Neural Comput & Applic (2014) 25:1569–1584
DOI 10.1007/s00521-014-1640-y

ORIGINAL ARTICLE

Adaptive gbest-guided gravitational search algorithm

Seyedali Mirjalili · Andrew Lewis

Received: 3 August 2013 / Accepted: 24 March 2014 / Published online: 4 June 2014

Abstract One heuristic evolutionary algorithm recently proposed is the gravitational search algorithm (GSA), inspired by the gravitational forces between masses in nature. This algorithm has demonstrated superior performance among other well-known heuristic algorithms such as particle swarm optimisation and genetic algorithm. However, slow exploitation is a major weakness that might result in degraded performance when dealing with real engineering problems. Due to the cumulative effect of the fitness function on mass in GSA, masses get heavier and heavier over the course of iteration. This causes masses to remain in close proximity and neutralise the gravitational forces of each other in later iterations, preventing them from rapidly exploiting the optimum. In this study, the best mass is archived and utilised to accelerate the exploitation phase, ameliorating this weakness. The proposed method is tested on 25 unconstrained benchmark functions with six different scales provided by CEC 2005. In addition, two classical, constrained, engineering design problems, namely welded beam and tension spring, are also employed to investigate the efficiency of the proposed method in real constrained problems. The results of benchmark and classical engineering problems demonstrate the performance of the proposed method.

Keywords Optimisation · Heuristics · Evolutionary algorithms · Exploration and exploitation · Constrained optimisation

S. Mirjalili (✉) · A. Lewis
School of Information and Communication Technology, Griffith University, Brisbane, QLD 4111, Australia
e-mail: seyedali.mirjalili@griffithuni.edu.au

1 Introduction

In recent years, various heuristic evolutionary optimisation algorithms have been developed mimicking natural phenomena. Some of the most popular are particle swarm optimisation (PSO) [1, 2], genetic algorithm (GA) [3], differential evolution (DE) [4, 5], ant colony optimisation (ACO) [6], and other, recent additions such as bio-geography optimisation algorithm (BBO) [7, 8], grey wolf optimiser (GWO) [9], and Krill Herd (KH) algorithm [10, 11]. The gravitational search algorithm (GSA) has been recently proposed by Rashedi et al. [12]. It has been shown that this algorithm is able to provide promising results compared with other well-known algorithms in this field. The similarity between these evolutionary algorithms is that they start the search process with an initial population as a set of candidate solutions. This population is evolved through a predefined number of steps with certain rules. For instance, PSO uses the social behaviour of birds flocking and GA utilises Darwin's theory of evolution. Regardless of structure, these algorithms basically divide the search process into two main phases—exploration and exploitation—to find the global optimum.

Exploration requires an algorithm to search the problem space broadly, whereas exploitation needs the algorithm to converge to the best solution from promising solutions found in the exploration phase. The ultimate goal was to find an efficient trade-off between exploitation and exploration [13]. However, it is challenging to find a good balance due to the stochastic behaviour of evolutionary algorithms [14]. Moreover, exploration and exploitation conflict: emphasising one generally weakens the other one.

In the literature, merging optimisation algorithms is a popular way to obtain a better balance between exploration and exploration, as in the hybrids PSO–GA [15, 16], PSO–

DE [17, 18], PSO–ACO [19, 20], KH–DE [21], and KH–BBO [22]. In improving the exploitation phase, local search [23, 24] and gradient descent [25, 26] have been utilised. Both of these methods are able to improve performance, but they bring additional computational cost. Moreover, gradient descent methods are also ill-defined for non-differentiable search spaces. Normally, in combining two algorithms, the computational time is greater than for either alone because both algorithms should be run either in parallel or sequentially. Local search is also computationally expensive itself. So these inevitable problems should be considered, especially for real-world engineering problems. There are also other methods for improving the performance of meta-heuristics: utilising chaotic maps [27–29] and employing different evolutionary operators [30, 31]. In this study, we try to enhance the exploitation of GSA with an extremely low-cost method since it appears to be the main drawback of this algorithm [32].

The search process in GSA slows as the number of iterations increases. Due to the cumulative effect of the fitness function on mass, masses get greater over the course of iteration. This prevents them from rapidly exploiting the best solutions in later iterations [33]. This problem has been addressed directly or indirectly in previous work. In 2010, Sinaie solved the travelling salesman problem (TSP) with GSA and neural networks (NN) sequentially, with the exploration and exploitation phases accomplished by GSA and NN, respectively [33]. Shaw et al. [34] used an opposition-based learning to improve the convergence speed of GSA. In 2011, GSA was employed as a global search algorithm, accompanied by a combined multi-type local improvement scheme as a local search operator [35]. A novel immune gravitation optimisation algorithm (IGOA) was proposed by Zhang et al. [32] in 2012, incorporating the characteristics of antibody diversity and vaccination to solve the slow convergence. Hatamlou et al. [36] developed a sequential method to solve clustering problems. They used GSA for finding a near-optimal solution and another heuristic method for improving the solutions obtained by GSA. In 2011, Li and Zhou [37] integrated social thinking and individual thinking of PSO to GSA and employed to solve parameter identification of hydraulic turbine governing system. However, there is no deep investigation of the proposed method in the paper especially on challenging benchmark problems.

All these studies show that GSA suffers from slow exploitation. However, none of them investigate this problem in detail. In addition, some of these studies adopted problem-specific solutions for this problem. The first author also has two publications [38, 39] on improving the performance of GSA, in which the social thinking of PSO was integrated with GSA in order to improve the exploitation. However, we observed that the constants proposed for balancing the effect of GSA and PSO in search need to be tuned adaptively due to the loss of exploration. In this work, the main reason for the slow search process is further discussed in detail and a general, low-cost solution is proposed, which utilises adaptive coefficients to balance between exploration and exploitation. In addition, we investigate the performance of both improved GSA and GSA in solving constrained problems.

The rest of the paper is organised as follows: Section 2 presents a brief introduction to the GSA. Section 3 discusses the problem of slow exploitation and proposes a method to overcome it. The experimental results for benchmark and classical engineering design problems are provided in Sects. 4 and 5, respectively. Finally, Section 6 concludes the work and suggests some directions for future research.

2 Gravitational search algorithm

The basic physical theory from which GSA is inspired is Newton's law of universal gravitation. The GSA performs search by employing a collection of agents (candidate solutions) that have masses proportional to the value of a fitness function. During iteration, the masses attract each other by the gravity forces between them. The heavier the mass, the bigger the attractive force. Therefore, the heaviest mass, which is possibly close to the global optimum, attracts the other masses in proportion to their distances.

