Soft Computing (2021) 25:10167–10180
https://doi.org/10.1007/s00500-021-05939-3

OPTIMIZATION

Exploratory cuckoo search for solving single-objective optimization problems

Bilal H. Abed-alguni[1] · Noor Aldeen Alawad[1] · Malek Barhoush[1] · Rafat Hammad[2]

Accepted: 3 June 2021 / Published online: 12 June 2021

Abstract
The cuckoo search (CS) algorithm is an effective optimization algorithm, but it is prone to stagnation in suboptimality because of some limitations in its exploration mechanisms. This paper introduces a variation of CS called exploratory CS (ECS), which incorporates three modifications to the original CS algorithm to enhance its exploration capabilities. First, ECS uses a special type of opposition-based learning called refraction learning to improve the ability of CS to jump out of suboptimality. Second, ECS uses the Gaussian perturbation to optimize the worst candidate solutions in the population before the discard step in CS. Third, in addition to the Lévy flight mutation method used in CS, ECS employs two mutation methods, namely highly disruptive polynomial mutation and Jaya mutation, to generate new improved candidate solutions. A set of 14 widely used benchmark functions was used to evaluate and compare ECS to three variations of CS:CS with Lévy flight (CS), CS with highly disruptive polynomial mutation (CS10) and CS with pitch adjustment mutation (CS11). The overall experimental and statistical results indicate that ECS exhibits better performance than all of the tested CS variations. Besides, the single-objective IEEE CEC 2014 functions were used to evaluate and compare the performance of ECS to six well-known swarm optimization algorithms: CS with Lévy flight, Grey wolf optimizer (GWO), distributed Grey wolf optimizer (DGWO), distributed adaptive differential evolution with linear population size reduction evolution (L-SHADE), memory-based hybrid Dragonfly algorithm and Fireworks algorithm with differential mutation. Interestingly, the results indicate that ECS provides competitive performance compared to the tested six well-known swarm optimization algorithms.

Keywords Cuckoo search · Refraction learning · Single-objective optimization · Mutation method

1 Introduction

The cuckoo search (CS) algorithm is nature-inspired optimization algorithm Yang and Deb (2009) that has been adapted to solve different types of optimization problems in application fields such as image processing (Roy et al. 2017; Ye et al. 2015), engineering (Zhang et al. 2019; Shehab et al. 2019) and networking (Mohamad et al. 2014; Sonia and Patterh 2014). It has two main features that make it stands out against other optimization algorithms. First, it uses a mutation function based on Lévy flight (i.e., a special random walk with step lengths that have a heavy-tailed probability distribution) to improve the quality of randomly selected solutions at each iteration of CS. Second, it uses one parameter called abandon fraction p_a that does not require fine-tuning. The optimization strategies of CS, the uniform selection operators and the Lévy flight operator, may not be able to generate improved solutions over consecutive iterations (Abed-alguni and Alkhateeb 2017). This situation is called premature convergence, and it may cause CS to get strangled in local optimality.

Several improved versions of the CS algorithm have been proposed recently (see Sect. 3) to reduce the chances of CS being stuck in local optima. For example, many hybrid CS algorithms have been discussed in the literature (e.g., hybrid Krill herd and CS Wang et al. 2016, hybrid CS and β-hill climbing Abed-alguni and Alkhateeb 2018) in which CS is usually combined with one or more search algorithms (local-based or population-based algorithms) to improve the exploration mechanism of CS. However, the main problem with the hybrid CS approach is that it involves intensive com-

✉ Bilal H. Abed-alguni
Bilal.h@yu.edu.jo

1 Department of Computer Sciences, Yarmouk University, Irbid, Jordan

2 Department of Computer Information Systems, Yarmouk University, Irbid, Jordan

putations, which make it slower than the basic CS algorithm (Alkhateeb and Abed-Alguni 2017; Abed-alguni and Alkhateeb 2018).

In this paper, we introduce a variation of CS, called exploratory CS (ECS), that incorporates three modifications to the original CS algorithm in an attempt to enhance its exploration capabilities. First, ECS uses refraction learning (i.e., an opposition-based learning mechanism, which simulates the principle of light refraction in physics) to improve CS ability to jump out of suboptimality. Second, ECS uses the Gaussian perturbation in an attempt to enhance the worst candidate solutions in the population before the discard step (i.e., the final steps at each iteration of CS). Third, in addition to the Lévy flight mutation method, ECS employs two mutation methods, namely highly disruptive polynomial mutation and Jaya mutation, to generate new improved candidate solutions.

We carried out several experiments to investigate the performance of ECS using well-known benchmark suits and a variety of well-known optimization algorithms. First, we used 14 popular benchmark functions to examine the performance of ECS compared to three variations of CS: original CS, CS10 and CS11 (Abed-Alguni and Paul 2019). The overall experimental and statistical results indicate that ECS exhibits better performance than all of the tested CS variations. Second, we used 30 IEEE CEC 2014 functions to examine the performance of ECS compared to six popular optimization algorithms: CS, GWO (Mirjalili et al. 2014), DGWO (Abed-alguni and Barhoush 2018), L-SHADE (Tanabe and Fukunaga 2014), memory-based hybrid Dragonfly algorithm (MHDA) (Sree Ranjini and Murugan 2017) and Fireworks algorithm with differential mutation (FWA-DM) (Yu et al. 2014). The results clearly show that ECS provides a competitive performance compared to the tested well-known optimization algorithms.

The remainder of this paper is organized as follows: Section 2 introduces the preliminaries of the research (including CS, refraction learning, three mutation methods and Gaussian perturbation). Section 3 provides a discussion about well-known variations of CS. Section 4 discusses the ECS algorithm. Section 5 discusses the experimental results of ECS compared to popular optimization algorithms using popular test suits. Finally, Sect. 6 presents the conclusion of this paper.

2 Preliminaries

This section provides a summary of some of the underlying concepts of the paper. Section 2.1 describes the mathematical notations used in the paper, Sect. 2.2 describes the cuckoo search algorithm, Sect. 2.3 discusses the main concepts of refraction learning and finally Sect. 2.4 describes three mutation methods: highly disruptive polynomial mutation, Jaya mutation and Gaussian perturbation.

2.1 Notations

We used the following mathematical notations in the discussions of this paper:

- $\boldsymbol{x}^i$: the ith feasible solution (candidate solution), which includes m decision variables $\langle x_1^i, x_2^i, \ldots, x_m^i \rangle$.
- x_j^i: the value of the j^{th} decision variable of $\boldsymbol{x}^i$. The value of x_j^i is produced using a uniform random-based function as follows:

$$x_j^i = LB_j + (UB_j - LB_j) \times U(0, 1) \tag{1}$$

  where $U(0, 1)$ is a uniform random number in the interval $[0, 1]$, LB_j is the lower bound and UB_j is the upper bound of x_j.
- $f(\boldsymbol{x}^i)$: the fitness value of $\boldsymbol{x}^i$.
- The population of n candidate solutions is conventionally represented as $\boldsymbol{x}^i$ $(i = 1, 2, \ldots, n)$.

2.2 Cuckoo search algorithm

One of the most effective iterative population-based optimization algorithms is the cuckoo Search (CS) algorithm (Yang and Deb 2009). The optimization process of CS simulates the opportunistic reproduction habits of cuckoo birds and the Lévy flight behavior of birds.

CS initially generates a population of n random candidate solutions $\boldsymbol{x}^i$ $(i = 1, 2, \ldots, n)$. The values of the decision variables are generated according to Eq. 1. The maximum number of iterations ($MaxItr$) of the optimization loop in CS depends on the complexity of the problem in hand.

