Neural Computing and Applications (2021) 33:1139–1157
https://doi.org/10.1007/s00521-020-05004-4

ORIGINAL ARTICLE

Battle royale optimization algorithm

Taymaz Rahkar Farshi[1]

Received: 31 March 2020 / Accepted: 2 May 2020 / Published online: 2 June 2020

Abstract
Recently, several metaheuristic optimization approaches have been developed for solving many complex problems in various areas. Most of these optimization algorithms are inspired by nature or the social behavior of some animals. However, there is no optimization algorithm which has been inspired by a game. In this paper, a novel metaheuristic optimization algorithm, named BRO (battle royale optimization), is proposed. The proposed method is inspired by a genre of digital games knowns as "battle royale." BRO is a population-based algorithm in which each individual is represented by a soldier/player that would like to move toward the safest (best) place and ultimately survive. The proposed scheme has been compared with the well-known PSO algorithm and six recent proposed optimization algorithms on nineteen benchmark optimization functions. Moreover, to evaluate the performance of the proposed algorithm on real-world engineering problems, the inverse kinematics problem of the 6-DOF PUMA 560 robot arm is considered. The experimental results show that, according to both convergence and accuracy, the proposed algorithm is an efficient method and provides promising and competitive results.

Keywords Battle royale · Optimization · Swarm intelligence · Metaheuristic

1 Introduction

Optimization is the process of finding the best possible solution among a set of candidate solutions for a particular problem. Optimization algorithms play a critical role in many practical problems in both scientific and industrial application fields. In the past two decades, metaheuristic algorithms have been widely used to solve optimization problems [1]. Evolutionary algorithms (EA), swarm intelligence (SI), and physical phenomena algorithms are the stochastic metaheuristics optimization tools [2–9].

Evolutionary optimization approaches are inspired by Darwin's theory of evolution; they share a mutual mechanism of mutation, selection, and recombination operation. Some well-known EA paradigms are the genetic algorithm (GA) [10], evolution strategies [11], tabu search [12], simulated annealing [13], biogeography-based optimizer (BBO) [14], forest optimization algorithm [15], bird mating optimizer (BMO) [16], and others.

The intelligent collective behavior of various species of living creatures, such as animals and insects, have attracted the attention of researches for last few decades. For example, some optimization approaches are based on Swarm Intelligence, inspired by the collective behavior of both animals and humans. Although one individual is not intelligent, intelligent features are carried out through communication with other individuals [17]. Examples of SI-based approaches are particle swarm optimization [18], ant colony optimization [19], cat swarm optimization [20], artificial bee colony(ABC) [21], animal migration optimization [22], dragonfly algorithm [23], and selfish herd optimizer (SHO) [24], as well as others.

On the other hand, physics-inspired approaches are another type of metaheuristic algorithm; these mimic physical laws such as electromagnetic force, gravitational force, inertia force. The movement and communication of individuals are carried out by means of these rules [17]. Some of well-known, physics-inspired approaches are central force optimization (CFO) [25], gravitational search algorithm (GSA) [26], optics inspired optimization (OIO) [27], water evaporation optimization (WEO) [28], Black

✉ Taymaz Rahkar Farshi
taymaz.farshi@altinbas.edu.tr

1 Software Engineering Department, Altinbas University, Istanbul, Turkey

Hole (BH) [29], charged system search (CSS) [30], Yin-Yang pair optimization (YYPO) [31], thermal exchange optimization [32], electromagnetic field Optimization (EFO) [33] and others.

Each optimization algorithm has own strategy to carry out the concepts of exploration and exploitation [26]. The balance between exploration and exploitation should be maintained, as it is effective in providing successful results in optimization algorithms. Exploration refers to the process of searching for new solutions in a wide area of search space. In other words, unvisited areas are checked whether they are potential escape areas from poor local optima. Exploitation, however, is the search for a new solution within the vicinity of the best solution [16]. For example, selection processes provide exploitation, while crossover and mutation processes provide exploration. On the other hand, at the beginning of iterations with large velocities, the particles expend more effort on exploration. In the following, as velocities decrease toward zero, the particles will expend more effort on exploitation. The coefficient parameters should be set properly in order to provide a good balance between exploration and exploitation [34]. Further, elitism is a significant property of evolutionary algorithms; it maintains the best solution(s) gained during the whole optimization process.

In recent years, a large number of new approaches have been developed and applied to different real-world, complex optimization problems. Despite the fact that a wide range of optimization algorithms have been introduced and they have been commonly accepted and utilized in many research areas and applications, there is still the question of if we need to develop more techniques [35]. The favorable outcome of the metaheuristic approaches in solving optimization problems depends on the balance of exploration and exploitation [36]. However, each metaheuristic algorithm has its own mechanism to utilize the exploration and exploitation attributes using different methodologies and procedures. The no-free-lunch (NFL) theorem [37] proves that an optimization scheme cannot perform well in dealing with all optimization challenges. This theory claims that even if a particular optimization algorithm may outperform others in solving a huge number of optimization problems, it may perform worst in a particular set of problems, among others. As yet, a heuristic algorithm that solves all optimization problems by providing superior performance does not exist. This theory proves that new techniques can deal with challenges and complex and unsolved problems. Hence, it is also needed to propose new approaches alongside enhancing the existing ones.

Although several optimization algorithms have been inspired by nature, society, culture, politics, humankind, etc. [27], there are no optimization algorithms based on a game strategy. In this paper, a new optimization algorithm, called battle royale optimization (BRO) algorithm, is proposed. The proposed method is inspired by the strategy of the battle royale video games.

Like other existing swarm-based approaches, BRO is also population-based; it uses a population of candidate solutions to achieve the best solution. Each solution is called a soldier who is trying to defeat its nearest neighbor. Therefore, a soldier located in a better position can defeat others. Hence, all individuals try to move toward a better position to have a better chance of survival. Finally, at the end of the iteration, the best soldier will be the winner.

This paper is organized as follows: Sect. 2 reviews the battle royale game strategy and describes the BRO algorithm. Section 3 discusses the experimental results of the algorithm. Section 4 includes the performance of the algorithm on a real-world problem. Finally, Sect. 5 presents the conclusions.

2 Battle royale optimizer

2.1 Battle royale game strategy

The genre of battle royal video games, inspired by the Japanese film Battle Royal, features competitive, last-man-standing gameplay. Often, the game area shrinks, and players are required to explore and utilize their surroundings for survival. One of the significant features of battle royale games is that a large number of people can play in a digital platform. These games can be played on an individual versus individual level, but they can also be played in teams of up to five players. The challenge of these games is not just to beat the other players, but the game itself, as the area of gameplay (the "safe area") decreases throughout the game. Any players suddenly caught outside the safe zone either incur damage or risk elimination from the game. These weakened players now need to compete in a smaller space and with injuries. Most battle royale games start with players at equal levels of strength and resources; players are often randomly placed in the game space and are not given much in terms of resources or equipment. Rather, exploration is a key component of the game, as players need to search around for tools to aid in survival (all while avoiding being killed by opponents). Another component of battle royale games that is utilized in the algorithm is the possibility of respawning. While not a feature of all battle royale games, some games give players a chance to respawn/revive themselves. Some games also reward players for the length of their survival, but ultimately only one player (or team) wins [38, 39]. PlayerUnknown's Battlegrounds (PUBG) [39], Call of Duty: Warzone [40], Apex Legends [41], Counter-Strike: Global

Offensive [42], and Ring of Elysium [43] are some of well-known examples manifested in such game types.

To better understand this type of game, a brief description of some of the features of these games is given in this section. Generally, the wars are fought on a specific battlefield map, which is chosen by the players. For instance, Sanhok (see Fig. 1) is one of the famous maps in the PUBG. The Playzone, also called "The Circle" is the area where players have to stay throughout the game. The players who are outside of the circle would be taken damaged every second. This forces players to be close together and tries to bring everyone together in the limited safe space. Moreover, Damage ticks are sent to each player at exactly the same time. On the map, out of the circle are