This algorithm is formulated as follows [12]:

Every mass has a position in search space as follows:

$$X_i = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^n\right), \; i = 1, 2, \ldots, N \tag{2.1}$$

where N is the number of masses, n is the dimension of the problem, and x_i^d is the position of the ith agent in the dth dimension.

The algorithm starts by randomly placing all agents in a search space. During all epochs, the gravitational force from agent j on agent i at a specific time t is defined as follows:

$$F_{\mathrm{ij}}^d(t) = G(t)\frac{M_{\mathrm{pi}}(t) \times M_{\mathrm{aj}}(t)}{R_{\mathrm{ij}}(t) + \varepsilon}\left(x_j^d(t) - x_i^d(t)\right), \tag{2.2}$$

where M_{aj} is the active gravitational mass related to agent j, M_{pi} is the passive gravitational mass related to agent i, $G(t)$ is a gravitational constant at time t, ε is a small constant, and $R_{\mathrm{ij}}(t)$ is the Euclidian distance between two agents i and j. It is recognised that Eq. 2.2 is not a correct expression of Newtonian gravitational forces, as has been explored in the critique of Gauci et al. [40]. However, the original formulation is retained here to allow direct comparison with the original GSA.

In order to emphasise exploration in the first iterations and exploitation in the final iterations, G has been designed with an adaptive value so that it is increased over iterations. In other words, G encourages the search agents to move with big steps in the initial iterations, but they are constrained to move slowly in the final iterations.

The gravitational factor (G) and the Euclidian distance between two agents i and j are calculated as follows:

$$G(t) = G_0 \times \exp\left(-\alpha \times \frac{\text{iter}}{\text{maxiter}}\right), \tag{2.3}$$

$$R_{\text{ij}}(t) = X_i(t), X_j(t)_2, \tag{2.4}$$

where α is the coefficient of decrease, G_0 is the initial gravitational constant, iter is the current iteration, and maxiter is the maximum number of iterations.

In a problem space with dimension equal to d, the total force that acts on agent i is calculated by the following equation:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} \text{rand}_j F_{\text{ij}}^d(t), \tag{2.5}$$

where rand_j is a random number in the interval [0,1]. The random component has been included in this formula to have a random movement step along the gravitational force of each agent and the final resultant force. This helps to have more diverse behaviours in moving the search agents.

Newton's law of motion has also been utilised in this algorithm, which states that the acceleration of a mass is proportional to the applied force and inverse to its mass, so the accelerations of all agents are calculated as follows:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{\text{ii}}(t)}, \tag{2.6}$$

where d is the dimension of the problem, t is a specific time, and M_{ii} is the inertial mass of agent i.

The velocity and position of agents are calculated as follows:

$$v_i^d(t+1) = \text{rand}_i \times v_i^d(t) + a_i^d(t), \tag{2.7}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{2.8}$$

where d is the problem's dimension and is a random number in the interval [0,1].

As can be inferred from (2.7) and (2.8), the current velocity of an agent is defined as a fraction of its last velocity ($0 \leq \text{rand}_i \leq 1$) added to its acceleration. Only a (random) fraction of the initial velocity has been used in order to prevent the search agents from over-shooting the boundaries of the search space. In addition, it is a random fraction in order to promote diversity in the behaviour of GSA.

Furthermore, the current position of an agent is set to its last position added to its current velocity. To correctly apply Newton's laws of motion, the velocity used should be the average of the velocities at time t and $t+1$. However, the original version of GSA used only the final velocity, and this same simplified method is used in this study to allow direct comparison.

Since agents' masses are defined by their fitness evaluation, the agent with heaviest mass is the fittest agent. According to the above equations, the heaviest agent has the highest attractive force and the slowest movement. Since there is a direct relation between mass and the fitness function, a normalisation method has been adopted to scale masses as follows:

$$m_i(t) = \frac{\text{fit}_i(t) - \text{worst}(t)}{\text{best}(t) - \text{worst}(t)}, \tag{2.9}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{2.10}$$

where $\text{fit}_i(t)$ is the fitness value of the agent i at time t, best (t) is the fittest agent at time t, and worst (t) is the weakest agent at time t.

In the GSA, at first, all agents are initialised with random values. During iteration, the velocities and positions are defined by (2.7) and (2.8). Meanwhile, other parameters such as the gravitational constant and masses are calculated by (2.3) and (2.10). Finally, the GSA is terminated by satisfying an end criterion.

3 Proposed method

In this section, the problem of slow exploitation is first clarified in detail. Following this, a method is proposed for overcoming the problem.

3.1 Slow exploitation

As mentioned earlier, there are two main, common characteristics between all population-based algorithms with evolutionary behaviour: an algorithm's exploration of search spaces and its exploitation of the most promising solution. An algorithm should support these two vital characteristics to guarantee a favourable optimisation process.

In GSA, the gravitational constant (G) defines the speed at which solutions change their location in solution space. According to Eq. 2.2, a high value of G results in high intensities of gravitational forces and resulting rapid movement in earlier iterations. However, G is progressively decreased according to Eq. 2.3, and this, combined with the slow movement of increasingly heavy agents, helps GSA during exploitation [12, 41]. So the exploitation phase coincides with less intensity of attractive force and