CS uses the population to produce a sequence of improving approximate populations for a specific complex problem, in which the new population is derived from the previous ones. At each iteration t of the optimization loop of CS, one candidate solution ($\boldsymbol{x}^i(t)$) is selected based on a random uniform function. The selected solution is then mutated using the Lévy-based mutation function as follows:

$$\boldsymbol{x}^i(t+1) = \boldsymbol{x}^i(t) + \beta \oplus \text{Lévy}(\lambda), \tag{2}$$

Here, $\beta > 0$ is the step size of mutation, the Lévy flight operator (Lévy(λ)) is a vector of m values drawn from Lévy distribution and $\oplus$ is an entry-wise product operation. The Lévy flight operator is a special type of the random walk method, where the step length of the Lévy flight operator is calculated as follows:

$$\text{Lévy} \sim u = S^{-\alpha}, \tag{3}$$

where S is the step size of the distribution and α is a uniform-random number in the range $(1, 3]$ related to the fractal dimension (i.e., a statistical index of the complexity of a given search space). The Lévy distribution function has two characteristics: it has an infinite variance and the value of D gets longer over the course of iterations (Yang and Deb 2009).

Each iteration of CS ends with the discard process, which aims to strengthen the diversity of the current solutions. In this process, CS selects a fraction (p_a) of the worst solutions in the population, and then, it tries to replace them with better solutions.

2.3 Refraction learning

The natural phenomenon of refraction is the bending of a light ray when it leaves a fast–medium (e.g., air) and enters a slow one (e.g., water). The light ray changes its velocity when it passes to the second medium and bends toward the normal to the boundary between two mediums.

Refraction learning is a special type of opposition-based learning that is based on the refraction principle in physics. It was used with GWO (Long et al. 2019) and the whale optimization algorithm (WOA) (Long et al. 2020). The experimental and statistical results in Long et al. (2019, 2020) indicate that it can improve the performance of optimization algorithms such as GWO and WOA. Figure 1 shows the refraction–learning process applied at iteration t to the global optima $\boldsymbol{x}^*$ that has a single decision variable. In the refraction learning algorithm, the opposite of optimal solution $\boldsymbol{x}^*$ is given as follows:

$$\boldsymbol{x}^{'*} = (\text{LB} + \text{UB})/2 + (\text{LB} + \text{UB})/(2k\eta) - \boldsymbol{x}^*/(k\eta), \tag{4}$$

where η represents the refraction index which is expressed as follows:

$$\eta = \frac{sin\,\theta_1}{sin\,\theta_2}, \tag{5}$$

where $sin\,\theta_1$ and $sin\,\theta_2$ (Fig. 1) are represented as follows:

$$\sin\theta_1 = ((\text{LB} + \text{UB})/2 - \boldsymbol{x}^*)/h, \tag{6}$$
$$\sin\theta_2 = (\boldsymbol{x}^{'*} - (\text{LB} + \text{UB})/2)/h^{'}, \tag{7}$$

where $\boldsymbol{x}$ is the incidence point (a candidate solution) and $\boldsymbol{x}^{'}$ is the refraction point (another candidate solution), LB and UB are the lower and upper boundaries of the decision variables, O is the center point between LB and UB, h is the distance between $\boldsymbol{x}$ and O and h' is the distance between $\boldsymbol{x}^{'}$ and O. $\boldsymbol{x}^{'*}$ is the opposite solution of $\boldsymbol{x}^*$ based on refraction learning (i.e., is the projection of $\boldsymbol{x}^{'}$ on the x-axis)

Interestingly, Eq. 4 can be simply generalized to manipulate n decision variables as follows:

$$x_j^{'*} = (\text{LB}_j + \text{UB}_j)/2 + (\text{LB}_j + \text{UB}_j)/(2k\eta) - x_j^*/(k\eta), \tag{8}$$

where x_j^* and $x_j^{'*}$ are the jth decision variable of $\boldsymbol{x}^*$ and $\boldsymbol{x}^{'*}$, respectively, $[\text{LB}_j, \text{UB}_j]$ is the search interval of the jth decision variable.

2.4 Mutation methods

2.4.1 Highly disruptive polynomial mutation

The highly disruptive polynomial (HDP) mutation Deb and Tiwari (2008) is a well-recognized mutation method (Abed-alguni 2019; Abed-Alguni and Paul 2019). It provides efficient exploration of the search range of each decision variable in a candidate solution regardless of the current value of the decision variable.

The HDP method is used to optimize a candidate solution $\boldsymbol{x}^j(t)$ at iteration t by attempting to alter each decision variable in $\boldsymbol{x}^j(t)$ with probability $P_m \in [0, 1]$ as follows (Doush et al. 2014):

$$x_i^j(t+1) \leftarrow x_i^j(t) + \delta_k.(UB_i - LB_i), \quad \text{where} \tag{9}$$

$$\delta_k = \begin{cases} (2r) + (1-2r) \times (1-\delta_1)^{\eta_m+1}]^{\frac{1}{\eta_m+1}-1} & r \le P_m \\ 1 - [2(1-r) + 2(r-0.5) \times (1-\delta_2)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} & r > P_m \end{cases} \tag{10}$$

$$\delta_1 \leftarrow \frac{x_i^j(t) - LB_i}{UB_i - LB_i} \tag{11}$$

$$\delta_2 \leftarrow \frac{UB_i - x_i^j(t)}{UB_i - LB_i} \tag{12}$$

The following are definitions of the parameters of HDP mutation:

- r: a random number generated within the interval [0, 1]
- P_m: the mutation probability
- LB_i: the lower boundary of $x_i^j(t)$
- UB_i: the upper boundary of $x_i^j(t)$
- δ_1: the distance between $x_i^j(t)$ and LB_i divided by the distance between UB_i and LB_i
- δ_2: the distance between UB_i and $x_i^j(t)$ divided by the distance between UB_i and LB_i
- η_m: a nonnegative number that represents the distribution index

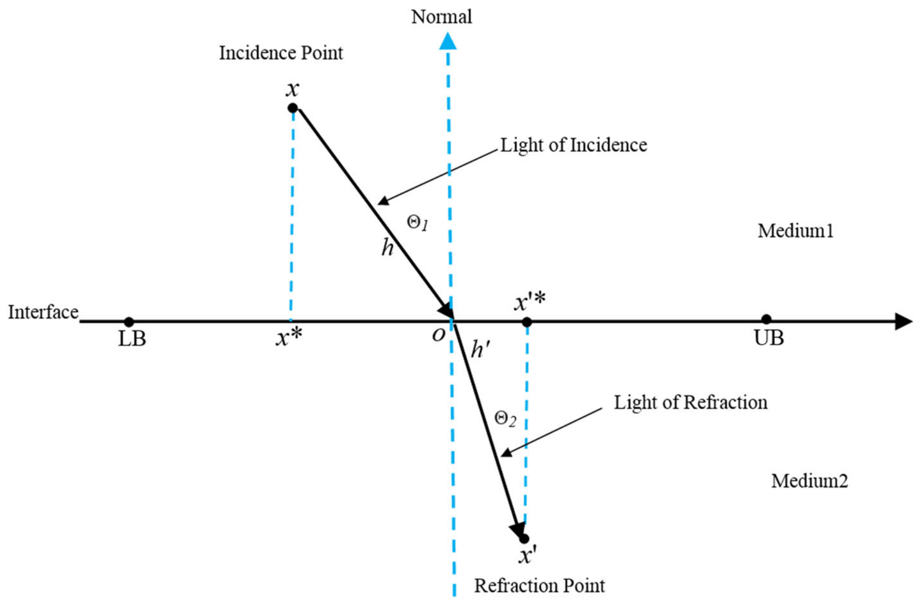

Fig. 1 Refraction learning for the global optimal x^*

2.4.2 Jaya mutation

The Jaya algorithm (Rao 2016) is an iterative population-based optimization algorithm. It uses the Jaya mutation method to mutate all the candidate solutions in the population of solutions by moving them between the best and worst candidate solutions at each iteration of Jaya. The decision variable $x^i_j(t)$ of candidate solution $\boldsymbol{x}^i(t)$ at iteration t is mutated as follows:

$$x^i_j(t+1) = x^i_j(t) + r_1\, x^B_j(t) + r_2\, x^W_j(t), \tag{13}$$

where $x^B_j(t)$ and $x^W_j(t)$ are the values of the jth decision variables of the best and worst candidate solutions, respectively, at iteration t. The parameters r_1 and r_2 are two uniformly distributed random numbers in the range [0,1]. After updating the population, if the fitness value of $\boldsymbol{x}^i(t+1)$ is better than the fitness of $\boldsymbol{x}^i(t)$, the solution $\boldsymbol{x}^i(t+1)$ is accepted to replace $\boldsymbol{x}^i(t)$.

