Soft Computing (2022) 26:1279–1314
https://doi.org/10.1007/s00500-021-06401-0

OPTIMIZATION

# Hunter–prey optimization: algorithm and applications

Iraj Naruei[1] · Farshid Keynia[2] · Amir Sabbagh Molahosseini[1]

Accepted: 4 October 2021 / Published online: 1 December 2021


**Abstract**

This paper proposes a new population-based optimization algorithm called hunter–prey optimizer (HPO). This algorithm is inspired by the behavior of predator animals such as lions, leopards and wolves, and preys such as stag and gazelle. There are many scenarios of animal hunting behavior, and some of them have transformed into optimization algorithms. The scenario used in this paper is different from the scenario of the previous algorithms. In the proposed approach, a prey and predator population, and a predator attacks a prey that moves away from the prey population. The hunter adjusts his position toward this far prey, and the prey adjusts his position toward a safe place. The search agent's position that was the best value of the fitness function considered a safe place. The HPO algorithm implemented on several test functions to evaluate its performance. Also, to performance verification, the proposed algorithm is applied to several engineering problems. The results showed that the proposed algorithm performed effective in solving test functions and engineering problems.

**Keywords** Hunter–Prey Optimizer · HPO Algorithm · Optimization · Meta-heuristic · Optimizer

## 1 Introduction

Optimization refers to the process of finding optimal values for the parameters of a given system from all the possible values to maximize or minimize its output (Mirjalili 2016). Optimization problems can be found in various fields which makes optimization methods essential, and provides an exciting research direction for researchers (Hussain et al. 2018). Optimization algorithms are an effective field of research with exceedingly important improvements in the resolve of intractable optimization problems. Significant advances have been made since the first algorithm was proposed, and many new algorithms are still being proposed (Dokeroglu et al. 2019). Conventional optimization methods such as the Newton method (Deuflhard 2011) and quadratic programming (Hillier and Hillier 2003) have problems such as local optimization stagnation and the need to derive the search space (Simpson et al. 1994). Stochastic optimization methods have become popular in the last two decades (Spall 2003; Parejo et al. 2012; Boussaïd et al. 2013). A meta-heuristic algorithm is an algorithmic framework that can be applied to various optimization problems with slight modifications. The use of meta-algorithms significantly increases the ability to find high-quality solutions to hybrid optimization problems. In other words, a meta-heuristic algorithm is a heuristic method that can search the search space to find high-quality answers. The meta-algorithms' common goal is to solve the well-known challenging optimization problems (Dorigo and Stützle 2004). Meta-heuristic methods have the following common characteristics (Crawford et al. 2017):

- These methods are somewhat probabilistic. This approach avoids placing the algorithm in the optimal local trap.
- A meta-heuristic is a high-level strategy that guides a heuristic search process.

✉ Farshid Keynia
f.keynia@kgut.ac.ir

Iraj Naruei
irajnaruei@iauk.ac.ir

Amir Sabbagh Molahosseini
sabbagh@iauk.ac.ir

[1] Department Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran

[2] Department of Energy Management and Optimization, Institute of Science and High Technology and Environmental Sciences, Graduate University of Advanced Technology, Kerman, Iran

- The goal is to efficiently explore the search space in order to find (close to) optimal solutions.
- Meta-heuristics are not problem-special.
- The basic concepts of meta-heuristics permit an abstract level of description.
- Meta-heuristic algorithms are approximate and generally nondeterministic.
- One of the common disadvantages in these methods is the difficulty of adjusting and matching of parameters.

Different criteria are used to classify meta-heuristic algorithms (Talbi 2009). In general, meta-heuristic algorithms are divided into two categories: one-solution-based algorithms and population-based algorithms (Boussaïd et al. 2013). A one-solution-based algorithm changes a solution during the search process (Fig. 1), whereas in the population-based algorithms, a population of solutions is considered (Fig. 2). The characteristics of these two types of algorithms are complementary to each other. One-solution-based meta-heuristic algorithms can focus on local search areas. In contrast, population-based meta-heuristic algorithms can lead the search to different solution space regions (Zhou et al. 2011).

Population-based optimization methods are inspired by human phenomena, collective intelligence, evolutionary concepts and physical phenomena (Shen et al. 2016; Wang et al. 2017; Zhang et al. 2018; Rizk-Allah 2018). Many studies have tried to classify optimization algorithms based on their type of inspiration. Figure 3 shows a type of this classification. The algorithms start with a random initial population (Heidari et al. 2017; Mafarja et al. 2018a), and this population is directed to the optimal areas in the search space by search mechanisms (Aljarah et al. 2018; Mafarja et al. 2018b). The search process involves two stages of exploration and exploitation. The well-designed algorithm and enriched random nature should explore different parts of the search space at the exploration stage. The exploitation stage is usually performed after the exploration phase. The algorithm focuses on good solutions and improves the search operation by searching around these right solutions in this stage. A good algorithm should balance the two steps to prevent premature convergence or belated convergence. The structure of optimization algorithms is almost similar, and their main difference is in how the exploration and exploitation phases are performed. How to balance the exploration and operation phases is another indicator that differentiates the performance of algorithms.

## 2 Related works

This section introduces a number of popular optimization algorithms. These methods include genetic algorithms (GA) (Holland 1967; Holland and Reitman 1977), evolutionary programming(EP) (Fogel et al. 1966; Xin Yao et al. 1999), particle swarm optimization (PSO) (Eberhart and Kennedy 2002), ant colony optimization (ACO) (Colorni et al. 1991), differential evolution (DE) (Storn and Price 1995) and harmony search (HS) (Manjarres et al. 2013). Although these algorithms can solve many real and challenging problems, there are still issues that these algorithms have not been able to solve. Therefore, an algorithm can help solve one set of problems, but it is ineffective in another set of problems. Some of the new algorithms are gray wolf optimizer (GWO) (Mirjalili et al. 2014), artificial bee colony (ABC) algorithm (Basturk and Karaboga 2006), firefly algorithm (FA) (Yang 2010), imperialist competitive algorithm (ICA) (Atashpaz-Gargari and Lucas 2007), cuckoo search algorithm (CS) (Yang and Suash Deb 2009; Yang and Deb 2010; Rajabioun 2011), gravitational search algorithm (GSA) (Rashedi et al. 2009), charged system search (CSS) (Kaveh and Talatahari 2010), magnetic

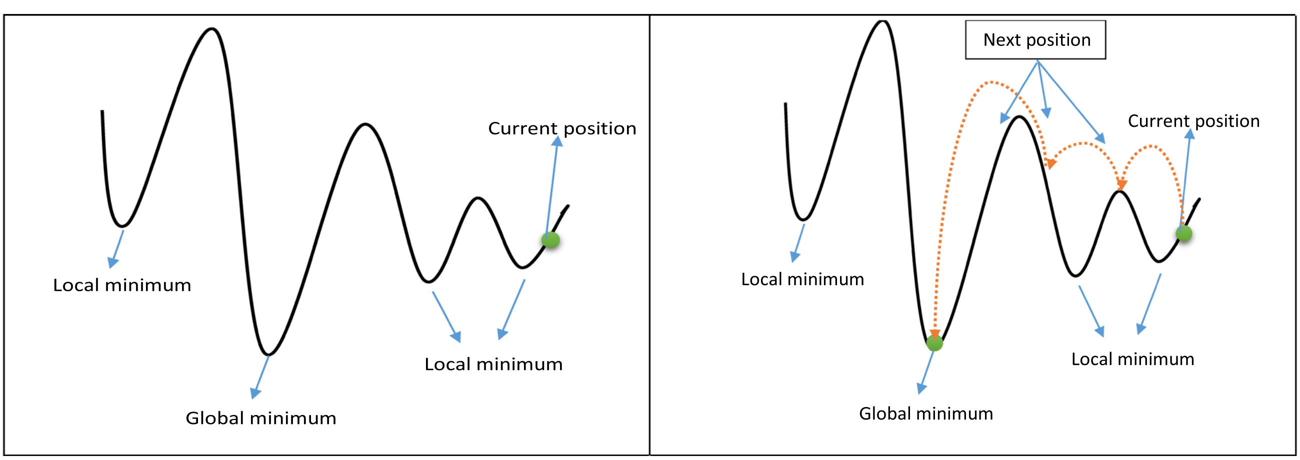



**Fig. 1** One-solution-based meta-heuristic algorithm

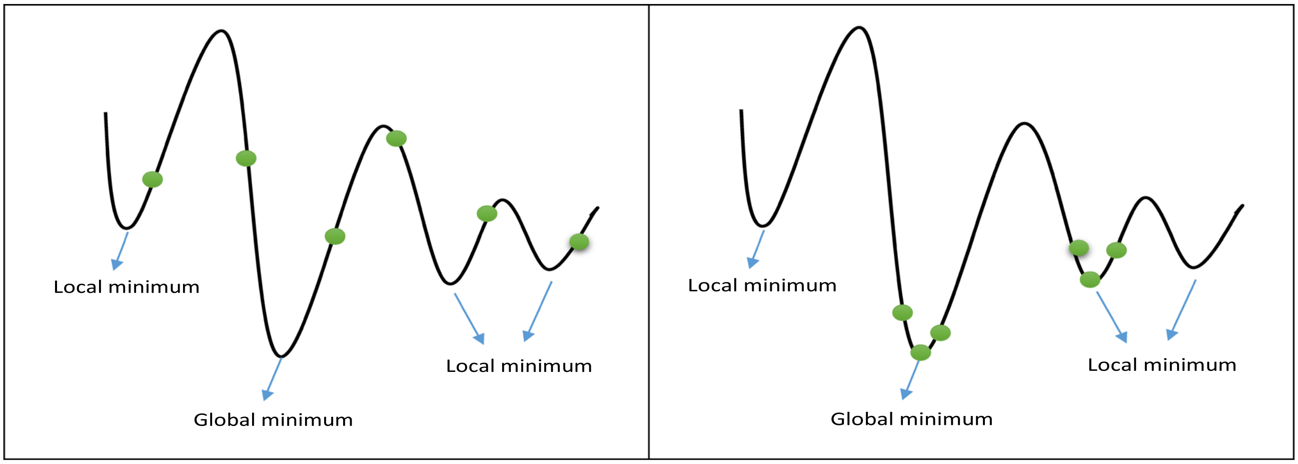
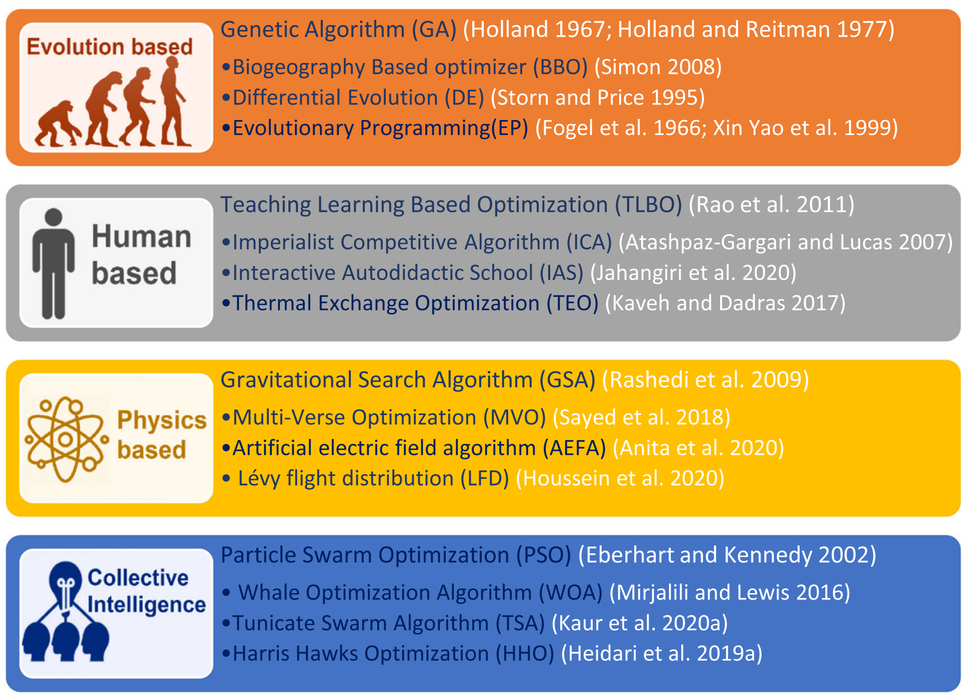



**Fig. 2** Population-based meta-heuristic algorithm

**Fig. 3** Categorization of meta-heuristic algorithms




charged system search (Kaveh et al. 2013), thermal exchange optimization (TEO) (Kaveh and Dadras 2017), ray optimization (RO) algorithm (Kaveh and Khayatazad 2012), colliding bodies optimization (CBO) (Kaveh and Mahdavi 2014), sea lion optimization algorithm (SLOA) (Masadeh et al. 2019), Biogeography-based optimizer (BBO) (Simon 2008), dolphin echolocation (DE) (Kaveh and Farhoudi 2013) and bat algorithm (BA) (Yang and Hossein Gandomi 2012). The ant lion optimizer (ALO) algorithm mimics the hunting mechanism of ant lions (Mirjalili 2015a). The whale optimization algorithm (WOA) is inspired by the social behavior of humpback whales and mimics the bubble-net hunting method (Mirjalili and Lewis 2016). The Harris hawks optimizer (HHO) mimics the cooperative behavior, and chasing style of Harris' hawks in nature called surprise pounce (Heidari et al. 2019). The Lévy flight distribution (LFD) algorithm mimics the Lévy flight random walk to find the optimal solution (Houssein et al. 2020). The tunicate swarm algorithm (TSA) is inspired by the swarm behaviors of tunicates and jet propulsion during the food search and navigation process (Kaur et al. 2020). The wild horse optimizer (WHO) algorithm mimics the social life behavior of wild horses (Naruei and Keynia 2021a).

In recent years, many optimization methods have been proposed. The question now arises why, in spite of these algorithms, it is still necessary to provide new algorithms or improve previous algorithms. This question can be

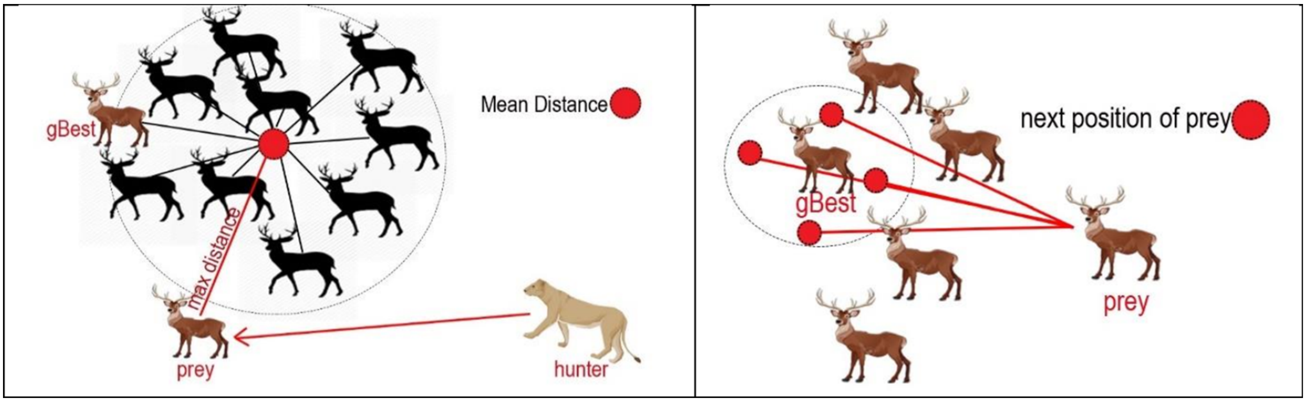



**Fig. 4** Hunter behavior (left image **a**), prey behaviors (right image **b**) and next position adjustment

answered by referring to the No Free Lunch (NFL) theory (Wolpert and Macready 1997). The NFL theorem logically proves that no one can present an algorithm that can solve all problems of optimization. According to this theorem, it can be concluded that the ability of an optimization algorithm to solve a specific set of issues does not guarantee that the algorithm is able to solve other problems. Therefore, all optimizers act on average considering all optimization problems despite higher performance in a subset of optimization issues. The NFL theorem allows researchers to suggest new optimization methods or improve existing methods to solve a subset of problems in different fields. This theory motivated us to propose a novel optimization method to solve challenging problems in this area.

This paper proposes a new population-based optimization algorithm called the hunter–prey optimization algorithm. This algorithm mimics the behavior of hunters such as lions, leopards, wolves and prey such as deer, stag, gazelle with a different scenario than the previously proposed algorithms in this field.

The rest of the paper is as follows. Section 3 provides the inspiration and scenario for this work and proposes the HPO algorithm. In Sect. 4, the results of the proposed method on different test functions are presented. In Sect. 5, the performance of the proposed algorithm for solving real-world problems was evaluated. Finally, conclusions and possible new researches are presented in Sect. 6.

## 3 Hunter–prey optimization algorithms

In this part, first, the inspiration and hunting scenario of HPO algorithm is explained, and then the mathematical model and HPO algorithm are described in detail.

### 3.1 Inspiration

Nature inspiration may be a good idea to solve problems. In nature, organisms interact with each other in different ways (Han et al. 2001; Krebs 2009). One of these interactions can occur between hunter and prey behavior. Hunter–prey cycles are one of the most remarkable observations in population biology, and thus, serious debate is taking place among ecologists (Berryman 2002; Turchin 2003). There are many scenarios of how animals hunt. Some of these scenarios have been converted into optimization algorithms. The scenario considered in this paper is different from others scenarios. Our scenario is that the hunter searches for prey, and since prey is usually swarmed, the hunter chooses a prey that is far from the swarm (average herd position). After the hunter finds his prey, he chases and hunts it. At the same time, the prey searches for food and escapes in a predator attack and reaches a safe place (Berryman 1992; Krohne 2000). We consider this safe place as the place of the best prey in terms of a fitness function. Figure 4 illustrates these behaviors.

### 3.2 Mathematical model and algorithm

As mentioned in the previous section, the general structure of all optimization techniques is the same. First, the initial population is randomly set to $(\vec{x}) = \{\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n\}$, and then the objective function is computed as $(\vec{O}) = \{O_1, O_2, \ldots O_n\}$ for all members of the population. The population is controlled and directed in the search space using a series of rules and strategies inspired by the proposed algorithm. This process is repeated until the algorithm is stopped. In each iteration, the position of each member of the population is updated according to the rules of the proposed algorithm, and the new position is evaluated with the objective function. This process causes the solutions to improve with each iteration. The position of

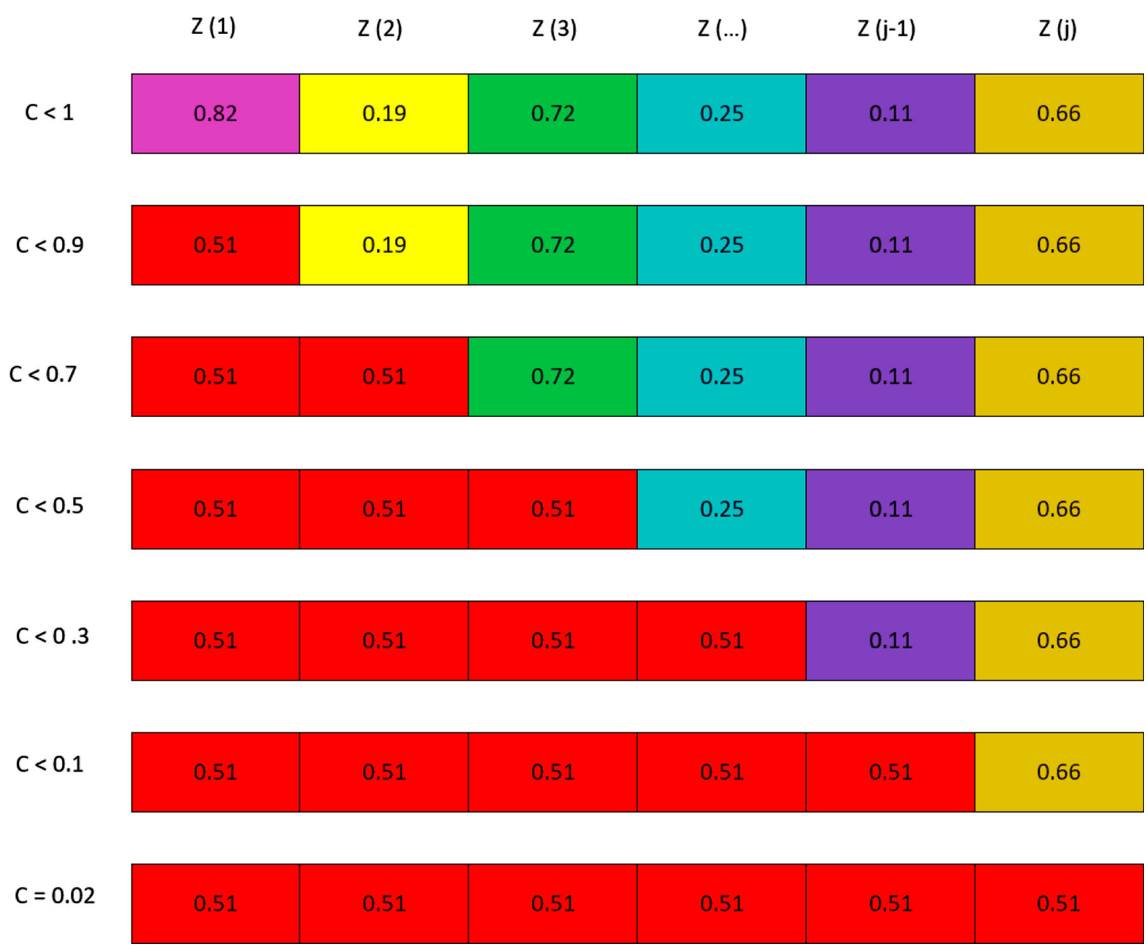



**Fig. 5** Z parameter structure. $R_2$ is random vector (for example: [0.82, 0.19, 0.72, 0.25, 0.11 and 0.66]) and $R_3$ is random number (for example: with value 0.51). (j is the length of the dimension with value 6)

each member of the initial population is randomly generated in the search space by Eq. (1)

$$x_i = rand(1,d).*(ub - lb) + lb \quad (1)$$

where $X_i$ is the hunter position or prey, *lb* is the minimum value for the problem variables (lower boundary), *ub* is the maximum value for the problem variables (upper boundary), and *d* is the number of variables (dimensions) of the problem. Equation (2) defines the lower boundary and the upper boundary of the search space. It should be noted that a problem may have the same or different lower and upper bounds for all its variables.