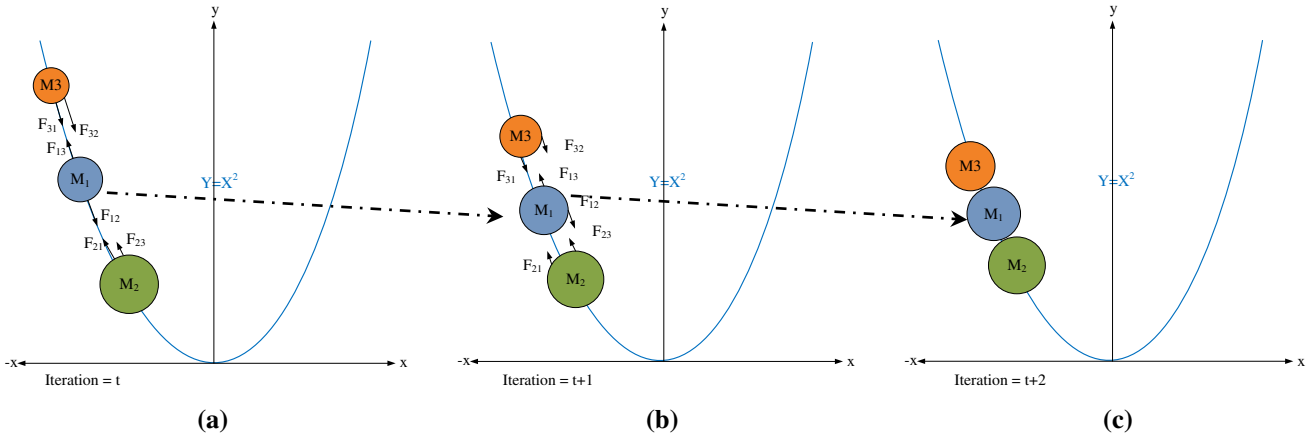

Fig. 1 Movement behaviour of masses in GSA

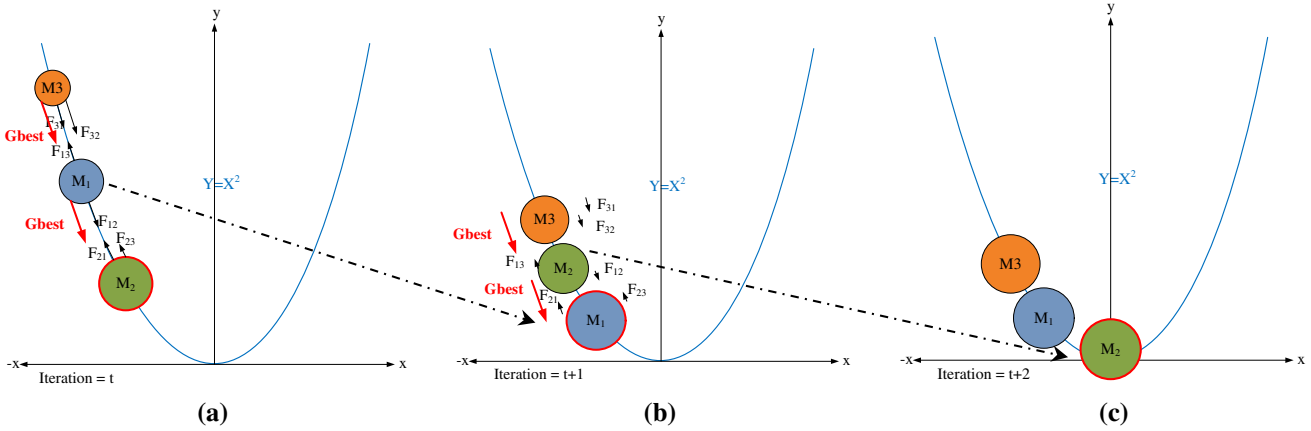

Fig. 2 Movement behaviour of masses in the proposed method

slow movement. Unfortunately, heavy masses with slow movement and less intensity of attractive force significantly degrade the speed of convergence as well. Therefore, it seems that GSA suffers from slow search speed in the exploitation phase originating from these factors.

Figure 1 shows a simple, one-dimensional problem where the fitness function is $y = x^2$. As may be seen in this figure, the masses $M1$ and $M3$ are attracted by $M2$ at iterations $t + 1$ and $t + 2$. However, these masses also attract $M2$ and move it slightly away from the optimum. So as particles approach an optimum, they are not necessarily able to accelerate towards it, instead moving towards the centre of mass of all the particles in the neighbourhood. It is worth mentioning here that GSA has no memory for saving the best solution obtained so far so the best solution might be lost as the best mass is attracted away by other less fit masses. All these problems motivate us to develop the solution discussed in the following section.

3.2 Improving the exploitation

The basic idea of the proposed method is to save and use the location of the best mass to speed up the exploitation phase. Figure 2 shows the effect of using the best solution to accelerate movement of agents towards the global optimum. As shown in this figure, the gbest element applies an additional velocity component towards the last known location for the best mass. In this way, the external gbest "force" helps to prevent masses from stagnating in a suboptimal situation. There are two benefits in this method: accelerating the movement of particles towards the location of the best mass, which may help them to surpass it and be the best mass in the next iteration, as illustrated in Fig. 2b and c and saving the best solution attained so far.

Equation 3.1 is proposed for mathematically modelling the proposed method, as follows:

$$V_i(t+1) = \text{rand} \times V_i(t) + c_1' \times \text{ac}_i(t) + c_2' \times (\text{gbest} - X_i(t)), \tag{3.1}$$

Generate initial population
while *(the end criterion is not satisfied)*
Evaluate the fitness of all agents
Update G by equation (2.3) and gbest
Calculate M using equation (2.10)
Calculate forces and accelerations using equations (2.5) and (2.6)
Update velocity and position by equation (3.1) and (3.2)
end while
return *gbest*

Fig. 3 General steps of the proposed method

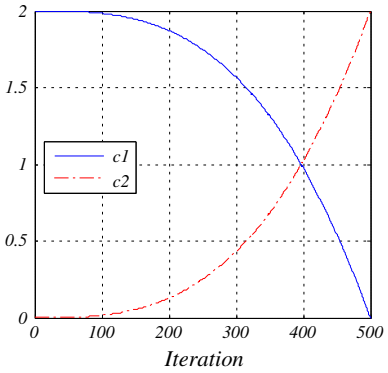

Fig. 4 Coefficients curves

where $V_i(t)$ is the velocity of agent i at iteration t, c'_1 and c'_2 are accelerating coefficients, rand is a random number between 0 and 1, $\mathrm{ac}_i(t)$ is the acceleration of agent i at iteration t, and gbest is the position of the best solution acquired so far. In Eq. 3.1, the first component $(\mathrm{rand} \times V_i(t) + c'_1 \times \mathrm{ac}_i(t) + c'_2)$ is the same as that of GSA, in which the exploration of the masses is emphasised. The second component $(c'_2 \times (\mathrm{gbest} - X_i(t)))$ is responsible for attracting masses towards the best masses obtained so far. The distance of each mass from the best mass is calculated by $\mathrm{gbest} - X_i(t)$. The final force towards the best mass is a random fraction the distance defined by c'_2 that will be the final external force applied to each mass.