2.4.3 Gaussian perturbation

The Gaussian perturbation is a mutation method derived from the Gaussian distribution (Feng et al. 2018). It is commonly used to improve the diversity of a population of candidate solutions to help it to escape local optimality. For example, it is possible to use the Gaussian perturbation to optimize the worst candidate solutions in a population. The one-dimensional Gaussian density function is expressed as follows:

$$\text{Gaussian}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{14}$$

where $\mu = 1.0$ and $\sigma = 0.5$.

Then Gaussian perturbation function is as follows:

$$x^i_j(t+1) = r\,\text{Gaussian}(x^i_j(t)), \tag{15}$$

where x^i_j is the jth decision variable of the ith candidate solution $\boldsymbol{x}^i$ and r is a randomly generated real number between (0,1).

3 Related work

This section provides a discussion about recently proposed research studies that focus on improving the exploration capabilities or convergence rate to optimality for the CS algorithm.

Abed-alguni (2019) proposed the *i*CSPM algorithm, which is an efficient modified CS algorithm that is built on the island model and HDP mutation. The island model is a parallel model that is used to parallelize the execution of an optimization algorithm and at the same time improve the diversity of its population (Lardeux and Goëffon 2010; Abed-alguni 2019). Besides, the HDP mutation is a well-recognized mutation method (Sect. 2.4.1). In *i*CSPM, the population (i.e., a group of candidate solutions) of an optimization problem is organized into subpopulations (islands) to allow the candidate solutions in each island to evolve using the original CS algorithm for a predefined period. However, the islands are permitted to connect and share solutions after a certain number of iterations. According to Abed-alguni (2019), *i*CSPM shows better performance than the basic CS algorithm and island-based genetic algorithm (Lardeux and Goëffon 2010). However, it is better to execute *i*CSPM algorithm across parallel machines to reduce its computational complexity. This is because its computational complexity is higher than the original CS algorithm (Abed-alguni and Barhoush 2018; Abed-alguni et al. 2019). Another problem of *i*CSPM is that the reliability of the island model is strongly dependent on the initial configuration of its parameters (Abed-alguni and Barhoush 2018; Abed-alguni et al. 2019; Abed-alguni 2019).

Alkhateeb and Abed-Alguni (2017) proposed four approaches to integrate CS and simulated annealing (SA) algorithms in one hybrid algorithm. The common factor among these approaches is that they all use the exploration mechanism of the SA algorithm. In this mechanism, the solution space is explored by replacing some current solutions by worse solutions with a decreasing probability. However, the proposed algorithms require higher computational time than the computational times of CS and SA algorithms.

Wang et al. (2016) suggested the integration of the krill herd (KH) and CS algorithms in one algorithm by the name the krill-herd cuckoo-search (KHCS) algorithm. This algorithm is basically used to solve the continuous optimization problems. It utilizes the optimization operators of CS in the optimization loop of KH. These operators aim to improve the exploitation ability of the original KH algorithm. It is important to note that the KHCS algorithm does not perform much better compared to the CS and KH algorithms (Wang et al. 2016). Furthermore, fine-tuning of the parameters of KH for each class of a given optimization problem is important and is a key factor in the success of KHCS.

CSARL (Shehab et al. 2019) is a variation of CS that is based on the basics of reinforcement learning (i.e., a machine learning approach that is based on trial and error Abed-Alguni et al. 2016). The use of reinforcement learning in CSARL aims to improve the exploration mechanism of CS, but it incurs a heavy computation cost compared to standard optimization algorithms. This means that the average length of an iteration in CSARL is longer than the one of that standard CS algorithm. According to Shehab et al. (2019), CSARL was only compared to standard optimization algorithms (standard genetic, harmony search (HS) and krill-heard algorithms) using standard benchmark functions, which is insufficient to evaluate the performance of new proposed algorithms.

Li et al. (2019) introduced TOB-DCS, which is a CS algorithm that uses the Taguchi opposition-based learning and dynamic evaluation. These methods are employed to explore different regions in the problem space and produce possibly better solutions from the current population. The Taguchi search is basically a random opposition-based learning approach. The dynamic evaluation strategy has the potential to reduce the number of times a function is evaluated by merging the updated decision variable with the other decision variables to make a new candidate solution. In Li et al. (2019), TOB-DCS was compared to basic optimization algorithms using basic test functions, which is considered insufficient to reliably examine the performance behavior of TOB-DCS.

Several variations of the ECSEE algorithm (i.e., enhanced CS with enhanced exploration and exploitation properties) have been proposed in Salgotra et al. (2018). All of the variations of ECSEE use the Cauchy operator to determine the step size of mutation instead of the Lévy flights, and they also use a local search method to improve the exploration of CS. Besides, the population in ECSEE may be divided into n subpopulations by following a process called division of the population or divided into two subpopulations through a process called the division of generations. The goal of the division process is to increase the diversity of population in ECSEE by allowing unfitted solutions in each subpopulation to develop in isolation away from other subpopulations. The ECSEE has been compared to standard optimization algorithms (Grey wolf optimization, differential evolution, firefly algorithm, flower pollination algorithm and bat algorithm (BA)) using 25 standard test functions and IEEE CEC 2015 benchmark suite. However, according to Salgotra et al. (2018) ECSEE did not perform well with IEEE CEC 2015 benchmark suite, which may be because of the stochastic nature of ECSEE.

The β-hill climbing (BHC) algorithm is a single-solution optimization algorithm that employs an operand called the β-operand in the hill climbing (HC) algorithm for more efficient exploration of the potential solutions of a given minimization algorithm. The BHC algorithm has been recently integrated inside the CS algorithm in Abed-alguni and Alkhateeb (2018). The new algorithm was called CSBHC. Unlike most well-known hybrid optimization algorithms, CSBHC uses the BHC algorithm inside the improvement loop of CS based on a decreasing probability. This way of integration reduces the computational cost that is normally associated with most hybrid optimization algorithms. However, the performance of CSBHC degrades with an increase in the dimensions of complex optimization function (Abed-alguni and Alkhateeb 2018). Furthermore, the performance of CSBHC was only evaluated using standard benchmark functions. This means that more experiments are required to assess the performance of CSBHC.

Ali and Tawhid (2016) proposed a hybrid optimization algorithm called HCSNM. It combines the CS algorithm with the Nelder–Mead method. The Nelder–Mead approach is a nonlinear optimization algorithm that is derivative-free and built on the idea of a simplex. The HCSNM algorithm calls the Nelder–Mead method periodically (i.e., each few number of iterations of CS) to refine the best candidate solution. However, the HCSNM algorithm does not specify a formula that calculates the period (i.e., the number of iterations) that should pass before calling the Nelder–Mead method.

Besides, the authors of the algorithm did not conduct experiments that study the sensitivity of HCSNM to its parameters (e.g., the period before calling the Nelder–Mead method).

Chaotic maps (i.e., chaotic functions that exhibit some chaotic behaviors El-Shorbagy et al. (2016)) are commonly utilized in the optimization field to explore the possibility of improving the convergence speed of optimization algorithms (Gandomi and Yang 2014). Wang and Deb (2016) have recently incorporated twelve chaotic maps (e.g., Chebyshev map, circle map, Liebovich map, iterative map, Tent map, sinusoidal map) with the CS algorithm in a new algorithm called chaotic CS (CCS). In CCS, the maps are employed to generate the step size β for the Lévy mutation function (Eq. 2). This is because the ergodicity and mixing properties of chaos maps may speed up the convergence speed of CS (Gandomi and Yang 2014). A problem that occurs when using chaotic maps with an optimization algorithm is that choosing a suitable chaos map for a given optimization problem requires extensive simulations and analysis (Fister et al. 2015).