$$lb = [lb_1, lb_2, \ldots, lb_d],\ ub = [ub_1, ub_2, \ldots, ub_d] \quad (2)$$

After generating the initial population and determining each agent's position, each solution's fitness is calculated using $O_i = f(\vec{x})$, the objective function. *F (x)* can be maximum (efficiency, performance, etc.) or minimum (cost, time, etc.). Calculating the fitness function determines which solution is good or bad, but we do not achieve the optimal solution with a single run. A search mechanism must be defined and repeated several times to guide the search agents to the optimal position. The search mechanism usually involves two steps: exploration and exploitation. Exploration refers to the algorithm's tendency to highly random behaviors so that solutions change significantly. The significant changes in solutions cause further exploration of the search space and discover its promising areas. After promising regions have been found, random behaviors must be reduced so that the algorithm can search around the promising regions, and this refers to exploitation. For the hunter search mechanism, we propose Eq. (3)

$$x_{i,j}(t+1) = x_{i,j}(t) + 0.5\left[\left(2CZP_{pos(j)} - x_{i,j}(t)\right) + \left(2(1-C)Z\mu_{(j)} - x_{i,j}(t)\right)\right]. \quad (3)$$

Equation (3) updates the hunter position, where *x (t)* is the current hunter position, *x (t + 1)* is the hunter next position, Ppos is the prey position, $\mu$ is the mean of all positions, and $Z$ is an adaptive parameter calculated by Eq. (4)

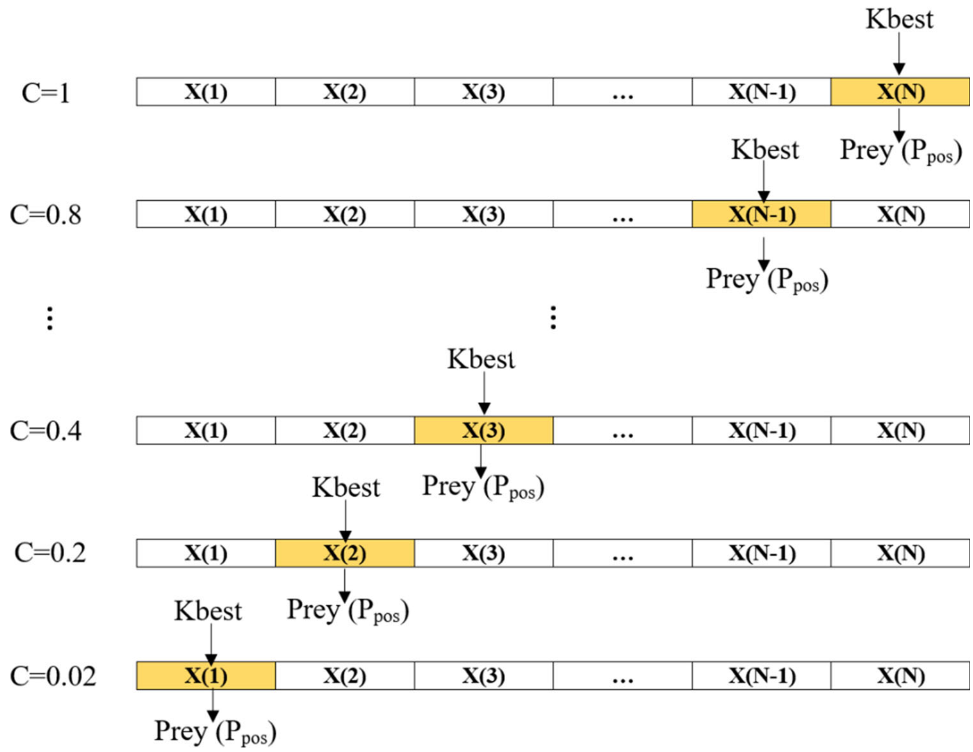



**Fig. 6** How to calculate *Kbest* and select prey ($P_{pos}$) during algorithm running

$$P = \vec{R}_1 < C\,; \quad IDX = (P == 0); \qquad (4)$$
$$Z = R_2 \otimes IDX + \vec{R}_3 \otimes (\sim IDX)$$

where $\vec{R}_1$ and $\vec{R}_3$ are random vectors in the range [0,1], P is a random vector with values 0 and 1 equal to the number of problem variables, $R_2$ is a random number in the range [0,1], and IDX is the index numbers of the vector $\vec{R}_1$ which satisfies the condition (P = = 0). The structure of parameter $Z$ and its relationship with parameter $C$ is shown in Fig. 5. It should be noted that Fig. 5 is just an example for the reader to understand. Numbers are random, and their value and order change in each iteration.

$C$ is the balance parameter between exploration and exploitation, whose value decreases from 1 to 0.02 over the course of iterations. $C$ is calculated as follows:

$$C = 1 - it\left(\frac{0.98}{MaxIt}\right) \qquad (5)$$

where it is the current iteration value and MaxIt is the maximum number of iterations. As shown in Fig. (4a), the position of the prey ($Ppos$) is calculated so that we first calculate the average of all positions ($\mu$) based on Eq. (6) and then the distance of each of the search agents from this mean position

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\vec{x}_i. \qquad (6)$$

We calculate the distance based on Euclidean distance according to Eq. (7)

$$D_{euc(i)} = \left(\sum_{j=1}^{d}(x_{i,j} - \mu_j)^2\right)^{\frac{1}{2}}. \qquad (7)$$

According to Eq. (8), the search agent with the maximum distance from the mean of positions is considered prey ($P_{pos}$)

$$\vec{P}_{pos} = \vec{x}_i | i \, is \, index \, of \, Max(end) sort(D_{euc}). \qquad (8)$$

If we always consider the search agent with the maximum distance from the average position ($\mu$) in each iteration, the algorithm will have late convergence. According to the hunting scenario, when the hunter takes the prey, the prey dies, and the next time, the hunter moves to the new prey. To solve this problem, we consider a decreasing mechanism as Eq. (9)

$$kbest = round(C \times N) \qquad (9)$$

*where* $N$ is the number of search agents.

Now, we change Eq. (8) and calculate the prey position as Eq. (10)

$$\vec{P}_{pos} = \vec{x}_i | i \, is \, sorted \, D_{euc}(kbest). \qquad (10)$$

Figure 6 shows how to calculate *Kbest* and select prey ($P_{pos}$) during algorithm running. At the beginning of the algorithm, the value of *Kbest* is equal to $N$ (number of search agents). Therefore, the last search agent that is

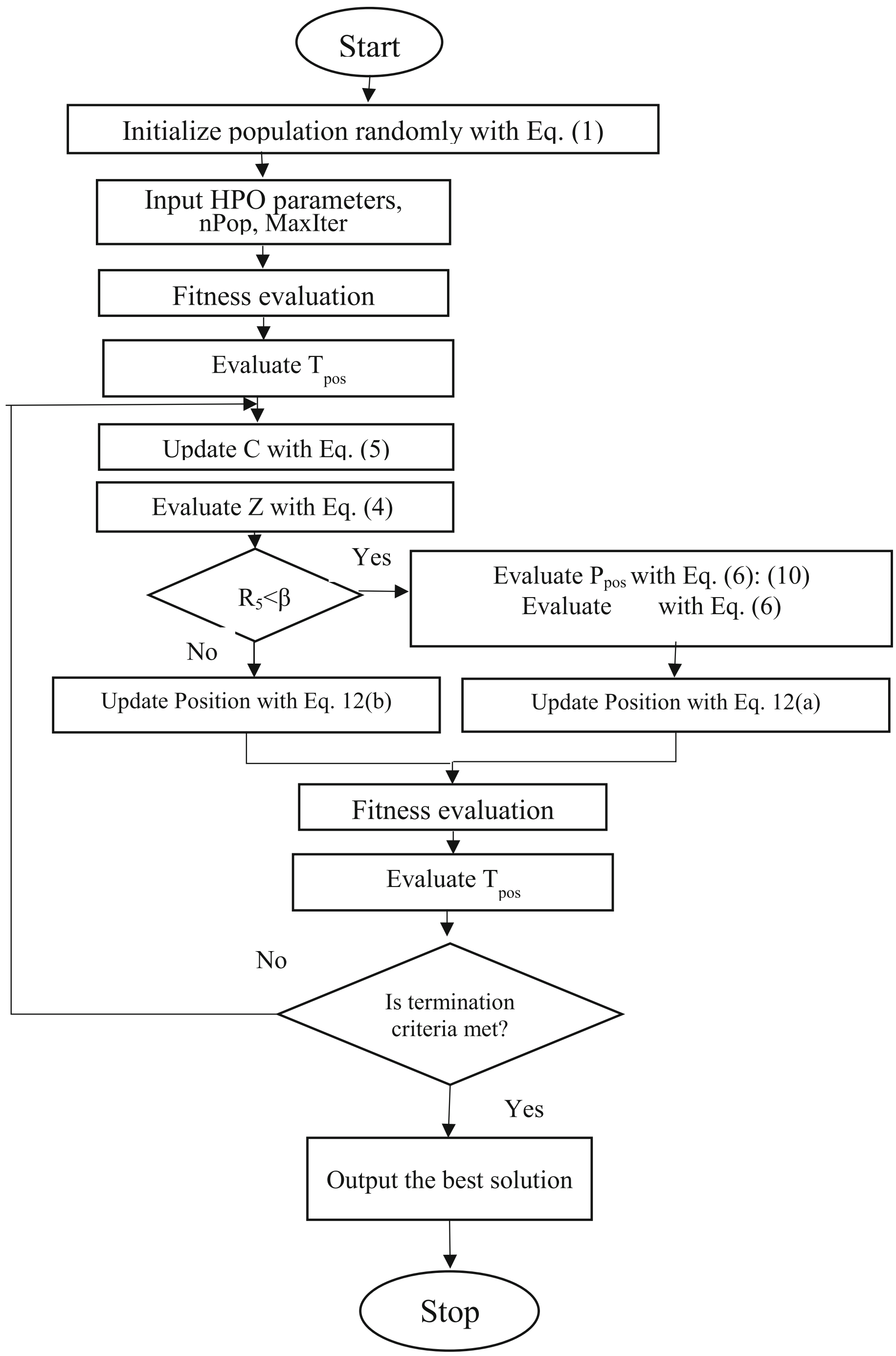



**Fig. 7** Flowchart of the hunter–prey optimization

farthest from the search agents' average position ($\mu$) is selected as prey and attacked by the hunter. As shown in Fig. 6, the *Kbest* value gradually decreases so that at the end of the algorithm, the *Kbest* value equals the first search agent (the shortest distance from the average position of the search agents ($\mu$)). It should be mentioned that the search agents are sorted in each iteration based on the distance from the search agents' average position ($\mu$).

As shown in Fig. 4b, when prey is attacked, it tries to escape and reach its safe place.

We assume that the best safe position is the optimal global position because it will give the prey a better chance of survival, and the hunter may choose another prey. Equation (6) is proposed to update the prey position

$$x_{i,j}(t+1) = T_{pos(j)} + CZ\cos(2\pi R_4) \times (T_{pos(j)} - x_{i,j}(t)) \tag{11}$$

where *x (t)* is the current position of the prey, *x (t + 1)* is the next position of the prey, Tpos is the optimum global

**Table 1** Unimodal benchmark functions

| Function | Dim | Range | $f_{MIN}$ |
|---|---|---|---|
| $f_1(x) = \sum_{i=1}^{n} x_i^2$ | 30 | [− 100,100] | 0 |
| $f_2(x) = \sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$ | 30 | [− 10,10] | 0 |
| $f_3(x) = \sum_{i=1}^{n} \left(\sum_{j-1}^{i} x_j\right)^2$ | 30 | [− 100,100] | 0 |
| $f_4(x) = \max\{\lvert x_i\rvert, 1 \le i \le n\}$ | 30 | [− 100,100] | 0 |
| $f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ | 30 | [− 30,30] | 0 |
| $f_6(x) = \sum_{i=1}^{n} ([x_i + 0.5])^2$ | 30 | [− 100,100] | 0 |
| $f_7(x) = \max\{\lvert x_i\rvert, 1 \le i \le n\}$ | 30 | [− 1.28,1.28] | 0 |

**Table 2** Multi-modal benchmark functions

| Function | Dim | range | $F_{min}$ |
|---|---|---|---|
| $F_8(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i\rvert})$ | 30 | [− 500,500] | − 418.9829*5 |
| $F_9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$ | 30 | [− 5.12,5.12] | 0 |
| $F_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)) + 20 + e$ | 30 | [− 32,32] | 0 |
| $F_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$ | 30 | [− 600,600] | 0 |
| $F_{12}(x) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4) + \sum_{i=1}^{n} u(x_i, 10, 100, 4) \quad y_i = 1 + \frac{x_i + 1}{4}$ $u(x_i, a, k, m) = \left\{\begin{array}{ll} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{array}\right\}$ | 30 | [− 50,50] | 0 |
| $F_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2[1 + \sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$ | 30 | [− 50,50] | 0 |

position, $Z$ is an adaptive parameter calculated by Eq. (4), and $R4$ is a random number in the range [−1, 1]. $C$ is the balance parameter between exploration and exploitation, whose value decreases during the algorithm's iteration. It is calculated according to Eq. (5). The COS function and its input parameter allow the next prey position to be positioned at global optimum different radials and angles and increase the exploitation phase's performance.

The question that arises here is how to choose the hunter and prey in this algorithm.

To answer this question, we combine Eqs. (3) and (11) as Eq. (12)

*where R5* is a random number in the range [0, 1], and β is a regulatory parameter whose value in this study is set to 0.1. If the *R5* value is smaller than β, the search agent is considered a hunter, and the next position of the search agent is updated with Eq. (12a); if the *R5* value is larger than β, the search agent will be considered prey, and the next position of the search agent will be updated with Eq. (12b). The flowchart of the proposed algorithm is shown in Fig. 7.

$$x_i(t+1) = \begin{cases} x_i(t) + 0.5\left[(2CZP_{pos} - x_i(t)) + (2(1-c)Z\mu - x_i(t))\right] & if\ R_5 < \beta\ (13a) \\ T_{pos} + CZ\cos(2\pi R_4) \times (T_{pos} - x_i(t)) & else\ (13b) \end{cases} \tag{12}$$

### 3.3 Assumptions of the HPO algorithm

Theoretically, the proposed HPO algorithm can provide suitable solutions to various problems for the following reasons:

- Exploring the search space by selecting the farthest search agent relative to the average search agent's position as prey.
- Exploring the search space is guaranteed by the random selection of hunter and prey and hunters' random movements around the prey.
- Due to random movements and random selection of hunter and prey, the probability of getting stuck in a local optimal is low.
- The prey selection mechanism with the highest prey distance from the average search agent position is adaptively reduced during the iterations, ensuring both algorithm convergence and HPO algorithm exploitation.
- The severity of the hunter and prey movement during the iterations is reduced by the adaptive parameter, ensuring the HPO algorithm's convergence.
- During optimization, the hunter gradually moves toward the best prey position, and the balance between the exploration and exploitation phases is maintained.
- The calculation of the adaptive parameter and the random parameter for each hunter and prey in each dimension increases the population diversity.
- Each search agent (hunter or prey) in each iteration is compared with the best solution obtained so far, and the best solution is stored.
- The hunter directs the prey to promising positions in the search space.
- The HPO algorithm is a non-gradient approximation algorithm that treats the problem as a black box.
- The number of adjustment parameters of HPO algorithm is few, and some parameters are adjusted adaptively.

### 3.4 Computational complexity analysis

In general, the complexity of calculating the HPO algorithm depends on four components, namely initialization, updating of the hunter, updating prey and fitness evaluation. Note that with N search agents, the initialization process's computational complexity is O (N). The computational complexity of the update process is $O(T \times N) + O((1-\beta) \times T \times N \times D) + O(\beta \times T \times N \times D)$, which includes updating the position vector of all prey and hunters to find the best optimal position. The computational complexity of the update process is $O(T \times N) + O((1-\beta) \times T \times N \times D) + O(\beta \times T \times N \times D)$, T denotes the maximum number of iterations, D denotes the number of problem variables, and β is a regulatory parameter whose value in this study is set to 0.1. Therefore, the total complexity of HPO is $O(N \times (T + (1-\beta) TD + \beta TN + 1))$.

**Table 3** Set parameters of algorithms

| Algorithm | Parameter | Value |
|---|---|---|
| HPO | C | $\in[1,0.02]$ |
| | $\beta$ | 0.1 |
| PSO | Social and cognitive coefficient | C1 = 2, C2 = 2 |
| | Inertial coefficient | Adaptively decreases from 0.9 to 0.4 |
| WOA | a | $\in[0,2]$ |
| | a2 | $\in[-1,-2]$ |
| HHO | $\beta$ | 1.5 |
| | $E_0$ | $\in[-1,-1]$ |
| TSA | $P_{min}$ | 1 |
| | $P_{max}$ | 4 |
| LFD | Threshold | 2 |
| | CSV | 0.5 |
| | $\beta$ | 1.5 |
| | $\alpha_1,\alpha_2,\alpha_3$ | 10,0.00005,0.005 |
| | $\partial_1,\partial_2$ | 0.9,0.1 |
| ALO | r(t) | $\in[0,1]$ |

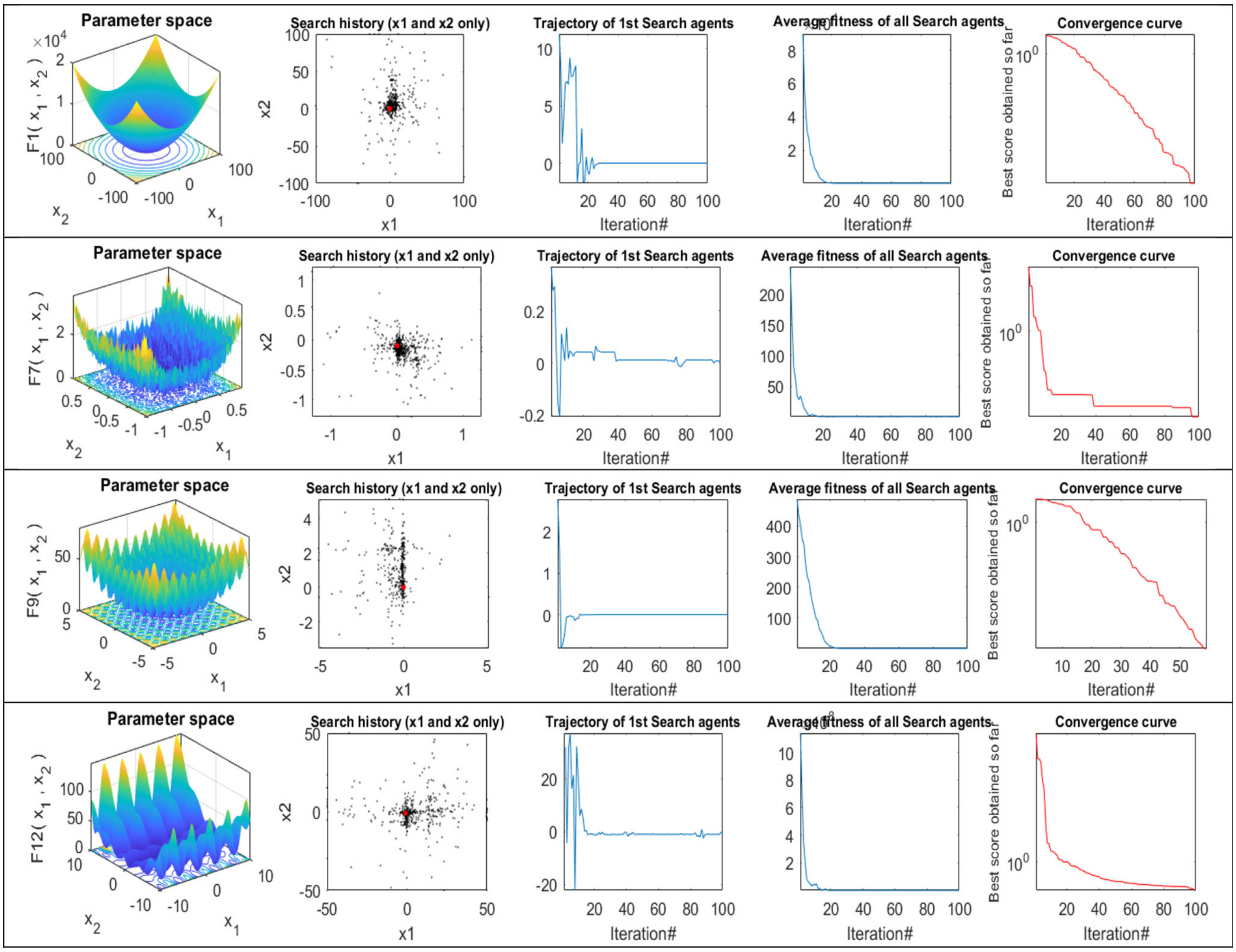



**Fig. 8** Convergence behavior and search history of the proposed HPO algorithm

## 4 Results and discussion

In this section, the HPO algorithm is evaluated on 43 test functions. The HPO has been compared to advanced swarm-based optimization algorithms. In general, benchmark functions can be divided into four groups such as unimodal, multi-modal, hybrid functions and composition function. The first 13 benchmark functions are the classical functions used by many researchers (Mirjalili 2016; Saremi et al. 2017). From these 13 classical functions, the first seven are unimodal, and the second six are multi-modal. The unimodal functions (f1–f7) are suitable for determining algorithms' exploitation because they have a global optimum and no local optimum. Multi-modal functions (f8–13) have many local optimal and are useful for examining the exploration and avoiding the algorithms' local optimal. These benchmark functions are given in Tables 1 and 2, where *DIM* represents the dimensions of the function, *RANGE* is the boundary of the function's search space, and *FMIN* is the optimal value. The hybrid and composite functions are a combination of different unimodal and multi-modal test functions, rotary and displacement, from the CEC2017 session (Awad et al. 2017). These functions' search space is very challenging; they are very similar to real search spaces and are useful for evaluating algorithms in terms of the balance of exploration and exploitation. To further evaluate the proposed algorithm, the HPO algorithm is implemented on nine real engineering problems such as rolling element bearing problem, reducer design problem, cantilever beam design, multi-plate disk clutch brake, welded beam, three-bar truss, step-cone pulley problem, pressure vessel designs and tension/compression spring. For justly comparison, the algorithm runs 30 times. The Wilcoxon statistical test will examine the null hypothesis that two populations are equal from the same distribution. The similarity objective can be used to determine whether two sets of solutions are statistically different. The result obtained by using the Wilcoxon statistical test is a parameter called p-value, which