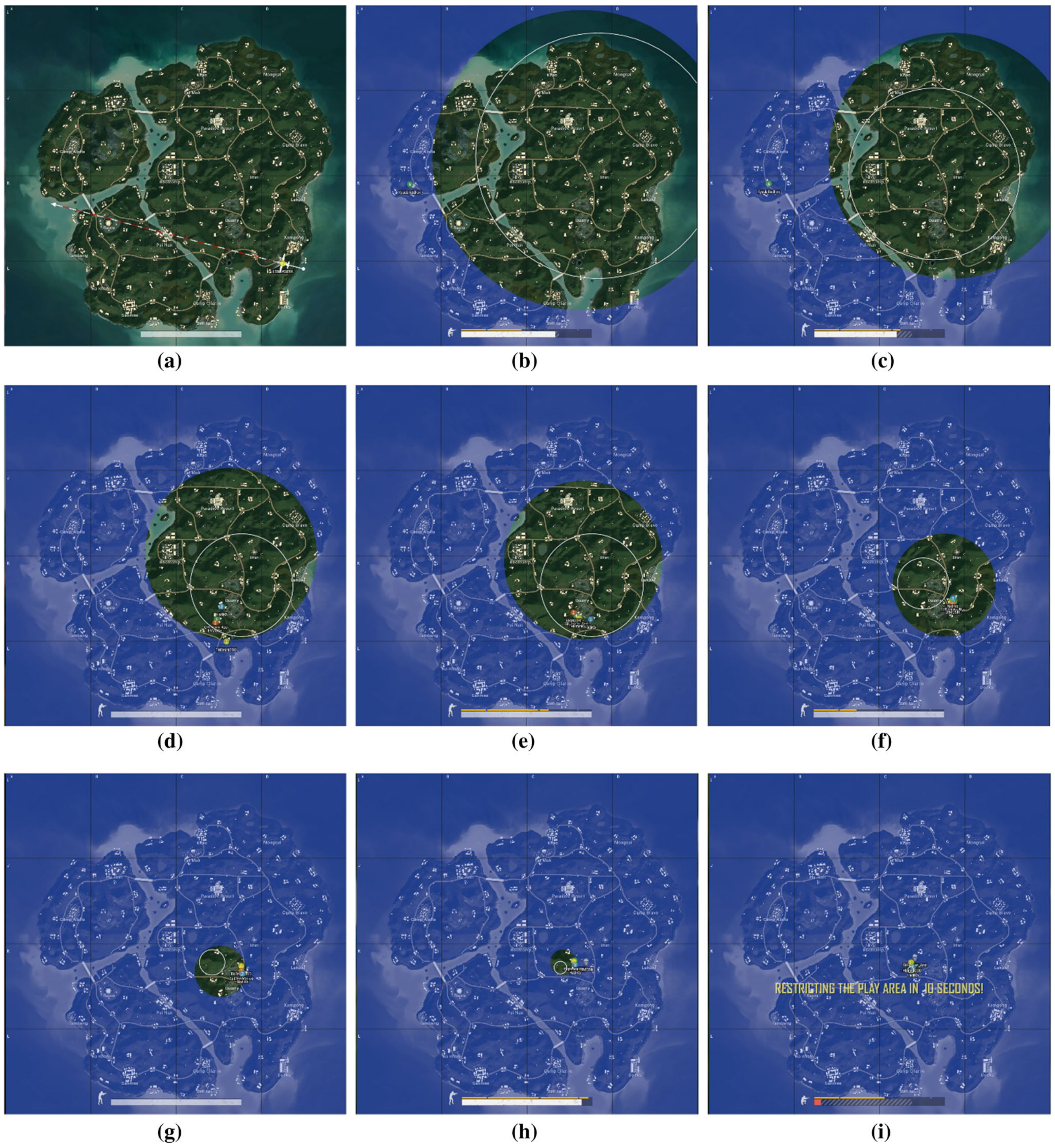


Fig. 1 The playzone after each phase of game

overlaid with blue color. In each phase, the playzone starts shrinking. Before the playzone begins shrinking, the next restricted playzone is marked by a white smaller circle. This process is demonstrated in Fig. 1. During this process, players try to kill each other and survive until the last phase. On the other hand, PUBG has another play mode called deathmatch. The aim of this mode is to kill/frag as many other players as possible until a kill limit or a time limit. During the game, if a player gets killed by opponents, it respawns at a random point of the battlefield. Finally, the winner will be the player who has the highest number of kills among the others.

2.2 BRO algorithm

First, this section will cover the inspiration of BRO, so that the proposed methodology can be better understood. Some versions of battle royale games begin as players jump from a plane and parachute down onto the map. Moreover, like other swarm-based algorithms, BRO starts with a random population, which will be uniformly distributed throughout the problem space. Next, each individual (soldier/player) tries to hurt the nearest soldier by shooting a weapon. Soldiers in better positions therefore cause damage to their nearest neighbors. When one soldier is hurt by another, its damage level increases by one. These interactions are mathematically calculated per $x_i.\text{damage} = x_i.\text{damage} + 1$, where x_i.damage is the damage level of the ith soldier among the population. Moreover, soldiers want to change their position immediately after experiencing damage and therefore attack opponents from another side. Hence, to focus on exploitation, the damaged solder moves toward a point somewhere between the previous position and best position found so far (elite player). These interactions are mathematically calculated per:

$$x_{\text{dam},d} = x_{\text{dam},d} + r\left(x_{\text{best},d} - x_{\text{dam},d}\right), \tag{1}$$

where r is a randomly generated number uniformly distributed in the range [0,1] and $x_{\text{dam},d}$ is the position of the damaged soldier in dimension d. Moreover, if damaged soldiers can hurt their opponent in the next iteration, x_i.damage will be reset to zero. Also, to focus on exploration, if the damage level of a soldier surpasses the predefined threshold value, the soldier dies and respawn randomly from the feasible problem space, and x_i.damage will reset to zero. Through trial and error, we determined that a value of threshold = 3 was appropriate. This action avoids premature convergence and provides better exploration. The soldier coming back to problem space after being killed is as follows:

$$x_{\text{dam},d} = r(ub_d - lb_d) + lb_d \tag{2}$$

where lb_d and ub_d is the lower and upper bounds of dimension d in problem space, respectively. Furthermore, in every Δ iteration the feasible search space of the problem begins to shrink down toward the best solution. The initial value was $\Delta = \log_{10}(\text{MaxCicle})$ but then $\Delta = \Delta + \text{round}\left(\frac{\Delta}{2}\right)$. Here MaxCicle is the maximum number of generations.

This interaction contributes to both exploration and exploitation. So, the lower and upper bound will be updated as follows:

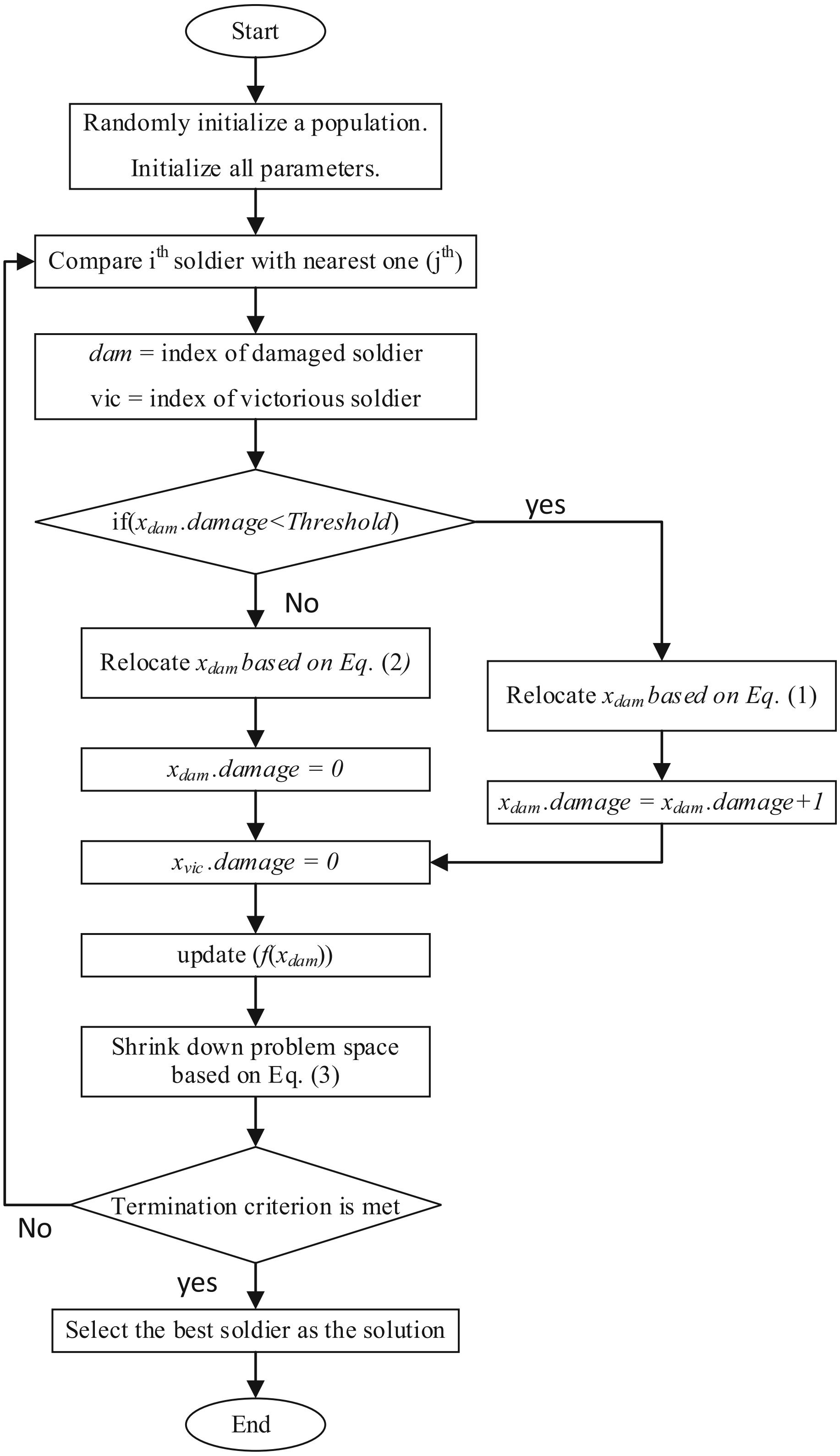
$$lb_d = x_{\text{best},d} - SD(\overline{x_d})$$
$$ub_d = x_{\text{best},d} + SD(\overline{x_d}) \tag{3}$$

Here $SD(\bar{x}_d)$ is the standard deviation of whole population in dimension d and $x_{\text{best},d}$ is the position of the best solution found so far. Hence, if the lb_d/ub_d exceeds the original lower/upper bound, then it sets to the original lb_d/ub_d. Furthermore, to focus on elitism, the best player/soldier found in each iteration is kept and considered as an elite.

Apart from dimensions of the problem, the computational complexity of the proposed approach depends on the population size, and maximum number of iterations. Since each solution must be compared with all others in order to calculate its Euclidean distance from all other solutions, given the population size n the complexity of calculating for all solutions is $O(n^2)$. Hence, given the number of iterations m the computational complexity of BRO is $O(n^3)$.