Chi et al. (2019) proposed a hybrid optimization algorithm called CSPSO that incorporates the particle optimization algorithm (PSO) into the CS algorithm. In CSPSO, the Lévy mutation function (Eq. 2) of CS and the update equation of PSO are used together to increase the diversity of the population and improve the convergence rate to optimality. The CSPSO algorithm was compared in Chi et al. (2019) to basic optimization algorithms (PSO, CS, BA, HS) using standard benchmark functions which is still insufficient to evaluate the performance of CSPSO.

Cheng et al. (2019) introduced a new CS algorithm named the ensemble cuckoo search variant (ECSV). At each iteration of ECSV, three variations of CS (Chaos-enhanced cuckoo search Huang et al. 2016, nearest-neighbor cuckoo search Wang et al. 2016, peer-learning cuckoo search Yang et al. 2017) are used synchronously to evolve a population of candidate solutions to better solutions. ECSV assigns at the end of each iteration distinct candidate solutions to each CS algorithm based on its performance. ECSV was evaluated using 42 benchmark functions from CEC 2005 and CEC 2013. The simulation results indicate that ECSV provides competitive performance compared to CS, chaos-enhanced cuckoo search, nearest-neighbor cuckoo search and peer-learning cuckoo search.

It is a known fact that the performance of most of the optimization algorithms degrades with an increase in the number of dimensions (decision variables of a candidate solutions) (Abed-alguni and Barhoush 2018; Abed-alguni 2019). The DDECS (short for dimension-by-dimension enhanced CS) algorithm is a recently proposed variation of CS that makes the update strategy of CS more suitable to solve high-dimensionality optimization problems (Chen et al. 2019). DDECS merges each updated decision variable with the other decision variables to create a new candidate solution. The new solution is only accepted, if it enhances the current solution. According to Chen et al. (2019), DDECS was found to be a better optimization algorithm than CS, DECS (CS based on differential evolution Xiao and Duan 2014) and DDICS (CS with dimension by dimension improvement Wang et al. 2013). However, the experiments in Chen et al. (2019) were conducted using 18 basic benchmark functions, and no CEC benchmark suits have been used.

Rakhshani and Rahati (2017) introduced SDCS (snap-drift cuckoo search) that has two execution modes: snap and drift modes. SDCS was designed to address the performance issues of CS. In the snap mode, global search is conducted more than local search, while more local search is conducted in the drift mode. The SDCS algorithm was compared in Rakhshani and Rahati (2017) to basic optimization algorithms (e.g., CS, MCS Walton et al. 2011) using basic benchmark functions.

4 Proposed algorithm: exploratory cuckoo search

4.1 Algorithmic details of exploratory cuckoo search

In this section, we describe the exploratory cuckoo search (ECS) algorithm. The ECS algorithm attempts to increase the probability of finding globally optimal solutions by incorporating three modifications to the original CS algorithm. First, the refraction learning mechanism is used in the initialization and discard steps of CS. Second, a Gaussian perturbation method is used to update the part of the worst candidate solutions in the population before the discard step in an attempt to improve them. Finally, in addition to the Lévy flight mutation method used in CS, ECS employs two mutation methods (highly disruptive polynomial mutation and Jaya mutation) to generate new improved candidate solutions.

Algorithm 1 shows the algorithmic details of the ECS algorithm. In the beginning, ECS generates a population of n candidate solutions, where half of the n solutions are generated using a random generation function (Eq. 2) and the other half is the opposite solutions of the first half. The refraction learning mechanism is used at this step to generate the opposite solutions as described in Sect. 2.3.

In the improvement loop of ECS, the initial population is used to generate a sequence of improving approximate populations for a given optimization problem. There are five main steps in each iteration of this loop. First, ECS randomly selects three candidate solutions ($\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{x}^k$) and then attempts to improve $\boldsymbol{x}^i$ using a Lévy function (Eq. 2), $\boldsymbol{x}^j$ using the Jaya mutation (Eq. 13) and $\boldsymbol{x}^k$ using the HDP mutation (Eq. 9). Second, it calculates the p_a worst solutions and updates them using Gaussian perturbation (Eq. 15). Then, it recalculates the p_a worst solutions and replaces them with new randomly generated solutions. Third, it ranks the candidate solutions to find the current best solution ($\boldsymbol{x}_t^*$). Fourth, it generates the opposite solution $\boldsymbol{x}_t^{'*}$ of $\boldsymbol{x}_t^*$ using refraction learning and then replaces $\boldsymbol{x}_t^*$ with $\boldsymbol{x}_t^{'*}$, if $f(\boldsymbol{x}_t^{'*}) < f(\boldsymbol{x}_t^*)$. This step aims to explore the opposite region to the region where $\boldsymbol{x}_t^*$ is located, which may help to jump out of suboptimality if $\boldsymbol{x}_t^*$ is not improving throughout iterations. The improvement loop of ECS keeps repeating, while the stop criterion is not satisfied or the maximum number of iterations has not been reached. The final output of ECS is the global best solution $\boldsymbol{x}^*$.

4.2 Computational complexity of ECS

In this section, we show the calculations of the computational complexity of ECS. We assumed that the computational cost of any basic vector operation is O(1) and we denoted $MaxItr$ as M.

The computational complexity of ECS (Algorithm 1) can be computed as follows:

- The number of operations required to randomly generate $n/2$ candidate solutions is $n/2$ operations (Line 3(a)). Similarly, the number of operations required to calculate the opposite solutions of the $n/2$ candidate solutions using refraction learning (Eq. 13) is $n/2$ operations (Line 3(b)).
- The random selection of three candidate solutions in line 5 costs 3 operations.
- The update process of the three selected solutions in lines 6–8 costs 3 operations.
- Calculating the fitness value of each of the three updated candidate solutions costs 3 operations (line 9).
- The random selection of another three candidate solutions in line 10 costs 3 operations.
- The comparisons in lines 11–19 cost between 3 and 6 operations.
- The computational cost in lines 20–22 is $2 \times p_a \times n$ operations. It can be further simplified to $p_a \times n$.
- The cost of the calculations in lines 23 and 24 is $n \log_2 n +$ 1 operations, which can be simplified to $n \log_2 n$.
- The comparison process in lines 25–27 costs at maximum 2 operations.
- Overall, the cost of the operations in the optimization loop (lines 4–28) is $M(3{+}3{+}3{+}3{+}6{+}p_a \times n{+}n \log_2 n{+}2)$. This can be simplified to $Mn \log_2 n$ because $n \log_2 n$ is greater than $3 + 3 + 3 + 3 + 6 + p_a \times n + 2$
- The cost of line 29 is one operation.
- The overall computational cost of ECS is $n/2 + n/2 + Mn \log_2 n + 1$, which can be simplified to $Mn \log_2 n$ because $Mn \log_2 n$ is greater than $n + 1$.

In summary, the computational complexity of ECS is $O(Mn \log_2 n)$, which is more than the computational complexity of the original CS algorithm $O(Mn)$ Alawad and Abed-alguni (2021).

4.3 Limitations

We proposed in this section ECS to solve continuous single-objective optimization problems. However, many real-world decision-making problems have multiple criteria and therefore are modeled as multi-objective optimization problems. In a multi-objective optimization problem, each criterion is modeled as an objective function (Ding et al. 2017). For example, the resource-scheduling problem in cloud computing has multiple criteria such as minimizing the completion time and cost and maximizing the profit (Wang et al. 2018; Alawad and Abed-alguni 2021; Abed-alguni and Alawad 2021). Therefore, we will further develop ECS in the near future to solve multi-objective optimization problems. In addition, we will incorporate a discretization method (e.g., the smallest position value method Alawad and Abed-alguni 2021; Abed-alguni and Alawad 2021) into ECS to enable it to deal with discrete optimization problems such as the resource-scheduling problem.

5 Experiments

5.1 Benchmark functions

Table 1 describes 14 well-known benchmark functions (Hasan et al. 2014; Doush et al. 2014) that were utilized to asses the overall performance of ECS compared to three CS algorithms: CS, CS10 and CS11 Abed-Alguni and Paul (2019).