**Table 4** Results for the unimodal test functions

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F1 | min | 4.1929e-08 | 9.1868e-86 | 2.0758e-04 | 1.5743e-07 | 1.5625e-23 | 9.6839e-117 | **1.3483e-186** |
| | max | 4.3606e-05 | 1.3513e-72 | 0.0036 | 5.3000e-07 | 3.8843e-20 | 3.0576e-92 | **1.1673e-167** |
| | avg | 2.7368e-06 | 7.2644e-74 | 0.0010 | 3.1904e-07 | 3.5086e-21 | 1.1030e-93 | **3.8960e-169** |
| | std | 7.8979e-06 | 2.6139e-73 | 8.3478e-04 | 7.0385e-08 | 7.5860e-21 | 5.5816e-93 | **0** |
| F2 | min | 8.7537e-05 | 7.9595e-56 | 0.8653 | 2.2253e-04 | 1.0225e-14 | 4.0014e-60 | **4.2018e-99** |
| | max | 0.0228 | 1.8836e-49 | 129.1467 | 5.0105e-04 | 1.0057e-12 | 7.4750e-48 | **6.0415e-91** |
| | avg | 0.0023 | 1.2605e-50 | 43.2560 | 3.3647e-04 | 1.1593e-13 | 2.5192e-49 | **2.9108e-92** |
| | std | 0.0042 | 3.9289e-50 | 47.8404 | 6.0363e-05 | 1.8668e-13 | 1.3643e-48 | **1.1177e-91** |
| F3 | min | 48.6588 | 1.5867e + 04 | 657.0275 | 5.0909e-07 | 1.4390e-07 | 6.0866e-102 | **2.4279e-161** |
| | max | 676.5631 | 8.5120e + 04 | 1.2392e + 04 | 2.8215e-06 | 0.0059 | 5.4466e-70 | **2.0432e-142** |
| | avg | 236.4627 | 4.6712e + 04 | 4.4373e + 03 | 1.3700e-06 | 6.1660e-04 | 1.8158e-71 | **6.8323e-144** |
| | std | 147.1275 | 1.5757e + 04 | 2.5847e + 03 | 5.0401e-07 | 0.0013 | 9.9440e-71 | **3.7299e-143** |
| F4 | min | 1.0616 | 0.8494 | 8.2454 | 2.8538e-04 | 0.0049 | 6.0051e-57 | **2.4541e-83** |
| | max | 4.5613 | 89.0149 | 35.5695 | 4.7177e-04 | 1.3984 | 3.4493e-46 | **8.2385e-76** |
| | avg | 2.6549 | 47.0941 | 16.6688 | 3.4946e-04 | 0.3257 | 1.3491e-47 | **5.4057e-77** |
| | std | 0.9912 | 28.9341 | 5.2473 | 4.5172e-05 | 0.3510 | 6.3104e-47 | **1.6203e-76** |
| F5 | min | 8.7916 | 27.2605 | 26.2370 | 27.8536 | 26.1806 | **7.1629e-07** | 22.8180 |
| | max | 193.4814 | 28.7582 | 2.0738e + 03 | 28.2583 | 30.3516 | **0.0895** | 25.9952 |
| | avg | 48.5377 | 28.0400 | 330.3476 | 28.0628 | 28.3860 | **0.0171** | 23.7090 |
| | std | 41.4386 | 0.4365 | 535.2123 | 0.1291 | 0.8620 | **0.0223** | 0.7436 |
| F6 | min | 6.8884e-08 | 0.0846 | 1.9036e-04 | 0.8371 | 2.8156 | 7.7275e-07 | **6.3266e-10** |
| | max | 1.3346e-05 | 1.1841 | 0.0027 | 2.1610 | 5.3077 | 9.4810e-04 | **2.2300e-06** |
| | avg | 1.5340e-06 | 0.3742 | 0.0012 | 1.7965 | 3.8706 | 1.2635e-04 | **1.2538e-07** |
| | std | 2.7959e-06 | 0.2228 | 7.4550e-04 | 0.3084 | 0.6509 | 2.3701e-04 | **4.1350e-07** |
| F7 | min | 0.0118 | 1.4784e-04 | 0.1053 | 0.2353 | 0.0037 | **1.7007e-06** | 9.1589e-06 |
| | max | 0.0522 | 0.0200 | 0.4769 | 2.8537 | 0.0249 | **3.9630e-04** | 0.0020 |
| | avg | 0.0252 | 0.0036 | 0.2518 | 1.0900 | 0.0109 | **1.3213e-04** | 3.3854e-04 |
| | std | 0.0091 | 0.0040 | 0.0917 | 0.5959 | 0.0046 | **1.0620e-04** | 4.1809e-04 |
| Friedman mean rank | | 5 | 4.29 | 6.14 | 4.71 | 4.57 | 2 | 1.29 |
| Rank | | 6 | 3 | 7 | 5 | 4 | 2 | 1 |

measures the significance level of the algorithms. In other words, if the p-value is less than 0.05, the two algorithms are statistically significant. A nonparametric Wilcoxon statistical test was performed on 30 runs to obtain statistically significant conclusions. Such statistical tests should be performed considering the meta-heuristic algorithms' random nature (García et al. 2009; Derrac et al. 2011).

## 4.1 Experimental setup

The experimentations were run on a PC with a Windows 10 64-bit professional and 12 GB RAM. The algorithms were implemented by MATLAB R2017b. The maximum iteration and population size in all methods are 500 and 30, respectively. For a fair comparison, all methods are run 30 times independently and then compared based on statistical indicators such as minimum, maximum, average and standard deviation. To verify performance, the proposed algorithm is compared with famous and new algorithms including Harris hawks optimizer, particle swarm optimization, ant lion optimizer, whale optimization algorithm, Lévy flight distribution and tunicate swarm algorithm. The parameters of the algorithms are set according to Table 3.

## 4.2 Convergence analysis of the proposed HPO algorithm

In optimization techniques, the search agents have sudden and abrupt movements at the beginning of the search process (exploration) and gradually shrinking their motions (exploitation) (van den Bergh and Engelbrecht 2006). Figure 8 shows the convergence behavior of the proposed

**Table 5** P-values of the Wilcoxon test overall runs and number function evaluation (NFE)

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F1 | p-value | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 30,649 | 15,000 |
| F2 | p-value | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 30,002 | 15,000 |
| F3 | p-value | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 34,045 | 15,000 |
| F4 | p-value | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 33,416 | 15,000 |
| F5 | p-value | 1.2477e-04 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 36,756 | 15,000 |
| F6 | p-value | 1.5581e-08 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.3384e-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 36,114 | 15,000 |
| F7 | p-value | 3.020E-11 | 2.3897e-08 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 0.0156 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 33,562 | 15,000 |

HPO algorithm, the first column showing the three-dimensional figure of the test functions. The search history of the search agents (only in the first and second dimensions of the solution) is shown in the second column. It can be seen that while dealing with different cases, HPO exhibits a similar pattern, in which search agents attempt to maximize diversity in exploring desirable areas and then exploit around best areas. The third column shows the search paths. In the early stages of the algorithm, the movements of the search agents are sudden, and gradually the movements become slower, so that in the final stages of the algorithm, the search agents gather at one point. This is due to the comparative reduction in prey selection based on Eq. (9), making the search agents gradually converge to the best point during the iterations. This behavior ensures that the HPO algorithm changes during optimization from the exploration phase to the exploitation phase. The fourth column in Fig. 8 shows all search agents' average fitness in each iteration and shows that the other search agents behave as the first search agent. The fifth column is the convergence curve that represents the best value of each iteration. In this graph, fast convergence can also be seen.

### 4.3 HPO algorithm exploitation analysis

The results in Table 4 shows that the hunter–prey optimizer performs better in most of the unimodal functions (F1, F2, F3, F4 and F6) than the other algorithms compared in Table 4. The HPO performed better in F5 and F7 compared to others except for the HHO algorithm. The p-values and number function evaluation (NFE) in Table 5 also prove that the proposed algorithm's superiority is significant in most cases. The point to be noted is that the HHO algorithm achieved these results with about 30,000 number function evaluations (NFE), while the HPO algorithm with 15,000 number function evaluations (NFE). This demonstrates the exploitation ability of the HPO algorithm due to the movement of prey toward the best position. An example of this is shown in Fig. 9.

### 4.4 HPO algorithm exploration analysis

The results in Table 6 indicate that the HPO method provides competitive results compared with other methods. In the F9, F10, F11 and F12 test functions, the proposed HPO algorithm performs better than others. The p-values in Table 7 also prove that the superiority of the HPO algorithm is significant in most cases. This indicates that the HPO method is capable of exploration, and this is due to the fact that the hunter moves toward a prey that is far from the group. This allows search agents (hunters) to explore different areas of the search space.

The results in Table 6 for the F9, F10 and F11 functions reveal that the proposed HPO algorithm and the HHO algorithm have obtained the same results. The convergence diagrams of these functions in Fig. 9 indicate that the HPO algorithm has reached the optimal point in fewer iterations compared to the HHO algorithm. Also, the number of NFEs of the HPO algorithm is less than half of the HHO algorithm. This shows that the HPO algorithm performs much better than the HHO.

### 4.5 Scalability analysis of the HPO algorithm

This part explains the effect of scalability on different test functions by using proposed HPO algorithm. The dimensions of the test functions varies as 10, 30, 50 and 100. The scalability results for all methods are also shown in Fig. 10. This reveals the impact of dimension on the quality of solutions for the HPO algorithm to diagnose its efficacy for

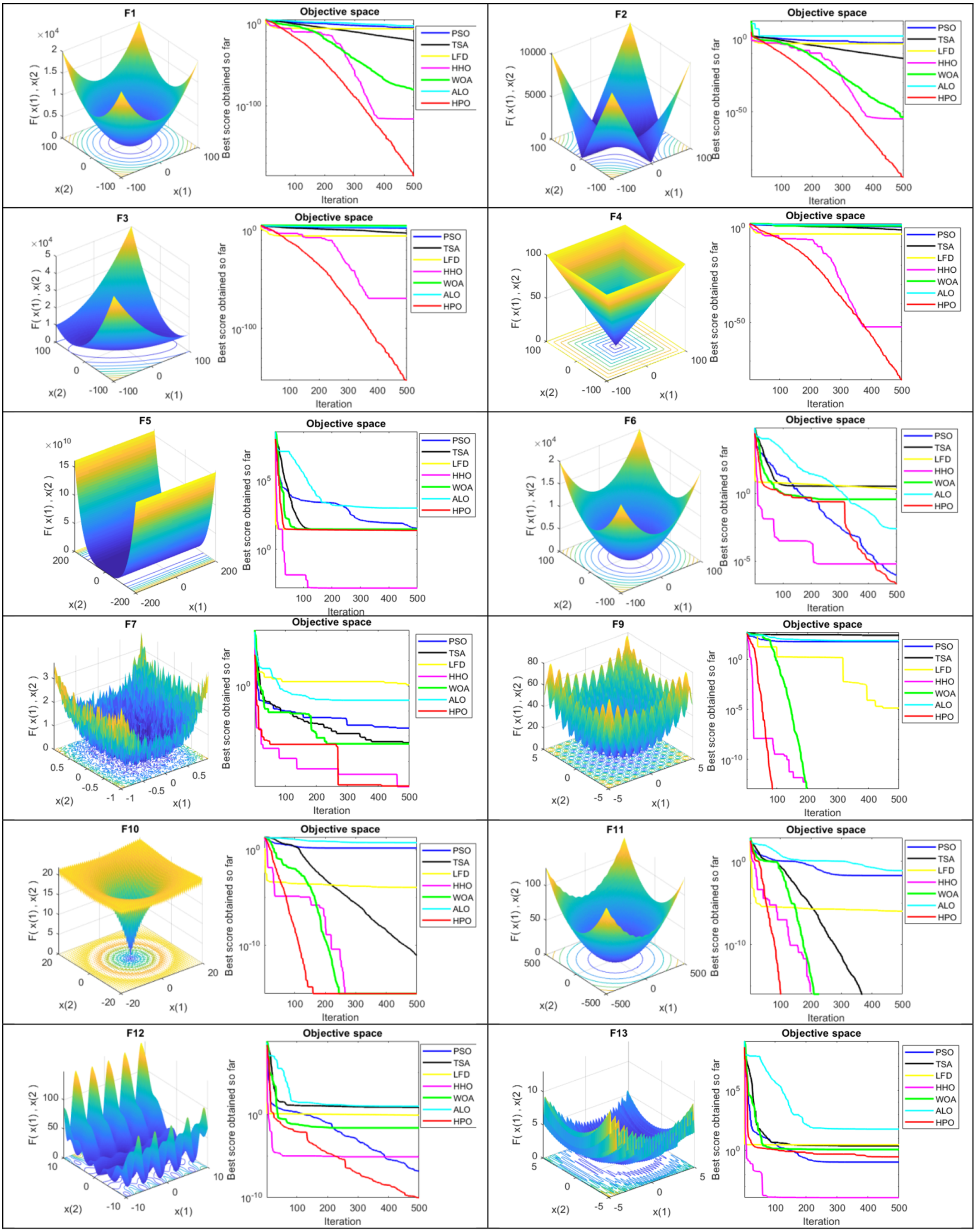



**Fig. 9** Convergence curve of the methods on classical functions

Table 6 Results for the multi-modal benchmark functions

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F8 | min | − 7.8120e + 03 | − 1.2565e + 04 | − 8.5214e + 03 | **− 5.0651e + 03** | − 6.9100e + 03 | − 1.2569e + 04 | − 9.7233e + 03 |
| | max | − 4.9494e + 03 | − 7.0219e + 03 | − 5.4177e + 03 | **− 3.2773e + 03** | − 4.3158e + 03 | − 1.2565e + 04 | − 7.3509e + 03 |
| | avg | − 6.2641e + 03 | − 1.0020e + 04 | − 5.6298e + 03 | **− 4.0512e + 03** | − 5.9093e + 03 | − 1.2569e + 04 | − 8.8439e + 03 |
| | std | 773.1355 | 1.7649e + 03 | 598.9912 | **413.8118** | 562.0738 | 0.9315 | 598.0833 |
| F9 | min | 21.8891 | 0 | 39.7995 | 9.5044e-08 | 115.9126 | 0 | **0** |
| | max | 67.6571 | 5.6843e-14 | 151.2331 | 2.2914e-05 | 260.9939 | 0 | **0** |
| | avg | 46.5640 | 1.8948e-15 | 81.3609 | 5.1798e-06 | 184.6362 | 0 | **0** |
| | std | 10.1715 | 1.0378e-14 | 24.3112 | 5.4679e-06 | 40.2045 | 0 | **0** |
| F10 | min | 1.5447e-04 | 8.8818e-16 | 1.3414 | 9.7034e-05 | 2.0579e-12 | 8.8818e-16 | **8.8818e-16** |
| | max | 2.7383 | 7.9936e-15 | 13.2214 | 1.8362e-04 | 4.5968 | 8.8818e-16 | **8.8818e-16** |
| | avg | 1.1543 | 3.8488e-15 | 4.3800 | 1.2910e-04 | 1.2947 | 8.8818e-16 | **8.8818e-16** |
| | std | 0.8514 | 2.8119e-15 | 2.7934 | 2.0618e-05 | 1.6500 | 0 | **0** |
| F11 | min | 2.7242e-07 | 0 | 0.0167 | 4.2184e-07 | 0 | 0 | **0** |
| | max | 0.1350 | 0 | 0.1202 | 1.3533e-06 | 0.0211 | 0 | **0** |
| | avg | 0.0231 | 0 | 0.0625 | 8.5391e-07 | 0.0053 | 0 | **0** |
| | std | 0.0281 | 0 | 0.0265 | 2.2683e-07 | 0.0074 | 0 | **0** |
| F12 | min | 9.7696e-09 | 0.0060 | 6.4120 | 0.5363 | 1.3866 | 1.8766e-08 | **5.4631e-11** |
| | max | 1.5771 | 0.1155 | 36.9798 | 1.0026 | 17.3209 | 5.4219e-05 | **1.1089e-08** |
| | avg | 0.2156 | 0.0289 | 13.5841 | 0.7195 | 9.2604 | 6.4135e-06 | **8.1369e-10** |
| | std | 0.3789 | 0.0233 | 6.8544 | 0.1128 | 4.1775 | 1.0672e-05 | **2.0305e-09** |
| F13 | min | 1.7055e-08 | 0.1375 | 0.1117 | 2.8223 | 1.7243 | 1.1862e-06 | **6.4274e-09** |
| | max | 0.5407 | 1.2581 | 64.2053 | 2.9661 | 4.2009 | **0.0012** | 0.5926 |
| | avg | 0.0297 | 0.4814 | 30.1711 | 2.9581 | 2.9450 | **1.1887e-04** | 0.1478 |
| | std | 0.1000 | 0.2599 | 17.9875 | 0.0309 | 0.6237 | **2.2897e-04** | 0.1409 |
| Friedman mean rank | | 3.29 | 2.72 | 4.57 | 2.71 | 4 | 1.72 | 1.71 |
| Rank | | 5 | 4 | 7 | 3 | 6 | 2 | 1 |

Table 7 P-values of the Wilcoxon test overall runs and number function evaluation (NFE)

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F8 | p-value | 4.9752e-11 | 0.0392 | 4.7456e-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 37,137 | 15,000 |
| F9 | p-value | 1.2118e-12 | 0.3337 | 1.2118e-12 | 1.2118e-12 | 1.2118e-12 | NaN | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 36,775 | 15,000 |
| F10 | p-value | 1.2118e-12 | 7.6453e-07 | 1.2118e-12 | 1.2118e-12 | 1.2118e-12 | NaN | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 36,219 | 15,000 |
| F11 | p-value | 1.2118e-12 | NaN | 1.2118e-12 | 1.2118e-12 | 2.9329e-05 | NaN | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 36,280 | 15,000 |
| F12 | p-value | 3.6897e-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | 3.020E-11 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 36,358 | 15,000 |
| F13 | p-value | 2.6806e-04 | 1.3594e-07 | 7.3891e-11 | 3.020E-11 | 3.020E-11 | 6.3560e-05 | – |
| | nfe | 15,000 | 15,000 | 15,000 | 15,030 | 15,000 | 36,274 | 15,000 |

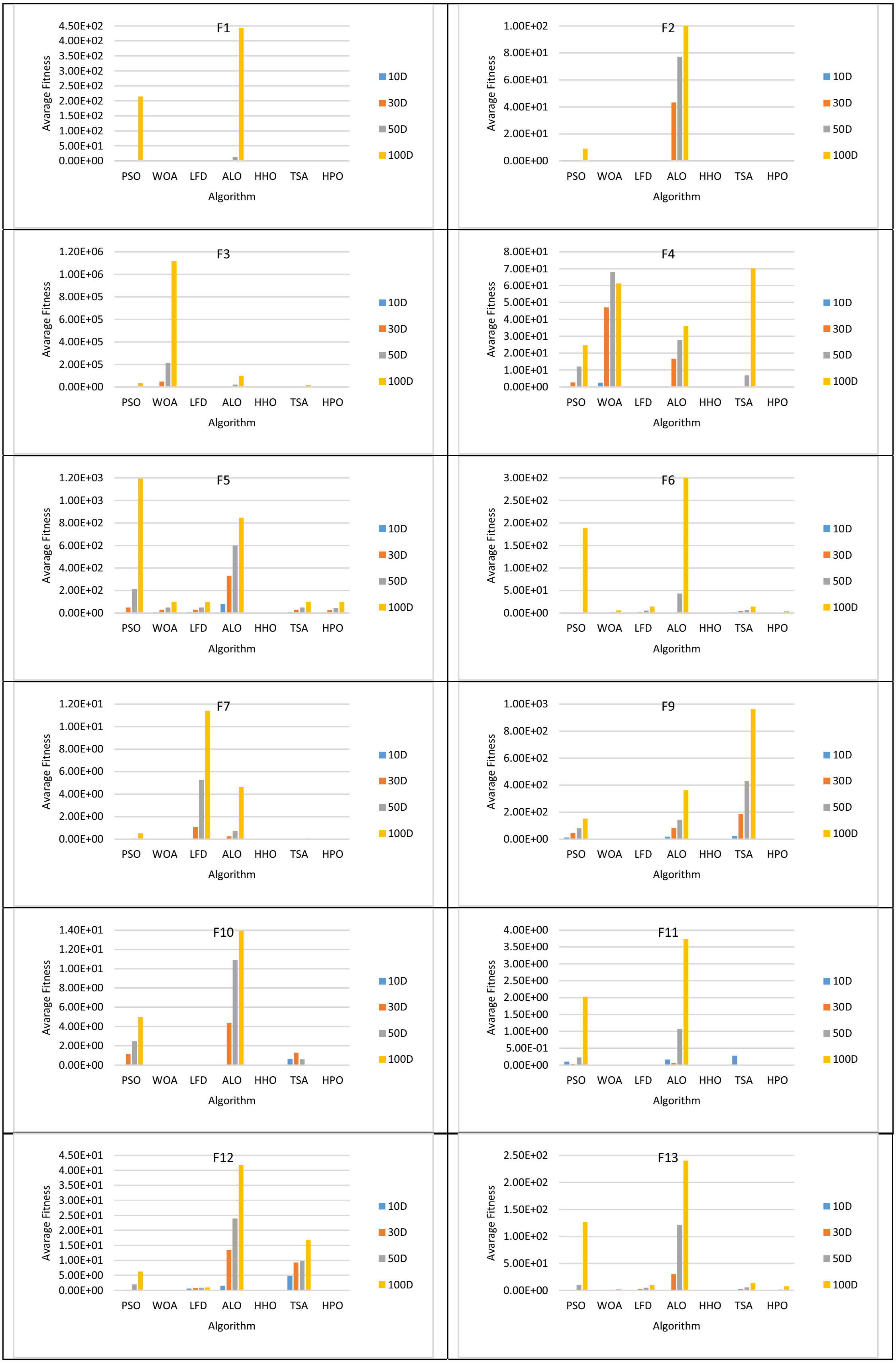



**Fig. 10** Scalability of different methods on classical functions

**Table 8** Comparison of average running time (with 1000 dimensions and 30 independent runs)

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F1 | avg | 0.8868 | 1.6677 | 477.0267 | 18.4953 | 1.8273 | **0.7652** | 1.1179 |
| | std | 0.0350 | 0.0806 | 3.2761 | 0.1376 | 0.0549 | **0.0839** | 0.0491 |
| F2 | avg | 0.8968 | 1.6666 | 476.6731 | 3.2290 | 1.8667 | **0.7750** | 1.1621 |
| | std | 0.0390 | 0.0544 | 3.1742 | 0.0806 | 0.0249 | **0.0335** | 0.0597 |
| F3 | avg | **24.0046** | 25.0240 | 502.2110 | 41.9845 | 25.4720 | 55.1191 | 24.3754 |
| | std | **0.1258** | 0.2702 | 2.3877 | 0.1457 | 0.1650 | 2.3150 | 0.1582 |
| F4 | avg | **0.8810** | 1.6398 | 479.5388 | 18.6519 | 2.0522 | 0.9421 | 1.1163 |
| | std | **0.0538** | 0.0439 | 2.2317 | 0.1377 | 0.1818 | 0.0444 | 0.0654 |
| F5 | avg | **0.9615** | 1.7738 | 480.0975 | 18.2776 | 1.8925 | 1.3174 | 1.1857 |
| | std | **0.0824** | 0.0919 | 1.9048 | 0.0500 | 0.0365 | 0.0651 | 0.0331 |
| F6 | avg | **0.8961** | 1.6361 | 475.6420 | 18.5080 | 2.0374 | 1.1213 | 1.1211 |
| | std | **0.0356** | 0.0406 | 1.9587 | 0.1201 | 0.0979 | 0.0277 | 0.0460 |
| F7 | avg | **2.2686** | 2.7341 | 477.3803 | 33.8539 | 2.9844 | 3.4807 | 2.3877 |
| | std | **0.1341** | 0.0449 | 2.1679 | 0.1937 | 0.0284 | 0.0589 | 0.0711 |
| F8 | avg | **1.4018** | 1.9342 | 479.8049 | 18.1480 | 2.2017 | 2.1021 | 1.5796 |
| | std | **0.0581** | 0.0739 | 1.9587 | 0.2370 | 0.0479 | 0.0544 | 0.0258 |
| F9 | avg | 1.3984 | 1.7663 | 477.0510 | 8.2852 | 2.0892 | 1.6453 | **1.3559** |
| | std | 0.1611 | 0.0736 | 1.5111 | 0.1298 | 0.0470 | 0.0637 | **0.0490** |
| F10 | avg | **1.2925** | 1.7626 | 477.5186 | 18.8905 | 2.1152 | 1.6899 | 1.3732 |
| | std | **0.0405** | 0.0220 | 1.4021 | 0.1299 | 0.0847 | 0.0683 | 0.0405 |
| F11 | avg | **1.4080** | 1.9048 | 478.0581 | 19.2788 | 2.1679 | 1.9587 | 1.5241 |
| | std | **0.0550** | 0.0582 | 1.7926 | 0.1010 | 0.0493 | 0.0460 | 0.0370 |
| F12 | avg | **3.7003** | 4.2255 | 479.8028 | 21.4206 | 4.5373 | 7.5103 | 3.8722 |
| | std | **0.0582** | 0.0463 | 2.0802 | 0.1860 | 0.0724 | 0.0437 | 0.0629 |
| F13 | avg | **3.7426** | 4.4563 | 481.2474 | 21.2718 | 4.5273 | 7.5211 | 3.8573 |
| | std | **0.0340** | 0.3063 | 2.0844 | 0.1397 | 0.0494 | 0.0667 | 0.2430 |