In each iteration, the positions of agents are updated as follows:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{3.2}$$

In the proposed method, all agents are randomly initialised. Then, gravitational force, gravitational constant and resultant forces between them are calculated using (2.2), (2.3) and (2.5), respectively. After that, the accelerations of particles are defined as in (2.6). At each iteration, the best solution obtained so far should be updated. After calculating the accelerations and updating the best solution, the velocities of all agents can be calculated using (3.1). Finally, the positions of agents are updated as (3.2). The process terminates by satisfying an end criterion. The general steps of the proposed method are represented in Fig. 3.

A problem may arise in that this method could affect the exploration phase as well, since it establishes a permanent element of velocity updating. In order to prevent the new updating velocity method from degrading the exploration ability, we use adaptive values for c'_1 and c'_2 as shown in Fig. 4. We adaptively decrease c'_1 and increase c'_2 so that the masses tend to accelerate towards the best solution as the algorithm reaches the exploitation phase. Since there is no clear border between the exploration and exploitation phases in evolutionary algorithms, the adaptive method is the best option for allowing a gradual transition between these two phases. In addition, this adaptive approach emphasises exploration in the first iterations and exploitation in the final iterations.

It is worth mentioning that the second part of Eq. 3.1, $c'_2 \times (\mathrm{gbest} - X_i(t))$, is quite similar to the social component of PSO, so the proposed method could be also considered as a hybrid of GSA and PSO. A high value of c'_1 biases towards GSA behaviour, while a high value of c'_2 emphasises the social component of PSO in performing the search process. The adaptive method allows GSA to explore the search space and a PSO-like exploitation of the best solution discovered by GSA.

Some remarks on the proposed method and its advantages are the following:

- The proposed method uses a memory (gbest) for saving the best solution obtained so far, in contrast to the unmodified GSA, so it is accessible at any time and will not be lost.
- Each agent can observe the best solution (gbest) and move towards it, so masses are provided with a sort of social intelligence.
- The effect of gbest is emphasised in the exploitation phase by adapting c'_1 and c'_2.
- The effect of gbest on agents is independent of their masses and so can be considered as an external force not subject to gravitational rules. This effectively prevents particles from gathering together and having extremely slow movement.
- The computational cost of this method is extremely low.

Because of these features, the proposed method has the potential to provide superior results compared to GSA. In the following section, various benchmark functions are employed to explore the effectiveness of the proposed method in action.

Table 1 Benchmark functions

Benchmark functions	Range	Dim	f_{min}
Unimodal functions			
F_1: Shifted sphere function	[−100,100]	10,30 50, 100, 200, 300	−450
F_2: Shifted Schwefel's	[−100,100]	10,30 50, 100, 200, 300	−450
F_3: Shifted rotated high conditioned elliptic function	[−100,100]	10,30 50, 100, 200, 300	−450
F_4: Shifted Schwefel's with noise in fitness	[−100,100]	10,30 50, 100, 200, 300	−450
F_5: Schwefel's with global optimum on bounds	[−100,100]	10,30 50, 100, 200, 300	−310
Multimodal functions			
F_6: Shifted Rosenbrock's function	[−100,100]	10,30 50, 100, 200, 300	390
F_7: Shifted rotated Griewank's function without bounds	[0,600]	10,30 50, 100, 200, 300	−180
F_8: Shifted rotated Ackley's function with global optimum on bounds	[−32,32]	10,30 50, 100, 200, 300	−140
F_9: Shifted Rastrigin's function	[−5,5]	10,30 50, 100, 200, 300	−330
F_{10}: Shifted rotated Rastrigin's function	[−5,5]	10,30 50, 100, 200, 300	−330
F_{11}: Shifted rotated Weierstrass function	[−0.5,0.5]	10,30 50, 100, 200, 300	90
F_{12}: Schwefel's	[−100,100]	10,30 50, 100, 200, 300	−460
F_{13}: Expanded extended Griewank's plus Rosenbrock's function (F_8F_2)	[−3,1]	10,30 50, 100, 200, 300	−130
F_{14}: Shifted rotated expanded Scaffer's F_6	[−100,100]	10,30 50, 100, 200, 300	−300
Composite functions			
F_{15}: Hybrid composition function	[−5,5]	10,30 50, 100, 200, 300	120
F_{16}: Rotated hybrid composition function	[−5,5]	10,30 50, 100, 200, 300	120
F_{17}: Rotated hybrid composition function with noise in fitness	[−5,5]	10,30 50, 100, 200, 300	120
F_{18}: Rotated hybrid composition function	[−5,5]	10,30 50, 100, 200, 300	10
F_{19}: Rotated hybrid composition function with a narrow basin for the global optimum	[−5,5]	10,30 50, 100, 200, 300	10
F_{20}: Rotated hybrid composition function with the global optimum on the bounds	[−5,5]	10,30 50, 100, 200, 300	10
F_{21}: Rotated hybrid composition function	[−5,5]	10,30 50, 100, 200, 300	360
F_{22}: Rotated hybrid composition function with high condition number matrix	[−5,5]	10,30 50, 100, 200, 300	360
F_{23}: Non-continuous rotated hybrid composition function	[−5,5]	10,30 50, 100, 200, 300	360
F_{24}: Rotated hybrid composition function	[−5,5]	10,30 50, 100, 200, 300	260
F_{25}: Rotated hybrid composition function without bounds	[−2,5]	10,30 50, 100, 200, 300	260

4 Experimental results and discussion

To evaluate the performance of the proposed method, termed gbest-guided GSA (GGSA), 25 standard benchmark functions, the CEC 2005 test functions, are employed in this section [42]. These benchmark functions are the shifted, rotated, expanded and combined variants of the classical functions, the most challenging forms of the test functions. They can be divided into three groups: unimodal, multimodal and composite functions. Table 1 lists the functions, where Dim indicates dimension of the function, Range is the boundary of the function's search space, and f_{min} is the minimum return value of the function. We compare GGSA and GSA on problems of six different dimensions to verify the performance of both algorithms dealing with problems of different scale. A detailed description of the benchmark functions is available in the technical report by Suganthan et al. [42].

The GGSA and GSA have several parameters that were defined as in Table 2.