Table 2 describes the single-objective real-parameter optimization-benchmark suit of CEC2014 that comprises 30 minimization functions. The function of this suite is very powerful functions that simulate real-world optimization problems. The search boundaries of each function in the suit are [-100, 100]^D. The complete definitions of these functions are available in Liang et al. (2014).

The comparison results reported in Abed-alguni and Barhoush (2018) using the CEC2014 suit were used to compare the performance of ECS to the performance of six well-known optimization algorithms: CS (Yang and Deb 2009), GWO (Mirjalili et al. 2014), DGWO (Abed-alguni and Barhoush 2018), L-SHADE (Tanabe and Fukunaga 2014), MHDA (Sree Ranjini and Murugan 2017) and FWA-DM (Yu et al. 2014).

Table 1 Selected benchmark functions Hasan et al. (2014), Abed-alguni (2019), Abed-alguni and Barhoush (2018)

Function name	Abb	Search range	D	$f(\overrightarrow{X^*})$
De Jong's first function	f_1	$[-100, 100]$	m	0
Schwefel 2.22 function	f_2	$[-100, 100]$	m	0
Step function	f_3	$[-100, 100]$	m	0
Rosenbrock's function	f_4	$[-2.048, 2.048]$	m	0
Rotated hyper-ellipsoid function	f_5	$[-100, 100]$	m	0
Schwefel 2.26 function	f_6	$[-500, 500]$	m	$-418.98 * \mathrm{m}$
Rastrigin's function	f_7	$[-5.12, 5.12]$	m	-1
Ackley's function	f_8	$[-32.77, 32.77]$	m	0
Griewank's function	f_9	$[-600, 600]$	m	0
Six-hump camel-back	f_{10}	$[-5, 5]$	2	-1.031628
Shifted sphere function	f_{11}	$[-100, 100]$	m	-450
Shifted Schwefel's problem 1.2	f_{12}	$[-100, 100]$	m	-450
Shifted Rosenbrock's function	f_{13}	$[-100, 100]$	m	390
Shifted Rastrigin's function	f_{14}	$[-5, 5]$	m	-330

Table 2 Single-objective real-parameter optimization-benchmark suit of CEC2014

Function	Function type
f_1–f_3	Unimodal functions
f_4–f_{16}	Multimodal functions
f_{17}–f_{22}	Hybrid functions
f_{23}–f_{30}	Composite functions

5.2 Setup

The experiments were conducted using a 10th-generation Intel® Core i7 processor (3.4 GHz) with 32 GB RAM running macOS 10.13, High Sierra. The Java programming language was used to write the source codes of all of the algorithms.

The parameter settings of the CS algorithms were: $n = 10$, $D = 1$ and $p_a = 0.25$. The mutation rate $r = 0.05$ for CS10 and CS11. In addition, CS11 used PAR = 0.3. These values are based on the recommendations in Yang and Deb (2009), Yang and Deb (2010), Abed-alguni and Alkhateeb (2017).

The parameters of FWA-DM were as suggested in Yu et al. (2014). The parameters of L-SHADE were dynamically adjusted as in Tanabe and Fukunaga (2014). The parameters of MHDA were as suggested in Sree Ranjini and Murugan (2017). The best experimental findings in the tables of this section are highlighted with bold font.

5.3 Comparison between ECS and other well-known variations of cuckoo search

Tables 3 and 4 show the experimental results of CS, ECS, CS10 and CS11 using the test functions that are described in Table 1. The format of the results is as follows: the first row is the average of the lowest acquired fitness values, the second row is the standard deviation and the third row is the FEV for 30 independent runs. FEV is the error value of a function, which is the distance between its true optimal value and the average of best objective values of the function over multiple runs. Note that in this section we say that an optimization algorithm outperforms the other algorithms when it produces the lowest mean fitness value and error value compared to the other algorithms over 30 independent runs.

Table 3 shows a summarization of the experimental results of the 30D (30 decision variables) problems, while Table 4 shows a summarization of the experimental results of the 50D (50 decision variables) problems. We can obviously see that ECS achieved the best fitness values for all of the benchmark functions. This observation clearly indicates that ECS has a robust performance even when the problem size increases. The outstanding performance of ECS is expected because it uses several exploration methods. First, it uses the HDP mutation, which provides efficient exploration capabilities over the search range of each decision variable in a candidate solution regardless of the current value of the decision variable. In contrast, CS and CS11 may get trapped closer to the middle between the two boundaries. Second, it uses refraction learning to improve the ability of CS to jump out of suboptimality. Third, it tries to improve a portion of the worst candidate solutions using Gaussian perturbation before the discard step of CS. Fourth, it uses Jaya mutation to move randomly selected solutions between the worst and best solutions in the population. Fifth, it uses the original mutation method in CS (Lévy flight mutation) to improve randomly selected candidates solutions.

We can also see that CS10 (CS with polynomial mutation) is the second-best performing algorithm in Tables 3 and 4. The performance of CS10 is expected because it also uses

Algorithm 1 Exploratory Cuckoo Search (ECS).

```
    Begin
2:  Objective function $f(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i = \langle x_1, ..., x_m \rangle$ is a candidate solution
    Generate initial population of $n$ candidate solutions $\boldsymbol{x}_i (i = 1, 2, ..., n)$ as follows:
      (a) Generate $n/2$ random candidate solutions
      (b) Generate the opposite solutions of the $n/2$ generated solutions using refraction learning (apply Equation 8 to every decision variable in every generated solution)
4:  while ($t$ <MaxItr) or (stop criterion) do
        Select three candidate solutions randomly (say, $\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{x}^k$) from the current population
6:      Update the solution $\boldsymbol{x}^i$ using Lévy flights (apply equation 2)
        Update the solution $\boldsymbol{x}^j$ using the Jaya mutation (apply Equation 13 to every decision variable in $\boldsymbol{x}^j$)
8:      Update the solution $\boldsymbol{x}^k$ using the HDP mutation (Equation 9)
        Calculate the fitness values: $f(\boldsymbol{x}^i)$, $f(\boldsymbol{x}^j)$ and $f(\boldsymbol{x}^k)$
10:     Select three candidate solutions randomly from the current population (say, $\boldsymbol{x}^{l_1}, \boldsymbol{x}^{l_2}, \boldsymbol{x}^{l_3}$)
        if $f(\boldsymbol{x}^i)$ is better than $f(\boldsymbol{x}^{l_1})$ then
12:         Replace $\boldsymbol{x}^{l_1}$ by $\boldsymbol{x}^i$
        end if
14:     if $f(\boldsymbol{x}^j)$ is better than $f(\boldsymbol{x}^{l_2})$ then
            Replace $\boldsymbol{x}^{l_2}$ by $\boldsymbol{x}_j$
16:     end if
        if $f(\boldsymbol{x}^k)$ is better than $f(\boldsymbol{x}^{l_3})$ then
18:         Replace $\boldsymbol{x}^{l_3}$ by $\boldsymbol{x}^k$
        end if
20:     Calculate the $p_a$ worst solutions and update them using Gaussian Perturbation (Equation 15)
        Rank the solutions based on their fitness values and replace the worst $p_a$ solutions with new randomly generated solutions.
22:     Keep the best solutions in the population
        Rank the solutions and find the current best $\boldsymbol{x}_t^*$
24:     Generate the opposite solution $\boldsymbol{x}_t^{'*}$ of $\boldsymbol{x}_t^*$ using refraction learning (i.e., apply Equation 8 to every decision variable in $\boldsymbol{x}_t^*$)
        if $f(x_t^{'*})$ is better than $f(\boldsymbol{x}_t^*)$ then
26:         Replace $\boldsymbol{x}_t^*$ by $\boldsymbol{x}_t^{'*}$
        end if
28: end while
    Return the global best solution $\boldsymbol{x}^*$
30: End
```

the HDP mutation to sample the search space. Moreover, ECS and CS10 have the lowest standard deviations for all the functions. This strongly indicates that they provide more stable and consistent results over multiple runs compared to the other tested algorithms.