The general flow and pseudo-code of BRO are presented in Fig. 2 and Algorithm 1, respectively

Fig. 2 Flowchart of BRO


Start
Randomly initialize a population.
Initialize all parameters.
Compare i^th soldier with nearest one (j^th)
dam = index of damaged soldier
vic = index of victorious soldier
if(x_dam.damage<Threshold)
yes
No
Relocate x_dam based on Eq. (2)
Relocate x_dam based on Eq. (1)
x_dam.damage = 0
x_dam.damage = x_dam.damage+1
x_vic.damage = 0
update (f(x_dam))
Shrink down problem space based on Eq. (3)
Termination criterion is met
No
yes
Select the best soldier as the solution
End

Algorithm 1 pseudocode of BRO

```
Begin
Randomly initialize a population ($\vec{x}$)
Initialize all parameters;
$shrink = \text{ceil}(\log_{10}(MaxXicle)))$
$\Delta$= round ($MaxXicle$ /Shrink)
Iter = 0;
while termination criterion is not met do
  iter = iter +1
   for i=1: population_size
     //compare i^th soldier with nearest one (j^th)
      dam = j
      vic = i
       if $f(x_i) < f(x_j)$
         dam = i
         vic =j
      end if
      if $x_{dam}.damage$ < Threshold
        for d=1: Dimension
            change the position of damaged solder based on:
            $x_{dam,d} = r(\max(x_{dam,d}, x_{best,d}) - \min(x_{dam,d}, x_{best,d})) + \max(x_{dam,d}, x_{best,d})$.
        end for d
        $x_{dam}.damage = x_i.damage + 1$
        $x_{vic}.damage = 0$
      else
        for d=1: Dimension
            $x_{dam,d} = r(ub_d - lb_d) + lb_d$
        end for d
        update $f(x_{dam})$
        $x_{dam}.damage = 0$
   end for i
      if iter>=$\Delta$
        update ($ub - lb$) based on Eq. (3)
        $\Delta$=$\Delta$+ round ($\Delta$/2);
      end if
       if the $lb_d$ or $ub_d$ exceeds the original lower/upper bound then it set to the original $lb_d$ or $ub_d$.
end while
Select the best soldier as the solution.
```

Table 1 Unimodal benchmark test functions

Function	Name	Range	Shift position	$f_{\min}$
$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	Sphere	[− 100, 100]	[− 30, − 30, − 30,.. . ., − 30]	
$f_2(\mathbf{x}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	Schwefel 2.20	[− 10, 10]	[− 3, − 3, − 3,.. . ., − 3]	
$f_3(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	Rotated hyper-ellipsoids	[− 100, 100]	[− 30, − 30, − 30,.. . ., − 30]	
$f_4(\mathbf{x}) = \max_{i=1,\ldots,n} \lvert x_i \rvert$	Schwefel 2.21	[− 100, 100]	[− 30, − 30, − 30,.. . ., − 30]	
$f_5(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$	Rosenbrock	[− 30, 30]	[− 15, − 15, − 15,.. . ., − 15]	
$f_6(\mathbf{x}) = \sum_{i=1}^{n} ([x_i + 00.5])^2$	Step	[− 100, 100]	[− 750, − 750,.. . ., − 750]	
$f_7(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + \text{rand}[0, 1)$	Quartic	[− 128, 128]	[− 25, −25,.. . ., − 25]	

Table 2 Multimodal benchmark test functions

Function	Name	Range	Shift position	$f_{\min}$
$f_9(\mathbf{x}) = \sum_{i=1}^{n} x_i sin(\sqrt{\lvert x_i \rvert})$	Schwefel	[− 500, 500]	[− 300, − 300,…, − 300]	− 418.9829 × dim
$f_{10}(\mathbf{x}) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$	Rstrigin	[− 5.12, 5.12]	[− 2, − 2, − 2,…, − 2]	0
$f_{11}(\mathbf{x}) = -20.exp\left(-0,2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i) + 20 + exp(1)$	Ackley	[− 32, 32]		0
$f_{12}(\mathbf{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	Griewank	[− 600, 600]	[− 400, − 400,…, − 400]	0
$f_{13}(\mathbf{x}) = \frac{\pi}{n} \times \{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 [1 + 10\sin^2(\pi y_{i+1})]$	Penalized	[− 50, 50]	[− 30, − 30, − 30,…, -30]	0
$+ (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)\, y_i = 1 + \frac{1}{4}(x_i + 1)\ \ u(x_i, a, k, m)$				
$= \begin{cases} k(x_i - a)^m & \text{if } x_i > a \\ 0 & \text{if } -a \leq x_i \leq a \\ k(-x_i - a)^m & \text{if } x_i < -a \end{cases}$				
$f_{14}(\mathbf{x}) = 0.1\{\sin^2(3\pi x) + (x_i - 1)^2(1 + \sin^2(3\pi y)) + (x_n - 1)^2(1 + \sin^2(2\pi x_n))\}$	Levi	[− 50, 50]	[− 100, 100]	0
$+ \sum_{i=1}^{n} u(x_i, 5, 100, 4)$				

Table 3 Composite benchmark test functions

Function	Name	Range	$f_{\min}$
$f_{14}(\mathrm{CF1}):$ $f_1, f_2, f_3, \ldots, f_{10} =$ Sphere Function $[\sigma_1, \sigma_2, \ldots \sigma_{10}] = [1, 1, \ldots, 1]$ $[\lambda_1, \lambda_2, \ldots \lambda_{10}] = \left[\frac{5}{100}, \frac{5}{100}, \ldots, \frac{5}{100}\right]$	CF1	[− 5, 5]	0
$f_{15}(\mathrm{CF2}):$ $f_1, f_2, f_3, \ldots, f_{10} =$ Griewank's Function $[\sigma_1, \sigma_2, \ldots \sigma_{10}] = [1, 1, \ldots, 1]$ $[\lambda_1, \lambda_2, \ldots \lambda_{10}] = \left[\frac{5}{100}, \frac{5}{100}, \ldots, \frac{5}{100}\right]$	CF2	[− 5, 5]	0
$f_{16}(\mathrm{CF3}):$ $f_1, f_2, f_3, \ldots, f_{10} =$ Griewank's Function $[\sigma_1, \sigma_2, \ldots, \sigma_{10}] = [1, 1, \ldots, 1]$ $[\lambda_1, \lambda_2, \ldots \lambda_{10}] = [1, 1, \ldots, 1]$	CF3	[− 5, 5]	− 1.0316
$f_{17}(\mathrm{CF4}):$ $f_1, f_2 =$ Ackley$'s$ Function $f_3, f_4 =$ Rastrigin$'s$ Function $f_5, f_6 =$ Weierstrass Function $f_7, f_8 =$ Griewank's Function $f_9, f_{10} =$ Sphere Function $[\sigma_1, \sigma_2, \ldots \sigma_{10}] = [1, 1, \ldots, 1]$ $[\lambda_1, \lambda_2, \ldots \lambda_{10}] = \left[\frac{5}{32}, \frac{5}{32}, 1, 1, \frac{5}{0.5}, \frac{5}{0.5}, \frac{5}{100}, \frac{5}{100}, \frac{5}{100}, \frac{5}{100}\right]$	CF4	[− 5, 5]	0
$f_{18}(\mathrm{CF5}):$ $f_1, f_2 =$ Rastrigin$'s$ Function $f_3, f_4 =$ Weierstrass Function $f_5, f_6 =$ Griewank's Function $f_7, f_8 =$ Ackley$'s$ Function $f_9, f_{10} =$ Sphere Function $[\sigma_1, \sigma_2, \ldots \sigma_{10}] = [1, 1, \ldots, 1]$ $[\lambda_1, \lambda_2, \ldots \lambda_{10}] = \left[\frac{1}{5}, \frac{1}{5}, \frac{5}{0.5}, \frac{5}{0.5}, \frac{5}{100}, \frac{5}{100}, \frac{5}{32}, \frac{5}{32}, \frac{5}{100}, \frac{5}{100}\right]$	CF5	[− 5, 5]	0

Table 3 continued

Function	Name	Range	$f_{\min}$
$f_{19}(\mathrm{CF6})$:	CF6	[− 5, 5]	− 1.032
$f_1, f_2 =$ Rastrigin's Function			
$f_3, f_4 =$ Weierstrass Function			
$f_5, f_6 =$ Griewank's Function			
$f_7, f_8 =$ Ackley's Function			
$f_9, f_{10} =$ Sphere Function			
$[\sigma_1, \sigma_2, \ldots \sigma_{10}] = [1, 1, \ldots, 1]$			
$[\lambda_1, \lambda_2, \ldots \lambda_{10}] = \left[0.1 \times \frac{1}{5}, 0.2 \times \frac{1}{5}, 0.3 \times \frac{5}{0.5}, 0.4 \times \frac{5}{0.5}, 0.5 \times \frac{5}{100}, 0.6 \times \frac{5}{100}, 0.7 \times \frac{5}{32}, 0.8 \times \frac{5}{32}, 0.9 \times \frac{5}{100}, \frac{5}{100}\right]$			

3 Experimental results and performance evaluation

To evaluate the performance of the proposed BRO algorithm, a comprehensive experimental evaluation and comparison have been performed with the well-known PSO (particle swarm optimization) algorithm and six recent proposed optimization algorithms: ALO (the ant lion optimizer) [44], DA (dragonfly algorithm) [23], MFO (moth–flame optimization algorithm) [45], MVO (multi-verse optimizer) [46], SCA (A Sine Cosine Algorithm) [47], and GOA (Grasshopper optimization algorithm) [48]. The controlling parameters for all algorithms were set according to the recommendations in the original papers. Moreover, all algorithms were implemented in MATLAB and performed on a Core i7-7700 HQ 2.81 processor with 32 GB of RAM. In this study, 19 benchmark mathematical functions (unimodal, multimodal, and composite), listed in Tables 1, 2 and 3, were used for comparison. Additionally, the search space of some of the employed benchmark functions are illustrated in Fig. 3. Tables 1 and 2 also include shifted positions of global optima; the simplicity of unimodal and multimodal test functions meant that they needed to be shifted so that the difficulty would increase.

To do a deep investigation of the performance of BRO in comparison with other algorithms, it was important to apply all algorithms in more complex domains. Hence, to increase the complexity of the optimization task, all unimodal and multimodal functions were employed in 30 dimensions space. Furthermore, the composition functions F14–F19 have been employed in 2, 4, 2, 2, 2, and 3 dimensions space, respectively. These composition functions are the combination of various rotated, shifted, and biased multimodal functions. Eventually, all these benchmark functions became minimization problems, where the proposed method is a maximization scheme. Furthermore, although the proposed algorithm is a maximization scheme, it can be converted to a minimization scheme.

