A Hybrid CS–GSA Algorithm for Optimization

Manoj Kumar Naik, Leena Samantaray and Rutuparna Panda

Abstract The chapter presents a hybridized population-based Cuckoo search–Gravitational search algorithm (CS–GSA) for optimization. The central idea of this chapter is to increase the exploration capability of the Gravitational search algorithm in the Cuckoo search (CS) algorithm. The CS algorithm is common for its exploitation conduct. The other motivation behind this proposal is to obtain a quicker and stable solution. Twenty-three different kinds of standard test functions are considered here to compare the performance of our hybridized algorithm with both the CS and the GSA methods. Extensive simulation-based results are presented in the results section to show that the proposed algorithm outperforms both CS and GSA algorithms. We land up with a faster convergence than the CS and the GSA algorithms. Thus, best solutions are found with significantly less number of function evaluations. This chapter also explains how to handle the constrained optimization problems with suitable examples.

Keywords Cuckoo search · Gravitational search algorithm · Optimization

M.K. Naik (✉)
Department of Electronics & Instrumentation Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan University, Bhubaneswar 751030, India
e-mail: manojnaik@soauniversity.ac.in

L. Samantaray
Department of Electronics & Instrumentation Engineering, Ajaya Binaya Institute of Technology, Cuttack, India
e-mail: leena_sam@rediffmail.com

R. Panda
Department of Electronics & Telecommunication Engineering, VSS University of Technology, Burla 768018, India
e-mail: r_ppanda@yahoo.co.in

S. Bhattacharyya et al. (eds.), *Hybrid Soft Computing Approaches*, Studies in Computational Intelligence 611,
DOI 10.1007/978-81-322-2544-7_1

1 Introduction

Over the decades, the evolutionary computation (EC) algorithms have been successfully smeared to solve the different practical computational problems like optimization of the objective functions [1, 2], optimization of filter parameters [3, 4], optimization of different parameters for improvising image processing results [5, 6], optimal feature selection in pattern recognition [7–9], etc. Aforementioned engineering applications are basically motivated by the near-global optimization norm of the evolutionary computational methods. This permits those algorithms to accomplish the task within a very big search space (of a given problem). However, certain ECs are not yet investigated for solving a particular tricky efficiently. The room of the proposed idea is to conduit the slit. It may generate concern among the readers doing research in this area. There is a strong need to develop new hybrid algorithms to suit a particular problem in hand. Further, by presenting a widespread simulation results on the performance of 2 moderately firsthand ECs, we try to convince the readers of their importance. In this chapter, an attempt is made to attract the researchers regarding the aptness of the proposed technique for solving the delinquent of the function optimization.

Many types of heuristic search ECs were proposed by investigators Genetic algorithm (GA) [10], Ant colony algorithm (ACA) [11], Particle swarm optimization (PSO) algorithm [12], Bacterial foraging optimization (BFO) algorithm [13], Cuckoo search (CS) algorithm [14–19], Gravitational search algorithm (GSA) [20], etc. But a particular algorithm is not efficient to solve different types of optimization problems. They never provide us with the best solutions. Certain algorithms offer best solutions for particular (given) problems only. Thus, it is necessary to devise hybridized heuristic population-based optimization methods for solving different applications efficiently. In this chapter, a population-based hybridized optimization algorithm is proposed. In this work, the thrust is mainly to cartel the social thinking capability found in the Cuckoo birds and the local search capability observed in Gravitational search method. Actually, an efficient optimization scheme is well umpired by its two key features—(i) exploration and (ii) exploitation. Note that exploration is the capability of an EC to explore the complete search space, and exploitation is the skill to congregate to a better result. Combining these two features, the best solutions can be obtained with less number of function evaluations. Therefore, here an attempt is made to hybridize CS with GSA in order to provide equilibrium for both exploration and exploitation ability. This chapter describes the use of both the methods in a balanced manner. Note that hybridized approach provides significant improvements in the solutions. Interestingly, the CS is popular for its simplicity and its ability to search for a near-global solution. Further, GSA provides a better local search method with good initial estimates to solve a particular problem.

In fact, EC techniques were initially proposed by different researchers as an approach to the artificial intelligence. Later, it has become more popular and is used

directly to solve analytical and combinatorial optimization problems. However, the demerit of an EC method is its slow convergence in solving multimodal optimization problems, to find near (global) optimum values. In this context, various EC techniques are proposed. Recently, there has been a strong need to develop new hybridized EC techniques to find better results. This chapter discusses the application of a new hybrid EC method for optimization. Extensive empirical evaluations are important to measure the strength and weaknesses of a particular algorithm. In this connection, we consider 23 standard benchmark [21] functions. The proposed CS–GSA algorithm performs better than GSA and CS techniques for multimodal functions with many local minima. In addition, the proposed algorithm is faster than GSA. The levy flight incorporated in the algorithm helps us for a speed to reach the near (global) optimum very quickly. The idea behind this is simple and straightforward. Thinking ability of the Cuckoo birds is very useful for exploitation. In this study, we propose the hybridization of these two algorithms. Here, 23 standard benchmark functions [21] are utilized to relate the performance of our method. The results are compared to both the CS and GSA algorithms. In this work, the solutions presented in the chapter reveal the fact that our proposed algorithm is well suited for function minimization.

For this work, we consider 23 benchmark functions. These functions are given in Table 1. More details on such test functions are discussed in the Appendix. Note that the functions F_1–F_{13} refer to the high-dimensional problems, whereas the functions F_1–F_7 are known as the unimodal functions. The function F_5 is a step function. This has only one minimum. Further, the function is discontinuous. Here, the function F_7 is coined as the quartic function with noise. Here, rand (0, 1) is basically a uniformly distributed random variable in the range (0, 1). Here, we also consider multimodal functions F_8–F_{13} for our experimental study. For these functions, the amount of local minima surges exponentially with the surge in the problem size. It is important to solve these types of functions to validate our

Table 1 Unimodal benchmark functions

Benchmark function
$F_1(\mathrm{X}) = \sum_{i=1}^{n} x_i^2$
$F_2(\mathrm{X}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$
$F_3(\mathrm{X}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$
$F_4(\mathrm{X}) = \max_i \{ \lvert x_i \rvert, 1 \le i \le n \}$
$F_5(\mathrm{X}) = \sum_{i=1}^{n-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right]$
$F_6(\mathrm{X}) = \sum_{i=1}^{n} \left(\lfloor x_i + 0.5 \rfloor \right)^2$
$F_7(\mathrm{X}) = \sum_{i=1}^{n} i x_i^4 + \mathrm{random}[0,\ 1)$

algorithm for optimization. The functions F_{14}–F_{23} are called as the functions with the low dimension. These functions have only a few local minima. It is relatively easy to optimize the unimodal functions. However, the optimization is much more significant in the case of multimodal functions. The ability of an algorithm is well judged by its behavior, capable of escaping from the poor local optima while localizing the best near (global) solution. Note that the functions F_{14}–F_{23} are multimodal in nature, but are fixed-dimensional functions. These functions are also very useful for the measure of the accuracy of the hybrid soft computing techniques.

The organization of the rest of the chapter is as follows: Sect. 1 is the introduction part. Related work is discussed in Sect. 2. The hybridized population-based CS–GSA algorithm is discussed in Sect. 3. Extensive results and discussions are presented in Sect. 4. In this chapter, conclusions are presented in Sect. 5.

2 Related Works

The main objective of an optimization algorithm is to find a near or near-global optimal solution. Several EC algorithms are presented in the literature, but the CS algorithm [14–16] has its own importance. Interestingly, it consists of less parameters for search, and so it is a faster optimization method. Recently, Chetty and Adewumi [22] have done a case study on an annual crop planning problem using a CS algorithm along with the GA and glow worm swarm optimization (GSO); the former has shown superior results. Chen and Do [23] used the CS to train the feed-forward neural networks for predicting the student academic performances. Swain et al. [24] have proposed neural network based on CS and apply the same for the noise cancellation. Khodier [25] used the CS algorithm for optimization of the antenna array. Although the CS gives better performance, some researcher adds some more characteristics to the search process. Zhao and Li [26] proposed opposition-based learning to upsurge the exploration proficiency of the CS algorithm.

The other non-evolutionary algorithms are also good at finding the near-global optima. The GSA [20] is founded on the rule of gravity. It has better convergence in the search space. Saha et al. [27]; Rashedi et al. [28] used the GSA for the filter design and modeling. Many authors proposed new schemes to enhance the performance of the GSA such as Disruption operator [29], black hole operator [30], Niche GSA [31], and binary GSA (BGSA) [32].

Every algorithm has its own merits and demerits. To improve search performance of one algorithm, some researcher proposed hybrid algorithm combining the features of more than one algorithm. Mirjalili and Hashim [33] proposed hybrid algorithm PSOGSA by integrating the ability of exploitation in PSO and the ability of exploration in GSA. Jiang et al. [34] proposed HPSO-GSA for solving economic

emission load dispatch problems by updating the particle position with PSO velocity and GSA acceleration. Ghodrati and Lotfi [35] proposed hybrid algorithm CS/PSO by combining the idea of cuckoo birds awareness of each other position via swarm intelligence of PSO. A hybrid algorithm on GSA–ABC was proposed by Guo [36] to update the ant colony with the help of an artificial bee colony algorithm (ABC) and GSA. Sun and Zhang [37] have proposed GA–GSA hybrid algorithm for image segmentation based on the GA and GSA. Yin et al. [38] proposed a hybrid IGSAKHM approach for clustering by combining K-harmonic means (KHM) clustering technique and improved gravitational search algorithm (IGSA).

In this section, we discuss the potential features of two different algorithms used for optimization.