5.4 Comparison between ECS and other well-known optimization algorithms

The FEV values for the 30 CEC2014 functions are reported in Table 5. We can see that the simulation results in Table 5 confirm the conclusion in Abed-alguni and Barhoush (2018) that L-SHADE performs better than the other optimization algorithms by providing the lowest FEV for 10 functions of the 30 functions. This is maybe because L-SHADE continuously adjusts its internal parameters and population size over the course of its optimization process. We can also see that ECS is the second-best performing algorithm with only one function difference than L-SHADE. A possible explanation is that ECS uses several techniques to help CS to jump out of suboptimality.

The overall results suggest that ECS performs very well compared to the powerful optimization algorithms. Note that GWO and CS are the worst-performing algorithms compared to the other algorithms. This is expected because CS and GWO do not use any special techniques to improve their convergence behaviors compared to the other tested algorithms.

6 Conclusion

This paper presented a new version of cuckoo search (CS) called exploratory cuckoo search (ECS). ECS incorporates three modifications to the original CS algorithm in an attempt to enhance its exploration capabilities. First, it uses refraction learning for a better exploration of the space of all feasible solutions. Second, it applies a Gaussian perturbation method to a predefined fraction of the worst candidate solutions in the population before the discard step. Third, it uses three mutation methods, the Lévy flight mutation, HDP mutation and Jaya mutation, to produce new optimized candidate solutions.

Extensive simulations were conducted to examine the performance of ECS using different algorithms and different benchmark functions. First, ECS was evaluated and compared to three variations of CS (the original CS, CS10 and CS11) using 14 widely used benchmark functions. The overall experimental and statistical results indicate that ECS exhibits better performance than the three variations of CS. Second, ECS was evaluated and compared to the state-of-the-art optimization algorithms, CS, GWO, DGWO, L-SHADE, MHDA and FWA-DM, using 30 IEEE CEC 2014 functions. Although the results indicate that ECS is the second-best performing algorithm after L-SHADE, the results strongly indicate that ECS provides competitive performance compared to L-SHADE.

There are three interesting directions for future work. First, two improvements will be incorporated to ECS based on L-SHADE: success-history-based parameter adaptation and dynamic reduction of the population based on a linear function (Tanabe and Fukunaga 2014). Second, ECS will be applied to cooperative Q-learning (Abed-alguni et al. 2015a, b; Abed-alguni and Ottom 2018) as described in Abed-alguni (2018), Abed-alguni (2017), Abed-Alguni

Table 3 Summary of the experimental results of of ECS compared to three CS algorithms using 14 test functions

Function	CS	ECS	CS10	CS11
F1	5.13E−06	**0.00E+00**	**0.00E+00**	4.31E−06
	5.84E−06	**0.00E+00**	**0.00E+00**	6.29E−06
	5.13E−06	**0.00E+00**	**0.00E+00**	4.31E−06
F2	3.48E−03	**0.00E+00**	**0.00E+00**	1.26E−03
	4.03E−03	**0.00E+00**	**0.00E+00**	1.03E−03
	3.48E−03	**0.00E+00**	**0.00E+00**	1.26E−03
F3	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**
F4	2.58E+03	**2.27E+02**	2.83E+03	2.83E+03
	7.07E+02	**1.14E+02**	1.21E−02	2.71E−03
	2.58E+03	**2.27E+02**	2.83E+03	2.83E+03
F5	1.76E+01	**0.00E+00**	**0.00E+00**	1.19E+01
	1.57E+01	**0.00E+00**	**0.00E+00**	2.63E+01
	1.76E+01	**0.00E+00**	**0.00E+00**	1.19E+01
F6	1.97E+03	**1.00E+00**	5.87E+03	5.87E+03
	2.16E+02	**2.36E−03**	1.44E−03	8.44E−04
	1.45E+04	**1.25+04**	1.84E+04	1.84E+04
F7	3.46E−03	**0.00E+00**	**0.00E+00**	2.08E−04
	4.58E−03	**0.00E+00**	**0.00E+00**	2.94E−04
	3.46E−03	**0.00E+00**	**0.00E+00**	2.08E−04
F8	1.19E−02	**0.00E+00**	4.44E−16	3.72E−03
	9.51E−03	**0.00E+00**	1.04E−31	7.36E−03
	1.19E−02	**0.00E+00**	4.44E−16	3.72E−03
F9	1.84E−02	**0.00E+00**	**0.00E+00**	9.36E−03
	1.47E−02	**0.00E+00**	**0.00E+00**	8.35E−03
	1.84E−02	**0.00E+00**	**0.00E+00**	9.36E−03
F10	−1.03E+00	**− 1.03E+00**	−1.03E+00	−1.03E+00
	1.56E−03	**1.50E−03**	1.62E−03	1.97E−03
	1.52E−03	**1.26E−03**	2.04E−03	2.15E−03
F11	6.12E+04	**− 4.35E+02**	−2.86E+02	6.12E+04
	7.17E+03	**1.76E+01**	1.89E+01	1.00E+04
	6.17E+04	**0.15E+02**	1.64E+02	6.16E+04
F12	1.08E+07	**1.11E+04**	2.79E+04	1.14E+07
	1.36E+06	**5.14E+03**	4.59E+03	1.94E+06
	1.08E+07	**1.11E+04**	2.84E+04	1.14E+07
F13	3.02E+10	**2.03E+7**	7.01E+10	4.18E+10
	5.87E+10	**4.18E+7**	8.68E+10	4.38E+10
	3.02E+10	**2.03E+7**	7.01E+10	4.18E+10
F14	2.98E+01	**− 1.84E+02**	4.45E+01	9.55E+01
	1.76E+02	**1.40E+02**	1.25E+02	1.80E+02
	3.60E+02	**1.46E+02**	3.75E+02	4.26E+02

$D = 30$, runs = 30 and maximum number of iterations = 10,000
The best result for each benchmark function is highlighted with bold font

Table 4 Summary of the experimental results of ECS compared to three CS algorithms using 14 test functions

Function	CS	ECS	CS10	CS11
F_1	7.44E−06	**0.00E+00**	**0.00E+00**	2.99E−06
	9.12E−06	**0.00E+00**	**0.00E+00**	4.18E−06
	7.44E−06	**0.00E+00**	**0.00E+00**	2.99E−06
F_2	5.16E−03	**0.00E+00**	**0.00E+00**	1.03E−03
	4.76E−03	**0.00E+00**	**0.00E+00**	8.16E−04
	5.16E−03	**0.00E+00**	**0.00E+00**	1.03E−03
F_3	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**
F_4	4.83E+03	**2.21E+01**	4.83E+03	4.83E+03
	2.14E−01	**1.70E+01**	7.20E−03	3.48E−02
	4.83E+03	**2.21E+01**	4.83E+03	4.83E+03
F_5	1.12E+02	**0.00E+00**	**0.00E+00**	2.86E+00
	1.76E+02	**0.00E+00**	**0.00E+00**	7.18E+00
	1.12E+02	**0.00E+00**	**0.00E+00**	2.86E+00
F_6	6.43E+02	**7.87E+01**	5.87E+03	5.87E+03
	3.73E+02	**7.31E−04**	2.16E−03	5.78E−04
	1.32E+04	**1.26E+4**	1.84E+04	1.84E+04
F_7	1.58E−03	**0.00E+00**	**0.00E+00**	4.64E−04
	1.87E−03	**0.00E+00**	**0.00E+00**	5.36E−04
	1.58E−03	**0.00E+00**	**0.00E+00**	4.64E−04
F_8	6.91E−03	**0.00E+00**	**4.44E−16**	1.87E−03
	5.94E−03	**0.00E+00**	**1.04E−31**	8.60E−04
	6.91E−03	**0.00E+00**	**4.44E−16**	1.87E−03
F_9	1.40E−02	**0.00E+00**	**0.00E+00**	1.33E−02
	8.48E−03	**0.00E+00**	**0.00E+00**	1.64E−02
	1.40E−02	**0.00E+00**	**0.00E+00**	1.33E−02
F_{10}	−1.03E+00	− **1.03E+00**	−1.03E+00	−1.03E+00
	1.56E−03	**1.42E−03**	1.62E−03	1.97E−03
	1.52E−03	**1.66E−03**	2.04E−03	2.15E−03
F_{11}	1.42E+05	− **8.14E+01**	−6.40E+01	1.39E+05
	1.31E+04	**3.17E+01**	2.96E+01	1.64E+04
	1.43E+05	**3.61E+02**	3.86E+02	1.40E+05
F_{12}	8.08E+07	− **1.11E+02**	2.20E+05	8.44E+07
	1.04E+07	**1.66E+01**	2.98E+04	1.07E+07
	8.08E+07	**3.48E02**	2.21E+05	8.44E+07
F_{13}	1.11E+11	**2.16E+2**	1.87E+11	7.14E+10
	1.59E+11	**1.47E+2**	1.66E+11	1.39E+11
	1.11E+11	**2.16E+2**	1.87E+11	7.14E+10
F_{14}	3.90E+02	− **0.51E+02**	3.84E+02	2.97E+02
	3.14E+02	**3.65E+02**	3.38E+02	3.43E+02
	7.20E+02	**2.79E+02**	7.14E+02	6.27E+02