The maximum number of iteration and population size are 500 and 100, respectively, for all algorithms. For as much as the heuristic algorithms are stochastic optimization techniques, all the following experimental results are obtained by taking the average of 25 independent runs for each benchmark function. Furthermore, the mean and standard deviation of the fitness evaluation values after 25 independent runs are utilized as the performance metrics.

In order to find the best threshold value for the maximum amount of damage in the proposed BRO, this parameter has been tuned by trial and error based on some test functions. The numerical results of parameter tuning are illustrated in Table 4. Furthermore, according to the fitness value when the damage threshold is 3, the algorithm

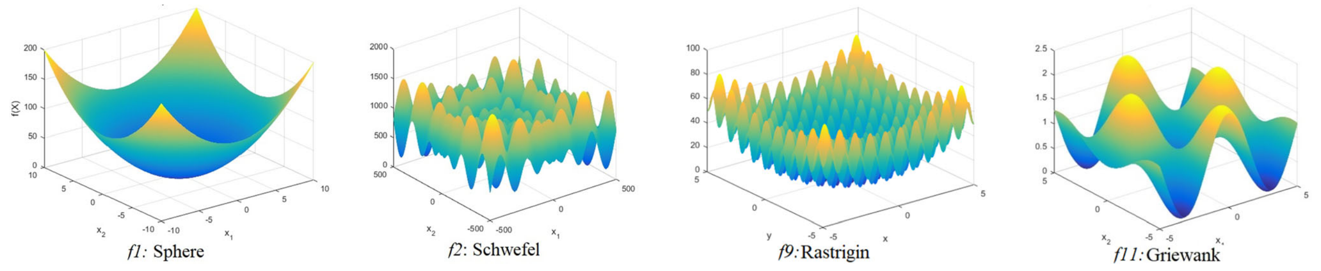


Fig. 3 Search space of some benchmark functions

Table 4 The performance evaluation of BRO for some benchmark functions with different damage thresholds

Function	Maximum damage					
	2	3	4	5	6	7
F1						
Mean	0.000051	**3.0353e−09**	4.0463e−08	2.0491e−08	0.0059696	0.000426
Std	0.000042	**4.1348e−09**	4.1481e−08	2.4788e-08	0.0046666	0.000600
F2						
Mean	0.0040	**0.000046**	0.000049	0.000379	0.1985	0.5778
std	0.0025	**0.000024**	0.000015	0.001117	0.3895	0.5543
F3						
Mean	78.1053	**54.865255**	112.2575	216.7282	448.4380	455.2066
Std	34.8479	**16.117329**	44.4166	79.8389	165.2054	129.2482
F9						
Mean	**32.3960**	48.275350	59.4587	63.2793	65.1498	76.4130
Std	**8.7547**	14.094585	14.0099	15.1982	15.4462	22.5190
F11						
Mean	0.0134	**0.013373**	0.0145	0.0134	0.0223	0.0341
Std	0.0113	0.015796	**0.0122**	0.0132	0.0282	0.0369
F17						
Mean	0.3979	**0.397887**	0.3979	0.3979	0.3979	0.3979
Std	2.2450e-10	**0**	**0**	**0**	**0**	**0**

The best results are shown in bold

provides better performance on more than 80% of the test functions listed in Table 4. For instance, in F3, when the damage threshold is 3, the algorithm gained better fitness value compared to when the value was 2. Moreover, as the damage threshold grows, the fitness values also gradually get worse. So, by getting away from this threshold value, the efficiency of the algorithm is reduced.

The experimental results over all algorithm have been provided in Tables 5, 6, and 7 for all unimodal, multimodal and composition test functions, respectively.

As Table 5 shows, the proposed BRO provides the best results on seven unimodal functions. The algorithms are ranked based on the mean value of fitness value. The proposed method ranks first overall, whereas PSO and MFO rank second and third, respectively. Additionally, the fitness values obtained by ALO, MVO, and SCA are competitive, and they may have the same rank in terms of mean fitness value. MFO also ranks first according to average run time. Furthermore, GOA not only ranks last according to fitness value but also it ranks last according to average run time among all compared algorithms. BRO provides more promising standard deviation than the others.

Table 6 reports the results earned by all algorithms through 25 independent runs for all multimodal functions. BRO clearly performs better the other algorithms for most

Table 5 Results for unimodal benchmark test functions of 30 dimension

F	BRO	PSO	ALO	DA	MFO	MVO	SCA	GOA
F1								
Mean	**3.0353e−09**	8.1310e−08	0.000001	389.045669	0.937433	0.241901	0.139829	1028.24219
Std	**4.1348e−09**	1.5670e−07	0.000001	592.570629	0.547602	0.048150	0.213883	2921.282914
Time	3.536958	1.1516	76.82223	217.8481	0.5428634	0.7795	**0.213883**	1264.051
F2								
Mean	**0.000046**	0.1282	20.528498	7.567446	28.501319	0.298135	0.000979	77.977975
Std	**0.000024**	0.2889	31.566660	5.428482	19.846522	0.070119	0.001209	48.421113
Time	3.577353	1.1460	77.20241	235.3072	**0.5510458**	0,7136389	0. 5723780	1332.949
F3								
Mean	54.865255	103.5336	127.413983	1717.558488	12432.852883	**13.225865**	1524.366568	5095.51855
Std	**16.117329**	77.8071	43.867083	1966.729643	8927.487452	5.383609	1339.012958	5875.806955
Time	3.710890	1.1226	77.78005	226.9669	1.449506	1.628910	**1.436205**	1206.811
F4								
Mean	**0.518757**	3.0470	8.659458	10.208916	30.626058	0.528809	16.267051	1.775508
Std	**0.403657**	1.9104	3.145536	8.917023	10.680434	0.227575	9.442185	0.989782
Time	3.532575	1.1399	77.50493	219.4741	**0.5653759**	0,7420762	0.5887966	1303.754
F5								
Mean	99.936848	**59.0941**	70.598422	27929.483349	1128.704203	329.447404	782.057738	754.861651
Std	82.862358	**44.7770**	71.723742	48870.521184	1327.980927	566.273879	2102.680740	2748.332118
Time	3.948993	1.2316	77.04758	224.6990	**0.6677657**	0. 8423920	0. 7207685	423.1110
F6								
Mean	**2.8731e−08**	9.3320e−08	0.000001	336.899243	1601.281851	0.233792	4.097225	0.000003
Std	**1.8423e−08**	2.2551e−07	0.000001	456.912541	3741.607562	0.053234	0.599685	0.000002
Time	3.551993	1.1685	77.13036	226.5670	**0.5411606**	0.7408868	5.861539	424.5775
F7								
Mean	**0.000368**	0.0215	0.023584	0.182126	2.211140	0.007987	0.021428	4.330057
Std	**0.000094**	0.0097	0.008495	0.297804	3.461601	0.003273	0.018127	9.455933
Time	4.307858	1.5415	77.52549	228.3993	**1.082169**	1.268602	1.102313	1242.990

The best results are shown in bold

multimodal functions. BRO's performance is the best for F10–F12, but PSO and SCA perform better on F8 and F9, respectively. Moreover, it can be observed that in most cases BRO provides a better standard deviation when compared to the other algorithms. The experiments show that MFO provides faster results overall, however. Although BRO does not rank first in runtime, it consistently provides swift results. Furthermore, its rank is never worse than ALO, DA, and GOA. As mentioned above, GOA performs worse than the other algorithms in fitness value, standard deviation, and average run time.

For the composition functions (F14–F19), Table 7 reports the results obtained from the algorithms over 25 independent runs. According to this table, it can be concluded that BRO outperforms all the other algorithms for all composition functions, in terms of mean fitness value. Moreover, not only does BRO rank first according to mean fitness value, but it also ranks first with respect to standard deviation. While BRO provides better standard deviation for F15–F19, MFO had a better standard deviation on F14. Furthermore, SCA is also capable of reaching the solution. As with the multimodal functions, BRO does not rank first in runtime, but it still consistently provides swift results. Also, its rank is never worse than the ALO, DA and GOA algorithms. Finally, it is worth noting that the average run time for GOA is also remarkably high for composition functions.

Table 6 Results for multimodal benchmark test functions of 30 dimension

F	BRO	PSO	ALO	DA	MFO	MVO	SCA	GOA
F8								
Mean	−7035.21079	**−9.3970e + 31**	−5601.86434	−6374.392	−9439.1055	−7872.021223	−4118.07249	−6759.624483
Std	712.332699	4.2289e + 32	362.171676	901.223838	933.416905	651.301060	**295.868016**	899.291586
Time	3.782519	1.1632	77.28404	284.3768	**0. 7050792**	0.7452922	0. 7323907	1259.423
F9								
Mean	48.275350	54.2471	60.851573	82.081617	107.591049	108.044835	**17.584379**	174.769757
Std	14.094585	**13.4287**	22.120831	257.2315	24.913857	32.191667	20.956338	29.134588
Time	3.771521	1.2822	77.06642	46.250730	**0. 6211414**	0. 8954284	0.6358331	1270,283
F10								
Mean	**0.350724**	0.9906	2.053675	5.521769	6.363405	0.712974	10.133719	3.366269
Std	**0.688712**	0.8038	0.796278	244.3684	8.534955	0.693826	9.537759	4.154672
Time	3.784324	1.2443	77.20121	2.956078	**0.6478177**	0. 9157972	0.6841579	1209.365
F11								
Mean	**0.001373**	0.0089	0.010494	3.883619	0.751211	0.479614	0.455533	7.231566
Std	**0.010796**	0.0113	0.016548	3.754029	0.179238	0.159060	0.275454	23.778806
Time	3.844791	1.2318	77.07877	217.4880	0.7355857	1.028005	0.7752492	1213.085
F12								
Mean	**0.369497**	30.4105	6.933983	616.84318	1.734171	0.553024	1.184349	4,323,306.373458
Std	**0.601450**	9.4299	3.807565	3061.497548	1.328050	0.624414	1.270975	1,616,521.886052
Time	5.278253	1.3266	78.24491	218.3771	**1.965486**	2.268537	2.036552	1214.762
F13								
Mean	**0.000004**	41.1520	0.006889	1050.3921	2.047761	0.032335	19.143104	18,404,721.85227
Std	**0.000020**	9.6473	0.009962	5154.472197	1.909983	0.013044	71.923444	81.308,523.29251
Time	3.303456	1.3337	78.24868	218.9030	**1.986649**	2.291993	2.154423	1191.996

The best results are shown in bold

As can be seen from Tables 5, 6 and 7, BRO presents satisfactory results and has a lower summary rank compared to the other algorithms.