2.1 Cuckoo Search (CS) Algorithm

Cuckoo search (CS) method is basically a nature-inspired technique. This method is introduced by Yang and Deb [14–17]. The CS is inspired by an interesting event how the Cuckoo bird leaves eggs in the nest of another horde bird. The available host nests are fixed. The egg laid by the Cuckoo bird may be exposed by the horde bird with a probability $p_a \in [0, 1]$. Then, the horde birds either throw those eggs or abandon the present nest. Sometimes, the horde bird builds a new nest in a totally different location [18] to deceive the Cuckoo bird. Here, each egg in the nest represents a solution. It is interesting to note here that the CS has similarity to the well-known hill climbing algorithm. Here, in the Cuckoo search, note that a particular pattern corresponds to a nest, whereas an individual feature of that pattern resembles to that of an egg of the Cuckoo bird.

Interestingly, the CS method is founded on the succeeding three more idealized strategies:

i. Every Cuckoo bird puts an egg at a time, it junk yards its egg in a randomly selected nest.
ii. Nests having the best class of eggs are carried over to the subsequent generations.
iii. Always the quantity of available horde nests is fixed. Note that the egg placed by a Cuckoo bird is exposed by the horde bird with a probability p_a € [0, 1]. Then, the worst nests are revealed and discarded from future calculations.

The CS algorithm is mathematically modeled as follows: For a new search space $X_i(t + 1)$ for Cuckoo i (for $i = 1, 2, \ldots, N$) at time $t + 1$,

$$X_i^{t+1} = X_i^t + \alpha \oplus \text{Lévy}(\lambda), \tag{1}$$

where $X_i(t)$ is the current search space at time t, represented as $X_i = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^n\right)$, $\alpha > 0$ is the step size connected to the range of the problem, $\oplus$ is the entry wise multiplication, and Lévy(λ) is the random walk through the Lévy flight. The Lévy flight [19] provides random walk for step size from the Lévy distribution $\left(\text{Lévy} \sim u = t^{-\lambda}\right)$ by considering λ such that it satisfies $1 < \lambda < 3$.

Here, time t denotes the number for a recent group (t = 1, 2, 3, …, $t_{\max}$) and $t_{\max}$ represents the pre-determined extreme cohort position. Here, the initial values of the dth attribute of the ith pattern are found by

$$x_i^d(t=0) = \text{rand} \cdot \left(u^d x_i^d - l^d x_i^d\right) + l^d x_i^d, \tag{2}$$

where l^d and u^d are called as the lower and the upper search space limits of the dth attributes, respectively. These attributes are useful for implementations. This method is used to provide control over the boundary conditions in every calculation step. Note that the value for the interrelated attribute is restructured, when it exceeds the allowed search space limits. This is achieved by considering the value for the nearby limit corresponding to the linked trait. In this discussion, it is seen that the CS method identifies the best fruitful pattern as the X_{best} pattern. This process is accomplished before starting the iterative search method. Actually, here in the CS algorithm, the iterative growth part of the pattern matrix initiates by the discovery step of the Φ as

$$\Phi = \left(\frac{\Gamma(1+\gamma) \cdot \sin(\pi \cdot \gamma/2)}{\Gamma\left(\left(\frac{1+\gamma}{2}\right) \cdot \gamma \cdot 2^{\frac{\gamma-1}{2}}\right)} \right)^{\frac{1}{\gamma}}. \tag{3}$$

The Γ in the above expression represents a gamma function and $\gamma - 1 = \lambda$.

The evolution phase of the X_i pattern is defined by the donor vector v, where $v = X_{i.}$ The required step size value is calculated as

$$\text{stepsize}_d = 0.01 \cdot \left(\frac{u_d}{v_d}\right)^{\frac{1}{\gamma}} \cdot (v - X_{\text{best}}).$$

Note that $u = \Phi$ randn[n] and v = randn[n].

Here, a donor pattern is randomly mutated as

$$v = v + \text{stepsize}_d \cdot \text{randn}[n]. \tag{4}$$

In this scheme, the update procedure of X_{best} in CS is stated as

$$X_{\text{best}} \leftarrow f(X_{\text{best}}) \leq f(X_i). \tag{5}$$

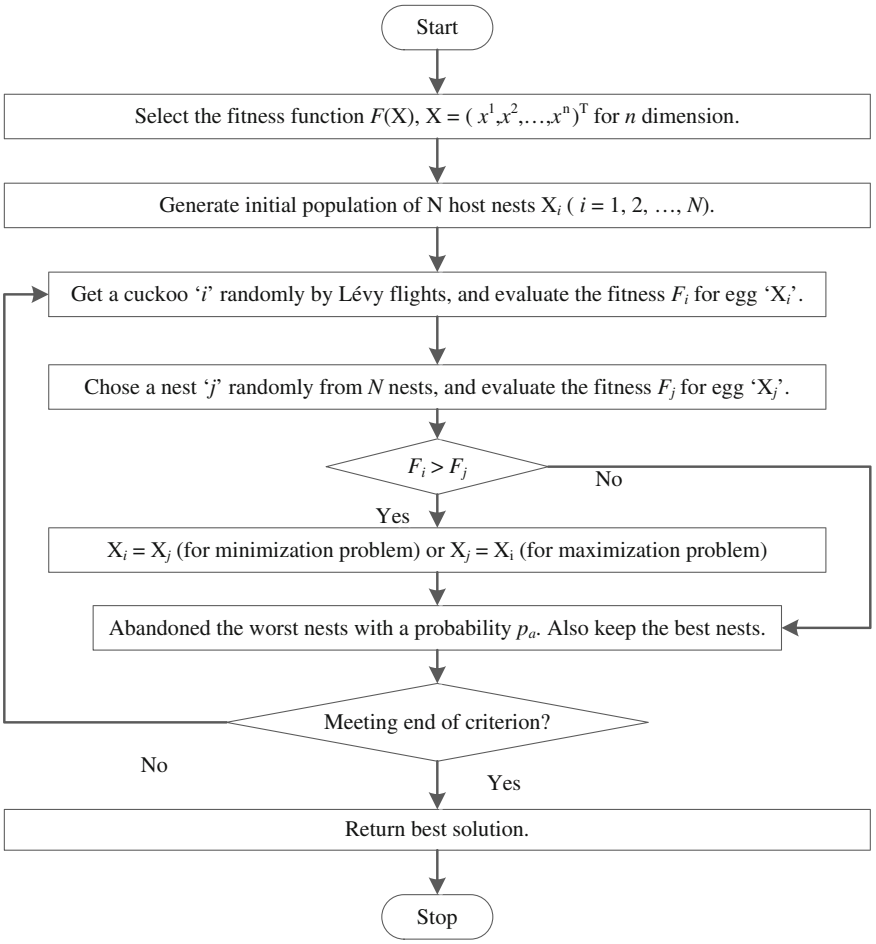

Fig. 1 Flow chart of CS

It is noteworthy to mention here that the controller constraints of the CS technique are coined as scale factor (λ) followed by the mutation probability (p_a). The flow chart for the CS algorithm is shown in Fig. 1.

PseudoCode for CS

First identifies the search space-like dimension of search problem '*n*,' the range of the objective function, and objective function $F(X)$. Let us choose some important parameters N, p_a, α, λ, $t_{\max}$, and $t = 1$. Also, randomly initialize the population of N host nests $X_i(t) = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^n\right)$ with n dimension for $i = 1, 2, \ldots, N$.

do {

(a) *Get a cuckoo 'i' randomly by Lévy flights, and evaluate the fitness* F_i *for egg* X_i.
(b) *Chose a nest 'j' randomly from N nests, and evaluate the fitness* F_j *for egg* X_j.
(c) *If* $(F_i > F_j)$
Replace X_i *with* X_j *for minimization problem, or Replace* X_j *with* X_i *for maximization problem.*
End
(d) *The worst nests are abandoned with a probability* (p_a). *The new ones are built and keep the best ones.*
(e) $t = t + 1$.

} *while* $(t < (t_{max} + 1))$ *or End criterion not satisfied).*

2.2 Gravitational Search Algorithm (GSA)

The GSA is based on the underlying principle of the Newton's theory. This was introduced in [20]. The Newton's theory states that 'Every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them.' Note that the force is inversely proportional to the distance between them.

This algorithm is quite interesting and considered as a collection of N agents (masses) only. Here, the masses relate to the solution of an optimization (given) problem. It is interesting to note here that the heavier mass has greater force of attraction. This may be very near to the global optima. Let us initialize the search space of the ith agent as $X_i = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^n\right)$(for $i = 1, 2, \ldots, N$), where n signifies the dimension of the search space. Note that at a time t, the force of attraction between mass 'i' by mass 'j' is defined as

$$F_{ij}(t) = G(t)\frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t) + \varepsilon}\left(x_j(t) - x_i(t)\right), \tag{6}$$

where M_{aj} and M_{pj} are the active and passive gravitational masses connected to the agent i, $G(t)$ is coined as a gravitational constant at a particular time t, and $R_{ij}(t)$ is the Euclidian distance between agents 'i' and 'j,' which is written as

$$R_{ij} = \left\|X_i(t) \cdot X_j(t)\right\|_2. \tag{7}$$

It is noteworthy to mention here that the gravitational constant G gradually decreases with respect to the time. This event actually helps us to reach at the minima in the search space. Therefore, the gravitational constant G is considered as a function of the initial value G_0 together with the time t. The phenomenon can mathematically be modeled as

$$G(t) = G_0 \times e^{\left(-\beta \frac{t}{t_{\max}}\right)}, \tag{8}$$

where β is the descending coefficients. Note that $t_{\max}$ is the maximum number of iterations considered for the simulation work.