$D = 50$, runs = 30 and maximum number of iterations = 10,000
The best result for each benchmark function is highlighted with bold font

Table 5 Summary of the experimental results of ECS compared to six optimization algorithms

Function	ECS	GWO	DGWO	CS	L-SHADE	MHDA	FWA-DM
f_1	**9.00E−15**	2.00E+03	4.36E+00	3.47E + 07	**9.00E−15**	3.59E+03	4.91E+05
f_2	**2.45E−16**	2.12E+03	2.36E+00	2.50E + 07	8.50E−11	3.82E+03	2.50E−16
f_3	**1.78E−16**	2.89E−01	2.54E−04	4.10E + 04	5.83E−10	5.80E−07	1.88E−16
f_4	1.82E−09	8.75E−03	**1.63E−09**	4.22E+02	2.58E−09	1.42E−08	2.23E+01
f_5	5.67E+00	3.96E+02	2.00E+02	5.00E+01	2.00E+01	**2.36E**+00	2.11E+01
f_6	**8.75E−15**	5.19E+01	1.21E+00	3.63E+01	1.25E−06	8.52E−14	1.82E+01
f_7	1.26E−10	2.89E−03	8.53E−10	1.86E+00	7.25E−09	**2.25E−11**	2.53E−03
f_8	8.55E−17	1.33E+00	**1.51E−19**	3.89E+02	1.25E−09	2.20E−19	9.53E−15
f_9	1.13E+00	1.82E+01	**1.03E+00**	3.00E+03	8.96E+00	5.30E+00	6.54E+01
f_{10}	**3.18E−03**	9.79E+00	3.20E−03	4.37E+03	2.36E−02	1.22E+03	1.13E+01
f_{11}	**1.50E+02**	1.99E+04	2.95E+03	4.00E+03	2.30E+03	1.52E+02	2.19E+03
f_{12}	6.35E−02	8.50E+00	**6.30E−02**	4.78E−01	9.00E−01	1.42E−01	3.25E−01
f_{13}	3.15E−01	2.19E+00	4.59E−01	4.81E−01	6.50E−01	4.78E−01	**3.11E−01**
f_{14}	2.00E−01	2.35E−01	**1.99E−01**	4.28E−01	8.60E−01	5.43E−01	2.99E−01
f_{15}	1.67E+00	1.01E+02	7.23E+01	9.94E+01	**1.60E+00**	3.25E+00	8.36E+00
f_{16}	**9.50E+00**	1.90E+01	9.53E+00	1.53E+01	1.02E+01	1.06E+01	1.10E+01
f_{17}	2.55E+00	1.66E+02	4.55E+03	3.47E+06	**2.20E+00**	4.53E+02	6.59E+03
f_{18}	2.95E+00	8.77E+00	3.94E+01	3.90E+03	**1.90E+00**	3.69E+00	7.24E+01
f_{19}	5.70E+00	4.96E+01	1.22E+02	6.14E+01	**5.30E+00**	3.78E+02	1.04E+01
f_{20}	**4.21E+00**	6.20E+01	4.73E+02	3.97E+04	4.30E+00	7.09E+02	4.37E+01
f_{21}	2.78E+02	1.04E+03	7.09E+02	3.57E+05	3.69E+02	**2.57E+02**	8.75E+02
f_{22}	**1.32E+02**	2.42E+02	2.73E+02	9.47E+02	**1.32E+02**	2.73E+02	1.62E+02
f_{23}	3.89E+01	3.65E+02	**3.69E+01**	3.78E+02	3.26E+02	3.10E+03	3.16E+02
f_{24}	1.98E+02	2.24E+02	2.25E+02	2.89E+02	**1.93E+02**	2.26E+02	2.96E+02
f_{25}	2.10E+02	2.45E+02	2.11E+02	3.26E+02	**2.00E+02**	2.11E+02	2.09E+02
f_{26}	1.89E+02	3.29E+02	2.10E+02	2.22E+02	2.69E+02	1.00E+02	**9.93E+01**
f_{27}	1.29E+02	2.95E+02	4.09E+02	5.22E+02	**1.26E+02**	4.05E+02	4.10E+02
f_{28}	3.89E+02	5.36E+02	1.65E+03	3.86E+03	**3.62E+02**	1.54E+03	4.22E+02
f_{29}	2.31E+02	2.39E+02	**2.29E+02**	2.59E+05	7.33E+02	7.86E+02	2.78E+02
f_{30}	2.93E+00	3.32E+02	**2.83E+00**	2.39E+04	6.99E+02	2.63E+03	4.69E+02

$D = 30$, runs = 30 and number of iterations = 10,000
The best result for each benchmark function is highlighted with bold font

(2014). Finally, ECS will be further developed to solve multi-objective optimization problems.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Human and animal rights This article does not contain any studies with human participants or animals performed by any of the authors.

References

Abed-Alguni BHK (2014) Cooperative reinforcement learning for independent learners. Ph.D. Thesis, Faculty of Engineering and Built Environment, School of Electrical Engineering and Computer Science, The University of Newcastle, Australia

Abed-alguni BH, Klaib AF, Nahar KM (2019) Island-based whale optimization algorithm for continuous optimization problems. Int J Reason Based Intell Syst 1–11

Abed-Alguni BH, Paul DJ (2019) Hybridizing the cuckoo search algorithm with different mutation operators for numerical optimization problems. J Intell Syst

Abed-alguni HB, Alkhateeb F (2018) Intelligent hybrid cuckoo search and β-hill climbing algorithm. J King Saud Univ Comput Inf Sci 1–43

Abed-alguni BH (2017) Bat Q-learning algorithm. Jordanian J Comput Inf Technol (JJCIT) 3(1):56–77

Abed-alguni BH (2018) Action-selection method for reinforcement learning based on cuckoo search algorithm. Arab J Sci Eng 43(12):6771–6785

Abed-alguni BH (2019) Island-based cuckoo search with highly disruptive polynomial mutation. Int J Artif Intell 17(1):57–82

Abed-alguni BH, Alawad NA (2021) Distributed grey wolf optimizer for scheduling of workflow applications in cloud environments. Appl Soft Comput 102:107113

Abed-alguni BH, Alkhateeb F (2017) Novel selection schemes for cuckoo search. Arab J Sci Eng 42(8):3635–3654

Abed-alguni BH, Barhoush M (2018) Distributed grey wolf optimizer for numerical optimization problems. Jordanian J Comput Inf Technol (JJCIT) 4:130–149

Abed-alguni BH, Barhoush M (2018) Distributed grey wolf optimizer for numerical optimization problems. Jordanian J Comput Inf Technol 4(3):130–149

Abed-alguni BH, Ottom MA (2018) Double delayed Q-learning. Int J Artif Intell 16(2):41–59

Abed-alguni BH, Chalup SK, Henskens FA, Paul DJ (2015) Erratum to: A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers. Vietnam J Comput Sci 2(4):227