Table 2 Initial parameters

Algorithm	Parameter	Value
GGSA	Number of particles	30 (dim = 10, 30, 50), 60 (dim = 100, 200, 300)
	c_1'	$(-2t^3/T^3) + 2$
	c_2'	$(2t^3/T^3)$
	G_0	1
	α	20
	Max iterations	500
	Stopping criteria	Max iteration
GSA	Number of masses	30 (dim = 10, 30, 50), 60 (dim = 100, 200, 300)
	G_0	1
	α	20
	Max iterations	500
	Stopping criteria	Max iteration

T indicates the maximum number of interactions

t is the current iteration

Table 3 Minimisation results of the unimodal benchmark functions

Dim	*F*1	*F*2	*F*3	*F*4	*F*5
GGSA					
10	**−450.00000 ± 0.00000**	**−450.00000 ± 0.00000**	**−450.00000 ± 0.00000**	**−450.00000 ± 0.00000**	**−310.00000 ± 0.00000**
30	**1,038.47460 ± 975.055**	**1,410.45330 ± 919.248**	**922.223800 ± 683.538**	**1,127.84270 ± 1,217.68**	**2,115.65050 ± 1,965.03**
50	**31,012.0700 ± 5,200.31**	**31,020.6090 ± 6,791.73**	**30,232.2660 ± 6,218.12**	**30,564.1930 ± 3,753.42**	**30,084.5240 ± 6,179.49**
100	**157,574.471 ± 8,320.58**	**155,456.001 ± 10,458.1**	**156,098.961 ± 6,401.34**	**151,617.639 ± 5,460.79**	**161,084.308 ± 5,502.25**
200	**587,011.907 ± 11,179.7**	**584,665.605 ± 15,890.6**	**590,912.910 ± 13,248.5**	**586,355.823 ± 13,329.0**	**589,741.240 ± 20,028.2**
300	**858,222.694 ± 9,447.49**	**860,288.803 ± 18,289.4**	**866,201.591 ± 15,625.2**	**858,551.587 ± 19,348.6**	**853,933.492 ± 16,969.4**
GSA					
10	**−450.00000 ± 0.00000**	**−450.00000 ± 0.00000**	**−450.00000 ± 0.00000**	**−450.00000 ± 0.00000**	**−310.00000 ± 0.00000**
30	23,756.9400 ± 3,610.59	25,459.6480 ± 3,065.40	25,025.3420 ± 5,838.77	23,659.0870 ± 3,076.43	25,321.0640 ± 3,328.68
50	78,615.7800 ± 7,561.45	78,115.8710 ± 6,946.47	79,719.0030 ± 6,142.03	79,545.7910 ± 8,253.67	79,734.3280 ± 6,411.98
100	223,847.517 ± 10,639.9	229,005.346 ± 6,203.36	227,534.626 ± 5,379.42	225,582.846 ± 7,253.33	224,507.487 ± 5,893.19
200	670,413.528 ± 16,093.9	673,780.129 ± 18,084.8	686,825.022 ± 13,363.0	673,645.855 ± 14,617.9	677,916.400 ± 12,712.1
300	949,270.405 ± 18,348.1	956,634.832 ± 12,447.4	948,794.770 ± 13,517.7	965,324.325 ± 16,913.2	954,244.460 ± 12,784.8

Table 4 *p* values of Wilcoxon's rank-sum test for unimodal benchmark functions

Dim	*F*1	*F*2	*F*3	*F*4	*F*5
10	N/A	N/A	N/A	N/A	N/A
30	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)
50	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)
100	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)
200	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)
300	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)

The experimental results are presented in Tables 3, 4, 5, 6, 7 and 8. The results are averaged over 30 independent runs, and the best results are indicated in bold type. The average (AVE) and standard deviation (STD) of the best solution obtained in the last iteration are reported in the form of AVE ± STD. Note that the source codes of the GGSA algorithm can be found in http://www.alimirjalili.com/Projects.html.

According to Derrac et al. [43], to improve the evaluation of evolutionary algorithms' performance, statistical tests should be conducted to prove that a proposed new algorithm presents a significant improvement over other existing methods for a particular problem. In order to judge whether the results of the GGSA and GSA differ from each other in a statistically significant way, a nonparametric statistical test, Wilcoxon's rank-sum test [44], is carried out at 5 % significance level. The *p* values calculated in the Wilcoxon's rank-sum comparing the algorithms over all the benchmark functions are given in Tables 4, 6 and 8. In these tables, N/A indicates "Not Applicable," which means the statistical test cannot be done because both algorithms found the optimum successfully in all the runs. According to Derrac et al., *p* values that are less than 0.05 can be considered as strong evidence against the null hypothesis. Note that the results are provided in the form of *Algorithm (p value),* where *Algorithm* is the better algorithm based on the average of the results (AVE). In the following subsections, the simulation results of benchmark functions are explained and discussed in terms of search performance and convergence behaviour.

4.1 Search performance analysis

As shown in Table 3, the GGSA and GSA provide similar results on 10-D unimodal benchmark functions. For the remaining dimensions, GGSA has the best results for all benchmark functions. The *p* values presented in Table 4 indicate that the results of GGSA are significantly better than GSA. The unimodal benchmark functions have only one global solution without any local optima so they are highly suitable for examining exploitation. Therefore, these results demonstrate that the proposed method provides greatly improved exploitation compared to the original GSA.

However, multimodal test functions have many local minima, with the number increasing exponentially with dimension, so they are well suited to test the exploration capability of an algorithm. The mean and STD values in

Table 5 Minimisation results of the multimodal benchmark functions

Dim	*F*6	*F*7	*F*8	*F*9	*F*10
GGSA					
10	**390.000000 ± 0.00000**	**−109.64880 ± 222.4702**	**9,632.5842 ± 199.9269**	**24,318.8556 ± 93.2272**	**24,318.991 ± 96.71750**
30	**2,610.51600 ± 1,491.60**	**84,637.5080 ± 23,391.52**	**38,054.967 ± 1,881.951**	**79,133.2785 ± 480.221**	**79,173.459 ± 404.7450**
50	**31,490.0170 ± 5,020.49**	**203,828.334 ± 48,436.87**	**76,491.616 ± 2,982.408**	**132,155.559 ± 862.244**	**132,401.20 ± 319.1275**
100	**156,998.075 ± 9,113.09**	**365,001.140 ± 50,728.39**	**177,772.69 ± 4,312.698**	**264,098.649 ± 689.162**	**264,435.70 ± 630.8460**
200	**588,559.860 ± 17,866.5**	**894,117.506 ± 83,959.53**	**587,851.76 ± 8,084.269**	**696,260.734 ± 1,437.11**	**695,557.84 ± 1,357.026**
300	**865,395.919 ± 18,627.4**	**1,356,170.09 ± 94,453.33**	**848,279.17 ± 11,595.40**	**951,549.525 ± 1,922.34**	**950,034.26 ± 2,055.630**
GSA					
10	**390.000000 ± 0.00000**	13,024.87 0 ± 13,821.88	10,203.996 ± 452.626	24,375.5194 ± 116.077	24,344.7388 ± 126.611
30	29,514.9520 ± 5,720.44	177,978.330 ± 40,563.92	47,536.183 ± 3,246.152	79,604.3470 ± 606.208	79,321.2020 ± 590.015
50	82,455.2530 ± 8,705.94	616,546.368 ± 61,610.10	93,310.367 ± 4,827.996	132,390.083 ± 356.120	132,911.000 ± 649.400
100	225,961.501 ± 14,860.9	2,695,994.54 ± 211,511.9	218,785.48 ± 5,668.755	264,948.737 ± 1,018.87	264,902.427 ± 1,030.81
200	671,946.558 ± 16,681.5	9,304,835.05 ± 539,337.5	658,328.24 ± 5,237.024	702,211.439 ± 1,669.58	700,882.302 ± 1,079.05
300	949,534.815 ± 12,623.6	16,841,021.0 ± 552,117.3	936,002.17 ± 8,489.646	963,609.162 ± 2,070.76	964,702.193 ± 1,559.11