Thus, the total amount of the force that acts on the agent i is $F_i(t)$, which can be computed from Eq. (6) as

$$F_i(t) = \sum_{j=1, j\neq i}^{N} \text{rand}_j \, F_{\text{ij}}(t). \tag{9}$$

Here, different masses are computed from the fitness values. Further, the masses are updated by the following set of equations:

$$M_{\text{ai}} = M_{\text{pi}} = M_{\text{ii}} = M_i, i = 1, 2, \ldots, N, \tag{10}$$

$$m_i(t) = \frac{\text{fit}_i(t) - \text{worst}(t)}{\text{best}(t) - \text{worst}(t)}, \tag{11}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{12}$$

where $\text{fit}_i(t)$ designates the fitness cost of an agent i at a particular time t. Here, best (t) signifies the best fitness value and worst(t) represents the worst fitness value among the N agents. Here, the acceleration of agent i at time t can be given by

$$a_i(t) = {}^{F_i(t)}/_{M_i(t)}, \tag{13}$$

where $M_i(t)$ is coined as the mass of an agent i.

Finally, the velocity and the position of an agent in the search space are computed as given below:

$$x_i(t+1) = x_i(t) + v_i(t+1), \tag{14}$$

$$\text{and}, v_i(t+1) = \text{rand}_i \times v_i(t) + a_i(t). \tag{15}$$

Note that the positions are updated iteratively using the above equations till the GSA algorithm reaches the global or near-global minima. Actually, no further change in the mass should undergo after attaining the global or near-global minima. The flow chart for GSA is displayed in Fig. 2.

PseudoCode for GSA

In the beginning, it recognizes the search space, dimension of the search problem '*n*,' the range of the objective function, and the objective function itself, i.e., F(X).

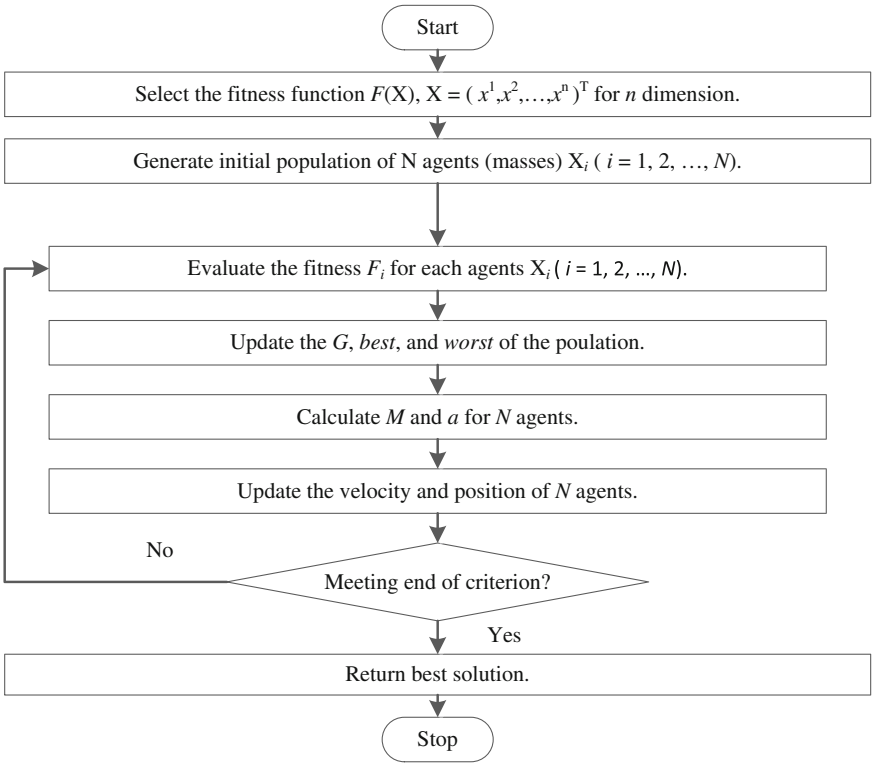

Fig. 2 Flow chart of GSA

Then choose some important parameters N, G_0, β, $t_{\max}$, and $t = 1$. After choosing the parameters, randomly initialize the population of N agents (or masses) $X_i(t) = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^n\right)$ with n dimension for $i = 1, 2, \ldots, N$.

do {

(a) Evaluate the objective function $F(X_i)$ *for* $i = 1,2,\ldots,N$.
(b) Update $G(t)$, *best*(t), *and worst*(t) *of the current population.*
(c) Calculate $M_i(t)$,$v_i(t)$ *and* $a_i(t)$ *for all N agents.*
(d) Then calculate the new position $X_i(t+1)$ *of agents for* $i = 1,2,\ldots,N$.
(e) $t = t + 1$.

} *while* $(t < (t_{max} + 1))$ *or End criterion not satisfied).*

3 A Hybrid CS–GSA Algorithm

It is well known from the literature that the CS is a heuristic search method based on evolutionary computational approach. The beauty of the CS algorithm is that it uses the randomized walk via a Lévy flight, as described in [14, 15]. Here, the Lévy flight is quite effectual in discovering the search space. Note that the step size is booked from the Lévy distribution as explained in [19].

Let us choose α as 1 (as $\alpha > 0$). So Eq. (1) is reduced to

$$X_i(t+1) = X_i(t) + \text{Lévy}(\lambda). \tag{16}$$

From the above Eq. (16), it is clearly showed that the new search space (the new solution) only rest on a Lévy distribution. Let us introduce a term lBest (t), which provides the best local solution among $i = 1, 2, \ldots, N$ at time t. Here, the lBest(t) can be expressed by

$$\begin{gathered} l\text{Best}(t) = X_j(t) \big| \forall j == i, \\ \text{for which } f(X_i(t)) \text{ is minimum,} \\ \text{for } i = 1, 2, \ldots, N \text{ at time } t. \end{gathered} \tag{17}$$

Let us again incorporate an additional term (coined as the proportionate term) to the new solution, thereby incorporating the difference between the current solution and the local best solution at time t. Therefore, Eq. (16) can be expressed by

$$X_i(t+1) = X_i(t) + \text{Lévy}(\lambda) \times (l\text{Best}(t) - X_i(t)). \tag{18}$$

Further, let us see how every solution differs from the other at time t. In this sense, the acceleration of an agent i at a time t is used to provide enhancement to the local search in the GSA algorithm. Once more, we incorporate Eq. (13) in Eq. (18) as expressed by

$$X_i(t+1) = X_i(t) + \text{Lévy}(\lambda) \times (l\text{Best}(t) - X_i(t)) + a_i(t). \tag{19}$$

It is noteworthy to mention here that $a_i(t)$ is defined in Eq. (13). If we choose α as the proportionate measure of the step size, then Eq. (19) can be re-organized as

$$X_i(t+1) = X_i(t) + \alpha \times \text{Lévy}(\lambda) \times (l\text{Best}(t) - X_i(t)) + a_i(t). \tag{20}$$

Thus, Eq. (20) provides the new solution space for Cuckoo Search–Gravitational Search Algorithm (CS–GSA) from the list of current solutions obtained in this method. The flow chart for our new hybrid CS–GSA algorithm is shown in Fig. 3.

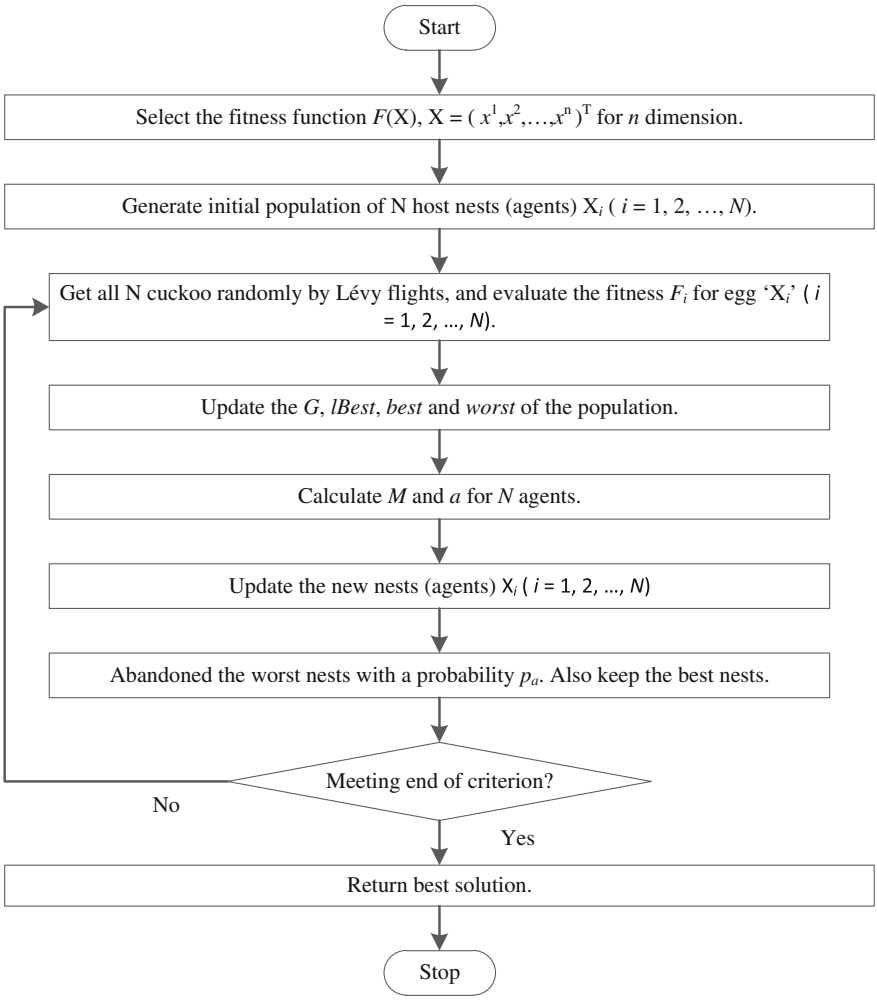

Fig. 3 Flow chart of CS–GSA

PseudoCode for CS–GSA

In the opening, the objective function $f(\mathrm{X})$ with the dimension n is recognized. Then choose the parameters N, p_a, G_0, α, λ, β, $t_{\max}$, and $t = 1$ to control the algorithm while in iteration. Let us initialize randomly the population of N horde nests $\mathrm{X}_i(t) = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^n\right)$ with n dimension for $i = 1, 2, \ldots, N$ at $t = 1$.

do {

(f) Evaluate the objective function $f(\mathrm{X}_i)$ *for i = 1,2,…,N.*

(g) Analyze all the fitness functions $f(\mathrm{X}_i)$ *for i = 1,2,…,N. Then find the lBest(t) from the Eq.(17).*

(h) Update G(t) from the Eq. (8), $M_i(t)$ *from the Eq. (12). Then compute acceleration* $a_i(t)$ *from the Eq. (13).*

(i) Then compute the new position of Cuckoo nests by using the Eq. (20).