problems with lower dimensions and higher dimension. This is due to the better capability of the proposed HPO for balancing between exploitation and exploration.

To more scrutinize the performance of HPO algorithm, the algorithms were tested on classical functions (F1–F13) with high dimensions (1000 variables), and the execution time of each algorithm was calculated and given in Table 8. Looking at the results in Table 8, it can be seen that the proposed HPO algorithm showed competitive and logical performance in solving high-dimensional classical functions than other algorithms. According to the results of the average execution time of algorithms on classical functions, the HPO performs faster than WOA, ALO, LFD and TSA algorithms. The HPO algorithm performed faster than the HHO algorithm in all functions except F 1, F 2 and F 4.

## 4.6 The performance of the hunter and prey optimizer on the CEC2017

In this section, we evaluate the performance of the proposed HPO algorithm on the CEC2017 challenging function set. The CEC2017 set includes 30 functions, of which the first ten are unimodal and multi-modal functions, the second ten are hybrid functions, and the last ten are composition functions (Awad et al. 2017). Details of these functions are given in Table 9.

The hunter–prey optimization (HPO) was tested on CEC2017 test functions and compared with whale optimization algorithm (WOA), particle swarm optimization (PSO), ant lion optimizer (ALO), Harris hawks optimizer (HHO), tunicate swarm algorithm (TSA) and Lévy flight distribution (LFD). Each algorithm tested 30 times with 60 search agents and 1000 iterations. The dimensions of all functions are considered 10. The functions were divided into three groups, such as unimodal–multi-modal, hybrid and composition. The result of unimodal and multi-modal functions given in Table 10. The Friedman test is used to find the differences in treatments or algorithms across multiple test attempts. This test ranks the data within each row (or block) and tests for a difference across columns. We adopt Friedman test to compare the comprehensive performance of every algorithm on each group of problems of CEC 2017 benchmark (Derrac et al. 2011). Hence, the Friedman test method allows us to determine which

**Table 9** CEC 2017 test functions (Range = [-100, 100], Dimension = 10)

| Type | NO | Functions | Global min |
|---|---|---|---|
| Unimodal function | F1 | Bent cigar function (shifted and rotated) | 100 |
| | F2 | Sum of different power function (shifted and rotated) | 200 |
| | F3 | Zakharov function (shifted and rotated) | 300 |
| Multi-modal functions | F4 | Rosenbrock's function (shifted and rotated) | 400 |
| | F5 | Rastrigin's function (shifted and rotated) | 500 |
| | F6 | Expanded Schaffer's function (shifted and rotated) | 600 |
| | F7 | Lunacek bi-Rastrigin function (shifted and rotated) | 700 |
| | F8 | Non-continuous Rastrigin's function (shifted and rotated) | 800 |
| | F9 | Levy function (shifted and rotated) | 900 |
| | F10 | Schwefel's function (shifted and rotated) | 1000 |
| Hybrid functions | F11 | Rastrigin's, Rosenbrock and Zakharov | 1100 |
| | F12 | Bent cigar, modified Schwefel and high-conditioned elliptic | 1200 |
| | F13 | Lunache bi-Rastrigin, Rosenbrock and Bent ciagr | 1300 |
| | F14 | Rastrigin, Schaffer, Ackley and elliptic | 1400 |
| | F15 | Bent cigar, HGBat, Rastrigin and Rosenbrock | 1500 |
| | F16 | Modified Schwefel, Rosenbrock, HGBat and expanded Schaffer | 1600 |
| | F17 | Rastrigin, modified Schwefel, expanded Griewank plus Rosenbrock, Ackley and Katsuura | 1700 |
| | F18 | Discus, HGBat, Rastrigin, Ackley and high-conditioned elliptic | 1800 |
| | F19 | Expanded Schaffer, Weierstrass, expanded Grienwank plus Rosenbrock, Rastrigin and bent cigar | 1900 |
| | F20 | Schaffer, modified Schwefel, Rastrigin, Ackley, Katsuura and Happycat | 2000 |
| Composition functions | F21 | Rastrigin, high-conditioned elliptic and Rosenbrock | 2100 |
| | F22 | Modified Schwefel's and Rastrigin's, Griewank's | 2200 |
| | F23 | Rastrigin, modified Schwefel, Ackley and Rosenbrock | 2300 |
| | F24 | Rastrigin, Girewank, high-conditioned elliptic and Ackley | 2400 |
| | F25 | Rosenbrock, discus, Ackley, Happycat and Rastrigin | 2500 |
| | F26 | Rastrigin, Rosenbrock, Griewank, modified Schwefel and expanded Scaffer | 2600 |
| | F27 | Expanded Schaffer, high-conditioned elliptic, bent cigar, modified Schwefel, Rastrigin and HGBat | 2700 |
| | F28 | Expanded Schaffer, HappyCat, Rosenbrock, Griewank, discus and Ackley | 2800 |
| | F29 | Lunacek bi-Rastrigin, expanded Schaffer and shifted and rotated Rastrigin | 2900 |
| | F30 | Levy function, non-continuous Rastrigin and shifted and rotated Rastrigin | 3000 |

algorithms are significantly better/worse. The results in Table 10 show that the PSO algorithm performed better than the other algorithms. However, the HPO algorithm ranks second, and it was able to discover the closest optimal value in most functions.

The algorithms were implemented with the same conditions of the first group on the functions of the second group of CEC2017, which are hybrids. The results of this experiment are shown in Table 11. From the results of Table 11, it can be seen that the HPO algorithm has the best performance in most hybrid functions compared to the PSO, WOA, ALO, LFD, TSA and HHO algorithms. The proposed algorithm was able to take the first rank in solving the hybrid functions of CEC2017. This shows the HPO algorithm has a good balance between exploration and exploitation phase.

Table 12 shows the results of the implementation of algorithms on the third group of functions that are composition. The results show that the HPO algorithm and PSO algorithm show better performance than other algorithms on the composition functions. However, the proposed algorithm is ranked first.

The nonparametric Wilcoxon statistical test can effectively evaluate the overall performance of algorithms. The Wilcoxon statistical test was performed at 95% significance level (a = 0.05) to detect significant differences between the results obtained from different algorithms. The results of the Wilcoxon statistical test are shown in Table 13.

The convergence curve of the methods in all 30 functions is shown in Fig. 11. It can be found that in most

**Table 10** Results for the CEC2017 test functions (unimodal and multi-modal)

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F1 | min | 100.3335 | 6.6725e + 04 | 8.8078e + 09 | 2.0455e + 09 | 7.6935e + 06 | 6.7222e + 04 | **100.1119** |
| | max | **1.1318e + 04** | 1.4432e + 07 | 3.3940e + 10 | 7.0880e + 09 | 7.7544e + 09 | 6.8564e + 05 | 1.2741e + 04 |
| | avg | **1.4407e + 03** | 1.2024e + 06 | 1.9224e + 10 | 3.8239e + 09 | 1.7501e + 09 | 3.0476e + 05 | 5.6629e + 03 |
| | std | **2.1777e + 03** | 2.6326e + 06 | 6.2907e + 09 | 1.1656e + 09 | 1.9578e + 09 | 1.7304e + 05 | 4.5371e + 03 |
| F2 | min | **200** | 1104 | 3.7030e + 10 | 9,934,158 | 8064 | 200 | **200** |
| | max | **200** | 2,446,619 | 1.7844e + 16 | 1.1117e + 10 | 3.1292e + 10 | 15,268 | 201 |
| | avg | **200** | 143,220 | 7.1056e + 14 | 2.6541e + 09 | 5.0113e + 09 | 1.9650e + 03 | 200.1000 |
| | std | **0** | 4.4677e + 05 | 3.2502e + 15 | 3.1110e + 09 | 8.0403e + 09 | 4.0898e + 03 | 0.3051 |
| F3 | min | **300** | 328.1045 | 2.5125e + 04 | 5.2120e + 03 | 526.4445 | 300.1734 | 300.0000 |
| | max | **300.0000** | 2.1768e + 03 | 2.8661e + 06 | 1.8234e + 04 | 1.8664e + 04 | 302.7382 | 300.0000 |
| | avg | **300** | 882.0293 | 2.7020e + 05 | 1.2784e + 04 | 1.0477e + 04 | 301.2657 | 300.0000 |
| | std | **3.9495e-14** | 553.4807 | 5.9332e + 05 | 3.2152e + 03 | 5.4428e + 03 | 0.6645 | 8.6024e-12 |
| F4 | min | 400.2036 | 403.5653 | 1.1264e + 03 | 487.8636 | 402.5429 | **400.0196** | 400.1992 |
| | max | 408.3034 | 572.4386 | 3.4610e + 03 | 773.1190 | 792.4691 | 488.0464 | **400.8251** |
| | avg | 403.7248 | 424.1228 | 2.3370e + 03 | 650.8894 | 494.0990 | 423.0511 | **400.3703** |
| | std | 2.9488 | 39.2210 | 654.5091 | 68.9828 | 88.3054 | 31.0169 | **0.1613** |
| F5 | min | 515.9193 | 519.7263 | 594.0551 | 539.8437 | 524.3893 | 526.4103 | **503.9798** |
| | max | 559.6973 | 605.5651 | 698.7312 | 575.2589 | 622.8365 | 575.0000 | **549.7477** |
| | avg | 532.6014 | 549.3775 | 647.9928 | 560.4726 | 556.8053 | 551.1613 | **526.4024** |
| | std | 10.4046 | 22.1945 | 26.6382 | 9.0874 | 24.8914 | 14.5799 | **10.3670** |
| F6 | min | **600** | 617.3973 | 658.0840 | 618.4575 | 608.3568 | 609.7232 | 600.0000 |
| | max | 622.0582 | 666.1594 | 726.7402 | 646.1767 | 666.9672 | 655.2942 | **612.8558** |
| | avg | 606.6721 | 633.7535 | 698.2065 | 634.8130 | 628.0914 | 631.2847 | **602.3421** |
| | std | 6.5948 | 12.3247 | 16.9382 | 5.4716 | 13.3051 | 12.2824 | **3.1498** |
| F7 | min | **714.0140** | 736.0757 | 921.5228 | 790.5101 | 740.1315 | 753.7464 | 725.6064 |
| | max | **741.4200** | 854.7155 | 1.2984e + 03 | 833.3167 | 846.8011 | 827.0057 | 768.2235 |
| | avg | **726.4628** | 778.2208 | 1.1531e + 03 | 813.2098 | 783.1846 | 780.0214 | 740.9441 |
| | std | **7.1699** | 31.2622 | 85.5322 | 9.7414 | 23.5888 | 19.5062 | 11.6220 |
| F8 | min | **808.9546** | 815.9955 | 891.0628 | 840.9710 | 818.1020 | 817.0544 | 805.9698 |
| | max | **836.8134** | 880.6324 | 958.5254 | 873.6454 | 862.7627 | 847.1642 | 841.7881 |
| | avg | **817.4118** | 844.9085 | 936.6075 | 860.0993 | 838.0162 | 829.8183 | 823.6945 |
| | std | **6.6449** | 19.3296 | 16.5164 | 8.6212 | 11.9368 | 6.9328 | 9.6310 |
| F9 | min | **900** | 1.0216e + 03 | 2.4766e + 03 | 1.1221e + 03 | 918.2225 | 956.7325 | 900.0000 |
| | max | **918.6377** | 2.3434e + 03 | 7.3050e + 03 | 2.0510e + 03 | 3.2197e + 03 | 1.8342e + 03 | 1.7123e + 03 |
| | avg | **900.6213** | 1.5025e + 03 | 4.5175e + 03 | 1.5375e + 03 | 1.4280e + 03 | 1.4439e + 03 | 979.1277 |
| | std | **3.4028** | 325.6067 | 1.2298e + 03 | 233.4753 | 523.0850 | 288.6303 | 154.2818 |
| F10 | min | 1.2374e + 03 | **1.0159e + 03** | 2.9441e + 03 | 1.7631e + 03 | 1.5864e + 03 | 1.5359e + 03 | 1.1224e + 03 |
| | max | 2.4820e + 03 | 2.7987e + 03 | 4.1909e + 03 | 2.6908e + 03 | 2.5668e + 03 | 2.3913e + 03 | **2.4365e + 03** |
| | avg | 1.8659e + 03 | 2.0068e + 03 | 3.6914e + 03 | 2.3014e + 03 | 2.0250e + 03 | 1.9048e + 03 | **1.8050e + 03** |
| | std | 309.5024 | 420.3168 | 256.7451 | 189.3770 | 274.3471 | 234.7766 | **308.3789** |
| Friedman mean rank | | 1.4 | 4.2 | 7 | 5.8 | 4.5 | 3.5 | 1.6 |
| Rank | | 1 | 4 | 7 | 6 | 5 | 3 | 2 |

**Table 11** Results for the CEC2017 hybrid test functions

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F11 | min | 1.1117e + 03 | 1.1187e + 03 | 2.7434e + 03 | 1.2210e + 03 | 1.1128e + 03 | 1.1117e + 03 | **1.1040e + 03** |
| | max | 1.1783e + 03 | 1.4428e + 03 | 9.7013e + 04 | 1.6675e + 03 | 5.6665e + 03 | 1.3690e + 03 | **1.1336e + 03** |
| | avg | 1.1355e + 03 | 1.2042e + 03 | 2.0267e + 04 | 1.4752e + 03 | 1.7909e + 03 | 1.1629e + 03 | **1.1172e + 03** |
| | std | 15.0440 | 82.4060 | 1.9601e + 04 | 116.1770 | 1.5193e + 03 | 53.9183 | **8.7811** |
| F12 | min | **1.8848e + 03** | 7.3867e + 03 | 2.6430e + 08 | 1.1885e + 07 | 6.6769e + 04 | 4.4734e + 03 | 3.0170e + 03 |
| | max | **5.5241e + 04** | 1.6165e + 07 | 4.5537e + 09 | 1.7882e + 08 | 3.4632e + 06 | 7.9466e + 06 | 6.2324e + 04 |
| | avg | **1.4715e + 04** | 4.3501e + 06 | 1.8460e + 09 | 7.4129e + 07 | 1.0463e + 06 | 2.1511e + 06 | 2.0882e + 04 |
| | std | **1.3570e + 04** | 5.0030e + 06 | 1.0696e + 09 | 4.6075e + 07 | 1.0316e + 06 | 2.2410e + 06 | 1.7196e + 04 |
| F13 | min | 1.3934e + 03 | 2.1368e + 03 | 2.9212e + 06 | 2.0314e + 03 | 4.5065e + 03 | 2.1721e + 03 | **1.3586e + 03** |
| | max | 2.2513e + 04 | 4.9296e + 04 | 9.3756e + 08 | 6.9288e + 04 | 2.8878e + 04 | 3.1948e + 04 | **4.5066e + 03** |
| | avg | 9.7998e + 03 | 1.7524e + 04 | 1.9327e + 08 | 1.7738e + 04 | 1.3902e + 04 | 1.5389e + 04 | **1.8643e + 03** |
| | std | 6.8674e + 03 | 1.3921e + 04 | 2.2074e + 08 | 1.8122e + 04 | 7.2055e + 03 | 9.0987e + 03 | **608.8909** |
| F14 | min | 1.4258e + 03 | 1.4766e + 03 | 3.3010e + 03 | 1.4774e + 03 | 1.4513e + 03 | 1.4595e + 03 | **1.4352e + 03** |
| | max | 3.3814e + 03 | 5.0046e + 03 | 1.0508e + 08 | 2.8387e + 03 | 5.2743e + 03 | 1.7477e + 03 | **1.7121e + 03** |
| | avg | 1.6818e + 03 | 1.7129e + 03 | 1.1467e + 07 | 1.8098e + 03 | 3.4798e + 03 | 1.5303e + 03 | **1.5007e + 03** |
| | std | 473.7783 | 653.5951 | 2.0925e + 07 | 292.0676 | 1.8847e + 03 | 50.5499 | **55.9308** |
| F15 | min | 1.5095e + 03 | 1.7400e + 03 | 3.6033e + 04 | 1.9506e + 03 | 1.5547e + 03 | 1.5752e + 03 | **1.5026e + 03** |
| | max | 3.2460e + 03 | 1.8667e + 04 | 1.0625e + 08 | 1.2991e + 04 | 2.3678e + 04 | 6.3665e + 03 | **1.9290e + 03** |
| | avg | 1.7290e + 03 | 6.0473e + 03 | 1.7799e + 07 | 5.4373e + 03 | 7.7534e + 03 | 3.0154e + 03 | **1.5645e + 03** |
| | std | 321.2525 | 4.3270e + 03 | 3.1615e + 07 | 2.9027e + 03 | 7.1112e + 03 | 1.3153e + 03 | **87.5326** |
| F16 | min | **1.6006e + 03** | 1.6511e + 03 | 2.2702e + 03 | 1.7192e + 03 | 1.6466e + 03 | 1.6041e + 03 | 1.6015e + 03 |
| | max | 2.0570e + 03 | 2.1622e + 03 | 3.0826e + 03 | 2.0412e + 03 | 2.2501e + 03 | 2.1596e + 03 | **2.0174e + 03** |
| | avg | 1.8323e + 03 | 1.8418e + 03 | 2.6847e + 03 | 1.8719e + 03 | 1.9272e + 03 | 1.8273e + 03 | **1.7850e + 03** |
| | std | 127.2654 | 122.4871 | 204.5139 | 89.7046 | 151.5610 | 131.1514 | **128.4766** |
| F17 | min | 1.7219e + 03 | 1.7356e + 03 | 1.9878e + 03 | 1.7662e + 03 | 1.7431e + 03 | 1.7287e + 03 | **1.7182e + 03** |
| | max | **1.8031e + 03** | 2.1208e + 03 | 2.7484e + 03 | 1.8659e + 03 | 1.9576e + 03 | 1.9646e + 03 | 1.8654e + 03 |
| | avg | **1.7489e + 03** | 1.8178e + 03 | 2.3624e + 03 | 1.8125e + 03 | 1.8273e + 03 | 1.7990e + 03 | 1.7718e + 03 |
| | std | **20.3463** | 74.0312 | 211.2303 | 28.5784 | 62.8552 | 55.9787 | 47.2362 |
| F18 | min | 1.9290e + 03 | 2.2511e + 03 | 4.3338e + 06 | 2.7740e + 03 | 3.0223e + 03 | 1.8881e + 03 | **1.8881e + 03** |
| | max | 5.5240e + 04 | 3.7439e + 04 | 1.7160e + 09 | 1.5294e + 05 | 5.5740e + 04 | 3.4902e + 04 | **2.2481e + 04** |
| | avg | 1.1022e + 04 | 1.2381e + 04 | 4.3233e + 08 | 2.1067e + 04 | 2.2089e + 04 | 1.3984e + 04 | **4.1175e + 03** |
| | std | 1.0442e + 04 | 8.4396e + 03 | 4.2172e + 08 | 3.0507e + 04 | 1.5295e + 04 | 1.1304e + 04 | **3.9591e + 03** |
| F19 | min | 1.9110e + 03 | 1.9531e + 03 | 1.0019e + 04 | 1.9544e + 03 | 1.9317e + 03 | 2.0178e + 03 | **1.9035e + 03** |
| | max | 7.3022e + 03 | 4.3309e + 05 | 4.6308e + 08 | 3.1994e + 04 | 2.6794e + 05 | 3.2377e + 04 | **2.0422e + 03** |
| | avg | 3.2471e + 03 | 3.2402e + 04 | 4.2881e + 07 | 6.9176e + 03 | 2.3324e + 04 | 1.0561e + 04 | **1.9332e + 03** |
| | std | 1.5377e + 03 | 8.2526e + 04 | 8.4667e + 07 | 6.5469e + 03 | 6.6563e + 04 | 1.0059e + 04 | **39.2552** |
| F20 | min | 2.0223e + 03 | 2.0366e + 03 | 2.2322e + 03 | 2.0699e + 03 | 2.0467e + 03 | 2.0494e + 03 | **2.0213e + 03** |
| | max | 2.2298e + 03 | 2.3345e + 03 | 2.6731e + 03 | 2.1992e + 03 | 2.4306e + 03 | 2.2826e + 03 | **2.1602e + 03** |
| | avg | 2.0956e + 03 | 2.1442e + 03 | 2.4926e + 03 | 2.1254e + 03 | 2.1738e + 03 | 2.1540e + 03 | **2.0677e + 03** |
| | std | 62.2516 | 68.0855 | 105.9201 | 29.6452 | 85.3777 | 54.7183 | **48.6816** |
| Friedman mean rank | | 2 | 4.5 | 7 | 4.7 | 5.2 | 3.4 | 1.2 |
| Rank | | 2 | 4 | 7 | 5 | 6 | 3 | 1 |

