(1):814–830
84. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) Gsa: a gravitational search algorithm. Inf Sci 179(13):2232–2248
85. Rodriguez N, Gupta A, Zabala PL, Cabrera-Guerrero G (2018) Optimization algorithms combining (meta) heuristics and mathematical programming and its application in engineering. Math Probl Eng 2018:3967457. https://doi.org/10.1155/2018/3967457
86. Sadollah A, Bahreininejad A, Eskandar H, Hamdi M (2013) Mine blast algorithm: a new population based algorithm for solving constrained engineering optimization problems. Appl Soft Comput 13(5):2592–2612
87. Saremi S, Mirjalili S, Lewis A (2017) Grasshopper optimisation algorithm: theory and application. Adv Eng Softw 105:30–47
88. Saremi S, Mirjalili SZ, Mirjalili SM (2015) Evolutionary population dynamics and grey wolf optimizer. Neural Comput Appl 26(5):1257–1263
89. Shareef H, Ibrahim AA, Mutlag AH (2015) Lightning search algorithm. Appl Soft Comput 36:315–333
90. Tabari A, Ahmad A (2017) A new optimization method: electro-search algorithm. Comput Chem Eng 103:1–11
91. Uymaz SA, Tezel G, Yel E (2015) Artificial algae algorithm (aaa) for nonlinear global optimization. Appl Soft Comput 31:153–171
92. Wang G-G (2018) Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. Memetic Comput 10(2):151–164
93. Wang G-G, Guo L, Gandomi AH, Hao G-S, Wang H (2014) Chaotic krill herd algorithm. Inf Sci 274:17–34
94. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
95. Yang XS (2009) Firefly algorithms for multimodal optimization. In: International symposium on stochastic algorithms. Springer, pp 169–178
96. Yang X-S (2010) Nature-inspired metaheuristic algorithms. Luniver Press, London
97. Yang XS (2010) A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010), vol 284. Springer, Berlin, Heidelberg, pp 65–74

98. Yang XS (2012) Flower pollination algorithm for global optimization. In: International conference on unconventional computing and natural computation. Springer, pp 240–249
99. Yang XS, Deb S (2009) Cuckoo search via lévy flights. In: World congress on nature and biologically inspired computing (NaBIC). IEEE, pp 210–214
100. Yazdani M, Jolai F (2016) Lion optimization algorithm (loa): a nature-inspired metaheuristic algorithm. J Comput Des Eng 3(1):24–36
101. Ye Y, Li J, Li K, Hui F (2018) Cross-docking truck scheduling with product unloading/loading constraints based on an improved particle swarm optimisation algorithm. Int J Prod Res 56(16):5365–5385
102. Yong W, Tao W, Cheng-Zhi Z, Hua-Juan H (2016) A new stochastic optimization approach-dolphin swarm optimization algorithm. Int J Comput Intell Appl 15(02):1650011
103. Zaidan AA, Bayda Atiya MR, Bakar A, Zaidan BB (2019) A new hybrid algorithm of simulated annealing and simplex downhill for solving multiple-objective aggregate production planning on fuzzy environment. Neural Comput Appl 31(6):1823–1834
104. Zhalechian M, Tavakkoli-Moghaddam R, Rahimi Y, Jolai F (2017) An interactive possibilistic programming approach for a multi-objective hub location problem: Economic and environmental design. Appl Soft Comput 52:699–713