(j) The worst nests are abandoned with a probability (p_a)*. The new ones are built. Then, keep the best ones.*

(k) $t = t + 1$.

} while ($t < (t_{max} + 1)$ *or End criterion not satisfied).*

Finally, report the best $f(\mathrm{X}_i)$ with $i = 1, 2, \ldots, N$, also report the corresponding X_i.

4 Results and Discussions

In this work, the main thrust is to improve the CS algorithm in comparison to the standard CS methodology. Here, it is also important to bring some improvement over the GSA. In this context, a new CS–GSA algorithm has been developed in the previous Section.

4.1 Performance Evaluation Using Standard Benchmark Functions

In the performance evaluation of the proposed algorithm, we consider 23 standard benchmark functions [21] displayed in Tables 1, 2 and 3. Note that the convergence rate of the unimodal benchmark functions is important to validate an optimization algorithm. These useful functions are listed in Table 1.

It is noteworthy to mention here that the multimodal benchmark functions also have a significant role in validating optimization algorithms. These multimodal functions have many local minima, so it is difficult to optimize these functions. Such functions are displayed in Table 2 and are used for the performance measure.

Further, multimodal functions with fixed dimensions are also considered in this work. These types are displayed in Table 3. Generally, these functions have similar performances for all types of optimization algorithms.

Table 2 Multimodal benchmark functions

Benchmark function
$F_8(\mathrm{X}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$
$F_9(\mathrm{X}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$
$F_{10}(\mathrm{X}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
$F_{11}(\mathrm{X}) = \frac{\pi}{n}\left\{10\sin(\pi y_i) + \sum_{i=1}^{n}(y_i - 1)^2\left[1 + 10\sin^2(\pi \mathrm{y}_{\mathrm{i}+1})\right] + (\mathrm{y}_{\mathrm{n}} - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$
$F_{12}(\mathrm{X}) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_n - 1)^2 \cdot \left[1 + \sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$

Table 3 Multimodal benchmark functions with fixed dimension

Benchmark function	n
$F_{13}(\mathrm{X}) = \sum_{i=1}^{30} -x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)$	30
$F_{14}(\mathrm{X}) = \left(\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}\left(x_i - a_{\mathrm{ij}}\right)^6}\right)^{-1}$	2
$F_{15}(\mathrm{X}) = \sum_{i=1}^{11}\left[a_i - \frac{x_1\left(b_i^2 + b_i x_2\right)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4
$F_{16}(\mathrm{X}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2
$F_{17}(\mathrm{X}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right) + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	2
$F_{18}(\mathrm{X}) = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2\right)\right] \times \left[(2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right) + 30\right]$	2
$F_{19}(\mathrm{X}) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{\mathrm{ij}}\left(x_j - p_{\mathrm{ij}}\right)^2\right)$	3
$F_{20}(\mathrm{X}) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{\mathrm{ij}}\left(x_j - p_{\mathrm{ij}}\right)^2\right)$	6
$F_{21}(\mathrm{X}) = -\sum_{i=1}^{5}\left[(\mathrm{X} - a_i)(\mathrm{X} - a_i)^T + c_i\right]^{-1}$	4
$F_{22}(\mathrm{X}) = -\sum_{i=1}^{7}\left[(\mathrm{X} - a_i)(\mathrm{X} - a_i)^T + c_i\right]^{-1}$	4
$F_{23}(\mathrm{X}) = -\sum_{i=1}^{10}\left[(\mathrm{X} - a_i)(\mathrm{X} - a_i)^T + c_i\right]^{-1}$	4

Table 4 Parameter setting for GSA, CS, and CS–GSA for benchmark functions (F_1–F_{23})

Number of agents (masses or nests) $N = 50$
Number of maximum iteration $t_{max} = 1000$
Initial gravitational constant $G_0 = 100$
Mutation probability $p_a = 0.25$
Constant $\alpha = 1$, $\beta = 20$, $\lambda = 1.5$

In this simulation study, the range of x_i is different for different functions. The ranges of these functions are described in Appendix. The choice of parameters is important for evaluating the optimization algorithm. We have chosen best parameters for our study based on extensive simulation results. These parameters are used for all the three algorithms. The parameter setting for GSA, CS, and CS–GSA is shown in Table 4, for validating the benchmark functions given in Tables 1, 2 and 3.

The benchmark functions are categorized into three different tables as unimodal test functions (F_1–F_7), multimodal test functions (F_8–F_{12}), and multimodal test functions with fixed dimensions (F_{13}–F_{23}). The ranges of the objective functions and the global minima are given in the Appendix. The search dimension 'n' is taken as 10, 50 for the unimodal functions given in Table 1 and multimodal benchmark functions given in Table 2. The search dimension 'n' for the multimodal functions with fixed dimension is given in Table 3.

The performance evaluations of GSA, CS, and CS–GSA for the unimodal benchmark functions are presented in Tables 5 and 6.

From Table 5, it is observed that the performance of GSA for the unimodal benchmark functions F_1, F_2, and F_4 with $n = 10$ seems to be better than the other two algorithms. But for other functions, the performance of the CS–GSA algorithm is better. For all benchmark functions, final results are reflected as the 'Best,' 'Median,' and 'Ave' among 50 independent runs. Here, 'Best' implies the best fitness value obtained from 50 independent runs. 'Median' refers to the median of 50 fitness values obtained from 50 independent runs. The 'Ave' denotes the average value of 50 fitness values obtained from 50 independent runs. Within a function, the performance of GSA, CS, and CS–GSA is compared. The best solutions among all three algorithms are shown in boldface letters.

The performance evaluation of GSA, CS, and CS–GSA for the unimodal benchmark functions F_1 to F_7 with $n = 50$ is displayed in Table 6. Here, the proposed CS–GSA algorithm performs well as compared to other algorithms.

A comparison of these algorithms for the unimodal benchmark functions F_4 and F_7 with $n = 50$ is shown in Fig. 4. It is seen that CS–GSA offers us best values compared to other algorithms. Note that the maximum number of iterations considered here is 1000. Here, GSA is the second best.

The performance evaluation of GSA, CS, and CS–GSA for the multimodal benchmark functions F_8 to F_{12} with $n = 10$ is displayed in Table 7. Here, the proposed CS–GSA algorithm performs well for all functions except F_{12}.

The performance evaluation of GSA, CS, and CS–GSA for the multimodal benchmark functions F_8 to F_{12} with $n = 50$ is displayed in Table 8. Here, the new

Table 5 Performance evaluation of GSA, CS, and CS–GSA for the unimodal benchmark functions (displayed in Table 1) with $n = 10$

		GSA	CS	CS–GSA
F_1	Best	1.4442e-041	1.6724e-018	**3.2886e-042**
	Median	**1.5288e-040**	2.2921e-016	5.4848e-039
	Average	**1.5737e-039**	2.2606e-016	8.6919e-027
F_2	Best	**4.2795e-020**	7.2314e-005	9.7964e-020
	Median	**1.1788e-019**	2.2356e-004	4.8926e-019
	Average	**1.1491e-019**	1.1321e-003	7.1335e-016
F_3	Best	3.6769	4.4477	**2.2928**
	Median	29.0889	33.4532	**7.3735**
	Average	25.7047	28.9740	**18.1172**
F_4	Best	**1.3869e-020**	1.7780e-003	6.2616e-019
	Median	**9.5122e-020**	2.6564e-002	1.4920e-018
	Average	**1.5330e-019**	0.2551	1.2400e-012
F_5	Best	7.7969	12.4932	**7.7306**
	Median	**7.9811**	15.1342	8.2032
	Average	36.4058	39.2132	**35.7741**
F_6	Best	**0**	0.0121	**0**
	Median	**0**	0.0316	**0**
	Average	**0**	0.1276	**0**
F_7	Best	4.9725e-004	0.0217	**4.7790e-004**
	Median	0.0042	0.0319	**9.8222e-004**
	Average	0.0040	0.0332	**0.0011**

hybrid CS–GSA algorithm performs well for all functions except F_9. For the function F_9, GSA performs better.

A comparison of these algorithms for the multimodal benchmark functions F_8 and F_{12} with $n = 50$ is shown in Fig. 5. It is observed that the performance of the CS–GSA algorithm is better compared to other algorithms. Here, the maximum number of iterations is 1000. From Fig. 5, it is observed that the GSA is the second contestant.

The performance evaluation of GSA, CS, and CS–GSA for the multimodal benchmark functions F_{13} to F_{23} with *fixed dimension* is displayed in Table 9. Here, the new hybrid CS–GSA algorithm performs well for all functions except F_{14} and F_{15}. For the function F_{14}, CS performs better than GSA and CS–GSA algorithms. For the function F_{15}, GSA performs better than CS and CS–GSA algorithms. From the knowledge of the 'Best,' 'Median,' and 'Average' values, one can claim that CS–GSA can be used for optimization of such type of functions.