Table 8 reports the p values calculated by the non-parametric Wilcoxon rank-sum tests for the pair-wise comparison over two independent samples (BRO vs. PSO, ALO, DA, MFO, MVO, SCA, and GOA). The test was conducted at a 0.05 significance level, with 25 independent runs. 1^{+} indicates that the BRO performed better than the compared algorithms, while 1^{-} means it performs worse. On the other hand, '0' implies that both algorithms performed equally. This table proves that in most cases, BRO outperforms others.

To investigate the convergence rate of the proposed algorithm with its competitors, Fig. 4 provides convergence curves of the best cost function of some unimodal, multimodal, and composition test functions. Also, Fig. 4a–e shows that for F1, F3, F4, F5, and F6, BRO outperforms the other seven competitors. Moreover, for F11 and F13, the performance of BRO is better than others, as shown in Fig. 4g and h. Even though the convergence curve of BRO for F9 plunges rapidly, it still ranks third overall. The convergence rate for composition functions is competitive, but BRO converges faster than others, as shown in Fig. 4i and j. Lastly, as mentioned above, Tables 5, 6 and 7 show that GOA ranks last, and Fig. 4 illustrates that GOA has the worst convergence rate.

Figure 5 shows the status of the algorithm and how the algorithm converges to the global optimum. This figure illustrates the position of each soldier at four different stages of iterations. It clearly illustrates that after the problem space shrinks, the algorithm focuses more on the global optimum.

Table 7 Results for composite benchmark test functions of 2, 4, 2, 2, 2 and 3, respectively

F	BRO	PSO	ALO	DA	MFO	MVO	SCA	GOA
F14								
Mean	**0.998000**	**0.9980**	1.157048	1.435155	0.998004	0.998004	1.077373	1.791015
Std	**0**	**0**	0.371931	0.645898	**0**	4.4661e−12	0.396819	1.144130
Time	3.601619	2.7190	8.726093	93.80841	**0.2828155**	2.960769	0.2898902	824.6873
F15								
Mean	**0.000471**	0.00051058	0.001544	0.001308	0.000748	0.000624	0.000845	0.007727
Std	**0.000265**	0003.3323	0.003925	0.000479	0.000270	0.000315	0.000398	0.008465
Time	2.362441	0.9858	1.140260	122.5952	0.2774668	0.3253819	**0. 2726371**	158.6677
F16								
Mean	− **1.03160**	− **1.0316**	− 1.031628	− 1.031628	− 1.031628	− 1.031628	− 1.031619	− 1.031628
Std	**5.9959e−16**	6.7987e−16	6.8723e−16	0.000002	6.7987e−16	4.7670e−08	0.000007	7.1947e−16
Time	1.950645	0.8377	6.266332	91.32920	0. 2519939	0. 2871051	**0. 2421029**	79.82012
F17								
Mean	**0.397887**	0.3979	**0.397887**	0.397894	**0.397887**	**0.397887**	0.398227	0.410236
Std	**0**	**0**	**0**	0.000030	7.4200e−16	**0**	0.000260	0.064367
Time	1.975396	0.7929	5.986799	91.33857	0.2152625	0. 2524479	**0. 2069589**	117.7970
F18								
Mean	**3.000000**	**3.000000**	**3.000000**	**3.000000**	**3.000000**	3.000001	3.000002	**3.000000**
Std	**2.5895e−16**	1.3445e−15	9.0126e−14	3.0611e−04	**1.8819e−15**	**3.3333e−07**	0.000003	0,000101
Time	2.104709	0.7916	6.170522	91.04340	0.2069731	0.2461370	**0. 1920429**	8,481,287
F19								
Mean	−**3.862782**	−3.8628	−**3.862782**	−3.857556	−**3.862782**	−**3.862782**	−3.856386	− 3.8566
Std	**2.0066e-15**	2.2662e-15	6.2252e-15	0.003522	2.2662e−15	8.0539e−08	0.003206	0.0032
Time	4.461555	1.0814	9.011699	108.5330	0.3025137	0. 3353661	**0. 2996190**	167.7970

The best results are shown in bold

The good balance of BRO between exploration and exploitation makes it possible that BRO can not only locate the global optimum quickly but also cannot easily be trapped by local optima. The relocation of individuals after each damage, their ability to respawn and the shrinking of the feasible search space after some iterations provide a good balance of exploitation and exploration. The simulation and the numerical results for the 19 benchmark optimization problems show that the proposed BRO algorithm can outperform the other optimization algorithms compared in this paper. Also, it can be concluded that the proposed algorithm is well suited for high-dimensional optimization problems.

4 BRO for inverse kinematics of robot arms

Over the last two decades, optimization algorithms have been widely applied to solve optimization problems in many engineering areas [49–54]. More recently, the topic of the inverse kinematics of a robot arms has attracted attention. Various optimization techniques have been widely used to deal with this issue. To evaluate the performance of BRO on real-world engineering problems, the inverse kinematics problem of the 6-DOF PUMA 560 robot arm is considered below. The PUMA 560 robot has 6-DOF with six revolute joints. While the first three joints locate the end-effector, the other three joints provide the

Table 8 Pair-wise statistical comparison between BRO and all compared algorithms by Wilcoxon signed-rank test (α = 0.05)

	PSO		ALO		DA		MFO		MVO		SCA		GOA	
	p value	*h*	*p*-value	*h*	*p*-value	*h*	*p*-value	*h*	*p*-value	*h*	*p*-value	*h*	*p*-value	*h*
F1	1.23E−05	1^+	1.23E−05	1^+	1.39E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+
F2	1.23E−05	1^+	1.23E−05	1^+	1.39E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+
F3	1.23E−05	1^+	1.23E−05	1^+	1.77E−05	1^+	1.23E−05	1^+	0.000403	1^+	1.23E−05	1^+	1.23E−05	1^+
F4	1.23E−05	1^+	1.23E−05	1^+	1.39E−05	1^+	1.23E−05	1^+	0.000364	1^+	1.23E−05	1^+	1.23E−05	1^+
F5	0.016033	1^+	0.989266	1–	1.23E−05	1^+	1.23E−05	1^+	0.326049	1–	7.22E−05	1^+	0.001079	1^+
F6	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+
F7	1.23E−05	1^+	1.23E−05	1^+	2.26E−05	1^+	1.23E−05	1^+	0.087527	1–	2E−05	1^+	0.000446	1^+
F8	1.23E−05	1^+	0.002259	1^+	0.004927	1^+	1.23E−05	1^+	0.000157	1^+	1.23E−05	1^+	0.003507	1^+
F9	1.23E−05	1^+	0.002258	1^+	0.000602	1^+	1.23E−05	1^+	1.77E−05	1^+	0.001569	1^+	0.001186	1^+
F10	1.39E−05	1^+	0.000194	1^+	1.39E−05	1^+	3.22E−05	1^+	0.001885	1^+	0.000101	1^+	1.77E−05	1^+
F11	2E−05	1^+	0.777543	1^-	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+
F12	1.23E−05	1^+	1.23E−05	1^+	4.07E−05	1^+	0.000126	1^+	0.057836	1^-	0.000665	1^+	1.39E−05	1^+
F13	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+	1.23E−05	1^+
F14	1	0	0.03125	1^+	0.000122	1^+	1	0	1	0	1.81E−05	1^+	1	0
F15	0.13535	1^-	0.000194	1^+	1.39E−05	1^+	4.57E−05	1^+	0.000602	1^+	0.000194	1^+	2.54E−05	1^+
F16	5.73E−07	1^+	1	0	1	0	1	0	1	0	8.79E−05	1^+	1	0
F17	1	0	1	0	1	0	1	0	0.000488	1^+	1.23E−05	1^+	1	0
F18	1	0	1	0	0.25	1^-	1	0	0.000122	1^+	5.93E−05	1^+	1	0
F19	1	0	1	0	2.7E−05	1^+	1	0	1	1^-	1.23E−05	1^+	0.015625	1^+

orientation of the end-effector. The PUMA 560 manipulator is shown in Fig. 6.