**Table 12** Results for the CEC2017 composition test functions

| Function | | PSO | WOA | ALO | LFD | TSA | HHO | HPO |
|---|---|---|---|---|---|---|---|---|
| F21 | min | **2200** | 2.2110e + 03 | 2.3614e + 03 | 2.2131e + 03 | 2.2051e + 03 | 2.2036e + 03 | 2.2000e + 03 |
| | max | 2.3676e + 03 | 2.3840e + 03 | 2.4792e + 03 | **2.3072e + 03** | 2.3658e + 03 | 2.3685e + 03 | 2.3612e + 03 |
| | avg | 2.3023e + 03 | 2.3234e + 03 | 2.4464e + 03 | **2.2509e + 03** | 2.3237e + 03 | 2.3236e + 03 | 2.3043e + 03 |
| | std | 58.9363 | 61.4838 | 22.6579 | **20.8729** | 50.0590 | 54.8047 | 58.8420 |
| F22 | min | 2.2275e + 03 | 2.3051e + 03 | 3.1505e + 03 | 2.3679e + 03 | 2.3089e + 03 | 2.3046e + 03 | **2.2241e + 03** |
| | max | **2.3055e + 03** | 3.6662e + 03 | 4.9698e + 03 | 2.7317e + 03 | 3.6750e + 03 | 2.3273e + 03 | 2.3137e + 03 |
| | avg | **2.2977e + 03** | 2.3602e + 03 | 4.0072e + 03 | 2.5368e + 03 | 2.5391e + 03 | 2.3136e + 03 | 2.3012e + 03 |
| | std | **19.0744** | 246.8083 | 472.6296 | 106.2993 | 305.4145 | 6.3606 | 14.9762 |
| F23 | min | 2.6544e + 03 | 2.6149e + 03 | 2.7437e + 03 | 2.5762e + 03 | 2.6395e + 03 | 2.6236e + 03 | **2.6124e + 03** |
| | max | 2.7652e + 03 | 2.7042e + 03 | 2.9432e + 03 | 2.7180e + 03 | 2.7675e + 03 | 2.7124e + 03 | **2.6720e + 03** |
| | avg | 2.6894e + 03 | 2.6552e + 03 | 2.8392e + 03 | 2.6779e + 03 | 2.6828e + 03 | 2.6661e + 03 | **2.6323e + 03** |
| | std | 25.7810 | 23.0100 | 46.7610 | 22.8367 | 34.8402 | 25.6549 | **14.1667** |
| F24 | min | **2500** | 2.5103e + 03 | 2.8415e + 03 | 2.6343e + 03 | 2.5127e + 03 | 2.7470e + 03 | 2.5000e + 03 |
| | max | 2.8536e + 03 | 2.8341e + 03 | 3.2084e + 03 | 2.8281e + 03 | 2.8851e + 03 | 2.8972e + 03 | **2.7981e + 03** |
| | avg | **2.7223e + 03** | 2.7659e + 03 | 3.0065e + 03 | 2.7531e + 03 | 2.8005e + 03 | 2.8097e + 03 | 2.7452e + 03 |
| | std | 138.5768 | 69.1731 | 78.5597 | 51.4355 | 78.2055 | 43.1536 | 67.7618 |
| F25 | min | **2.6001e + 03** | 2.9037e + 03 | 3.2478e + 03 | 3.0049e + 03 | 2.8987e + 03 | 2.6113e + 03 | 2.8979e + 03 |
| | max | **2.9464e + 03** | 3.6025e + 03 | 5.1168e + 03 | 3.2064e + 03 | 3.3252e + 03 | 2.9735e + 03 | 3.0244e + 03 |
| | avg | **2.9126e + 03** | 2.9673e + 03 | 4.2638e + 03 | 3.1185e + 03 | 3.0303e + 03 | 2.9311e + 03 | 2.9406e + 03 |
| | std | 63.2366 | 121.3296 | 550.4555 | 52.8685 | 115.2854 | 63.0897 | 33.3675 |
| F26 | min | 2.8000e + 03 | 2.6055e + 03 | 3.6896e + 03 | 3.1271e + 03 | 3.0230e + 03 | 2.6085e + 03 | **2.6000e + 03** |
| | max | 4.5065e + 03 | 4.3860e + 03 | 5.5380e + 03 | **3.6117e + 03** | 4.4069e + 03 | 4.4757e + 03 | 4.2176e + 03 |
| | avg | 3.0874e + 03 | 3.2965e + 03 | 4.7471e + 03 | 3.4041e + 03 | 3.4251e + 03 | 3.3120e + 03 | **3.0802e + 03** |
| | std | 446.0531 | 446.2029 | 487.4479 | 115.1949 | 326.5004 | 482.0550 | 363.8518 |
| F27 | min | 3.1051e + 03 | 3.0925e + 03 | 3.2099e + 03 | 3.1068e + 03 | 3.0969e + 03 | 3.0951e + 03 | **3.0895e + 03** |
| | max | 3.3259e + 03 | 3.2282e + 03 | 3.6317e + 03 | 3.1520e + 03 | 3.2397e + 03 | 3.2361e + 03 | **3.1931e + 03** |
| | avg | 3.1665e + 03 | 3.1212e + 03 | 3.4340e + 03 | 3.1265e + 03 | 3.1433e + 03 | 3.1459e + 03 | **3.1023e + 03** |
| | std | 54.9809 | 33.3036 | 121.8437 | 13.3818 | 40.1819 | 41.5257 | **24.6708** |
| F28 | min | **2.8000e + 03** | 3.1020e + 03 | 3.4325e + 03 | 3.2683e + 03 | 3.1704e + 03 | 3.1678e + 03 | 3.1000e + 03 |
| | max | **3.4465e + 03** | 3.7362e + 03 | 4.6043e + 03 | 3.5088e + 03 | 3.8474e + 03 | 3.4611e + 03 | 3.7362e + 03 |
| | avg | **3.2105e + 03** | 3.3794e + 03 | 4.0653e + 03 | 3.4142e + 03 | 3.4499e + 03 | 3.3892e + 03 | 3.3340e + 03 |
| | std | 158.5970 | 140.8683 | 290.1687 | 62.3776 | 209.0705 | 80.3515 | 155.1064 |
| F29 | min | 3.1663e + 03 | 3.2043e + 03 | 3.3916e + 03 | 3.2119e + 03 | 3.1964e + 03 | 3.1763e + 03 | **3.1370e + 03** |
| | max | **3.3926e + 03** | 3.4403e + 03 | 4.3402e + 03 | 3.4254e + 03 | 3.4420e + 03 | 3.5315e + 03 | 3.4771e + 03 |
| | avg | **3.2383e + 03** | 3.3136e + 03 | 3.9282e + 03 | 3.2999e + 03 | 3.2945e + 03 | 3.3302e + 03 | 3.2704e + 03 |
| | std | **57.5350** | 67.6805 | 210.4884 | 53.9732 | 67.3381 | 92.8523 | 81.8499 |
| F30 | min | 5.0274e + 03 | 7.5637e + 03 | 2.5335e + 07 | 1.3086e + 05 | 2.3189e + 04 | 1.7377e + 04 | **3.9759e + 03** |
| | max | **1.2551e + 06** | 4.4022e + 06 | 1.7699e + 08 | 8.1772e + 06 | 1.0433e + 07 | 3.2042e + 06 | 1.2518e + 06 |
| | avg | **1.9623e + 05** | 8.7166e + 05 | 9.0789e + 07 | 2.9422e + 06 | 1.4958e + 06 | 6.4400e + 05 | 3.7477e + 05 |
| | std | **3.9119e + 05** | 1.2567e + 06 | 4.5756e + 07 | 2.2208e + 06 | 2.4098e + 06 | 8.9853e + 05 | 4.7518e + 05 |
| Friedman mean rank | | 2.2 | 3.6 | 7 | 4.2 | 5.1 | 4 | 1.9 |
| Rank | | 2 | 3 | 7 | 5 | 6 | 4 | 1 |

**Table 13** P-values of the Wilcoxon rank-sum test overall runs

| Function | PSO | WOA | ALO | LFD | TSA | HHO |
|---|---|---|---|---|---|---|
| F1 | 5.2637e-05 | 3.0180e-11 | 3.0180e-11 | 3.0180e-11 | 3.0180e-11 | 3.0180e-11 |
| F2 | 0.0814 | 3.1507e-12 | 3.1507e-12 | 3.1507e-12 | 3.1507e-12 | 1.0173e-09 |
| F3 | 1.4459e-11 | 3.0142e-11 | 3.0142e-11 | 3.0142e-11 | 3.0142e-11 | 3.0142e-11 |
| F4 | 6.3330e-07 | 3.4971e-09 | 3.0199e-11 | 3.0199e-11 | 2.1544e-10 | 2.0023e-06 |
| F5 | 0.0314 | 1.4294e-05 | 3.0180e-11 | 9.9127e-11 | 4.3088e-08 | 5.5305e-08 |
| F6 | 0.0042 | 3.0199e-11 | 3.0199e-11 | 3.0199e-11 | 4.5043e-11 | 4.5043e-11 |
| F7 | 8.1975e-07 | 1.8731e-07 | 3.0199e-11 | 3.0199e-11 | 7.3803e-10 | 5.5727e-10 |
| F8 | 0.0053 | 1.2486e-05 | 3.0161e-11 | 3.3342e-11 | 2.2768e-05 | 0.0044 |
| F9 | 1.9359e-11 | 7.3763e-10 | 3.0180e-11 | 2.3701e-10 | 6.5238e-07 | 4.1804e-09 |
| F10 | 0.3871 | 0.0232 | 3.0199e-11 | 5.5329e-08 | 0.0091 | 0.1624 |
| F11 | 8.1975e-07 | 7.3803e-10 | 3.0199e-11 | 3.0199e-11 | 1.0105e-08 | 1.1567e-07 |
| F12 | 0.1087 | 6.7220e-10 | 3.0199e-11 | 3.0199e-11 | 3.0199e-11 | 3.4971e-09 |
| F13 | 1.2541e-07 | 1.6132e-10 | 3.0199e-11 | 1.7769e-10 | 3.3384e-11 | 8.1527e-11 |
| F14 | 0.6520 | 2.3885e-04 | 3.0199e-11 | 6.0104e-08 | 3.9881e-04 | 0.0281 |
| F15 | 1.3250e-04 | 3.6897e-11 | 3.0199e-11 | 3.0199e-11 | 9.9186e-11 | 2.1544e-10 |
| F16 | 0.4119 | 0.0484 | 3.0199e-11 | 0.0083 | 1.7836e-04 | 0.2116 |
| F17 | 0.1494 | 0.0012 | 3.0199e-11 | 3.0059e-04 | 7.6588e-05 | 0.0156 |
| F18 | 5.6073e-05 | 1.5964e-07 | 3.0199e-11 | 7.0881e-08 | 1.3111e-08 | 1.5292e-05 |
| F19 | 4.3106e-08 | 4.9752e-11 | 3.0199e-11 | 9.9186e-11 | 5.0922e-08 | 4.0772e-11 |
| F20 | 0.1715 | 2.5974e-05 | 3.0199e-11 | 1.4067e-04 | 8.1975e-07 | 1.2860e-06 |
| F21 | 0.3040 | 0.0012 | 3.0199e-11 | 3.9881e-04 | 0.0207 | 0.0051 |
| F22 | 0.7283 | 5.4617e-09 | 3.0199e-11 | 3.0199e-11 | 1.2057e-10 | 1.6980e-08 |
| F23 | 1.3289e-10 | 3.5923e-05 | 3.0199e-11 | 9.7555e-10 | 1.6947e-09 | 1.6062e-06 |
| F24 | 0.0434 | 0.0061 | 3.0199e-11 | 0.9470 | 1.4294e-08 | 3.5708e-06 |
| F25 | 3.5905e-05 | 0.0615 | 3.0199e-11 | 4.5043e-11 | 0.0013 | 0.3953 |
| F26 | 0.1018 | 1.4932e-04 | 6.0658e-11 | 1.1567e-07 | 3.2555e-07 | 0.0421 |
| F27 | 3.8249e-09 | 1.4298e-05 | 3.0199e-11 | 2.1947e-08 | 5.5329e-08 | 3.0811e-08 |
| F28 | 3.9389e-05 | 0.0013 | 3.3876e-11 | 0.0023 | 0.0319 | 0.0043 |
| F29 | 0.1154 | 0.0339 | 3.6897e-11 | 0.1154 | 0.1907 | 0.0163 |
| F30 | 0.1809 | 0.0025 | 3.0180e-11 | 1.8492e-08 | 0.0011 | 0.0070 |

functions, the proposed algorithm has better convergence than other algorithms.

Figure 12 shows the Friedman mean rank of all the compared methods for unimodal and multi-modal functions (group 1), hybrid functions (group 2) and composition functions (group 3). As per results of Fig. 12, HPO has indicated a trustworthy and sure behavior in two groups (hybrid and composition function) compared to the other algorithms.

## 5 Performance of HPO algorithm on constrained problems

Engineers and decision makers face problems that increase their complexity daily. These problems create different fields such as research in operation, mechanical systems, image processing and electronics design (Kumar et al. 2020). In all these areas, the problem can be expressed as an optimization problem. In optimization problems, one or more objective functions are defined that should be minimized or maximized considering all the parameters. Usually, constraints are defined in optimization problems. All solutions must apply to these constraints. Otherwise, the solutions will not be justified. There are nine real-world problems in engineering design that are used by many researchers, namely reducer design problem, rolling element bearing problem, cantilever beam design, multi-plate disk clutch brake, welded beam, three-bar truss, step-cone pulley problem, pressure vessel designs and tension/compression spring. Unlike basic test functions, real-world problems have equality and inequality constraints, so HPO should be equipped with a constraints control method to optimize such problems. Unlike basic test functions, real-world problems have equality and inequality constraints, so

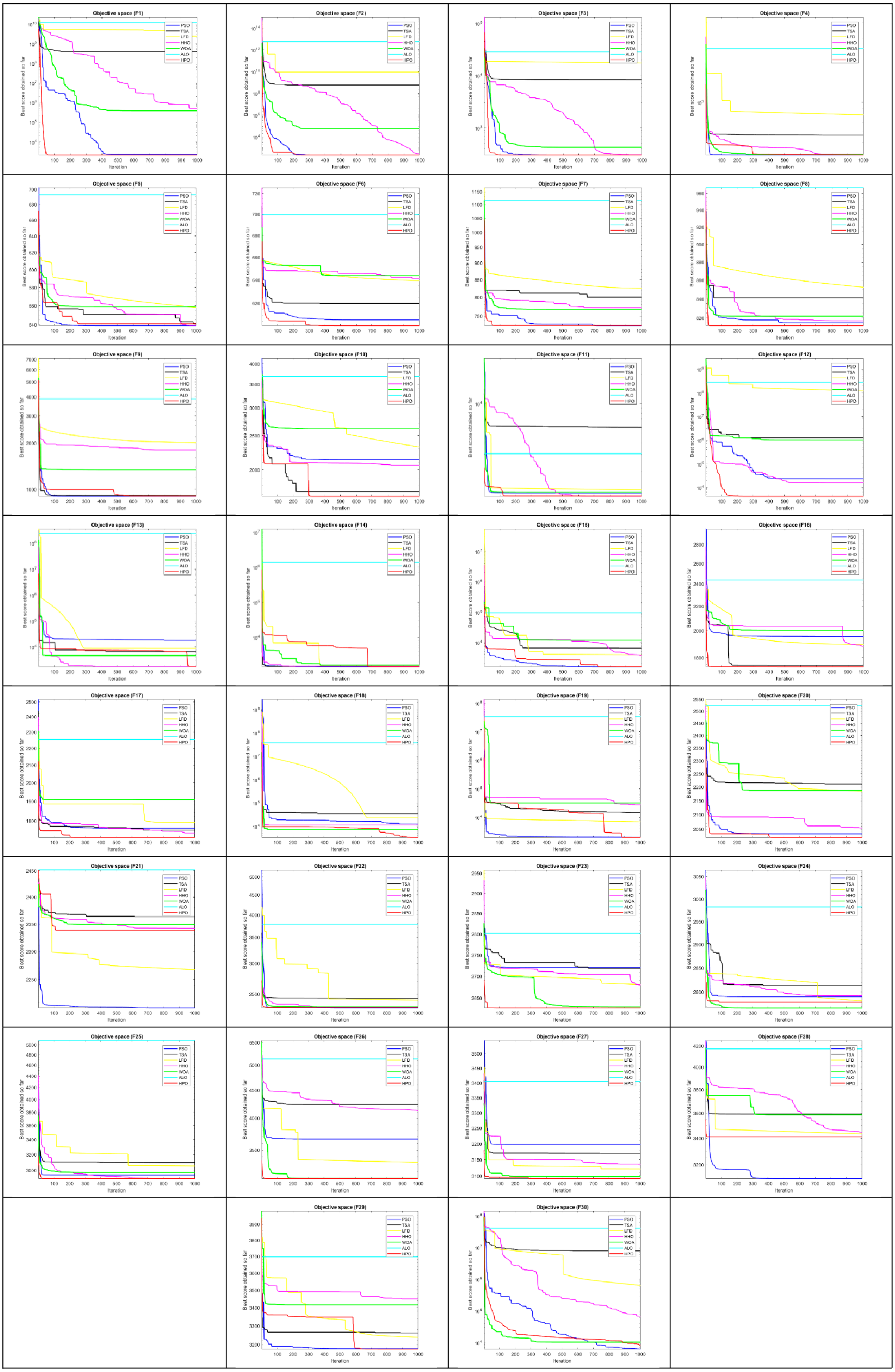



**Fig. 11** Convergence curve of the methods on the CEC2017 test function

HPO should be equipped with a constraints control method to optimize such problems.

The performance of the algorithm in dealing with constrained optimization problems is significantly influenced by the employed constraint handling technique (CHT). In recent decades, many constraint control methods have been developed for optimization algorithms. Some popular CHTs among them are death penalty, co-evolutionary, adaptive, annealing, dynamic and static (Coello Coello 2002). The death penalty function is the simplest method, which assigns a big objective value. This method eliminates impossible solutions by optimization algorithms during optimization process. The advantages of this method are low computational cost and simplicity (Mirjalili and Lewis 2016). To compare the methods, we used the death penalty method because most of the algorithms used the same. The results of the HPO algorithm were compared with the algorithms that previously solved these problems. For all problems, the number of search agents are set to 30, and the maximum number of iterations are set to 500. Details of these problems are given in Table 14.

Real-world problems have been used by many researchers, and any researcher may have tested these problems under different conditions, so only the best solution obtained by algorithms is reported.

## 5.1 Three-bar truss design problem

In general, one of the most important issues in the field of civil engineering is truss design. The goal in this problem is to design a truss with the least weight so that it does not violate any of the constraints of buckling, deflection and stress. The structure of this problem and its parameters are shown in Fig. 13.

**Table 14** Details of the nine constrained problems. h is the number of equality constraints, g is the number of inequality constraints, and D is the number of problem variables

| No | Name | h | g | D | Objective |
|---|---|---|---|---|---|
| 1 | Three-bar truss | 0 | 3 | 2 | Minimize |
| 2 | Reducer design problem | 0 | 11 | 7 | Minimize |
| 3 | Cantilever beam design | 1 | 1 | 5 | Minimize |
| 4 | Welded beam design | 0 | 7 | 4 | Minimize |
| 5 | Tension/compression spring | 0 | 4 | 3 | Minimize |
| 6 | Step-cone pulley problem | 0 | 1 | 4 | Minimize |
| 7 | Multi-plate disk clutch brake | 0 | 8 | 5 | Minimize |
| 8 | Pressure vessel design | 0 | 4 | 4 | Minimize |
| 9 | Rolling element bearing problem | 1 | 9 | 10 | Maximize |

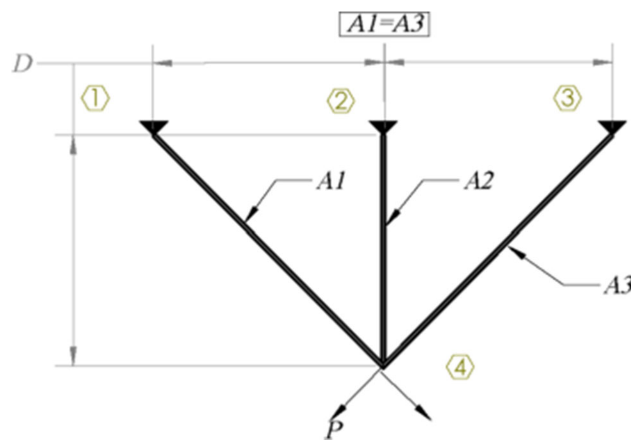



**Fig. 13** Three-bar truss design problem (Mirjalili et al. 2016)

The formula for this problem and its constraints are in the form of Eq. (13).