A comparison of these algorithms for the multimodal benchmark functions F_{15} and F_{17} with *fixed dimension* is shown in Fig. 6. It is observed that the performance of the CS–GSA algorithm is better compared to other algorithms. In this study, the

Table 6 Performance evaluation of GSA, CS, and CS–GSA for the unimodal benchmark functions (displayed in Table 1) with $n = 50$

		GSA	CS	CS–GSA
F_1	Best	17.9649	32.9181	**17.7891**
	Median	**204.0765**	449.6065	210.6723
	Average	235.1073	438.9923	**235.0924**
F_2	Best	**0.0055**	0.2223	0.0063
	Median	0.3428	0.4013	**0.1391**
	Average	0.4440	0.6051	**0.3131**
F_3	Best	**1.1520e+003**	5.4531e+003	1.8343e+003
	Median	2.3741e+003	8.0569e+003	**2.0492e+003**
	Average	**2.5979e+003**	7.9411e+003	2.7163e+003
F_4	Best	6.9320	6.0721	**4.6871**
	Median	9.6256	9.7903	**6.1135**
	Average	9.9463	8.9143	**5.9281**
F_5	Best	486.0379	473.9232	**290.1693**
	Median	1.4569e+003	7.8442e+003	**1.0498e+003**
	Average	1.8367e+003	7.8224e+003	**1.6808e+003**
F_6	Best	**84**	221	231
	Median	447	463	**422**
	Average	492.3667	498.4113	**460.2314**
F_7	Best	0.0704	0.1822	**0.0150**
	Median	0.1502	0.1890	**0.0347**
	Average	0.1659	0.3907	**0.0361**

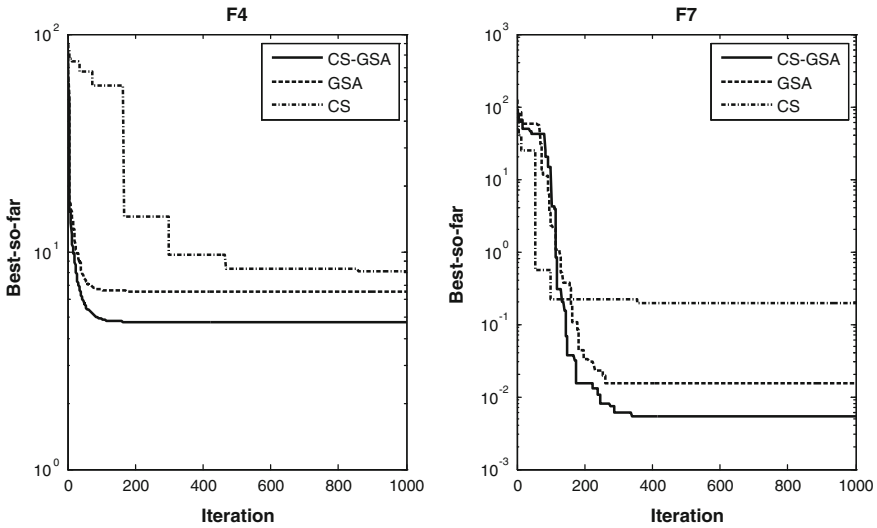

Fig. 4 Performance comparison of GSA, CS, and CS–GSA for the unimodal functions F_4 and F_7 with n = 50

Table 7 Performance evaluation of GSA, CS, and CS–GSA for the multimodal benchmark functions given in Table 2 with $n = 10$

		GSA	CS	CS–GSA
F_8	Best	0.9950	2.2075	**0**
	Median	2.9849	5.0666	**0.9950**
	Average	3.5276	5.5697	**0.8203**
F_9	Best	4.4409e-015	1.1146e-008	**8.8818e-016**
	Median	4.4409e-015	1.5178e-006	**4.4409e-015**
	Average	4.4409e-015	1.4342e-004	**4.1179e-015**
F_{10}	Best	1.9788	1.9305	**1.5922**
	Median	4.3241	4.8115	**2.0312**
	Average	4.4531	5.1299	**2.0954**
F_{11}	Best	4.7116e-032	2.4777	**2.8302e-004**
	Median	4.1906e-005	3.2834	**0.0012**
	Average	0.0585	3.0768	**0.0028**
F_{12}	Best	**1.3498e-032**	1.7255e-005	5.630×10^{-19}
	Median	**1.3498e-032**	1.0319e-004	1.106×10^{-18}
	Average	**1.3498e-032**	1.0445e-002	3.666×10^{-4}

Table 8 Performance evaluation of GSA, CS, and CS–GSA for the multimodal benchmark functions given in Table 2 with $n = 50$

		GSA	CS	CS–GSA
F_8	Best	19.9081	48.6626	**6.0071**
	Median	36.9289	53.0276	**12.2704**
	Average	35.7943	52.6585	**13.6570**
F_9	Best	0.3921	1.8128	**0.2686**
	Median	**0.4660**	2.3657	1.9741
	Average	**0.5669**	2.6760	1.9784
F_{10}	Best	**155.8180**	206.0770	185.4950
	Median	204.1554	281.4023	**199.5789**
	Average	203.1253	286.6942	**201.3960**
F_{11}	Best	**0.7625**	3.7943	0.9637
	Median	2.4986	5.0298	**1.7126**
	Average	2.7001	6.0326	**2.3431**
F_{12}	Best	**23.2165**	26.3823	24.5563
	Median	44.2780	41.5956	**39.8451**
	Average	**44.8839**	48.2117	47.2341

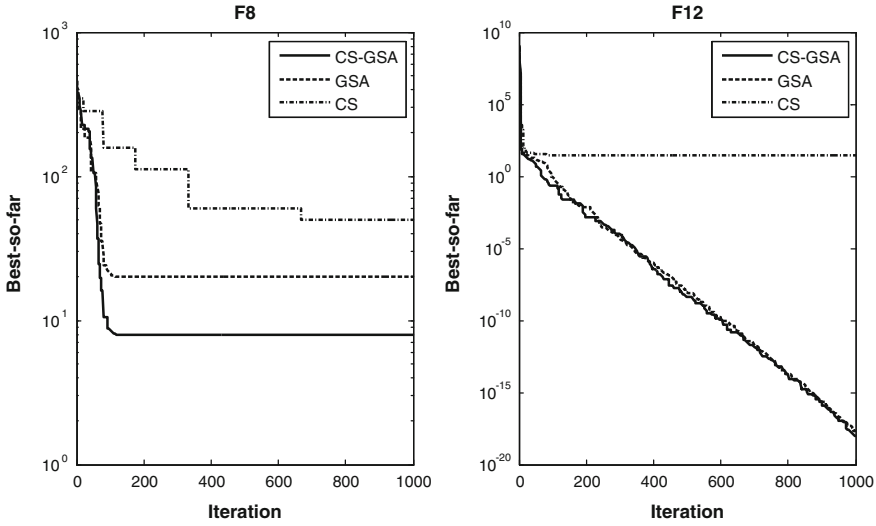

Fig. 5 Performance comparison of CS–GSA, CS, and GSA for the multimodal functions F_8 and F_{12} with n = 50

maximum number of iterations is 1000. From Fig. 6, it is seen that the GSA is the second contestant.

The performances of the proposed algorithm are summarized as follows:

- *For the unimodal test functions* (F_1–F_7): When the best results are concerned, CS–GSA outperforms GSA and CS. When the median and the average values are concerned, GSA outperforms CS–GSA and CS. However, CS–GSA has significant improvements over the CS.
- *For the multimodal test functions* (F_8–F_{12}): For functions F_8 to F_{12} (except F_{11}), the results are dominated by the CS–GSA over GSA and CS.
- *For the multimodal test functions with fixed dimensions* (F_{13}–F_{23}): The result in these functions is not varying so much, but still CS–GSA outperforms the other two algorithms GSA and CS.

The convergence of four benchmark functions, out of 23 such functions, is shown in Figs. 4, 5 and 6 using CS–GSA, GSA, and CS. Here, we consider 1000 iterations. In most of the cases, CS–GSA has shown a better convergence rate than GSA and CS. Reason is that CS has the ability to abandon the worst solutions, while searching for the best solutions quickly. From Figs. 4, 5 and 6, it is observed that CS–GSA provides best fitness function values compared to GSA and CS algorithms, because of the fact that the GSA has the ability to provide the best local search mechanism. Hence, by combining these features of CS and GSA in the hybridized CS–GSA, we get the best results.

Table 9 Performance evaluation of GSA, CS, and CS–GSA for the multimodal benchmark functions given in Table 3 with fixed dimension

		GSA	CS	CS–GSA
F_{13}	Best	−1.7928e+003	−2.5756e+003	**−1.7211e+003**
	Median	**−1.6104e+003**	−2.3242e+003	−1.8677e+003
	Average	**−1.5480e+003**	−2.3359e+003	−1.9888e+003
F_{14}	Best	**0.9980**	**0.9980**	**0.9980**
	Median	3.9711	**0.9985**	0.9984
	Average	5.1573	**1.0009**	1.0351
F_{15}	Best	**6.4199e-004**	9.4845e-004	7.4790e-004
	Median	**0.0038**	0.0016	0.0012
	Average	**0.0042**	0.0018	0.0014
F_{16}	Best	**−1.0316**	−1.0314	**−1.0316**
	Median	**−1.0316**	−1.0305	**−1.0316**
	Average	**−1.0316**	−1.0302	**−1.0316**
F_{17}	Best	**0.3980**	0.3979	**0.3980**
	Median	**0.3995**	0.3997	0.3994
	Average	**0.3999**	0.4007	0.4001
F_{18}	Best	**3.0000**	3.0014	**3.0000**
	Median	**3.0000**	3.0169	**3.0000**
	Average	**3.0000**	3.0235	**3.0000**
F_{19}	Best	**−3.8628**	−3.8623	**−3.8628**
	Median	−3.8596	−3.8593	**−3.8628**
	Average	−3.8593	−3.8590	**−3.8624**
F_{20}	Best	**−3.3220**	−3.2779	**−3.3220**
	Median	**−3.3220**	−3.0968	**−3.3220**
	Average	**−3.3220**	−3.1105	**−3.3220**
F_{21}	Best	**−10.1532**	−8.8466	**−10.1532**
	Median	−2.9417	−4.2211	**−10.1531**
	Average	−5.4547	−4.6898	**−7.3326**
F_{22}	Best	**−10.4029**	−9.3378	**−10.4029**
	Median	**−10.4029**	−5.1803	**−10.4029**
	Average	**−10.4029**	−5.4778	**−10.4029**
F_{23}	Best	**−10.5364**	−8.7847	**−10.5364**
	Median	**−10.5364**	−5.1446	**−10.5364**
	Average	−10.2659	−5.4009	**−10.5364**

4.2 Solving the Constrained Optimization Problems

In this Section, we discuss the use of CS–GSA algorithm for solving the constrained optimization problems. Here, we consider two different constrained optimization issues. These examples are very interesting and may create interest among the readers to explore the idea further.