Table 8 provides the Denavit–Hartenberg (D-H) parameters of PUMA 560. Below, Eq. 4 illustrates how the neighboring frames' transformations multiple, and it is used to examine the objective function of BRO:

$$^0T_6 = {}^0T_1{}^1T_2{}^2T_3{}^3T_4{}^4T_5{}^5T_6 \tag{4}$$

where the transfer matrix of link i is $^iT_{i+1}$. The matrix produced by $^0T_6$ gives a Cartesian coordinate for joint angles. The last two column of Table 9 indicate the constraints of the function. Furthermore, the cost function can be calculated with $^0T_6$ because, in this approach, the cost function is expressed as distance (Euclidian) between obtained and target points in space (Cartesian). Equation 5 shows the matrix which results from running the $^0T_6$ equation:

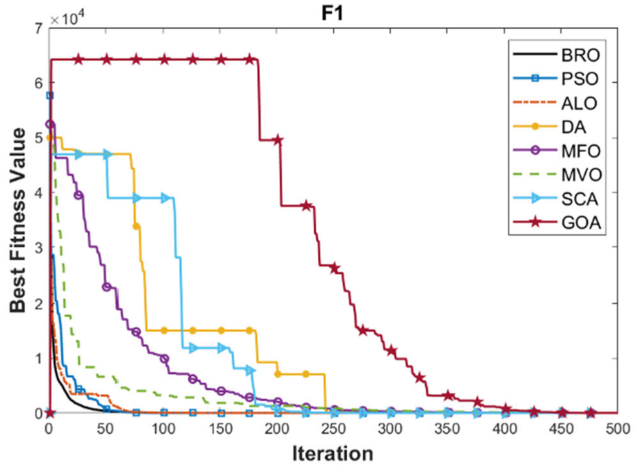
$$^0T_6 = {}^0T_1(\theta_1){}^1T_2(\theta_2){}^2T_3(\theta_3){}^3T_4(\theta_4){}^4T_5(\theta_5){}^5T_6(\theta_6) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where

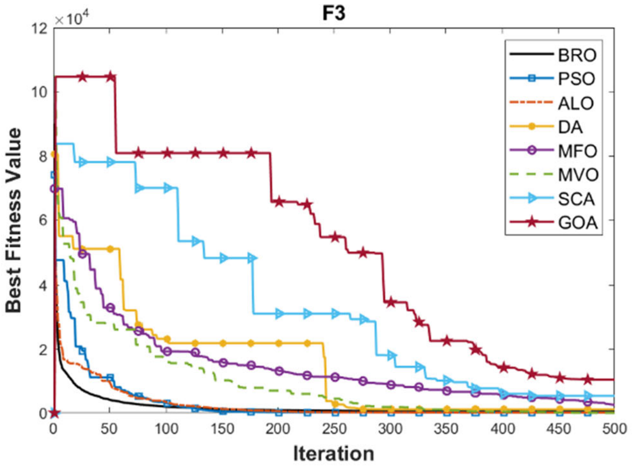
$$\begin{aligned}
r_{11} &= c_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] + s_1(s_4c_5c_6 + c_4s_6),\\
r_{21} &= s_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] - c_1(s_4c_5c_6 + c_4s_6),\\
r_{31} &= -s_{23}(c_4c_5c_6 - s_4s_6) - c_{23}s_5c_6,\\
r_{12} &= c_1[c_{23}(-c_4c_5s_6 - s_4c_6) + s_{23}s_5s_6] + s_1(c_4c_6 - s_4c_5s_6),\\
r_{22} &= s_1[c_{23}(-c_4c_5s_6 - s_4c_6) + s_{23}s_5s_6] - c_1(c_4c_6 - s_4c_5s_6),\\
r_{32} &= -s_{23}(-c_4c_5s_6 - s_4c_6) + c_{23}s_5s_6,\\
r_{13} &= -c_1(c_{23}c_4s_5 + s_{23}c_5) - s_1s_4s_5,\\
r_{23} &= -s_1(c_{23}c_4s_5 + s_{23}c_5) + c_1s_4s_5,\\
r_{33} &= s_{23}c_4s_5 - c_{23}c_5,\\
p_x &= c_1[a_2c_2 + a_3c_{23} - d_4s_{23}] - d_3s_1,\\
p_y &= S_1[a_2c_2 + a_3c_{23} - d_4s_{23}] + d_3c_1,\\
p_z &= -a_3s_{23} - a_2s_2 - d_4c_{23}
\end{aligned} \tag{6}$$

Here c_i and s_i indicate the $\cos(\theta_i)$ and $\sin(\theta_i)$, respectively. With both the given Cartesian coordinate (x_s, y_s, z_s) and the target coordinate (x, y, z), it is possible to find the optimum value for θ, which represents the angle value for each joint. With this value, the end-effector of each robot arm can be transferred to a better location. In this section, the proposed BRO algorithm is used to solve this optimization problem.

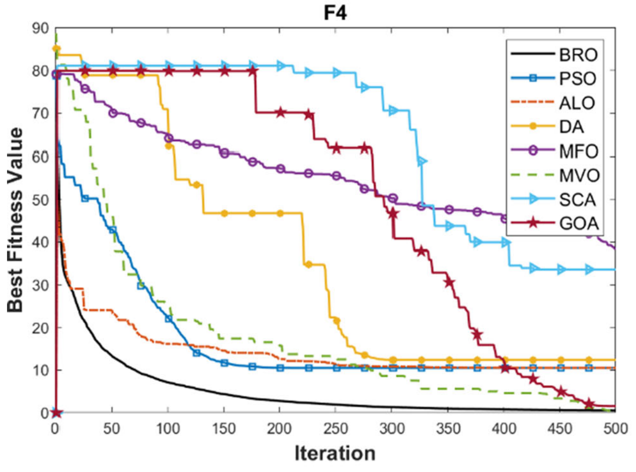
Equation 5, which calculates the forward kinematics of PUMA 560, is used to change the existing joint angles (θ) to the Cartesian coordinate (x', y', z'). The cost function in



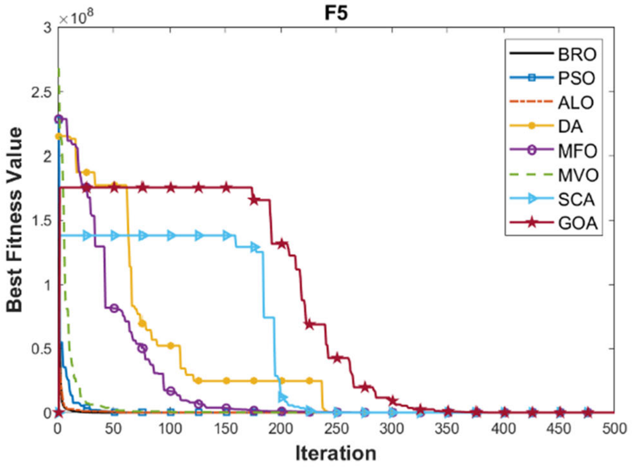
(a)



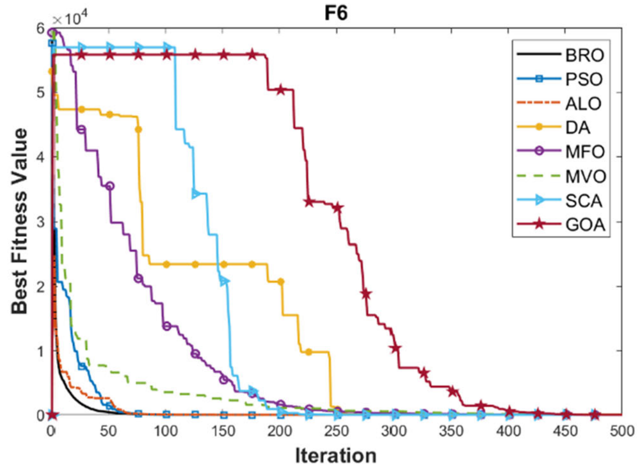
(b)



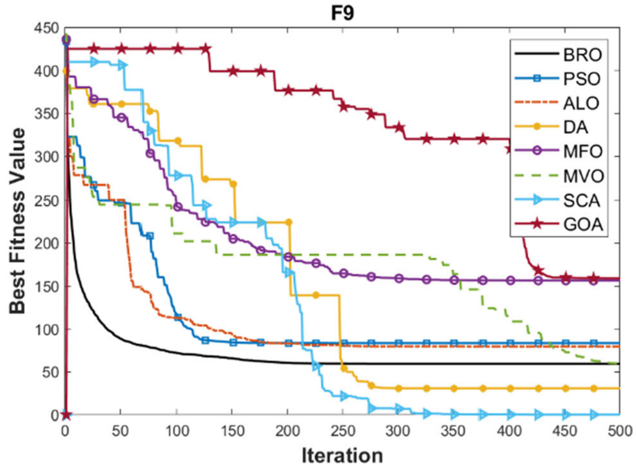
(c)



(d)



(e)



(f)

Fig. 4 Average results over 30 runs obtained by BRO and the other algorithms for some test function