Dim	*F*11	*F*12	*F*13	*F*14
GGSA				
10	27,701.2434 ± 26.2017	**25,635.8866 ± 72.44380**	**25,896.771 ± 38.54430**	**−300.0000 0 ± 0.00000**
30	**87,737.4482 ± 93.4266**	**82,910.8146 ± 197.4686**	**83,335.913 ± 169.3187**	**1,461.86660 ± 1,208.91**
50	**144,933.445 ± 86.9164**	**138,212.204 ± 277.2080**	**138,416.86 ± 368.0530**	**31,240.1757 ± 6,198.75**
100	**286,910.627 ± 134.111**	**274,620.631 ± 572.6490**	**275,058.44 ± 307.4,835**	**157,685.984 ± 12,433.0**
200	**738,697.539 ± 440.459**	**716,167.820 ± 858.7484**	**716,968.57 ± 874.5393**	**584,837.006 ± 15,460.2**
300	**1,004,054.91 ± 549.127**	**977,130.968 ± 1,433.060**	977,518.19 ± 889.0172	**858,549.906 ± 16,630.9**
GSA				
10	**27,693.0119 ± 22.9293**	25,670.9233 ± 90.31160	25,964.668 ± 45.01150	**−300.00000 ± 0.00000**
30	87,789.4,728 ± 104.725	83,083.2753 ± 297.4075	83,579.995 ± 236.4237	24,581.2390 ± 2,629.66
50	145,071.771 ± 80.7319	138,284.762 ± 314.8960	138,465.68 ± 402.3,130	78,622.8973 ± 6,696.06
100	287,015.125 ± 167.504	275,369.983 ± 424.1814	275,445.67 ± 334.6597	220,939.682 ± 7,343.51
200	738,917.428 ± 289.219	717,498.153 ± 1,046.253	717,332.39 ± 744.1265	680,207.251 ± 15,864.5
300	1,004,612.02 ± 267.922	978,733.440 ± 611.0777	**978,589.17 ± 1,364.674**	954,869.302 ± 11,192.9

Table 6 *p* values of Wilcoxon's rank-sum test for multi-modal benchmark functions

Dim	*F*6	*F*7	*F*8	*F*9	*F*10
10	N/A	GGSA (8.74e-05)	GGSA (0.004586)	GGSA (0.241322)	GGSA (0.850107)
30	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.053903)	GGSA (0.384673)
50	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.273036)	GGSA (0.053903)
100	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.045155)	GGSA (0.472676)
200	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)
300	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)	GGSA (0.000183)

Dim	*F*11	*F*12	*F*13	*F*14
10	GSA (0.570750)	GGSA (0.273036)	GGSA (0.002202)	N/A
30	GGSA (0.273036)	GGSA (0.185877)	GGSA (0.037635)	GGSA (0.000183)
50	GGSA (0.004586)	GGSA (0.677585)	GGSA (0.850107)	GGSA (0.000183)
100	GGSA (0.161972)	GGSA (0.002827)	GGSA (0.045155)	GGSA (0.000183)
200	GGSA (0.241322)	GGSA (0.004586)	GGSA (0.161972)	GGSA (0.000183)
300	GGSA (0.021134)	GGSA (0.002202)	GGSA (0.121225)	GGSA (0.000183)

Table 7 Minimisation results of the composite benchmark functions

Dim	*F*15	*F*16	*F*17	*F*18	*F*19	*F*20
GGSA						
10	**24,754.23 ± 90.364**	**24,739.34 ± 121.11**	**24,745.42 ± 86.265**	**24,582.19 ± 111.98**	**24,607.66 ± 106.24**	24,695.82 ± 56.557
30	**79,729.60 ± 253.37**	**79,270.13 ± 376.45**	**79,652.03 ± 515.15**	79,643.85 ± 554.11	**79,510.88 ± 180.21**	**79,496.72 ± 201.66**
50	**132,574.3 ± 845.51**	**132,715.3 ± 795.05**	**132,714.2 ± 517.72**	**132,745.0 ± 467.87**	**132,491.7 ± 682.34**	132,780.9 ± 674.19
100	**264,443.6 ± 1,058.7**	**264,339.3 ± 803.46**	**264,680.1 ± 906.96**	**263,975.7 ± 804.12**	**264,411.1 ± 643.53**	**264,082.4 ± 903.38**
200	**695,325.7 ± 1,729.7**	**696,547.7 ± 2,065.0**	**695,079.6 ± 1,378.1**	**694,975.8 ± 1,287.1**	**695,651.1 ± 1,477.4**	**697,005.9 ± 1,551.5**
300	**950,454.9 ± 3,019.7**	**952,388.6 ± 3,049.5**	**951,423.9 ± 1,884.8**	**952,689.1 ± 1,905.0**	**951,062.5 ± 1,917.3**	**951,928.1 ± 2,450.1**
GSA						
10	24,758.10 ± 120.06	24,842.93 ± 107.97	24,836.86 ± 94.745	24,670.36 ± 90.001	24,708.33 ± 98.683	**24,612.79 ± 103.88**
30	79,832.79 ± 185.68	79,760.23 ± 169.95	79,680.81 ± 330.34	**79,278.21 ± 446.29**	79,683.00 ± 405.66	79,536.02 ± 460.50
50	132,877.0 ± 850.01	133,343.2 ± 833.95	133,295.6 ± 469.39	132,797.2 ± 522.30	133,145.1 ± 411.43	**132,646.3 ± 571.68**
100	265,419.7 ± 1,004.5	265,396.2 ± 947.22	264,941.6 ± 791.68	265,054.1 ± 1,034.8	265,197.8 ± 878.11	265,218.6 ± 700.48
200	702,309.0 ± 2,675.8	702,654.8 ± 1,864.2	702,187.2 ± 1,345.0	702,309.0 ± 1,141.5	702,977.8 ± 2,435.8	702,277.9 ± 1,335.4
300	964,649.3 ± 1,923.1	963,174.6 ± 2,054.8	963,750.0 ± 1,405.4	963,769.0 ± 2,901.6	965,054.2 ± 2,598.6	963,298.8 ± 2,449.6