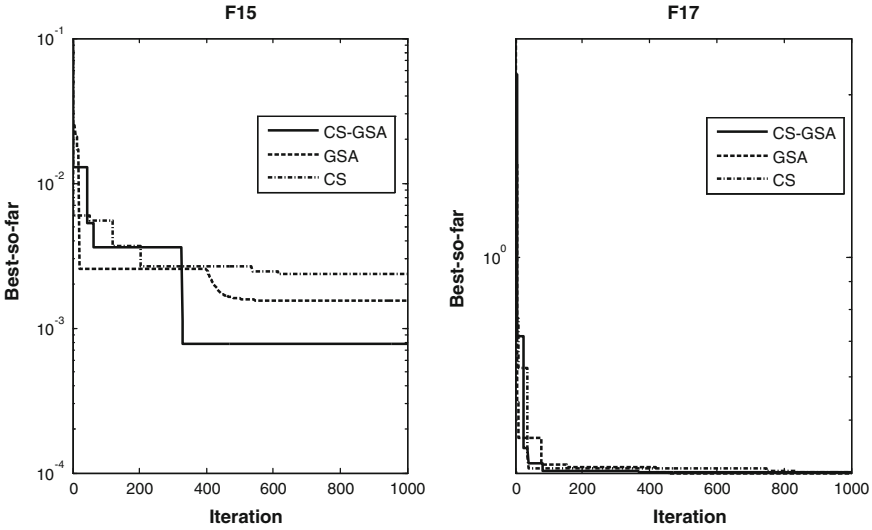

Fig. 6 Performance comparison of CS–GSA, CS, and GSA for the multimodal functions F_{15} and F_{17} with fixed dimension

4.2.1 Minimizing the Function

Here, we present an application of the proposed CS–GSA algorithm for function minimization. This is a constrained optimization problem. We like to minimize the function given in Eq. (21) [39]

$$f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \tag{21}$$

subject to the following constraints [39]:

$$\begin{aligned} g_1(x) &= 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0 \\ g_2(x) &= 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0 \\ g_3(x) &= 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0 \\ g_4(x) &= -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0 \\ & -10 \leq x_i \leq 10, \quad i = 1, 2, 3, 4, 5, 6, 7. \end{aligned}$$

For the evaluation of constrained problem described in this section, we have taken parameter for various algorithms given in Table 4. From Table 10, it is seen that the proposed CS–GSA scheme is well suited for this constrained optimization problem. The attribute values obtained by CS–GSA (marked as bold face letters)

Table 10 Comparison of the best solutions given by GSA, CS, and CS–GSA with the optimal solution for the constrained problem

EV	GSA	CS	CS–GSA	Optimal
x_1	1.619429	−0.658157	**2.169951**	2.330499
x_2	2.343357	1.408300	**2.041937**	1.951372
x_3	0.965166	−1.128276	**−0.936082**	−0.477541
x_4	2.737996	5.261455	**4.052961**	4.365726
x_5	−0.119048	0.671794	**−0.542999**	−0.624487
x_6	0.406691	0.824787	**0.594817**	1.038131
x_7	0.854488	1.690929	**1.462593**	1.594227
g_1(x)	0.934544	1.370761	**3.37333058**	4.464147e-05
g_2(x)	2.514614e+02	2.650624e+02	**2.473260e+02**	2.525617e+02
g_3(x)	1.591053e+02	2.186001e+02	**1.514995e+02**	1.448781e+02
g_4(x)	0.905996	5.433623	**1.650391**	7.632134e-06
f(x)	731.535628	761.432542	**688.856815**	680.630111

'EV' stands for the estimated value

Table 11 Comparisons of the statistical results of GSA, CS, and CS–GSA for the constrained problem

Algorithm	Best	Median	Average
GSA	731.535628	736.213432	742.234546
CS	761.432542	776.324511	780.753411
CS–GSA	**688.856815**	**696.062212**	**699.768549**

seem to be very close to the optimal values, presented in the table for a ready reference. From Table 11, it is seen that the 'Best,' the 'Median,' and the 'Average' values obtained by CS–GSA (marked as bold face letters) seem to be better than the other two algorithms.

4.2.2 One-Dimensional (1-D) Recursive Filter Design

Newly, an increasing interest is seen in the application of the EC algorithms for solving the problems of traditional filter design methods. There is a merit of not necessitating virtuous first estimate of the filter coefficients. Here, we present the design of 1-D recursive filters using GSA, CS, and CS–GSA algorithms. Note that the design of the IIR digital filter is well-thought-out here as a constrained optimization tricky. Inbuilt constraint handling is found to guarantee stability. Here, the best results are obtained through the convergence of the proposed CS–GSA method in order to confirm the quality. Interestingly, a faster result is achieved through the convergence of a meta-heuristic hybrid algorithm coined as CS–GSA algorithm. To be precise, the proposed constraint management competence makes the proposed method very eye-catching in the design of 1-D IIR digital filters. Results are compared to GSA and CS techniques.

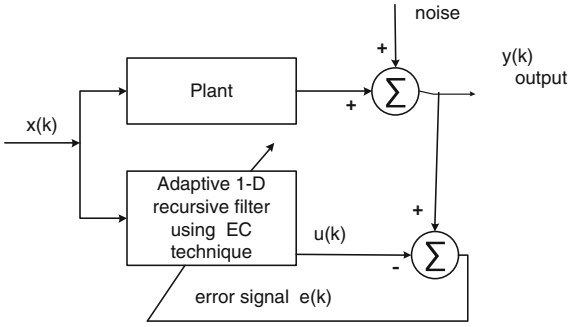

Fig. 7 One-dimensional (1-D) recursive filter optimization using EC methods

The block diagram showing IIR filter optimization is displayed in Fig. 7. Evolutionary computational technique is very useful for synthesis of digital IIR filters. The filter coefficients are chosen very accurately using evolutionary algorithms. To reduce the error, the filter coefficients are optimized. Then it is used for different applications. For this reason, EAs proved to be beneficial for the design of 1-D digital IIR filters. In this section, we discuss the design of 1-D IIR digital filter design by three fairly new EC methods.

The GSA was used for the design of 1-D recursive filters [27] and IIR filter design [40]. However, they are silent regarding the constraint treatment and diminishing tool, which is required to guarantee stability of the 1-D recursive filters. The design was not considered as the constrained optimization problem. Recently, the authors in [40] considered this as a constrained optimization work and then resolved this in designing the 1-D digital IIR filters. In this section, three optimization algorithms GSA, CS, and CS–GSA are deployed to design 1-D IIR filters.

Note that the system is called as recursive provided the preset output depends on the present input, the past input, and the past output of the system. So a 1-D system can be represented as

$$y(m) = \{x(m), x(m-1), \ldots, x(m-M), y(m-1), \ldots, y(m-M)\}. \tag{22}$$

For the evaluation of our proposed algorithm, let us consider a 1-D recursive filter transfer function. The transfer function can be represented as

$$H(z) = H_0 \frac{\sum_{i=0}^{S} a_i z^i}{\sum_{i=0}^{S} b_i z^i},\ a_0 = 1, \text{ and } b_0 = 1. \tag{23}$$

The stability conditions are described as

$$H(z) = \frac{A(z)}{B(z)}, \text{ with } B(z) \neq 0, \text{ and } |z| \geq 1. \tag{24}$$

For $S = 2$, one can write

$$H(z) = H_0 \frac{1 + a_1 z + a_2 z^2}{1 + b_1 z + b_2 z^2}. \tag{25}$$

In this section, the objective is to optimize the vector (X), where $X = [a_1, a_2, b_1, b_2, H_0]$, subject to the constraints given below:

$$(1 + b_i) > 0, \text{ and } (1 - b_i) > 0. \tag{26}$$

Hence, this problem is known as a constrained optimization problem. Here, an attempt is made to solve this problem using three different EC algorithms GSA, CS, and CS–GSA. To evaluate the proposed 1-D recursive function, let us consider M_d as the desired magnitude response of the digital IIR filter expressed as a function of frequency $\omega \in [0, \pi]$:

$$M_d(\omega) = \begin{cases} 1 & \text{if } \omega \leq 0.05\pi \\ \frac{1}{\sqrt{2}} & \text{if } 0.05\pi < \omega \leq 0.08\pi \\ \frac{1}{2\sqrt{2}} & \text{if } 0.08\pi < \varpi \leq 0.1\pi \\ 0 & \text{otherwise} \end{cases}. \tag{27}$$

Here, our objective is to find $H(z)$, such that it will closely approximate the desired magnitude response. The approximation can be attained by a fitness function J such as

$$J = \sum_{n=0}^{P} [|M(\omega)| - M_d(\omega)]^2, \tag{28}$$

where $M(\omega)$ is the Fourier transform of the $H(z)$, i.e., $M(\omega) = H(Z)|_{z=e^{-j\omega}}$ and $\omega = (\pi/P)n$.

The results (the filter parameters) are shown in Table 12.