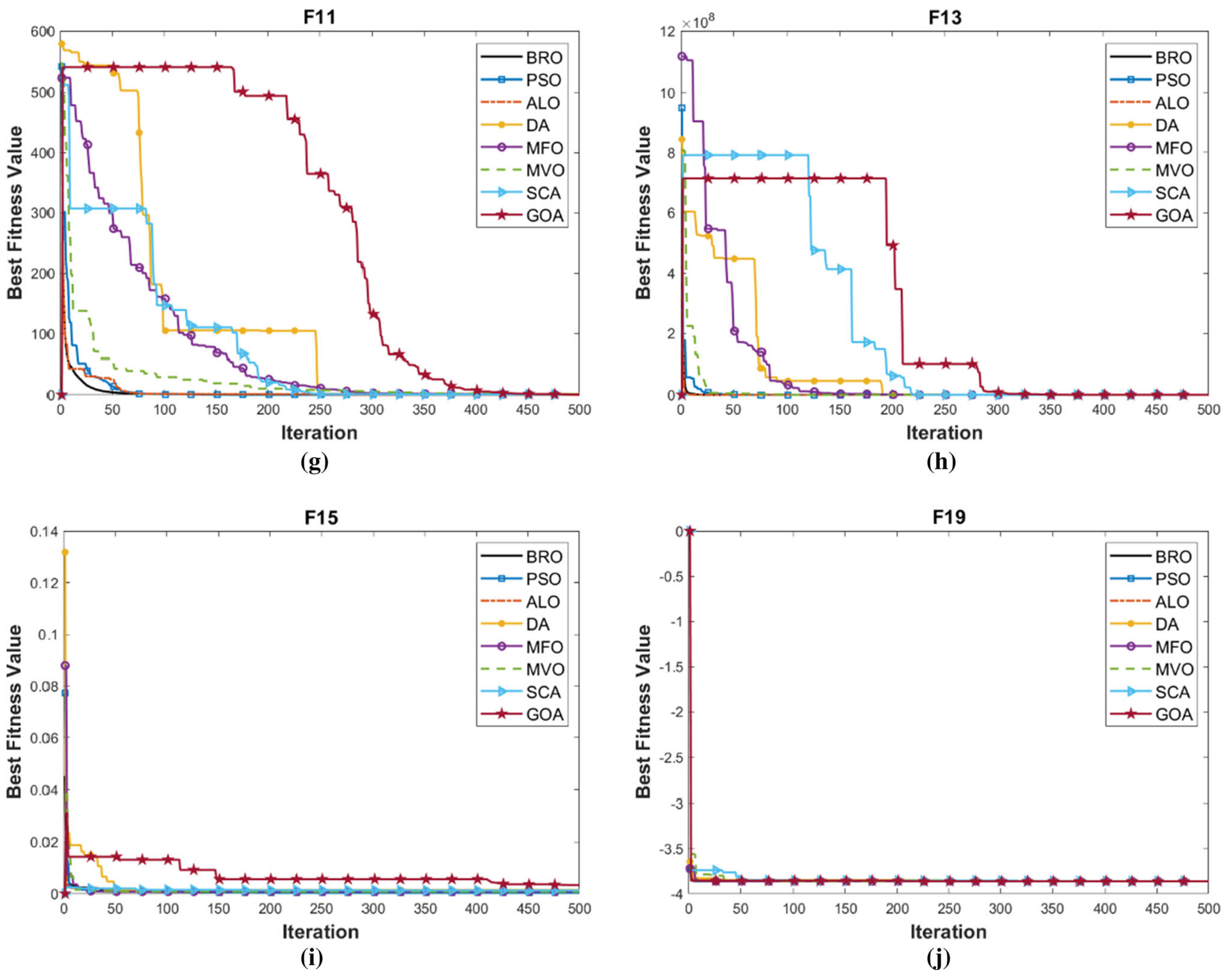


Fig. 4 continued

Eq. 5 comes from the Euclidean distance between the two locations—the starting coordinate and the desired coordinates. The Euclidean distance between the desired and current location is mathematically calculated by $d = \sqrt{(x' - x)^2 + (y' - y)^2 + (z' - z)^2}$

Table 10 reports the results of BRO over 50 independent runs for the inverse kinematics of PUMA 560 robot arm. To obtain trade-off parameters, experiments were conducted with different population sizes and iteration counts. According to Table 10, the results clearly validate the effectiveness of BRO.

5 Conclusion

This paper proposes a new optimization algorithm, named BRO. The proposed method is inspired by a digital game genre known as battle royale games. The well-known PSO algorithm and six recent proposed optimization algorithms—ALO, DA, MFO, MVO, SCA and GOA—were also tested for comparison. BRO and the other algorithms were examined on 19 benchmark test functions. As it mentioned before, the success of the metaheuristic approaches depends on the balance of exploration and exploitation. The death of the players causes them to escape from the local optima and respawn of the players in a random area leads to perform the exploration. On the other hand, moving the players toward the best player while shrinking the problem space has led to exploitation. According to above numerical results and analysis, it can be concluded that BRO outperforms all the compared algorithms in terms of accuracy convergence rate. Hence, it can be inferred that the BRO has succeeded in balancing exploration and exploitation. Furthermore, in order to verify the performance of the proposed algorithm, BRO was employed on a real-world engineering problem. The results show that not only is BRO effective for solving numerical optimization

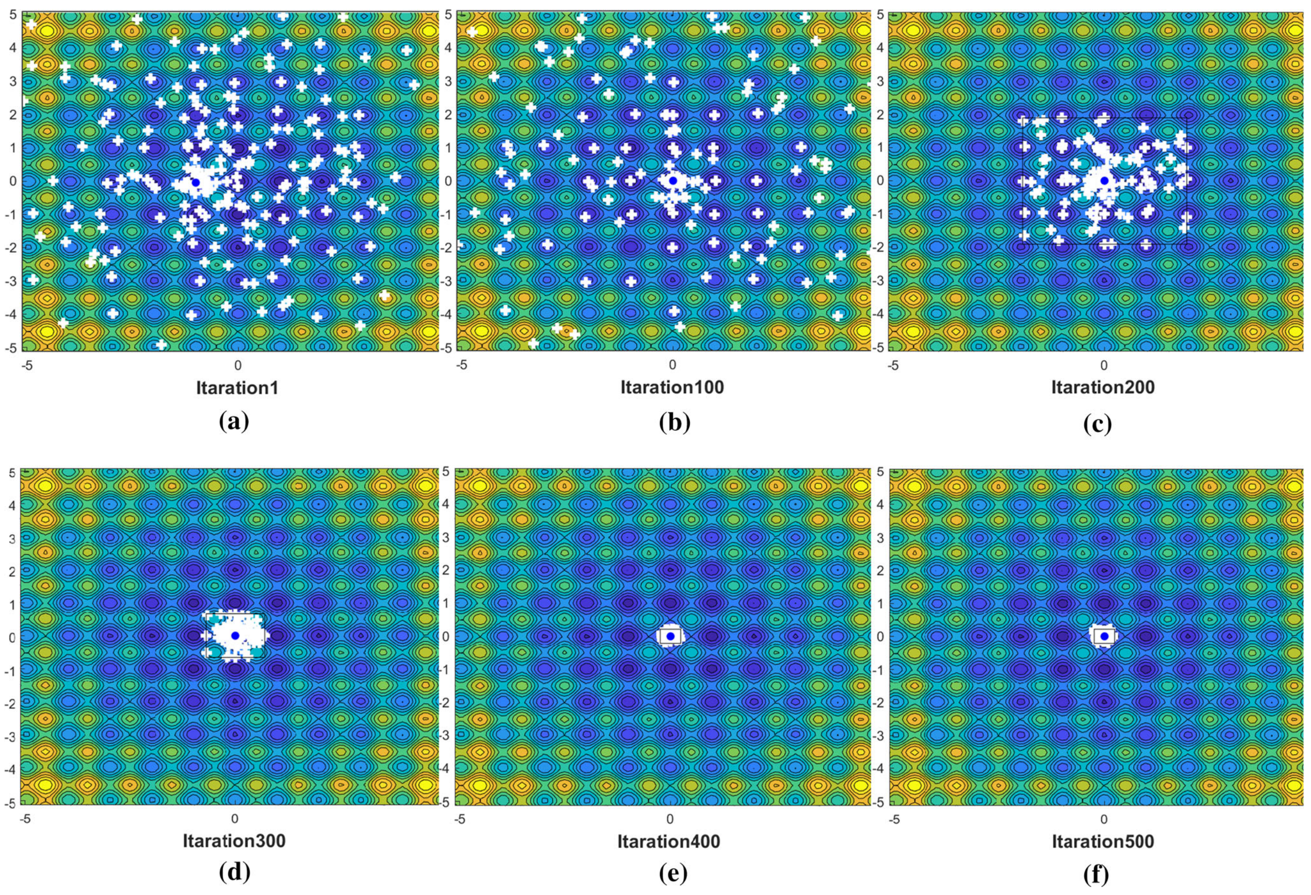


Fig. 5 The positions and status of the population in different iterations

Fig. 6 PUMA 560 Robot Arm

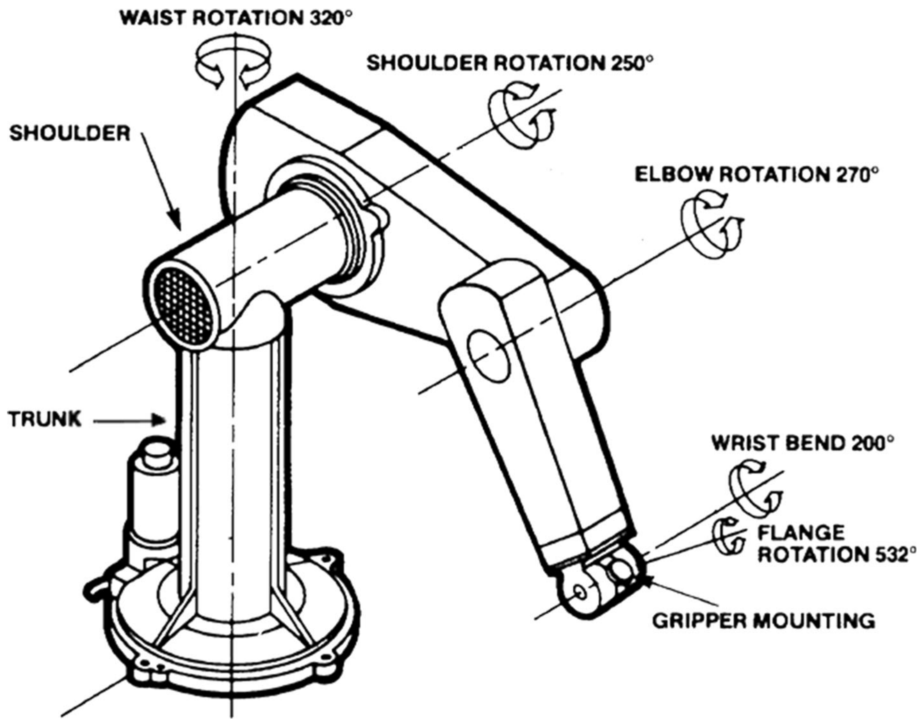


Table 9 D-H Kinematic Parameters of PUMA 560

NO	a_i(meters)	d_i(meters)	Range (°)	
1	0	0	− 160	+160
2	0	0	− 245	+45
2	0.4318	0.1491	− 45	225
4	0.0203	0.1331	− 110	170
5	0	0	− 100	100
6	0	0	− 266	266

Table 10 obtained results by means of BRO for Inverse kinematics of PUMA 560 robot arm

Population size	Maximum iteration count	Mean error value	SD of mean error value
25	100	0.00037	0.00035
25	300	0.00018	0.00021
50	100	0.00019	0.00021
50	300	7.2452e−05	0.00015
100	100	1.9146e−05	3.7751e−05
100	300	1.0689e–04	2.1312e−05
200	100	6.9530e−05	2.3740e−05
200	300	6.2040e−06	1.5333e−05
300	100	1.8821e−06	3.8133e−06
300	300	7.1473e−07	2.7865e−06
300	500	1.8914e−07	6.4582e−07

test functions, but it also provides very promising results for real-world applications and problems. The performance of BRO is promising. However, the successes of BRO in converging to the global optimum solution and in receiving high optimization results in tests does not mean that BRO is the most efficient or best optimization algorithm. Rather, it is a successful adaption of a game strategy that can be applied for some optimization problems.