Dim	*F*21	*F*22	*F*23	*F*24	*F*25
GGSA					
10	**24,953.41 ± 117.98**	**25,004.72 ± 92.801**	**24,998.07 ± 72.846**	**24,933.98 ± 125.66**	**24,870.91 ± 95.729**
30	**79,726.87 ± 187.21**	**79,592.34 ± 224.39**	**79,701.94 ± 353.10**	**79,757.11 ± 280.97**	**79,709.79 ± 364.14**
50	**132,753.7 ± 715.35**	**133,189.9 ± 412.32**	**132,971.1 ± 715.17**	**132,887.2 ± 502.82**	**132,874.6 ± 560.12**
100	**264,514.5 ± 930.59**	**264,479.8 ± 971.16**	**265,027.7 ± 758.81**	**264,309.5 ± 675.43**	**264,745.5 ± 1,091.3**
200	**696,218.2 ± 1,341.4**	**695,796.0 ± 835.54**	**695,220.6 ± 1,523.8**	**696,129.0 ± 1,352.0**	**696,973.2 ± 1,418.3**
300	**951,831.8 ± 1,941.1**	**951,873.2 ± 1,056.7**	**950,682.4 ± 2,081.1**	**951,058.4 ± 1,784.9**	**952,318.6 ± 2,089.0**
GSA					
10	25,043.82 ± 82.104	25,048.63 ± 65.284	**24,981.55 ± 83.522**	24,961.21 ± 64.549	24,963.03 ± 89.812
30	79,977.98 ± 264.92	79,820.98 ± 354.83	80,089.38 ± 272.27	79,907.19 ± 339.69	79,948.36 ± 161.12
50	133,202.4 ± 600.04	133,479.4 ± 875.04	133,062.5 ± 621.02	133,100.9 ± 724.07	132,904.8 ± 636.46
100	265,411.2 ± 625.06	265,123.1 ± 669.89	265,737.1 ± 720.56	265,406.6 ± 701.76	265,693.3 ± 886.62
200	702,844.7 ± 1,313.0	702,468.8 ± 1,830.7	702,742.0 ± 638.31	702,206.3 ± 1,267.5	703,801.7 ± 1,055.2
300	965,684.7 ± 1,831.3	963,284.4 ± 1,418.5	964,654.4 ± 1,874.3	965,154.6 ± 2,796.5	964,136.8 ± 2,412.6

Table 5 and p values in Table 6 are for this class of benchmark functions. These tables show that GGSA provided better results than GSA across the majority of dimensions. GGSA completely outperformed GSA on *F*7 and *F*9 for all dimensions. For some of the remaining benchmark functions, GSA provides apparently better results but the p values indicate the results for some cases were not statistically significant. The results of the third group of benchmark functions, composite benchmark functions, are provided in Tables 7 and 8.

The third class of benchmark functions, the composite functions, have extremely complex structures with many local minima, similar to real-world problems. These functions are thus suitable to test an algorithm in terms of both exploration and exploitation. Many of the results for these functions presented in Tables 7 and 8 are inconclusive. However, GGSA still showed generally better results, particularly for higher-dimensional functions.

To give an overall image of the performance of these algorithms, a summary of the results is provided in Figs. 5 and 6. It may be seen that the results of GGSA tend to be significantly better than GSA as the dimension and complexity of the problems rises (e.g., the ESGA provides significant improvement on almost all benchmark functions in 200 and 300 dimensions). This improving trend can be seen in both figures.

In summary, the results on unimodal functions strongly suggest that GGSA has superior exploitation to GSA. The results obtained on multimodal functions also show that the proposed method is capable of efficiently exploring

Table 8 *p* values of Wilcoxon's rank-sum test for composite benchmark functions

Dim	*F*15	*F*16	*F*17	*F*18	*F*19	*F*20
10	GGSA (0.96985)	GGSA (0.05390)	GGSA (0.05390)	GGSA (0.10411)	GGSA (0.04515)	GSA (0.03120)
30	GGSA (0.34470)	GGSA (0.00220)	GGSA (0.79133)	GSA (0.12122)	GGSA (0.34470)	GGSA (0.90972)
50	GGSA (0.27303)	GGSA (0.14046)	GGSA (0.01725)	GGSA (0.79133)	GGSA (0.01725)	GSA (0.67758)
100	GGSA (0.05390)	GGSA (0.02574)	GGSA (0.52052)	GGSA (0.03763)	GGSA (0.03120)	GGSA (0.01401)
200	GGSA (0.00018)	GGSA (0.00033)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)
300	GGSA 0.00018)	GGSA (0.00033)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)

Dim	*F*21	*F*22	*F*23	*F*24	*F*25
10	GGSA (0.05390)	GGSA (0.34470)	GSA (0.62317)	GGSA (0.47267)	GGSA (0.03763)
30	GGSA (0.03763)	GGSA (0.16197)	GGSA (0.01725)	GGSA (0.57075)	GGSA (0.01133)
50	GGSA (0.21229)	GGSA (0.38467)	GGSA (0.79133)	GGSA (0.67758)	GGSA (0.96985)
100	GGSA (0.03120)	GGSA (0.08897)	GGSA (0.06402)	GGSA (0.00728)	GGSA (0.05390)
200	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)
300	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)	GGSA (0.00018)

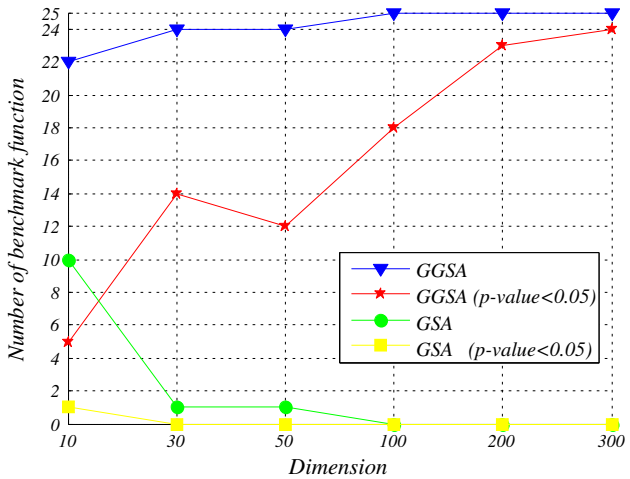

Fig. 5 Statistical results of all benchmark functions based on dimension

the search space. Moreover, the results on composite functions highly support this statement that the proposed adaptive values for c_1' and c_2' is a suitable method to balance exploration and exploitation. In the next section, the convergence behaviour of GGSA and GSA is investigated.