Table 12 Comparisons of the best solution given by GSA, CS, and CS–GSA for the 1-D recursive filter design. "EP" refers to the estimated parameters

EP	GSA	CS	CS–GSA
a_1	0.732505	0.746526	**0.883583**
a_2	0.580788	1.321402	**0.741019**
b_1	−0.852237	−0.928369	**−0.982292**
b_2	−0.274744	−0.197190	**−0.134745**
H_0	0.061419	0.041052	**0.050245**
$1 + b_1$	0.147762	0.071630	**0.017707**
$1 - b_1$	1.852237	1.928369	**1.982292**
$1 + b_2$	0.725255	0.802809	**0.865254**
$1 - b_2$	1.274744	1.197190	**1.134745**
J	0.596505	0.617075	**0.578251**

Table 13 Comparisons of statistical results of GSA, CS, and CS–GSA for 1-D recursive filter design

Algorithm	Best	Median	Average
GSA	0.596505	0.796212	0.857275
CS	0.617075	0.776405	0.785817
CS–GSA	**0.578251**	**0.669519**	**0.673789**

From Table 12, it is observed that the filter parameter obtained by CS–GSA algorithm is better than CS and GSA algorithms. They are optimized using CS–GSA algorithm to reduce the error. A comparison of the statistical results of GSA, CS, and CS–GSA for 1-D recursive filter design is presented in Table 13.

From Table 13, it is seen that the 'Best,' 'Median,' and 'Average' values of the filter parameters for 50 independent runs obtained by CS–GSA algorithm are better than CS and GSA algorithms. These statistical parameters are obtained using CS–GSA algorithm for a better performance.

Figure 8 displays the frequency responses of the 1-D recursive filter designed using CS–GSA, GSA, and CS as the three competitive methods. The above

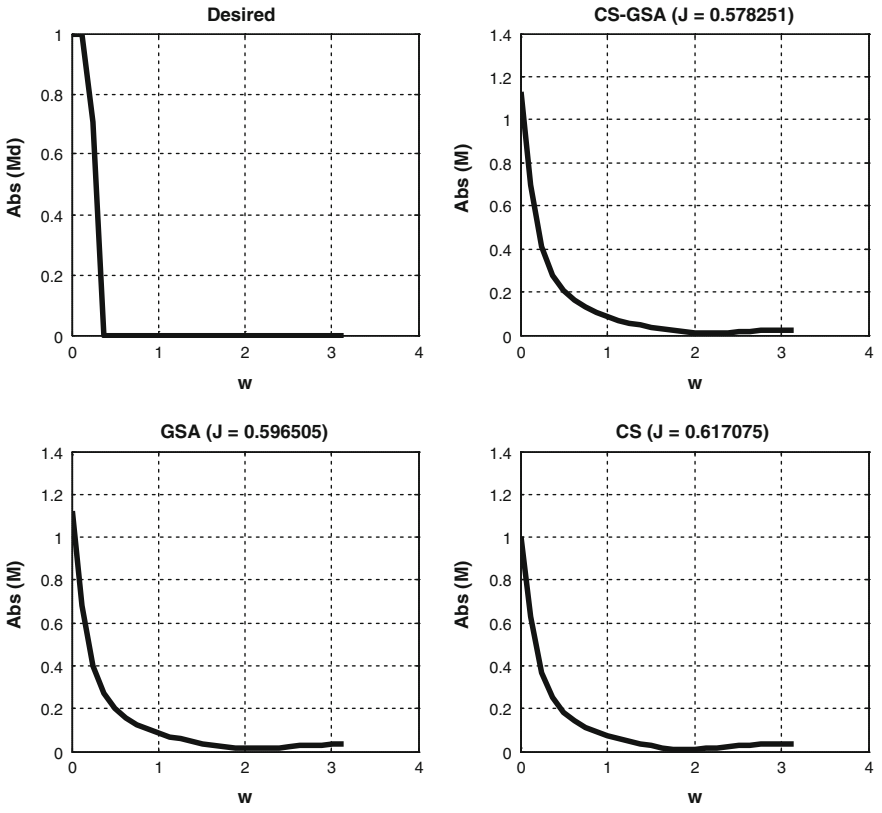

Fig. 8 Amplitude responses of 1-D recursive filter of desired and using CS–GSA, GSA, and CS

amplitude responses are plotted exhausting the solutions achieved by minimizing J for 50,000 function evaluations, and the parameter is given in Table 4. Note that in the cited numerical example, the solution attained by utilizing CS–GSA method offers us a better approximation of the proposed transfer function than latter methods. It seems closer to that of the desired frequency response. The GSA method is the second contestant in this work. The performance of the CS–GSA is quite better than the latter methods. From Fig. 8, we see that the frequency response attained by utilizing CS–GSA method is better than the frequency responses exhibited by other two algorithms CS and GSA.

5 Conclusions

In this chapter, the proposed hybrid algorithm CS–GSA outperforms both CS and GSA algorithms in terms of obtaining best solutions. In fact, the GSA is used to explore the local search ability, whereas CS is used to speed up the convergence rate. The convergence speed of the proposed hybrid algorithm is faster than CS and GSA algorithms. Interestingly, CS simulates the social behavior of Cuckoo birds, while GSA inspires by a physical phenomenon. This proposal can easily be extended to develop multi-objective optimization applications by considering different inbuilt constraints. Finally, it may be noted that the better convergence of CS algorithm and local search ability of the GSA produce good results that are beneficial.

Appendix: Benchmark Functions

a. Sphere Model

$$F_1(\mathrm{X}) = \sum_{i=1}^{n} x_i^2,\ -95 \le x_i \le 95,\ \text{ and } \min(F_1) = F_1(0,\ldots,0) = 0$$

b. Schwefel's Problem 2.22 [21, 42]

$$F_2(\mathrm{X}) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|,\ -12 \le x_i \le 12,\ \text{ and } \min(F_2) = F_2(0,\ldots,0) = 0$$

c. Schwefel's Problem 1.2

$$F_3(\mathrm{X}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2,\ -90 \le x_i \le 90,\ \text{ and } \min(F_3) = F_3(0,\ldots,0) = 0$$

d. Schwefel's Problem 2.21

$$F_4(\mathrm{X}) = \max_i\{|x_i|, 1 \le i \le n\},\ -90 \le x_i \le 90,\ \text{ and } \min(F_4) = F_4(0,\ldots,0) = 0$$

e. Generalized Rosenbrock's Function

$$F_5(\mathrm{X}) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right],\ -30 \le x_i \le 30$$
$$\min(F_5) = F_5(0,\ldots,0) = 0.$$

f. Step Function

$$F_6(\mathrm{X}) = \sum_{i=1}^{n}(\lfloor x_i + 0.5\rfloor)^2,\ -100 \le x_i \le 100,\ \text{ and } \min(F_6) = F_6(0,\ldots,0) = 0.$$

g. Quartic Function i.e. Noise

$$F_7(\mathrm{X}) = \sum_{i=1}^{n} i x_i^4 + \text{random}[0,1),\ -1.28 \le x_i \le 1.28$$
$$\min(\mathrm{F}_7) = F_7(0,\ldots,0) = 0$$

h. Generalized Rastrigin's Function

$$F_8(\mathrm{X}) = \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i) + 10\right],\ -5.12 \le x_i \le 5.12$$
$$\min(F_8) = F_8(0,\ldots,0) = 0.$$

i. Ackley's Function

$$F_9(\mathrm{X}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$
$$-32 \le x_i \le 32,\ \text{ and } \min(F_9) = F(0,\ldots,0) = 0.$$

j. Generalized Griewank Function

$$F_{10}(\mathrm{X}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1,\ -600 \le x_i \le 600$$
$$\min(F_{10}) = F_{10}(0,\ldots,0) = 0.$$

k. Generalized Penalized Function 1

$$F_{11}(\mathrm{X}) = \frac{\pi}{n}\left\{10\sin(\pi y_i) + \sum_{i=1}^{n}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4),$$

where

$$u(x_i,a,k,m)=\begin{cases} k(x_i-a)^m,\ x_i>a \\ 0,\ -a<x_i<a, \\ k(-x_i-a)^m,\ x_i<-a \end{cases} \quad \text{and } y_i=1+\frac{1}{4}(x_i+1)$$

$$-50\le x_i\le 50,\ \text{ and } \min(F_{11})=F_{11}(1,\ldots,1)=0.$$

l. Generalized Penalized Function 2

$$F_{12}(\mathrm{X})=0.1\Big\{\sin^2(3\pi x_1)+\sum_{i=1}^{n}(x_i-1)^2\left[1+\sin^2(3\pi x_i+1)\right]+(x_n-1)^2\cdot \left[1+\sin^2(2\pi x_n)\right]\Big\}+\sum_{i=1}^{n}u(x_i,5,100,4),$$

where

$$u(x_i,a,k,m)=\begin{cases} k(x_i-a)^m,\ x_i>a \\ 0,\ -a<x_i<a \\ k(-x_i-a)^m,\ x_i<-a \end{cases}, \text{ and } y_i=1+\frac{1}{4}(x_i+1)$$

$$-50\le x_i\le 50,\ \text{ and } \min(F_{12})=F_{12}(1,\ldots,1)=0.$$

m. Generalized Schwefel's Problem 2.26

$$F_{13}(\mathrm{X})=\sum_{i=1}^{30}-x_i\sin\left(\sqrt{|x_i|}\right),\ -500\le x_i\le 500$$

$$\min(F_{13})=F_{13}(420.9687,\ldots,420.9687)=-12569.5.$$

n. Shekel's Foxholes Function

$$F_{14}(\mathrm{X})=\left(\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}\left(x_i-a_{ij}\right)^6}\right)^{-1},\ -65.536\le x_i\le 65.536$$

$$\min(F_{14})=F_{14}(-32,-32)\approx 1,$$

where

$$a_{ij}=\begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}.$$

o. Kowalik's Function

$$F_{15}(\mathrm{X})=\sum_{i=1}^{11}\left[a_i-\frac{x_1\left(b_i^2+b_ix_2\right)}{b_i^2+b_ix_3+x_4}\right]^2,\ -5\le x_i\le 5$$