In future work, it is planned to use the squad mode of the game and extend this proposed approach to find multiple solutions (local/optima) as a multimodal optimization algorithm.

Compliance with ethical standards

Conflict of interest The author declares that he has no conflict of interest.

References

1. Lazar A (2002) Heuristic knowledge discovery for archaeological data using genetic algorithms and rough sets. In: Sarker R, Abbass H, Newton C (eds) Heuristic and optimization for knowledge discovery. IGI Global, Hershey, pp 263–278
2. Passino KM (2002) Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst Mag 22(3):52–67. https://doi.org/10.1109/MCS.2002.1004010
3. Yang X-S (2009) Firefly algorithms for multimodal optimization. In: Watanabe O, Zeugmann T (eds) Stochastic algorithms: foundations and applications. Springer, Berlin, pp 169–178
4. 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
5. Li MD, Zhao H, Weng XW, Han T (2016) A novel nature-inspired algorithm for optimization: virus colony search. Adv Eng Softw 92:65–88. https://doi.org/10.1016/j.advengsoft.2015.11.004
6. Sharafi Y, Khanesar MA, Teshnehlab M (2016) COOA: competitive optimization algorithm. Swarm Evolut Comput 30:39–63. https://doi.org/10.1016/j.swevo.2016.04.002
7. Savsani P, Savsani V (2016) Passing vehicle search (PVS): a novel metaheuristic algorithm. Appl Math Model 40(5):3951–3978. https://doi.org/10.1016/j.apm.2015.10.040
8. Jaddi NS, Alvankarian J, Abdullah S (2017) Kidney-inspired algorithm for optimization problems. Commun Nonlinear Sci Numer Simul 42:358–369. https://doi.org/10.1016/j.cnsns.2016.06.006
9. Seyyedabbasi A, Kiani F (2019) I-GWO and Ex-GWO: improved algorithms of the Grey Wolf Optimizer to solve global optimization problems. Eng Comput. https://doi.org/10.1007/s00366-019-00837-7
10. Holland J (1975) Adaptation in natural and artificial systems: an introductory analysis with application to biology. Control and artificial intelligence. MIT press, Cambridge
11. Schwefel H-P (1984) Evolution strategies: a family of non-linear optimization techniques based on imitating some principles of organic evolution. Ann Oper Res 1(2):165–167
12. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549. https://doi.org/10.1016/0305-0548(86)90048-1
13. Van Laarhoven PJ, Aarts EH (1987) Simulated annealing. In: Aart EH, van Laarhoven PJ (eds) Simulated annealing: theory and applications. Springer, Berlin, pp 7–15
14. Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12(6):702–713. https://doi.org/10.1109/TEVC.2008.919004
15. Ghaemi M, Feizi-Derakhshi M-R (2014) Forest optimization algorithm. Expert Syst Appl 41(15):6676–6687. https://doi.org/10.1016/j.eswa.2014.05.009
16. Askarzadeh A (2014) Bird mating optimizer: an optimization algorithm inspired by bird mating strategies. Commun Nonlinear Sci Numer Simul 19(4):1213–1228. https://doi.org/10.1016/j.cnsns.2013.08.027
17. Tang D, Dong S, Jiang Y, Li H, Huang Y (2015) ITGO: invasive tumor growth optimization algorithm. Appl Soft Comput 36:670–698. https://doi.org/10.1016/j.asoc.2015.07.045
18. Eberhart R, Kennedy JA (1995) New optimizer using particle swarm theory. In: MHS'95. proceedings of the sixth international symposium on micro machine and human science, 4-6 Oct. 1995 1995. pp 39–43. https://doi.org/10.1109/mhs.1995.494215

19. Dorigo M, Caro GD (1999) Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 Congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), 6-9 July 1999 1999. pp 1470–1477 Vol. 1472. https://doi.org/10.1109/cec.1999.782657
20. Chu S-C, Tsai P-w, Pan J-S (2006) Cat swarm optimization. In: Yang Q, Webb G (eds) PRICAI 2006: trends in artificial intelligence. Springer, Berlin, pp 854–858
21. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459–471. https://doi.org/10.1007/s10898-007-9149-x
22. Li X, Zhang J, Yin M (2014) Animal migration optimization: an optimization algorithm inspired by animal migration behavior. Neural Comput Appl 24(7):1867–1877. https://doi.org/10.1007/s00521-013-1433-8
23. Mirjalili S (2016) Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput Appl 27(4):1053–1073. https://doi.org/10.1007/s00521-015-1920-1
24. Fausto F, Cuevas E, Valdivia A, González A (2017) A global optimization algorithm inspired in the behavior of selfish herds. Biosystems 160:39–55. https://doi.org/10.1016/j.biosystems.2017.07.010
25. Formato RA (2007) Central force optimization. Prog Electromagn Res 77:425–491
26. Rashedi E, Nezamabadi-pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179(13):2232–2248. https://doi.org/10.1016/j.ins.2009.03.004
27. Husseinzadeh Kashan A (2015) A new metaheuristic for optimization: optics inspired optimization (OIO). Comput Oper Res 55:99–125. https://doi.org/10.1016/j.cor.2014.10.011
28. 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
29. Hatamlou A (2013) Black hole: a new heuristic optimization approach for data clustering. Inf Sci 222:175–184. https://doi.org/10.1016/j.ins.2012.08.023
30. Kaveh A, Talatahari S (2010) A novel heuristic optimization method: charged system search. Acta Mech 213(3):267–289. https://doi.org/10.1007/s00707-009-0270-4
31. Punnathanam V, Kotecha P (2016) Yin-Yang-pair optimization: a novel lightweight optimization algorithm. Eng Appl Artif Intell 54:62–79. https://doi.org/10.1016/j.engappai.2016.04.004
32. 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
33. Abedinpourshotorban H, Mariyam Shamsuddin S, Beheshti Z, Jawawi DNA (2016) Electromagnetic field optimization: a physics-inspired metaheuristic optimization algorithm. Swarm Evolut Comput 26:8–22. https://doi.org/10.1016/j.swevo.2015.07.002
34. Chen S, Montgomery J (2013) Particle swarm optimization with thresheld convergence. In: 2013 IEEE congress on evolutionary computation, 20-23 June 2013 2013. pp 510–516. https://doi.org/10.1109/cec.2013.6557611
35. Zhang Q, Wang R, Yang J, Ding K, Li Y, Hu J (2017) Collective decision optimization algorithm: a new heuristic optimization method. Neurocomputing 221:123–137. https://doi.org/10.1016/j.neucom.2016.09.068
36. Črepinšek M, Liu S-H, Mernik M (2013) Exploration and exploitation in evolutionary algorithms: a survey. ACM comput Surveys (CSUR) 45(3):1–33
37. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82. https://doi.org/10.1109/4235.585893
38. Contributors W (14 October 2018) Battle royale game. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Battle_royale_game&oldid=864010252
39. Contributors W (2020) PlayerUnknown's Battlegrounds—Wikipedia, The Free Encyclopedia
40. Contributors W (2020) Call of duty: Warzone—Wikipedia, The Free Encyclopedia
41. contributors W (2020) Apex Legends—Wikipedia, The Free Encyclopedia
42. Contributors W (2020) Counter-Strike: Global Offensive—Wikipedia, The Free Encyclopedia
43. Contributors W (2020) Ring of Elysium—Wikipedia, The Free Encyclopedia
44. Mirjalili S (2015) The ant lion optimizer. Adv Eng Softw 83:80–98. https://doi.org/10.1016/j.advengsoft.2015.01.010
45. Mirjalili S (2015) 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
46. Mirjalili S, Mirjalili SM, Hatamlou A (2016) Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Comput Appl 27(2):495–513. https://doi.org/10.1007/s00521-015-1870-7
47. 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
48. Saremi S, Mirjalili S, Lewis A (2017) Grasshopper optimisation algorithm: theory and application. Adv Eng Softw 105:30-47
49. Krohling RA, Jaschek H, Rey JP (1997) Designing PI/PID controllers for a motion control system based on genetic algorithms. In: Proceedings of 12th IEEE international symposium on intelligent control, 16-18 July 1997 1997. pp 125–130. https://doi.org/10.1109/isic.1997.626429
50. Richter CW, Sheble GB (1998) Genetic algorithm evolution of utility bidding strategies for the competitive marketplace. IEEE Trans Power Syst 13(1):256–261. https://doi.org/10.1109/59.651644
51. Elmi A, Solimanpur M, Topaloglu S, Elmi A (2011) A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. Comput Ind Eng 61(1):171–178. https://doi.org/10.1016/j.cie.2011.03.007
52. Foroughi A, Gökçen HA (2019) Multiple rule-based genetic algorithm for cost-oriented stochastic assembly line balancing problem. Assembly Automation. https://doi.org/10.1108/aa-03-2018-050
53. Çavdar T, Mohammad M, Milani RA (2013) A new heuristic approach for inverse kinematics of robot arms. Adv Sci Lett 19(1):329–333. https://doi.org/10.1166/asl.2013.4700
54. Milani MMRA, Çavdar T, Aghjehkand VF (2012) Particle swarm optimization—based determination of ziegler–Nichols parameters for PID controller of brushless DC motors. In: 2012 International symposium on innovations in intelligent systems and applications, 2-4 July 2012 2012. pp 1–5. https://doi.org/10.1109/inista.2012.6246984