**Fig. 12** Friedman mean rank (CEC2017)

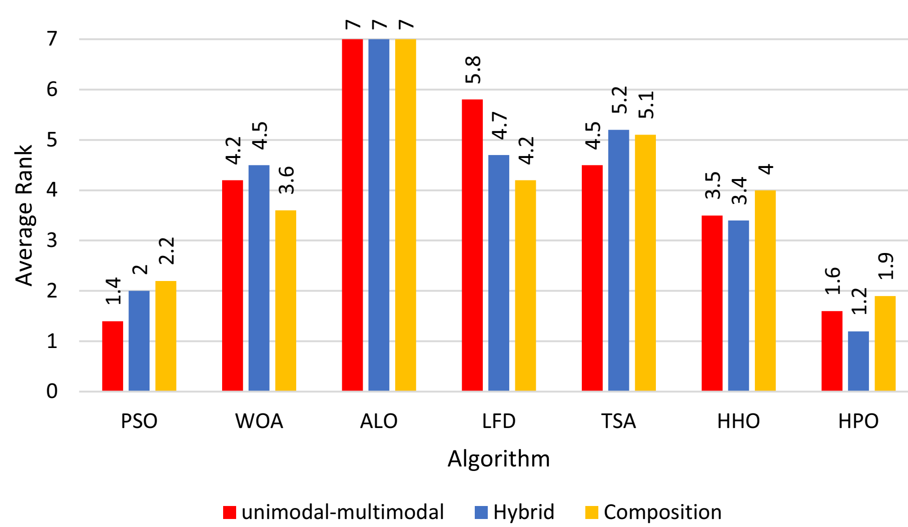


**Table 15** Results for the three-bar truss design problem

| Algorithm | Optimal values for variables | | Optimal weight |
|---|---|---|---|
| | $X_1$ | $X_2$ | |
| HPO | 0.788643655 | 0.4083373345 | 263.895844104 |
| CS | 0.78867 | 0.40902 | 263.9716 |
| DEDS | 0.78867513 | 0.40824828 | 263.8958434 |
| GOA | 0.788897555578973 | 0.407619570115153 | 263.8958814 |
| HHO | 0.788662816 | 0.408283133832900 | 263.8958434 |
| MBA | 0.7885650 | 0.4085597 | 263.8958522 |
| MFO | 0.788244771 | 0.409466905784741 | 263.8959797 |
| MVO | 0.78860276 | 0.408453070000000 | 263.8958499 |
| PRO | 0.7886475 | 0.4083262 | 263.8958439 |
| PSO-DE | **0.7886751** | **0.4082482** | **263.8958433** |
| Ray and Sain | 0.795 | 0.395 | 264.3 |
| SC-GWO | 0.78941 | 0.40617 | 263.8963 |
| SSA | 0.788665414 | 0.408275784444547 | 263.8958434 |

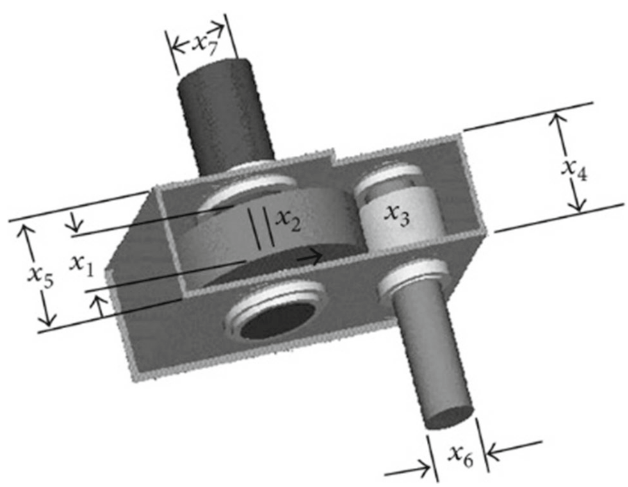



**Fig. 14** Speed reducer design (Hassan et al. 2005)

$$
\begin{aligned}
&Minimize: \quad f(A_1, A_2) = (2\sqrt{2A_1} + A_2) \times l \\
&Subject\ to: \\
&g_1 = \frac{\sqrt{2A_1} + A_2}{\sqrt{2}A_1^2 + 2A_1A_2} P - \sigma \le 0 \\
&g_2 = \frac{A_2}{\sqrt{2}A_1^2 + 2A_1A_2} P - \sigma \le 0 \\
&g_3 = \frac{1}{A_1 + \sqrt{2}A_2} P - \sigma \le 0 \\
&where \\
&0 \le A_1 \le 1 \quad and \quad 0 \le A_2 \le 1; \quad l = 100\,cm, \\
&P = 2KN/cm^2, \quad \sigma = 2KN/cm^2.
\end{aligned}
\tag{13}
$$

The performance of the proposed HPO algorithm on this problem was compared with cuckoo search algorithm (CS) (Gandomi et al. 2013), differential evolution with dynamic stochastic selection (DEDS) (Zhang et al. 2008), grasshopper optimization algorithm (GOA) (Saremi et al. 2017), Harris hawks optimizer (HHO) (Heidari et al. 2019), mine blast algorithm (MBA) (Sadollah et al. 2013), moth-flame optimization (MFO) algorithm (Mirjalili 2015c), multi-verse optimizer (MVO) (Mirjalili et al. 2016), poor and rich optimization (PRO) algorithm (Samareh Moosavi and Bardsiri 2019), particle swarm optimization with differential evolution (PSO-DE) (Liu et al. 2010), Ray and Sain (RAY and SAINI 2001), sine cosine gray wolf optimizer (SC-GWO) (Gupta et al. 2020) and salp swarm algorithm (SSA) (Mirjalili et al. 2017). The comparison results are shown in Table 15. The results in Table 15 show that the proposed HPO algorithm offers competitive results with the HHO, SSA, DEDS and PSO-DE algorithms. The HPO algorithm also performs better than other compared methods.

## 5.2 Speed reducer design problem

The speed reducer design problem has seven design variables (Gandomi et al. 2013), as shown in Fig. 14. This test problem's objective is to minimize the weight of a speed reducer with subject to different constraints on surfaces stress, bending stress, stresses in the shafts and transverse deflections of the shafts (Mezura-Montes and Coello, 2005). The formula for this problem and its constraints are in the form of Eq. (14)

$$
\begin{aligned}
&\textit{Consider } \bar{z} = [z_1\, z_2\, z_3\, z_4\, z_5\, z_6\, z_7] = [b\, m\, p\, l_1\, l_2\, d_1\, d_2],\\
&\textit{Minimize } f(\bar{z}) = 0.7854 z_1 z_2^2 (3.3333 z_3^2 + 14.9334 z_3 - 43.0934)\\
&- 1.508 z_1 (z_6^2 + z_7^2) + 7.4777 (z_6^3 + z_7^3) + 0.7854 (z_4 z_6^2 + z_5 z_7^2),\\
&\textit{Subject to}:\\
&g_1(\bar{z}) = \frac{27}{z_1 z_2^2 z_3} - 1 \le 0,\\
&g_2(\bar{z}) = \frac{397.5}{z_1 z_2^2 z_3^2} - 1 \le 0,\\
&g_3(\bar{z}) = \frac{1.93 z_4^3}{z_2 z_7^4 z_3} - 1 \le 0,\\
&g_4(\bar{z}) = \frac{1.93 z_4^3}{z_2 z_7^4 z_3} - 1 \le 0,\\
&g_5(\bar{z}) = \frac{[(745(z_4/z_2 z_3))^2 + 16.9 \times 10^6]^{1/2}}{110 z_6^3} - 1 \le 0,\\
&g_6(\bar{z}) = \frac{[(745(z_5/z_2 z_3))^2 + 157.5 \times 10^6]^{1/2}}{85 z_7^3} - 1 \le 0,\\
&g_4(\bar{z}) = \frac{1.93 z_4^3}{z_2 z_7^4 z_3} - 1 \le 0,\\
&g_7(\bar{z}) = \frac{z_2 z_3}{40} - 1 \le 0,\\
&g_8(\bar{z}) = \frac{5 z_2}{z_1} - 1 \le 0,\\
&g_9(\bar{z}) = \frac{z_1}{12 z_2} - 1 \le 0,\\
&g_{10}(\bar{z}) = \frac{1.5 z_6 + 1.9}{z_4} - 1 \le 0,\\
&g_{11}(\bar{z}) = \frac{1.1 z_7 + 1.9}{z_5} - 1 \le 0,\\
&\textit{where},\\
&2.6 \le z_1 \le 3.6,\ 0.7 \le z_2 \le 0.8,\ 17 \le z_3 \le 28,\ 7.3 \le z_4 \le 8.3,\\
&7.3 \le z_5 \le 8.3,\ 2.9 \le z_6 \le 3.9,\ 5.0 \le z_7 \le 5.5.
\end{aligned}
\tag{14}
$$

The proposed HPO algorithm was tested on this problem, and the results were compared with artificial ecosystem-based optimization (AEO) (Zhao et al. 2020), chaotic multi-verse optimization (CMVO) (Sayed et al. 2018), Coot optimization algorithm (COOT) (Naruei and Keynia 2021b), emperor penguin optimizer (EPO) (Dhiman and Kumar 2018), genetic algorithm (GA), gray prediction evolution algorithm based on accelerated even (GPEAae) (Gao et al. 2020), gray wolf optimizer (GWO) (Mirjalili et al. 2014), sine cosine gray wolf optimizer (SC-GWO) (Gupta et al. 2020), artificial bee colony with enhanced food locations (I-ABC) (Sharma and Abraham 2020), spotted hyena optimizer (SHO) (Dhiman and Kumar 2017) and tunicate swarm algorithm (TSA) (Kaur et al. 2020). The compared results are shown in Table 16. As it is clear from the results, the proposed HPO algorithm has shown a very good performance for this problem. The HPO algorithm has found a better optimal value than other compared methods.

### 5.3 Cantilever beam design problem

Cantilever beams are one of the most important problems in the field of mechanics and civil engineering. The purpose of this problem is to minimize the weight of the beam. As shown in Fig. 15, a cantilever beam consists of five hollow elements with a square cross section. Each element is defined by a variable, and the thickness of all of them is constant. This problem has a constraint that should not be violated. The formula for this problem and its constraints are in the form of Eq. (15)

$$
\begin{aligned}
&\textit{Minimize}:\\
&\bar{x} = [x_1 x_2 x_3 x_4 x_5],\\
&f(\bar{x}) = 0.6224 (x_1 + x_2 + x_3 + x_4 + x_5),\\
&g(\bar{x}) = \frac{61}{x_1^3} + \frac{37}{x_2^3} + \frac{19}{x_3^3} + \frac{7}{x_4^3} + \frac{1}{x_5^3} \le 1,\\
&0.01 \le x_1, x_2, x_3, x_4, x_5 \le 100.
\end{aligned}
\tag{15}
$$

The performance of the proposed HPO algorithm on this problem was compared with AEO, ALO, COOT, CS, GPEAae, MVO, interactive autodidactic school (IAS) (Jahangiri et al. 2020) and symbiotic organisms search (SOS) (Cheng and Prayogo 2014). The comparison results are shown in Table 17. The results of Table 17 show that the proposed HPO algorithm offers a better solution to solve this problem at the lowest cost.

### 5.4 Welded beam design

This test problem's objective is to minimize the welded beam's fabrication cost shown in Fig. 16. This problem has four constraints such as end deflection of the beam (δ), buckling load on the bar (Pc), bending stress in the beam (σ), shear stress in weld (τ) and side constraints.

Problem variables are thickness of the bar (b), the height of the bar (t), length of the attached part of the bar (l) and the thickness of weld (h). The formula for this problem and its constraints are in the form of Eq. (16)

**Table 16** Comparison results for speed reducer design problem

| Algorithm | Optimum variables | | | | | | | Optimum cost |
|---|---|---|---|---|---|---|---|---|
| | b | m | p | $L_1$ | $L_2$ | $D_1$ | $D_2$ | |
| HPO | **3.241297** | **0.7** | **17** | **7.3** | **7.7153199** | **3.350215** | **5.28665** | **2892.7292** |
| AEO | 3.5 | 0.7 | 17 | 7.3 | 7.7153199 | 3.3502146 | 5.2866545 | 2994.471066 |
| CMVO | 3.5 | 0.7 | 17 | 7.3 | 7.715319 | 3.350214 | 5.286654 | 2994.471 |
| COOT | 3.24132 | 0.7 | 17 | 7.3 | 7.715336 | 3.350215 | 5.28665 | 2892.7461 |
| EPO | 3.50123 | 0.7 | 17 | 7.3 | 7.8 | 3.33421 | 5.26536 | 2994.2472 |
| GA | 3.510253 | 0.7 | 17 | 8.35 | 7.8 | 3.362201 | 5.287723 | 3067.561 |
| GPEAae | 3.499997 | 0.7 | 17 | 7.300001 | 7.715311 | 3.350214 | 5.286653 | 2994.468240 |
| GWO | 3.506690 | 0.7 | 17 | 7.380933 | 7.815726 | 3.357847 | 5.286768 | 3001.288 |
| I-ABC greedy | 3.50021 | 0.7 | 17 | 7.3 | 7.71531189 | 3.3502147 | 5.2866554 | 2994.4710315 |
| PSO | 3.500019 | 0.7 | 17 | 8.3 | 7.8 | 3.352412 | 5.286715 | 3005.763 |
| SC-GWO | 3.50064 | 0.7 | 17 | 7.30643 | 7.80617 | 3.35034 | 5.28694 | 2996.9859 |
| SHO | 3.50159 | 0.7 | 17 | 7.3 | 7.8 | 3.35127 | 5.28874 | 2998.5507 |
| TSA | 3.50120 | 0.7 | 17 | 7.3 | 7.8 | 3.33410 | 5.26530 | 2990.9580 |

$$
\begin{aligned}
&\vec{x} = [x_1\, x_2\, x_3 x_4] = [hltb],\\
&f(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14.0 + x_2),\\
&g_1(\vec{x}) = \tau(\vec{x}) - \tau_{\max} \le 0,\\
&g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{\max} \le 0,\\
&g_3(\vec{x}) = \delta(\vec{x}) - \delta_{\max} \le 0,\\
&g_4(\vec{x}) = x_1 - x_4 \le 0,\\
&g_5(\vec{x}) = P - P_c(\vec{x}) \le 0,\\
&g_6(\vec{x}) = 0.125 - x_1 \le 0,\\
&g_7(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14.0 + x_2) - 5.0 \le 0\\
&0.1 \le x_1 \le 2,\\
&0.1 \le x_2 \le 10,\\
&0.1 \le x_3 \le 10,\\
&0.1 \le x_4 \le 2\\
&\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau \frac{x_2}{2R} + (\tau)^2},\\
&\tau' = \frac{p}{\sqrt{2} x_1 x_2}, \tau = \frac{MR}{J}, M = P(L + \frac{x_2}{2}),\\
&R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2},\\
&J = 2\left\{\sqrt{2} x_1 x_2 \left[\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\},\\
&\sigma(\vec{x}) = \frac{6PL}{x_4 x_3^2}, \delta(\vec{x}) = \frac{6PL^3}{E x_3^2 x_4}\\
&P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2 x_4^6}{36}}}{L^2}(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}),\\
&P = 6000\,lb, L = 14\,in., \quad \delta_{\max} = 0.25\,in.,\\
&E = 30 \times 1^6\,psi,\ G = 12 \times 10^6\,psi,\\
&\tau_{\max} = 13600\,psi,\ \sigma_{\max} = 30000\,psi.
\end{aligned}
\tag{16}
$$

The optimal results of HPO versus those attained by AEO, an effective co-evolutionary differential evolution (CDE) (Huang et al. 2007), CMVO, GPEAae, GSA, HHO, HS, I-ABC, IAS, LFD, SHO and TSA are given in Table 18. The results showed that the proposed HPO algorithm found a better optimal value than other compared methods. It is noteworthy that the optimal value found by the HPO algorithm is very different from the optimal values found by other methods.

### 5.5 Tension/Compression spring design problem

The engineering test problem used is the tension/ compression spring design problem. The goal is to minimize the cost of building a spring with three parameters, namely number of active loops *(N)*, average coil diameter *(D)* and wire diameter *(d)* (Arora 2017). Figure 17 shows the details of the spring and its parameters. The spring design problem has a number of inequality constraints, which are given in Eq. (17)

$$
\begin{aligned}
&\vec{x} = [x_1\, x_2\, x_3] = [dDN],\\
&f(\vec{x}) = (x_3 + 2) x_2 x_1^2,\\
&g_1(\vec{x}) = 1 - \frac{x_2^3 x_3}{71785 x_1^4} \le 0,\\
&g_2(\vec{x}) = \frac{4x_2^2 - x_1 x_2}{12566(x_2 x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} \le 0,\\
&g_3(\vec{x}) = 1 - \frac{140.45 x_1}{x_2^2 x_3} \le 0,\\
&g_4(\vec{x}) = \frac{x_1 + x_2}{1.5} - 1 \le 0,\\
&0.05 \le x_1 \le 2.00,\\
&0.25 \le x_2 \le 1.30,\\
&2.00 \le x_3 \le 15.0.
\end{aligned}
\tag{17}
$$

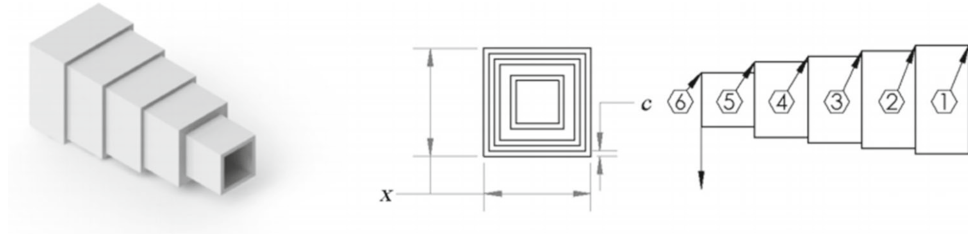



**Fig. 15** Cantilever beam design problem (Mirjalili et al. 2016)

**Table 17** Comparison results for the cantilever design problem

| Algorithm | Optimal values for variables | | | | | Optimal weight |
|---|---|---|---|---|---|---|
| | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | |
| HPO | **6.0055233569** | **5.30591367** | **4.49474956** | **3.51336235** | **2.154234** | **1.33652825** |
| AEO | 6.028850 | 5.316521 | 4.462649 | 3.508455 | 2.157761 | 1.339965 |
| ALO | 6.01812 | 5.31142 | 4.48836 | 3.49751 | 2.158329 | 1.33995 |
| COOT | 6.02743657 | 5.3385748 | 4.4904867 | 3.483437 | 2.134591 | 1.3365745 |
| CS | 6.0089 | 5.3049 | 4.5023 | 3.5077 | 2.1504 | 1.33999 |
| GPEAae | 6.014808 | 5.306724 | 4.493232 | 3.505168 | 2.153781 | 1.339982 |
| IAS | 5.99140 | 5.30850 | 4.51190 | 3.50210 | 2.16010 | 1.34000 |
| MVO | 6.0239402 | 5.3060112 | 4.4950113 | 3.496022 | 2.1527261 | 1.3399595 |
| SOS | 6.01878 | 5.30344 | 4.49587 | 3.49896 | 2.15564 | 1.33996 |

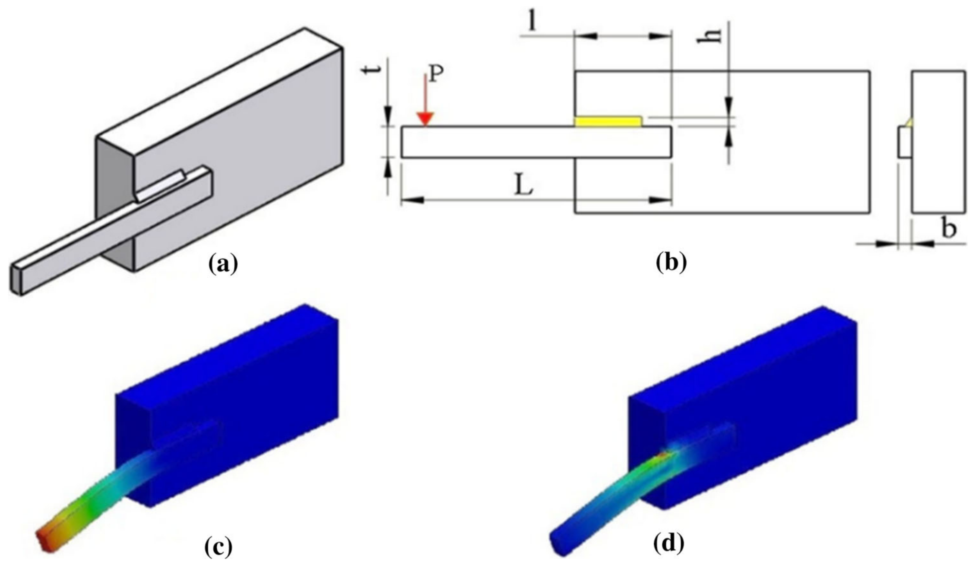



**Fig. 16** Welded beam design (Khalilpourazari and Khalilpourazary 2019)

The problem of spring design has been solved by many researchers. The proposed HPO algorithm was tested to solve this problem and compared with popular and new methods such as AEO, BA, COOT, co-evolutionary particle swarm optimization (CPSO) (He and Wang 2007a), GPEAae, GSA, GWO, HHO, stochastic fractal search (SFS) (Salimi 2015), SHO, SSA, water cycle algorithm (WCA) (Eskandar et al. 2012), water evaporation optimization (WEO) (Kaveh and Bakhshpoori 2016) and WOA. Table 19 shows the comparison results of different methods to solve the spring design problem. The proposed HPO algorithm was able to solve this problem better than all the compared methods and find better values for the problem variables with the least weight.

## 5.6 Step-cone pulley problem

One of the important problems in the field of engineering is the problem of step-cone pulley. The goal is to minimize the weight of the four step-cone pulleys. Figure 18 shows the structure and parameters of this problem. As shown in Fig. 18, this problem has five variables, four of which are related to the diameter of the pulley, and one variable is the width of the pulley. The step-cone pulley is to be designed for transmitting a power of at least 0.75 hp. The formula for this problem and its constraints are in the form of Eq. (17)

**Minimize:**

$$f(\bar{x}) = \rho\omega\left[d_1^2\left\{11+\left(\frac{N_1}{N}\right)^2\right\}+d_2^2\left\{1+\left(\frac{N_2}{N}\right)^2\right\}\right.$$
$$\left.+d_3^2\left\{1+\left(\frac{N_3}{N}\right)^2\right\}+d_4^2\left\{1+\left(\frac{N_4}{N}\right)^2\right\}\right]$$

**Subject to:**

$$\begin{aligned}
&h_1(\bar{x}) = C_1 - C_2 = 0,\\
&h_2(\bar{x}) = C_1 - C_3 = 0,\\
&h_3(\bar{x}) = C_1 - C_4 = 0,\\
&g_{i=1,2,3,4}(\bar{x}) = -R_i \le 2,\\
&g_{i=1,2,3,4}(\bar{x}) = (0.75\times 745.6998) - P_i \le 0\\
&where,\\
&C_i = \frac{\pi d_i}{2}\left(1+\frac{N_i}{N}\right)+\frac{\left(\frac{N_i}{N}-1\right)^2}{4a}+2a,\ i=(1,2,3,4),\\
&R_i = \exp\left(\mu\left\{\pi - 2\sin^{-1}\left\{\left(\frac{N_i}{N}-1\right)\frac{d_i}{2a}\right\}\right\}\right),\ i=(1,2,3,4),\\
&P_i = st\omega(1-R_i)\frac{\pi d_i N_i}{60},\ i=(1,2,3,4),\\
&t = 8mm,\quad s = 1.75MPa,\quad \mu = 0.35,\\
&\rho = 7200kg/m^3,\quad a = 3mm.
\end{aligned} \tag{18}$$

A schematic view of this problem is shown in Fig. 18.