$\min(F_{15}) \approx F_{15}(0.1928, 0.1908, 0.1231, 0.1358) \approx 0.0003075$. The coefficients are displayed in Table 14 [21, 41, 42].

p. Six-Hump Camel-Back Function

$$F_{16}(\mathrm{X}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4,\ -5 \le x_i \le 5$$

$$X_{\min} = (0.08983, -0.7126), (-0.08983, 0.7126)$$
$$\min(F_{16}) = -1.0316285.$$

q. Branin Function

$$F_{17}(\mathrm{X}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right) + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$
$$-5 \le x_1 \le 10,\ 0 \le x_2 \le 15$$

$$X_{\min} = (-3.142, 12.275), (3.142, 2.275), (9.425, 2.425)$$
$$\min(F_{17}) = 0.398.$$

r. Goldstein-Price Function

$$F_{18}(\mathrm{X}) = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2\right)\right] \times \left[(2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right) + 30\right]$$
$$-2 \le x_i \le 2, \text{ and } \min(F_{18}) = F_{18}(0, -1) = 3.$$

Table 14 Kowalik's function F_{15}

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

s. Hartman's Family

$$F(\mathrm{X}) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{n} a_{\mathrm{ij}}\left(x_j - p_{\mathrm{ij}}\right)^2\right),\ 0 \le x_j \le 1,\ n = 3, 6$$

for $F_{19}(\mathrm{X})$ and $F_{20}(\mathrm{X})$, respectively. $X_{\min}$ of $F_{19} = (0.114, 0.556, 0.852)$, and $\min(F_{19}) = -3.86$. $X_{\min}$ of $F_{20} = (0.201, 0.150, 0.477, 0.275, 0.311, 0.657)$, and $\min(F_{20}) = -3.32$.

The coefficients are shown in Tables 15 and 16, respectively.

t. Shekel's

Family$F(X) = -\sum_{i=1}^{m}\left[(X - a_i)(X - a_i)^T + c_i\right]^{-1}$, $m = 5, 7,$ and 10, for F_{21}, $F_{22},$ and $F_{23} 0 \le x_j \le 10$, $x_{\text{local\_optima}} \approx a_i$, and $F\left(x_{\text{local\_optima}}\right) \approx 1/c_i$ for $1 \le i \le m$.

These functions have five, seven, and ten local minima for $F_{21}, F_{22},$ and F_{23}, respectively. The coefficients are shown in Table 17.

Table 15 Hartman function F_{19}

i	a_{i1}	a_{i2}	a_{i3}	c_i	p_{i1}	p_{i2}	p_{i3}
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	4	0.038150	0.5743	0.8828

Table 16 Hartman function F_{20}

i	a_{i1}	a_{i2}	a_{i3}	a_{i4}	a_{i5}	a_{i6}	c_i
1	10	3	17	3.5	1.7	8	1
2	0.05	10	17	0.1	8	14	1.2
3	3	3.5	1.7	10	17	8	3
4	17	8	0.05	10	0.1	14	3.2
i	p_{i1}	p_{i2}	p_{i3}	p_{i4}	p_{i5}	p_{i6}	
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886	
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991	
3	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650	
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381	

Table 17 Shekel function F_{21}, F_{22}, and F_{23}

i	a_{i1}	a_{i2}	a_{i3}	a_{i4}	c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

References

1. Du W, Li B (2008) Multi-strategy ensemble particle swarm optimization for dynamic optimization. Inf Sci 178:3096–3109
2. Panda R, Naik MK (2012) A crossover bacterial foraging optimization algorithm. Appl Comput Intell Soft Comput, 1–7. Hindawi Publication
3. Mastorakis NE, Gonos IF, Swamy MNS (2003) Design of two-dimensional recursive filters using genetic algorithm. IEEE Trans Circuits Syst-I Fundam Theory Appl 50:634–639
4. Panda R, Naik MK (2013) Design of two-dimensional recursive filters using bacterial foraging optimization. In: Proceedings of the 2013 IEEE Symposium on Swarm Intelligence (SIS), pp 188–193
5. Cordon O, Damas S, Santamari J (2006) A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm. Pattern Recogn Lett 26:1191–1200
6. Panda R, Agrawal S, Bhuyan S (2013) Edge magnitude based multilevel thresholding using cuckoo search technique. Expert Syst Appl 40:7617–7628
7. Panda R, Naik MK, Panigrahi BK (2011) Face recognition using bacterial foraging strategy. Swarm Evol Comput 1:138–146
8. Liu C, Wechsler H (2000) Evolutionary pursuit and its application to face recognition. IEEE Trans Pattern Anal Mach Intell 22:570–582
9. Zheng WS, Lai JH, Yuen PC (2005) GA-Fisher: a new LDA-based face recognition algorithm with selection of principal components. IEEE Trans Syst Man Cybern Part B 35:1065–1078
10. Mitchell M (1998) An introduction to genetic algorithms. MIT Press, Cambridge
11. Dorigo M, Maniezzo V, Colorni A (1996) The ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B 26:29–41
12. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the IEEE international conference on neural networks vol 4, pp 1942–1948
13. Gazi V, Passino KM (2004) Stability analysis of social foraging swarms. IEEE Trans Syst Man Cybern Part B 34:539–557
14. Yang XS, Deb S (2009) Cuckoo search via Lévy flights. In: Proceedings of the world congress on nature and biologically inspired computing, (NaBIC 2009), pp 210–214
15. Yang XS, Deb S (2013) Cuckoo search: recent advances and applications. Neural Comput Appl 24(1):169–174
16. Cuckoo Search and Firefly Algorithm. http://link.springer.com/book/10.1007%2F978-3-319-02141-6

17. Pinar C, Erkan B (2011) A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artif Intell Rev, Springer. doi:10.1007/s10462-011-9276-0
18. Chakraverty S, Kumar A (2011) Design optimization for reliable embedded system using cuckoo search. In: Proceedings of the international conference on electronics, computer technology, pp 164–268
19. Barthelemy P, Bertolotti J, Wiersma DS (2008) A Lévy flight for light. Nature 453:495–498
20. Rashedi E, Nezamabadi S, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179:2232–2248
21. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3:82–102
22. Chetty S, Adewumi AO (2014) Comparison study of swarm intelligence techniques for annual crop planning problem. IEEE Trans Evol Comput 18:258–268
23. Chen J-F, Do QH (2014) Training neural networks to predict student academic performance: a comparison of cuckoo search and gravitational search algorithms. Int J Comput Intell Appl 13 (1):1450005
24. Swain KB, Solanki SS, Mahakula AK (2014) Bio inspired cuckoo search algorithm based neural network and its application to noise cancellation. In: Proceedings of the international conference on signal processing and integrated networks (SPIN), pp 632–635
25. Khodier M (2013) Optimisation of antenna arrays using the cuckoo search algorithm. IET Microwaves Antennas Propag 7(6):458–464
26. Zhao P, Li H (2012) Opposition based Cuckoo search algorithm for optimization problems. In: Proceedings of the 2012 fifth international symposium on computational intelligence and design, pp 344–347
27. Saha SK, Kar R, Mandal D, Ghosal SP (2013) Gravitational search algorithm: application to the optimal IIR filter design. Journal of King South University, 1–13
28. Rashedi E, Nezamabadi-pour H, Saryazdi S (2011) Filter modeling using gravitational search algorithm. Eng Appl Artif Intell 24:117–122
29. Rashedi E, Nezamabadi-pour H, Saryazdi S (2011) Disruption: A new operator in gravitational search algorithm. Sci Iranica D 18:539–548
30. Doraghinejad M, Nezamabadi-pour H, Sadeghian AH, Maghfoori M (2012) A hybrid algorithm based on gravitational search algorithm for unimodal optimization. In: Proceedings of the 2nd international conference on computer and knowledge engineering (ICCKE), pp 129–132
31. Yazdani S, Nezamabadi-pour H, Kamyab S (2013) A gravitational search algorithm for multimodal optimization. Swarm Evol Comput 1–14
32. Rashedi E, Nezamabadi-pour H, Saryazdi S (2009) BGSA: binary gravitational search algorithm. Nat Comput 9(3):727–745
33. Mirjalili S, Hashim SZM (2010) A new hybrid PSOGSA algorithm for function optimization. In: 2010 international conference on computer and information application, pp 374–377
34. Jiang S, Ji Z, Shen Y (2014) A novel hybrid particle swarm optimization and gravitational search algorithm for solving economic emission load dispatch problems with various practical constraints. Electr Power Energy Syst 55:628–644
35. Ghodrati A, Lotfi S (2012) A hybrid CS/PSO algorithm for global optimization. Lect Notes Comput Sci 7198:89–98
36. Guo Z (2012) A hybrid optimization algorithm based on artificial bee colony and gravitational search algorithm. Int J Digit Content Technol Appl 6(17):620–626
37. Sun G, Zhang A (2013) A hybrid genetic algorithm and gravitational using multilevel thresholding. Pattern Recognit Image Anal 7887:707–714
38. Yin M, Hu Y, Yang F, Li X, Gu W (2011) A novel hybrid K-harmonic means and gravitational search algorithm approach for clustering. Expert Syst Appl 38:9319–9324
39. Liu H, Cai Z, Wang Y (2010) Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl Soft Comput 10:629–640

40. Sarangi SK, Panda R, Dash M (2014) Design of 1-D and 2-D recursive filters using crossover bacterial foraging and cuckoo search techniques. Eng Appl Artif Intell 34:109–121
41. He J (2008) An experimental study on the self-adaption mechanism used by evolutionary programing. Prog Nat Sci 10:167–175
42. Ji M (2004) A single point mutation evolutionary programing. Inf Process Lett 90:293–299