Abed-alguni BH, Chalup SK, Henskens FA, Paul DJ (2015) A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers. Vietnam J Comput Sci 2(4):213–226

Abed-Alguni BH, Paul DJ, Chalup SK, Henskens FA (2016) A comparison study of cooperative Q-learning algorithms for independent learners. Int J Artif Intell 14(1):71–93

Alawad NA, Abed-alguni BH (2021) Discrete island-based cuckoo search with highly disruptive polynomial mutation and opposition-based learning strategy for scheduling of workflow applications in cloud environments. Arab J Sci Eng 46(4):3213–3233

Ali AF, Tawhid MA (2016) A hybrid cuckoo search algorithm with Nelder mead method for solving global optimization problems. SpringerPlus 5(1):473

Alkhateeb F, Abed-Alguni BH (2017) A hybrid cuckoo search and simulated annealing algorithm. J Intell Syst

Chen L, Lu H, Li H, Wang G, Chen L (2019) Dimension-by-dimension enhanced cuckoo search algorithm for global optimization. Soft Comput 23(21):11297–11312

Cheng J, Wang L, Xiong Y (2019) Ensemble of cuckoo search variants. Comput Ind Eng 135:299–313

Chi R, Su Y, Zhang D, Chi X, Zhang H (2019) A hybridization of cuckoo search and particle swarm optimization for solving optimization problems. Neural Comput Appl 31(1):653–670

Deb K, Tiwari S (2008) Omni-optimizer: a generic evolutionary algorithm for single and multi-objective optimization. Eur J Oper Res 185(3):1062–1087

Ding S, Xia C, Wang C, Wu D, Zhang Y (2017) Multi-objective optimization based ranking prediction for cloud service recommendation. Decis Support Syst 101:106–114

Doush IA, Hasan BHF, Al-Betar MA, Al Maghayreh E, Alkhateeb F, Hamdan M (2014) Artificial bee colony with different mutation schemes: a comparative study. Comput Sci J Moldova 22(1)

El-Shorbagy MA, Mousa AA, Nasr SM (2016) A chaos-based evolutionary algorithm for general nonlinear programming problems. Chaos Solitons Fractals 85:8–21

Feng Y, Wang G-G, Dong J, Wang L (2018) Opposition-based learning monarch butterfly optimization with gaussian perturbation for large-scale 0–1 knapsack problem. Comput Electr Eng 67:454–468

Fister I Jr, Perc M, Kamal SM, Fister I (2015) A review of chaos-based firefly algorithms: perspectives and research challenges. Appl Math Comput 252:155–165

Gandomi AH, Yang X-S (2014) Chaotic bat algorithm. J Comput Sci 5(2):224–232

Hasan BHF, Doush IA, Al Maghayreh E, Alkhateeb F, Hamdan M (2014) Hybridizing harmony search algorithm with different mutation operators for continuous problems. Appl Math Comput 232:1166–1182

Huang L, Ding S, Yu S, Wang J, Lu K (2016) Chaos-enhanced cuckoo search optimization algorithms for global optimization. Appl Math Model 40(5–6):3860–3875

Lardeux F, Goëffon A (2010) A dynamic island-based genetic algorithms framework. In: Asia-Pacific conference on simulated evolution and learning, Kanpur, India, SEAL'10. Springer, Berlin, pp 156–165

Li J, Li Y-X, Tian S-S, Zou J (2019) Dynamic cuckoo search algorithm based on Taguchi opposition-based search. Int J Bio-Inspired Comput 13(1):59–69

Liang JJ, Qu BY, Suganthan PN (2014) Problem definitions and evaluation criteria for the cec, special session and competition on single objective real-parameter numerical optimization. In: Computational intelligence laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore 635:490

Long W, Wu T, Cai S, Liang X, Jiao J, Xu M (2019) A novel grey wolf optimizer algorithm with refraction learning. IEEE Access 7:57805–57819

Long W, Wu T, Jiao J, Tang M, Xu M (2020) Refraction-learning-based whale optimization algorithm for high-dimensional problems and parameter estimation of PV model. Eng Appl Artif Intell 89:103457

Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Advances in engineering software 69:46–61

Mohamad AB, Zain AM, Bazin NEN (2014) Cuckoo search algorithm for optimization problems-a literature review and its applications. Appl Artif Intell 28(5):419–448

Rakhshani H, Rahati A (2017) Snap-drift cuckoo search: a novel cuckoo search optimization algorithm. Appl Soft Comput 52:771–794

Rao R (2016) Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int J Ind Eng Comput 7(1):19–34

Roy M, Chakraborty S, Mali K, Chatterjee S, Banerjee S, Chakraborty A, Biswas R, Karmakar J, Roy K (2017) Biomedical image enhancement based on modified cuckoo search and morphology. In: 2017 8th annual industrial automation and electromechanical engineering conference (IEMECON), pp 230–235. IEEE

Salgotra R, Singh U, Saha S (2018) New cuckoo search algorithms with enhanced exploration and exploitation properties. Expert Syst Appl 95:384–420

Shehab M, Khader AT, Alia MA(2019) Enhancing cuckoo search algorithm by using reinforcement learning for constrained engineering optimization problems. In: 2019 IEEE Jordan international joint conference on electrical engineering and information technology (JEEIT). IEEE, pp 812–816

Sonia G, Patterh MS (2014) Wireless sensor network localization based on cuckoo search algorithm. Wirel Pers Commun 79(1):223–234

Sree Ranjini KS, Murugan S (2017) Memory based hybrid dragonfly algorithm for numerical optimization problems. Expert Syst Appl 83:63–78

Tanabe R, Fukunaga AS (2014) Improving the search performance of shade using linear population size reduction. In: 2014 IEEE congress on evolutionary computation (CEC). IEEE, pp 1658–1665

Walton S, Hassan O, Morgan K, Brown MR (2011) Modified cuckoo search: a new gradient free optimisation algorithm. Chaos Solitons Fractals 44(9):710–718

Wang LJ, Yin YL, Zhong YW (2013) Cuckoo search algorithm with dimension by dimension improvement. J Softw 24(11):2687–2698

Wang G-G, Deb S, Gandomi AH, Zhang Z, AlaviAlavi AV (2016) Chaotic cuckoo search. Soft Comput 20(9):3349–3362

Wang L, Zhong Y, Yin Y (2016) Nearest neighbour cuckoo search algorithm with probabilistic mutation. Appl Soft Comput 49:498–509

Wang G-G, Gandomi AH, Yang X-S, Alavi AH (2016) A new hybrid method based on krill herd and cuckoo search for global optimisation tasks. Int J Bio-Inspired Comput. 8(5):286–299

Wang J, Li C, Xia C (2018) Improved centrality indicators to characterize the nodal spreading capability in complex networks. Appl Math Comput 334:388–400

Xiao H, Duan Y (2014) Cuckoo search algorithm based on differential evolution. J Comput Appl 34(6):1361–1635

Yang X-S, Deb S, (2009) Cuckoo search via Lévy flights. In: World congress on nature and biologically inspired computing, 2009. NaBIC 2009. IEEE, pp 210–214

Yang X-S, Deb S (2010) Engineering optimisation by cuckoo search. Int J Math Model Numer Optim 1(4):330–343

Yang Q, Gao H, Zhang W (2017) Biomass concentration prediction via an input-weighed model based on artificial neural network and peer-learning cuckoo search. Chemomet Intell Lab Syst 171:170–181

Ye Z, Wang M, Hu Z, Liu W (2015) An adaptive image enhancement technique by combining cuckoo search and particle swarm optimization algorithm. Comput Intell Neurosci

Yu C, Kelley L, Zheng S, Tan Y (2014) Fireworks algorithm with differential mutation for solving the CEC 2014 competition problems. In: 2014 IEEE congress on evolutionary computation (CEC). IEEE, pp 3238–3245

Zhang Z, Ding S, Jia W (2019) A hybrid optimization algorithm based on cuckoo search and differential evolution for solving constrained engineering problems. Eng Appl Artif Intell 85:254–268