The proposed HPO algorithm was tested on this problem and the results were compared with artificial electric field algorithm (AEFA) (Anita and Kumar 2020), passing vehicle search (PVS) (Savsani and Savsani 2016), artificial bee colony (ABC) (Rao et al. 2011) and Coot optimization algorithm (COOT). The results are shown in Table 20. As shown in Table 20, the HPO algorithm has discovered better optimal values than other algorithms and has been able to rank first.

## 5.7 Multiple disk clutch brake

The purpose of this test problem is to minimize the multi-disk clutch brake mass. This problem has five discrete decision variables, namely friction levels (Z), inner radius (ri), disk thickness (t), driving force (F) and outer radius (r0). The structure and parameters of this problem are shown in Fig. 19. The formula for this problem and its constraints are in the form of Eq. (19)

$$\begin{aligned}
&Minimize:\\
&f(\bar{x}) = \pi(x_2^2 - x_1^2)x_3(x_5+1)_\rho\\
&subject\ to:\\
&g_1(\bar{x}) = -p_{\max} + p_{rz} \le 0,\\
&g_2(\bar{x}) = p_{rz}v_{sr} - v_{sr,\max\,P\max} \le 0,\\
&g_3(\bar{x}) = \Delta R + x_1 - x_2 \le 0,\\
&g_4(\bar{x}) = -L_{\max} + (x_5+1)(x_3+\delta) \le 0,\\
&g_5(\bar{x}) = sM_s - M_h \le 0,\\
&g_6(\bar{x}) = T \ge 0,\\
&g_7(\bar{x}) = -v_{sr,\max} + v_{sr} \le 0,\\
&g_8(\bar{x}) = T - T_{\max} \le 0,\\
&where\\
&M_h = \frac{2}{3}\mu x_4 x_5 \frac{x_2^3 - x_1^3}{x_2^2 - x_1^2}N.mm,\\
&\omega = \frac{\pi n}{30}rad/s,\\
&A = \pi(x_2^2 - x_1^2)mm^2,\\
&p_{rz} = \frac{x_4}{A}N/mm^2,\\
&v_{sr} = \frac{\pi R_{sr} n}{30}mm/s,\\
&R_{sr} = \frac{2}{3}\frac{x_2^3 - x_1^3}{x_2^2 x_1^2}mm,\\
&T = \frac{I_z\omega}{M_h + M_f},\\
&\Delta R = 20mm, L_{\max} = 30mm, \mu = 0.6,\\
&V_{sr,\max} = 10m/s, \delta = 0.5mm, s = 1.5,\\
&T_{\max} = 15s, n = 250rpm, T_z = 55Kg.m^2,\\
&M_s = 40Nm, M_f = 2Nm, and\ p_{\max} = 1.\\
&with\ bounds:\\
&60 \le x_1 \le 80, 90 \le x_2 \le 110, 1 \le x_3 \le 3,\\
&0 \le x_4 \le 1000, 2 \le x_5 \le 9.
\end{aligned} \tag{19}$$

The proposed HPO algorithm was tested on this problem, and the results were compared with AEO, CMVO, flying squirrel optimizer (FSO) (Azizyan et al. 2019), HHO, I-ABC greedy, PVS, quantum-behaved simulated annealing algorithm-based moth-flame optimization (QSMFO) (Yu et al. 2020), TLBO and WCA. The compared results are shown in Table 21. As it is clear from the results, the proposed HPO algorithm has shown a very good performance for this problem. The HPO algorithm has found a better optimal value than other compared methods, while other algorithms have performed almost similarly.

**Table 18** Results for the welded beam

| Algorithms | τ | σ | Pc | δ | Optimal cost |
|---|---|---|---|---|---|
| HPO | **0.198811887** | **3.3377539** | **9.1920155** | **0.1988326** | **1.670240392337** |
| AEO | 0.2057296 | 3.4704886 | 9.0366239 | 0.2057296 | 1.7248520 |
| CDE | 0.203137 | 3.542998 | 9.033498 | 0.206179 | 1.733462 |
| CMVO | 0.20573 | 3.4705 | 9.03662 | 0.20573 | 1.724852 |
| GPEAae | 0.205731 | 3.470467 | 9.036624 | 0.205730 | 1.724851 |
| GSA | 0.182129 | 3.856979 | 10 | 0.202376 | 1.879952 |
| HHO | 0.204039 | 3.531061 | 9.027463 | 0.206147 | 1.73199057 |
| HS | 0.2442 | 6.2231 | 8.2915 | 0.2443 | 2.3807 |
| I-ABC greedy | 0.2057294 | 3.47048861 | 9.03662389 | 0.20572876 | 1.7248210 |
| IAS | 0.2057 | 3.4705 | 9.0366 | 0.2057 | 1.7249 |
| LFD | 0.1857 | 3.9070 | 9.1552 | 0.2051 | 1.77E + 00 |
| SHO | 0.205563 | 3.474846 | 9.035799 | 0.205811 | 1.725661 |
| TSA | 0.203290 | 3.471140 | 9.035100 | 0.201150 | 1.721020 |

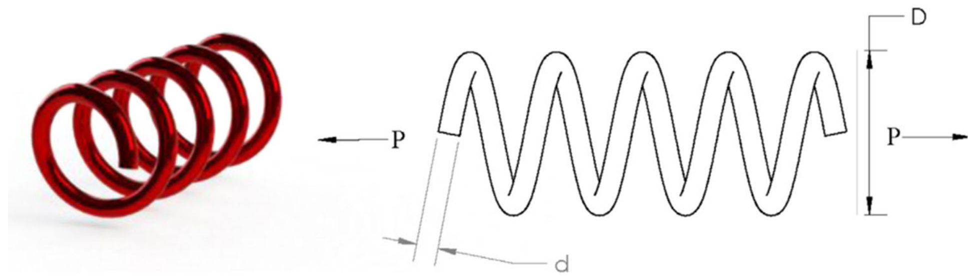



**Fig. 17** Tension/compression spring design problem (Mirjalili 2015b)

**Table 19** Results for tension/compression spring

| Algorithms | N | D | d | Weight |
|---|---|---|---|---|
| HPO | **11.21536452893** | **0.3579796674** | **0.0517414615** | **0.012665282823723** |
| AEO | 10.879842 | 0.361751 | 0.051897 | 0.0126662 |
| BA | 11.2885 | 0.35673 | 0.05169 | 0.012665 |
| COOT | 11.34038 | 0.35584 | 0.05165 | 0.012665293 |
| CPSO | 11.244543 | 0.357644 | 0.051728 | 0.012674 |
| GPEAae | 11.294000 | 0.356631 | 0.051685 | 0.012665 |
| GSA | 13.525410 | 0.323680 | 0.050276 | 0.012702 |
| GWO | 11.28885 | 0.356737 | 0.05169 | 0.012666 |
| HHO | 11.138859 | 0.359305355 | 0.051796393 | 0.012665443 |
| SFS | 11.288966 | 0.356717736 | 0.051689061 | 0.012665233 |
| SHO | 12.09550 | 0.343751 | 0.051144 | 0.012674000 |
| SSA | 12.004032 | 0.345215 | 0.051207 | 0.0126763 |
| WCA | 11.30041 | 0.356522 | 0.05168 | 0.012665 |
| WEO | 11.294103 | 0.356630 | 0.051685 | 0.012665 |
| WOA | 12.004032 | 0.345215 | 0.051207 | 0.0126763 |

## 5.8 Pressure vessel design

The problem of designing pressure vessels is another common engineering test problem in the optimization algorithm design. As shown in Fig. 20, one side of the dish is hemispherical, while the other side is flat. Problem parameters for optimization are length of cylindrical section without considering head (L), internal radius (R), head thickness (Th) and shell thickness (Ts). The problem of designing pressure vessels is formulated in Eq. (20)

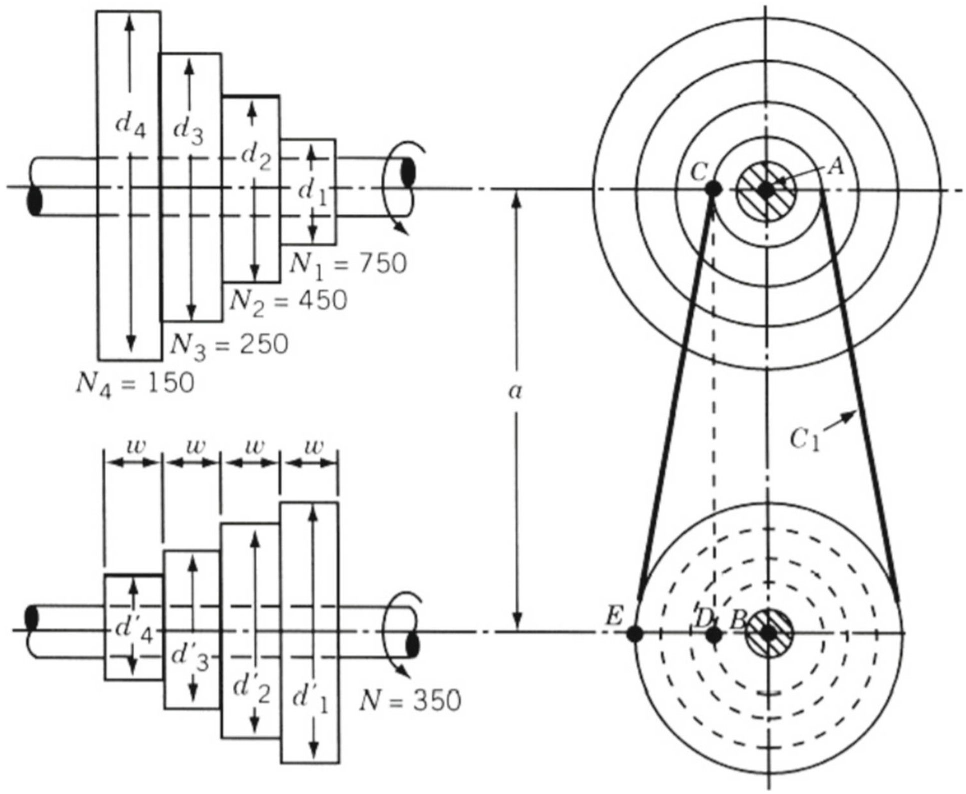



**Fig. 18** Step-cone pulley problem (Savsani and Savsani 2016)

$$\vec{x} = [x_1\, x_2\, x_3 x_4] = [T_s T_h R L],$$
$$f(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3,$$
$$g_1(\vec{x}) = -x_1 + 0.0193x_3 \leq 0,$$
$$g_2(\vec{x}) = -x_3 + 0.00954x_3 \leq 0,$$
$$g_3(\vec{x}) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0,$$
$$g_4(\vec{x}) = x_4 - 240 \leq 0,$$
$$0 \leq x_1 \leq 99,$$
$$0 \leq x_2 \leq 99,$$
$$10 \leq x_3 \leq 200,$$
$$10 \leq x_4 \leq 200. \quad (20)$$

The proposed HPO algorithm was tested to solve the pressure vessel design problem. The results of this experiment were compared with methods such as AEO, BA, charged system search (CSS) (Kaveh and Talatahari 2010), GA, GPEAae, Gaussian quantum-behaved particle swarm optimization (G-QPSO) (dos Santos Coelho 2010), GWO, HHO, hybrid particle swarm optimization (HPSO) (He and Wang 2007b), MFO, SC-GWO,WEO and WOA. Table 22 shows the results of this comparison. As can be seen from the results, the proposed HPO algorithm has found better values for the variables of this problem and solved it better than other methods.

## 5.9 Rolling element bearing design problem

Rolling element bearings have different geometric shapes that are optimized for different applications. The main goal of this problem is to maximize the dynamic load capacity by considering ten geometric design variables and nine constraints based on geometric and assembly constraints. Out of ten design variables, one design variable (number of balls in the bearing) is required to obtain the correct value. The formula for this problem and its constraints are in the form of Eq. (21)

**Maximize:**

$$f(\overline{x}) = \begin{cases} f_c Z^{2/3} D_b^{1.8} & , if\ D_b \leq 25.4\, mm \\ 3.647 f_c Z^{2/3} D_b^{1.4} & , otherwise \end{cases}$$

**Subject to:**

$$
\begin{aligned}
&g_1(\bar{x}) = Z - \frac{\phi_0}{2\sin^{-1}(D_b/D_m)} - 1 \le 0,\\
&g_2(\bar{x}) = 2D_b - K_{D\min}(D-d) > 0,\\
&g_3(\bar{x}) = K_{D\max}(D-d) - 2D_b \ge 0,\\
&g_4(\bar{x}) = \zeta B_\omega - D_b \le 0,\\
&g_5(\bar{x}) = D_m - 0.5(D+d) \ge 0,\\
&g_6(\bar{x}) = (0.5+e)(D+d) - D_m < 0,\\
&g_7(\bar{x}) = 0.5(D - D_m - D_b) - \varepsilon D_b \ge 0,\\
&g_8(\bar{x}) = f_i \ge 0.515,\\
&g_9(\bar{x}) = f_0 \ge 0.515,\\
&\textit{where}\\
&f_c = 37.91\left[1 + \left\{1.04\left(\frac{1-\gamma}{1+\gamma}\right)^{1.72}\left(\frac{f_i(2f_0-1)}{f_0(2f_i-1)}\right)^{0.41}\right\}^{10/3}\right]^{-0.3},\\
&\gamma = \frac{D_b}{D_m},\ f_i = \frac{r_i}{D_b},\ f_0 = \frac{r_0}{D_b},\\
&\phi_0 = 2\pi - 2\\
&\quad \times \cos^{-1}\left(\frac{\{(D-d)/2 - 3(T/4)\}^2 + \{D/2 - (T/4) - D_b\}^2 - \{d/2 + (T/4)\}^2}{2\{(D-d)/2 - 3(T/4)\}\{D/2 - (T/4) - D_b\}}\right)\\
&T = D - d - 2D_b,\ D = 160,\ d = 90,\ B_w = 30.
\end{aligned} \tag{21}
$$

**With bounds:**

$$
\begin{aligned}
&0.5(D+d) \le D_m \le 0.6(D+d),\\
&0.15(D-d) \le D_b \le 0.45(D-d),\\
&4 \le Z \le 50,\\
&0.515 \le f_i \le 0.6,\\
&0.515 \le f_0 \le 0.6,\\
&0.4 \le K_{D\min} \le 0.5,\\
&0.6 \le K_{D\max} \le 0.7,\\
&0.3 \le \varepsilon \le 0.4,\\
&0.02 \le e \le 0.1,\\
&0.6 \le \zeta \le 0.85.
\end{aligned}
$$

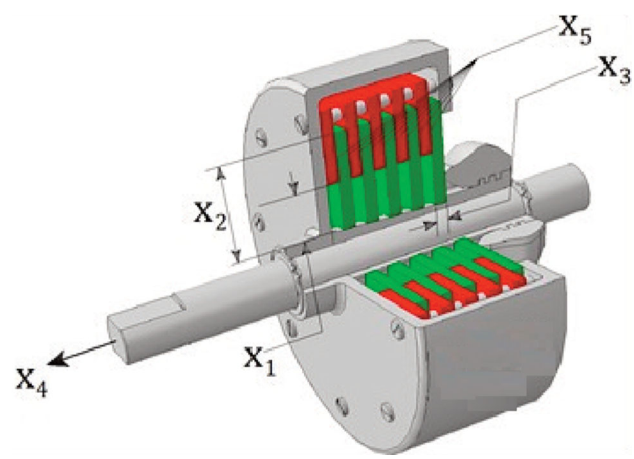



**Fig. 19** Multiple disk clutch brake (Azizyan et al. 2019)

**Table 20** Comparison of results for step-cone pulley problem

| Algorithm | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | Optimal weight |
|---|---|---|---|---|---|---|
| HPO | **38.414146** | **52.858892** | **70.473035** | **84.496123** | **90** | **16.09042816** |
| AEFA | 39.25346 | 54.01469 | 72.01386 | 86.34195 | 89.03809 | 16.6218705 |
| PVC | 40 | 54.76430219 | 73.013177 | 88.428419 | 85.98624 | 16.63450513 |
| ABC | NAN | NAN | NAN | NAN | NAN | 16.634655 |
| TLBO | 40 | 54.7643 | 73.01318 | 88.42842 | 85.98624 | 16.63451 |
| COOT | 38.58523 | 53.09449 | 70.78712 | 84.87239 | 89.82746 | 16.203291 |

**Table 21** Comparison of optimized designs for multi-plate disk clutch brake

| Algorithm | $r_i(x_1)$ | $r_0(x_2)$ | $t(x_3)$ | $F(x_4)$ | $Z(x_5)$ | Optimal cost |
|---|---|---|---|---|---|---|
| HPO | **70** | **90** | **1** | **907.769** | **2** | **0.23524245** |
| AEO | 70 | 90 | 1 | 810 | 3 | 0.3136566 |
| CMVO | 70 | 90 | 1 | 910 | 3 | 0.313656 |
| FSO | 70 | 90 | 1 | 870 | 3 | 0.313657 |
| HHO | 69.9999999992493 | 90 | 1 | 1000 | 2.3128 | 0.259768993 |
| I-ABC greedy | 70 | 90 | 1 | 900 | 3 | 0.313766 |
| PVS | 70 | 90 | 1 | 980 | 3 | 0.31366 |
| QSMFO | 80 | 101.3002 | 3 | 600 | 9 | 0.2902 |
| TLBO | 70 | 90 | 1 | 810 | 3 | 0.313656 |
| WCA | 70 | 90 | 1 | 910 | 3 | 0.313656 |

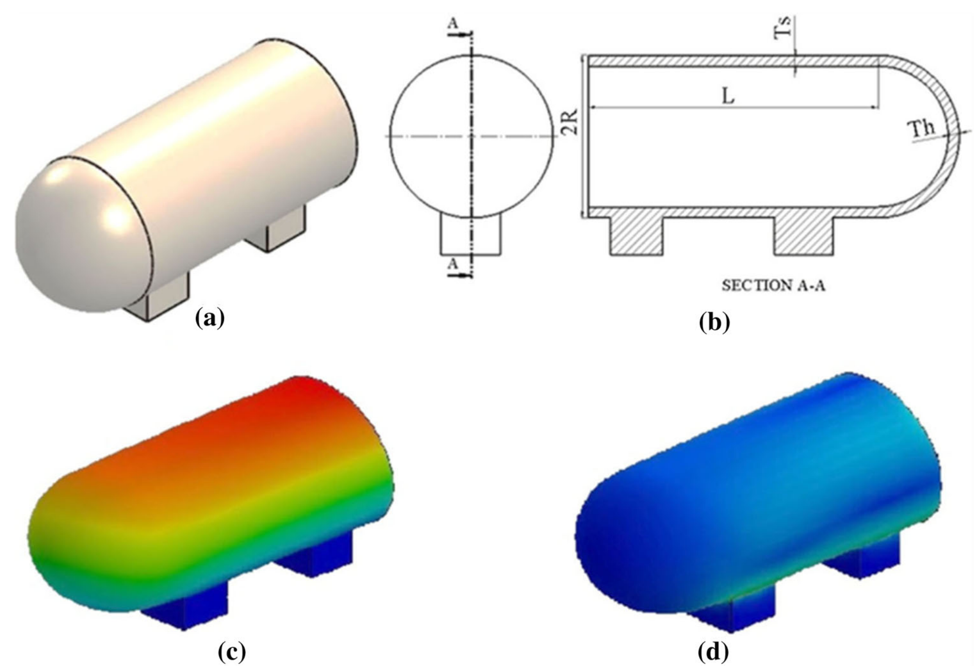



**Fig. 20** Pressure vessel design (Khalilpourazari and Khalilpourazary 2019)

**Table 22** Results for the pressure vessel

| Algorithms | Ts | Th | R | L | Weight |
|---|---|---|---|---|---|
| HPO | **0.778168** | **0.384649** | **40.3196187** | **200** | **5885.33277** |
| AEO | 0.8374205 | 0.413937 | 43.389597 | 161.268592 | 5994.50695 |
| BA | 0.812500 | 0.437500 | 42.098445 | 176.636595 | 6059.7143 |
| CSS | 0.812500 | 0.437500 | 42.103624 | 176.572656 | 6059.0888 |
| GA | 0.812500 | 0.437500 | 42.097398 | 176.654050 | 6059.9463 |
| GPEAae | 0.812500 | 0.437500 | 42.098497 | 176.635954 | 6059.708025 |
| G-QPSO | 0.812500 | 0.437500 | 42.0984 | 176.6372 | 6059.7208 |
| GWO | 0.8125 | 0.4345 | 42.089181 | 176.758731 | 6051.5639 |
| HHO | 0.81758383 | 0.4072927 | 42.09174576 | 176.7196352 | 6000.46259 |
| HPSO | 0.812500 | 0.437500 | 42.0984 | 176.6366 | 6059.7143 |
| MFO | 0.8125 | 0.4375 | 42.098445 | 176.636596 | 6059.7143 |
| SC-GWO | 0.8125 | 0.4375 | 42.0984 | 176.63706 | 6059.7179 |
| WEO | 0.812500 | 0.437500 | 42.098444 | 176.636622 | 6059.71 |
| WOA | 0 .812500 | 0.437500 | 42 .0982699 | 176 .638998 | 6059 .7410 |

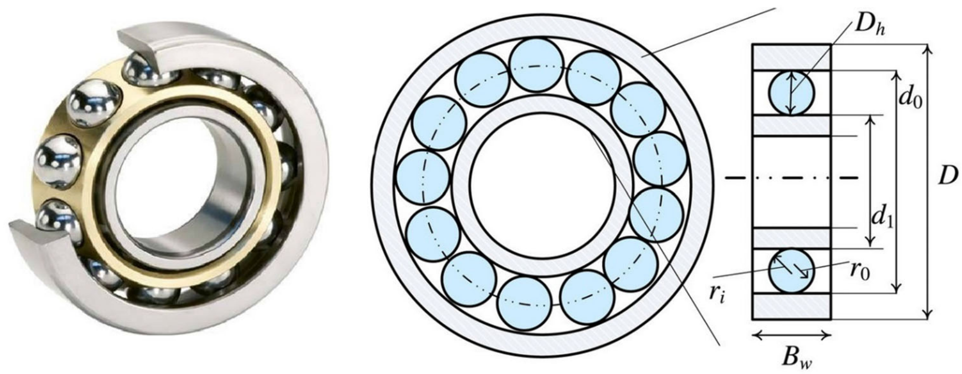



**Fig. 21** Rolling element bearing problem (Heidari et al. 2019)

**Table 23** Comparison of optimized designs for rolling element bearing design problem

| Algorithm | SCA | PVS | TLBO | GA4 | HHO | HPO |
|---|---|---|---|---|---|---|
| $\varepsilon$ | 0.3 | 0.300000 | 0.300000 | 0.300043 | 0.300000 | **0.3000** |
| $e$ | 0.02778 | 0.079990 | 0.068858 | 0.022300 | 0.050474 | **0.0290** |
| $\zeta$ | 0.62912 | 0.700000 | 0.799498 | 0.751000 | 0.600000 | **0.6000** |
| $D_b$ | 21.14834 | 21.425590 | 21.42559 | 21.423000 | 21.000000 | **21.8750** |
| $D_m$ | 125 | 125.719060 | 125.7191 | 125.717100 | 125.000000 | **125.0000** |
| $F_0$ | 0.515 | 0.515000 | 0.515000 | 0.515000 | 0.515000 | **0.5150** |
| $F_i$ | 0.515 | 0.515000 | 0.515000 | 0.515000 | 0.515000 | **0.5150** |
| $K_{Dmax}$ | 0.7 | 0.680160 | 0.633948 | 0.651000 | 0.600000 | **0.7000** |
| $K_{Dmin}$ | 0.5 | 0.400430 | 0.424266 | 0.415900 | 0.400000 | **0.4000** |
| Z | 10.92928 | 11.000000 | 11.000000 | 11.000000 | 11.092073 | **10.7770** |
| Maximum cost | 83,431.117 | 81,859.741210 | 81,859.74 | 81,843.30 | 83,011.88329 | **83,918.4925** |

The structure and parameters of the rolling element bearing design problem are shown in Fig. 21.

The proposed HPO algorithm was tested to solve the rolling element bearing design problem, which is a maximization problem. The results of this experiment were compared with methods such as Harris hawks optimizer (HHO), passing vehicle search (PVS), teaching–learning-based optimization (TLBO), genetic algorithm (GA) and sine cosine algorithm (SCA). The results of this comparison are shown in Table 23. As can be seen from the results, the proposed HPO algorithm was able to provide the best solution to solve this problem.

**Table 24** Results of the proposed HPO algorithm in solving real-world engineering problems

| No | Name | Optimal value | Rank |
|---|---|---|---|
| 1 | Three-bar truss | 263.895844104 | 7 |
| 2 | Reducer design problem | 2892.7292 | 1 |
| 3 | Cantilever beam design | 1.33652825 | 1 |
| 4 | Welded beam design | 1.670240392337 | 1 |
| 5 | Tension/compression spring | 0.012665282823723 | 1 |
| 6 | Step-cone pulley problem | 16.09042816 | 1 |
| 7 | Multi-plate disk clutch brake | 0.23524245 | 1 |
| 8 | Pressure vessel design | 5885.33277 | 1 |
| 9 | Rolling element bearing problem | 83,918.4925 | 1 |

As mentioned before, in real-world problems, the best solutions obtained by different algorithms are reported. However, many of these algorithms achieved these results in better conditions, including more iterations, more population and number function evaluations (NFE). Despite all this, the proposed HPO algorithm still performed better in solving these problems. In summary, the best optimal value obtained by the proposed HPO algorithm for real-world problems is given in Table 24.

## 6 Conclusion

In this paper, a new population-based optimization algorithm is proposed that is inspired by hunters' behavior such as lions and leopards and prey such as deer and gazelle.

The unique characteristics, such as hunting a prey out of the group and moving the prey toward the leader in front of the group, are the main motivation to create this optimization algorithm. In order to evaluate the algorithm in terms of exploration, exploitation and scalability, 43 test functions including seven unimodal test functions to evaluate algorithm exploitation, six multi-modal test functions to evaluate algorithm exploration, and CEC2017 test functions containing 30 functions which at least half of the functions are among the challenging hybrid and composition functions, to evaluate escape from the local optimal and the evaluation of the balance between the exploration phase and exploitation phase were used. The results showed that the proposed HPO algorithm has sufficient exploration and exploitation power to solve unimodal and multi-modal problems and establishes a good balance between these two phases. The HPO algorithm presented very competitive results compared to the well-known and new optimization algorithms. To further evaluate, the proposed algorithm was tested on nine real-world problems. The results showed that the proposed algorithm provided better solutions to solve these problems, while some algorithms were tested with better conditions. For future work, the proposed algorithm can be used to solve problems in different fields of study. The development of binary and the multi-objective versions is also proposed.

**Funding** No funding is provided for the preparation of manuscript.

**Data availability** We used own data and we used own coding.

## Declarations

**Conflict of interest** I. Naruei, Dr. F. Keynia and A. Sabbagh Molahosseini declared that they have no conflict of interest.

## References

Aljarah I, Mafarja M, Heidari AA et al (2018) Asynchronous accelerating multi-leader salp chains for feature selection. Appl Soft Comput 71:964–979. https://doi.org/10.1016/j.asoc.2018.07.040

Anita YA, Kumar N (2020) Artificial electric field algorithm for engineering optimization problems. Expert Syst Appl 149:113308. https://doi.org/10.1016/j.eswa.2020.113308

Arora JS (2017) Introduction to optimum design. Elsevier, Amsterdam

Atashpaz-Gargari E, Lucas C (2007) Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In: 2007 IEEE Congress on Evolutionary Computation. IEEE, pp 4661–4667

Awad NH, Ali MZ, Suganthan PN, Liang JJ, Qu BY (2017) Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective real-parameter numerical optimization. In: 2017 IEEE Congress on Evolutionary Computation (CEC)

Azizyan G, Miarnaeimi F, Rashki M, Shabakhty N (2019) Flying squirrel optimizer (FSO): a novel SI-based optimization algorithm for engineering problems. Iran J Optim 11:177–205

Basturk B, Karaboga D (2006) An artificial bee colony (ABC) algorithm for numeric function optimization. In: Proceedings of the IEEE swarm intelligence symposium, Indianapolis, IN, USA. In: May. pp 12–14

Berryman A (2002) Population cycles: the case for trophic interactions. Oxford University Press, Oxford

Berryman AA (1992) The orgins and evolution of predator-prey theory. Ecology 73:1530–1535. https://doi.org/10.2307/1940005

Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci (ny) 237:82–117. https://doi.org/10.1016/j.ins.2013.02.041

Cheng M-Y, Prayogo D (2014) Symbiotic organisms search: a new metaheuristic optimization algorithm. Comput Struct 139:98–112. https://doi.org/10.1016/j.compstruc.2014.03.007

Cdos Santos Coelho LS (2010) Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems. Expert Syst Appl 37:1676–1683. https://doi.org/10.1016/j.eswa.2009.06.044

Coello Coello CA (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191:1245–1287. https://doi.org/10.1016/S0045-7825(01)00323-1

Colorni A, Dorigo M, Maniezzo V (1991) Distributed Optimization by Ant Colonies. In: European Conference on artificial life. Cambridge, MA, pp 134–142

Crawford B, Soto R, Astorga G et al (2017) Putting continuous metaheuristics to work in binary search spaces. Complexity 2017:1–19. https://doi.org/10.1155/2017/8404231

Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 1:3–18. https://doi.org/10.1016/j.swevo.2011.02.002

Deuflhard P (2011) Newton methods for nonlinear problems. Springer, Berlin

Dhiman G, Kumar V (2018) Emperor penguin optimizer: a bio-inspired algorithm for engineering problems. Knowledge-Based Syst 159:20–50. https://doi.org/10.1016/j.knosys.2018.06.001

Dhiman G, Kumar V (2017) Spotted hyena optimizer: a novel bio-inspired based metaheuristic technique for engineering applications. Adv Eng Softw 114:48–70. https://doi.org/10.1016/j.advengsoft.2017.05.014

Dokeroglu T, Sevinc E, Kucukyilmaz T, Cosar A (2019) A survey on new generation metaheuristic algorithms. Comput Ind Eng 137:106040. https://doi.org/10.1016/j.cie.2019.106040

Dorigo M, Stützle T (2004) Ant Colony Optimization. Bradford Company, Scituate, MA, USA

Eberhart R, Kennedy J (2002) A new optimizer using particle swarm theory. In: MHS'95. In: Proceedings of the sixth international symposium on micro machine and human science. IEEE, pp 39–43

Eskandar H, Sadollah A, Bahreininejad A, Hamdi M (2012) Water cycle algorithm—a novel metaheuristic optimization method for solving constrained engineering optimization problems. Comput Struct 110–111:151–166. https://doi.org/10.1016/j.compstruc.2012.07.010

Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley, Oxford

Gandomi AH, Yang X-S, Alavi AH (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization

problems. Eng Comput 29:17–35. https://doi.org/10.1007/s00366-011-0241-y
Gao C, Hu Z, Xiong Z, Su Q (2020) Grey prediction evolution algorithm based on accelerated even grey model. IEEE Access 8:107941–107957. https://doi.org/10.1109/ACCESS.2020.3001194
García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. J Heuristics 15:617–644. https://doi.org/10.1007/s10732-008-9080-4
Gupta S, Deep K, Moayedi H et al (2020) Sine cosine grey wolf optimizer to solve engineering design problems. Eng Comput. https://doi.org/10.1007/s00366-020-00996-y
Han L, Ma Z, Hethcote HW (2001) Four predator prey models with infectious diseases. Math Comput Model 34:849–858. https://doi.org/10.1016/S0895-7177(01)00104-2
Hassan R, Cohanim B, de Weck O, Venter G (2005) A Comparison of Particle Swarm Optimization and the Genetic Algorithm. In: 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference. American Institute of Aeronautics and Astronautics, Reston, Virigina
He Q, Wang L (2007a) An effective co-evolutionary particle swarm optimization for constrained engineering design problems. Eng Appl Artif Intell 20:89–99. https://doi.org/10.1016/j.engappai.2006.03.003
He Q, Wang L (2007b) A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. Appl Math Comput 186:1407–1422. https://doi.org/10.1016/j.amc.2006.07.134
Heidari AA, Ali Abbaspour R, Rezaee Jordehi A (2017) An efficient chaotic water cycle algorithm for optimization tasks. Neural Comput Appl 28:57–85. https://doi.org/10.1007/s00521-015-2037-2
Heidari AA, Mirjalili S, Faris H et al (2019) Harris hawks optimization: Algorithm and applications. Futur Gener Comput Syst 97:849–872. https://doi.org/10.1016/j.future.2019.02.028
Hillier MS, Hillier FS (2003) Conventional optimization techniques. In: Evolutionary optimization. Kluwer Academic Publishers, Boston, pp 3–25
Holland JH (1967) Genetic algorithms understand genetic algorithms. Surprise 96(1):12–15. https://doi.org/10.2307/24939139
Holland JH, Reitman JS (1977) Cognitive systems based on adaptive algorithms. ACM SIGART Bull. https://doi.org/10.1145/1045343.1045373
Houssein EH, Saad MR, Hashim FA et al (2020) Lévy flight distribution: A new metaheuristic algorithm for solving engineering optimization problems. Eng Appl Artif Intell 94:103731. https://doi.org/10.1016/j.engappai.2020.103731
Huang F, Wang L, He Q (2007) An effective co-evolutionary differential evolution for constrained optimization. Appl Math Comput 186:340–356. https://doi.org/10.1016/j.amc.2006.07.105
Hussain K, Mohd Salleh MN, Cheng S, Shi Y (2018) Metaheuristic research: a comprehensive survey. Artif Intell Rev. https://doi.org/10.1007/s10462-017-9605-z
Jahangiri M, Hadianfard MA, Najafgholipour MA et al (2020) Interactive autodidactic school: a new metaheuristic optimization algorithm for solving mathematical and structural design optimization problems. Comput Struct 235:106268. https://doi.org/10.1016/j.compstruc.2020.106268
Kaur S, Awasthi LK, Sangal AL, Dhiman G (2020) Tunicate Swarm Algorithm: A new bio-inspired based metaheuristic paradigm for global optimization. Eng Appl Artif Intell 90:103541. https://doi.org/10.1016/j.engappai.2020.103541
Kaveh A, Bakhshpoori T (2016) Water Evaporation Optimization: A novel physically inspired optimization algorithm. Comput Struct 167:69–85. https://doi.org/10.1016/j.compstruc.2016.01.008
Kaveh A, Dadras A (2017) A novel meta-heuristic optimization algorithm: Thermal exchange optimization. Adv Eng Softw 110:69–84. https://doi.org/10.1016/j.advengsoft.2017.03.014
Kaveh A, Farhoudi N (2013) A new optimization method: Dolphin echolocation. Adv Eng Softw 59:53–70. https://doi.org/10.1016/j.advengsoft.2013.03.004
Kaveh A, Khayatazad M (2012) A new meta-heuristic method: ray optimization. Comput Struct 112–113:283–294. https://doi.org/10.1016/j.compstruc.2012.09.003
Kaveh A, Mahdavi VR (2014) Colliding bodies optimization: A novel meta-heuristic method. Comput Struct 139:18–27. https://doi.org/10.1016/j.compstruc.2014.04.005
Kaveh A, Motie Share MA, Moslehi M (2013) Magnetic charged system search: a new meta-heuristic algorithm for optimization. Acta Mech 224:85–107. https://doi.org/10.1007/s00707-012-0745-6
Kaveh A, Talatahari S (2010) A novel heuristic optimization method: charged system search. Acta Mech 213:267–289. https://doi.org/10.1007/s00707-009-0270-4
Khalilpourazari S, Khalilpourazary S (2019) An efficient hybrid algorithm based on Water Cycle and Moth-Flame Optimization algorithms for solving numerical and constrained engineering optimization problems. Soft Comput 23:1699–1722. https://doi.org/10.1007/s00500-017-2894-y
Krebs CJ (2009) Ecology: the experimental analysis of distribution and abundance. Pearson Benjamin Cummings
Krohne DT (2000) Ie General Ecology. Cengage Learning, Inc
Kumar A, Wu G, Ali MZ et al (2020) A test-suite of non-convex constrained optimization problems from the real-world and some baseline results. Swarm Evol Comput 56:100693. https://doi.org/10.1016/j.swevo.2020.100693
Mafarja M, Aljarah I, Heidari AA et al (2018a) Evolutionary population dynamics and grasshopper optimization approaches for feature selection problems. Knowl-Based Syst 145:25–45. https://doi.org/10.1016/j.knosys.2017.12.037
Mafarja M, Aljarah I, Heidari AA et al (2018b) Binary dragonfly optimization for feature selection using time-varying transfer functions. Knowle-Based Syst 161:185–204. https://doi.org/10.1016/j.knosys.2018.08.003
Manjarres D, Landa-Torres I, Gil-Lopez S et al (2013) A survey on applications of the harmony search algorithm. Eng Appl Artif Intell 26:1818–1831. https://doi.org/10.1016/j.engappai.2013.05.008
Masadeh R, Mahafzah BA, Sharieh A (2019) Sea Lion Optimization Algorithm. Int J Adv Comput Sci Appl. https://doi.org/10.14569/IJACSA.2019.0100548
Mirjalili S (2016) SCA: A Sine Cosine Algorithm for solving optimization problems. Knowl-Based Syst 96:120–133. https://doi.org/10.1016/j.knosys.2015.12.022
Mirjalili S (2015a) The ant lion optimizer. Adv Eng Softw 83:80–98. https://doi.org/10.1016/j.advengsoft.2015.01.010
Mirjalili S (2015b) Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. Knowl-Based Syst 89:228–249. https://doi.org/10.1016/j.knosys.2015.07.006
Mirjalili S, Gandomi AH, Mirjalili SZ et al (2017) Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. Adv Eng Softw 114:163–191. https://doi.org/10.1016/j.advengsoft.2017.07.002
Mirjalili S, Lewis A (2016) The Whale Optimization Algorithm. Adv Eng Softw 95:51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008
Mirjalili S, Mirjalili SM, Hatamlou A (2016) Multi-Verse Optimizer: a nature-inspired algorithm for global optimization. Neural

Comput Appl 27:495–513. https://doi.org/10.1007/s00521-015-1870-7
Mirjalili S, Mirjalili SM, Lewis A (2014) Grey Wolf Optimizer. Adv Eng Softw 69:46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007
Naruei I, Keynia F (2021a) Wild horse optimizer: a new meta-heuristic algorithm for solving engineering optimization problems. Eng Comput. https://doi.org/10.1007/s00366-021-01438-z
Naruei I, Keynia F (2021b) A new optimization method based on COOT bird natural life model. Expert Syst Appl 183:115352. https://doi.org/10.1016/j.eswa.2021.115352
Parejo JA, Ruiz-Cortés A, Lozano S, Fernandez P (2012) Metaheuristic optimization frameworks: a survey and benchmarking. Soft Comput 16:527–561. https://doi.org/10.1007/s00500-011-0754-8
Rajabioun R (2011) Cuckoo optimization algorithm. Appl Soft Comput 11:5508–5518. https://doi.org/10.1016/j.asoc.2011.05.008
Rashedi E, Nezamabadi-pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci (ny) 179:2232–2248. https://doi.org/10.1016/j.ins.2009.03.004
Rizk-Allah RM (2018) An improved sine–cosine algorithm based on orthogonal parallel information for global optimization. Soft Comput. https://doi.org/10.1007/s00500-018-3355-y
Salimi H (2015) Stochastic Fractal Search: A powerful metaheuristic algorithm. Knowl-Based Syst 75:1–18. https://doi.org/10.1016/j.knosys.2014.07.025
Saremi S, Mirjalili S, Lewis A (2017) Grasshopper optimisation algorithm: theory and application. Adv Eng Softw 105:30–47. https://doi.org/10.1016/j.advengsoft.2017.01.004
Savsani P, Savsani V (2016) Passing vehicle search (PVS): A novel metaheuristic algorithm. Appl Math Model 40:3951–3978. https://doi.org/10.1016/j.apm.2015.10.040
Sayed GI, Darwish A, Hassanien AE (2018) A new chaotic multi-verse optimization algorithm for solving engineering optimization problems. J Exp Theor Artif Intell 30:293–317. https://doi.org/10.1080/0952813X.2018.1430858
Sharma TK, Abraham A (2020) Artificial bee colony with enhanced food locations for solving mechanical engineering design problems. J Ambient Intell Humaniz Comput 11:267–290. https://doi.org/10.1007/s12652-019-01265-7
Shen L, Chen H, Yu Z et al (2016) Evolving support vector machines using fruit fly optimization for medical data classification. Knowl-Based Syst 96:61–75. https://doi.org/10.1016/j.knosys.2016.01.002
Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12:702–713. https://doi.org/10.1109/TEVC.2008.919004
Simpson AR, Dandy GC, Murphy LJ (1994) Genetic algorithms compared to other techniques for pipe optimization. J Water Resour Plan Manag 120:423–443. https://doi.org/10.1061/(ASCE)0733-9496(1994)120:4(423)
Spall JC (2003) Introduction to stochastic search and optimization. Wiley, Hoboken
Storn R, Price K (1995) Differential Evolution- A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Tech Rep TR-95–012 11:1–12. https://doi.org/10.1023/A:1008202821328
Talbi E-G (2009) Metaheuristics: from design to implementation. Wiley, Hoboken
Turchin P (2003) Complex population dynamics: a theoretical/empirical synthesis. Princeton University Press, Princeton
van den Bergh F, Engelbrecht AP (2006) A study of particle swarm optimization particle trajectories. Inf Sci (Ny) 176:937–971. https://doi.org/10.1016/j.ins.2005.02.003
Wang M, Chen H, Yang B et al (2017) Toward an optimal kernel extreme learning machine using a chaotic moth-flame optimization strategy with applications in medical diagnoses. Neurocomputing 267:69–84. https://doi.org/10.1016/j.neucom.2017.04.060
Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1:67–82. https://doi.org/10.1109/4235.585893
Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3:82–102. https://doi.org/10.1109/4235.771163
Yang X-S, Deb S (2009) Cuckoo Search via Lévy flights. In: 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC). IEEE, pp 210–214
Yang X-S, Deb S (2010) Engineering Optimisation by Cuckoo Search
Yang X, Hossein Gandomi A (2012) Bat algorithm: a novel approach for global engineering optimization. Eng Comput 29:464–483. https://doi.org/10.1108/02644401211235834
Yang XS (2010) Firefly algorithm, stochastic test functions and design optimisation. Int J Bio-Inspired Comput 2:78. https://doi.org/10.1504/IJBIC.2010.032124
Yu C, Heidari AA, Chen H (2020) A quantum-behaved simulated annealing algorithm-based moth-flame optimization method. Appl Math Model 87:1–19. https://doi.org/10.1016/j.apm.2020.04.019
Zhang M, Luo W, Wang X et al (2008) Differential evolution with dynamic stochastic selection for constrained optimization. Inf Sci Int J 178(15):3043–3074. https://doi.org/10.1016/j.ins.2008.02.014
Zhang Q, Chen H, Luo J et al (2018) Chaos Enhanced Bacterial Foraging Optimization for Global Optimization. IEEE Access 6:64905–64919. https://doi.org/10.1109/ACCESS.2018.2876996
Zhao W, Wang L, Zhang Z (2020) Artificial ecosystem-based optimization: a novel nature-inspired meta-heuristic algorithm. Neural Comput Appl 32:9383–9425. https://doi.org/10.1007/s00521-019-04452-x
Zhou A, Qu B-Y, Li H et al (2011) Multiobjective evolutionary algorithms: A survey of the state of the art. Swarm Evol Comput 1:32–49. https://doi.org/10.1016/j.swevo.2011.03.001