4.2 Convergence analysis

The convergence curves of both algorithms over 30-D benchmarks at all dimensions are illustrated in Fig. 11 in the Appendix. As may be seen from these figures, GGSA has a much better convergence rate than GSA on unimodal benchmark functions for all dimensions. Generally speaking, the convergence of GGSA tends to surpass GSA from the initial steps of a run. Considering this behaviour and the characteristics of unimodal functions indicates that the proposed method successfully improves the convergence rate compared with GSA and the consequent effectiveness of exploitation.

The convergence of the algorithms on the majority of multimodal and composite benchmark functions follows a different scenario. The GGSA has worse convergence than GSA in the initial iterations. However, the search process is progressively accelerated during iterations for this algorithm. This acceleration helps GGSA to surpass GSA after completing about a quarter of the iterations. This demonstrates how GGSA is capable of balancing exploration and exploitation to find the global optimum rapidly and effectively.

Overall, these results show that the proposed algorithm is able to significantly improve on the performance of GSA, overcoming its major shortcomings. In the next section, the performance of GGSA solving real engineering constraint problems is examined.

5 GGSA for classical design engineering problems

In the field of meta-heuristics, it is very common that an algorithm provides very promising results on test problems, but poor results on real challenging problems. In other words, there are possibilities that an algorithm utilises the known characteristics of test functions (symmetric for instance) and provides good superior results. In this case, some classical engineering design problems with unknown search spaces are available in the literature to benchmark meta-heuristics. Moreover, real problems usually have constraints, and an algorithm should be able to handle constraints in order to solve them. In this

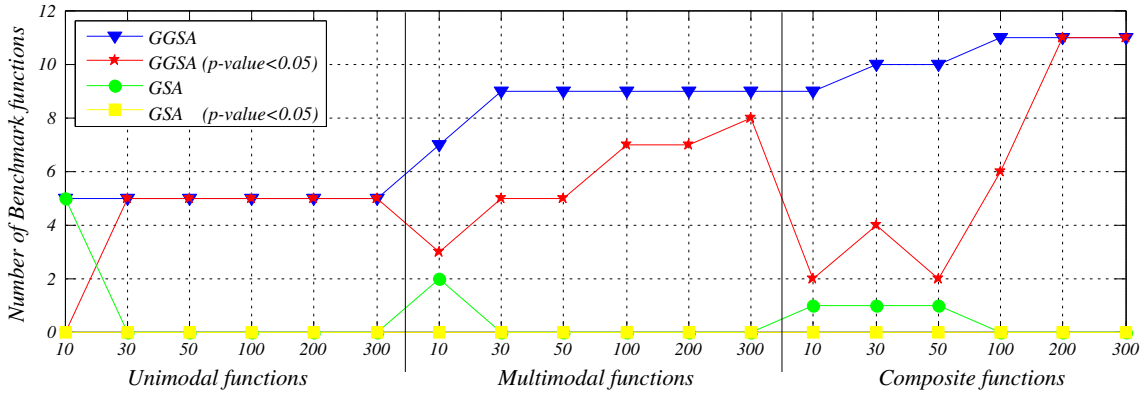

Fig. 6 Statistical results of unimodal, multimodal, and composite benchmark functions separately

Table 9 Comparison of results for tension/compression spring design problem

Algorithm	Optimum variables			Optimum weight
	d	*D*	*N*	
GGSA	**0.051319**	**0.347901**	**11.825211**	**0.0126677**
GSA	0.050276	0.323680	13.525410	0.0127022
PSO (Ha and Wang)	0.051728	0.357644	11.244543	0.0126747
ES (Coello and Montes)	0.051989	0.363965	10.890522	0.0126810
GA (Coello)	0.051480	0.351661	11.632201	0.0127048
HS (Mahdavi et al.)	0.051154	0.349871	12.076432	0.0126706
DE (Huang et al.)	0.051609	0.354714	11.410831	0.0126702
Mathematical optimisation (Belegundu)	0.053396	0.399180	9.1854000	0.0127303
Constraint correction (Arora)	0.050000	0.315900	14.250000	0.0128334

section, two constrained engineering design problems (a tension/compression spring and welded beam) are employed to further investigate the efficiency of the proposed method for real problems. We have chosen these classical engineering problems because they are constrained, so the ability of the proposed method would be benchmarked in terms of solving constrained problems. Moreover, the employed engineering problems are the simplified version of real problems with unknown search spaces, so the performance of the proposed method would be examined in terms of finding the optima of unknown challenging search spaces.

5.1 Tension/compression spring design

The objective of this problem is to minimise the weight of a tension/compression spring [45–47]. The minimisation process is subject to some constraints such as shear stress, surge frequency and minimum deflection. There are three variables in this problem: wire diameter (d), mean coil diameter (D) and the number of active coils (N). The mathematical formulation of this problem is as follows:

$$
\begin{aligned}
&\text{Consider} \quad \vec{x} = [x_1\ x_2\ x_3] = [d\ D\ N],\\
&\text{Minimize} \quad f(\vec{x}) = (x_3+2)x_2x_1^2,\\
&\text{Subject to} \quad g_1(\vec{x}) = 1 - \frac{x_2^3 x_3}{71785x_1^4} \le 0,\\
&g_2(\vec{x}) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} \le 0,\\
&g_3(\vec{x}) = 1 - \frac{140.45x_1}{x_2^2x_3} \le 0,\\
&g_4(\vec{x}) = \frac{x_1+x_2}{1.5} - 1 \le 0,\\
&\text{Variable range } 0.05 \le x_1 \le 2.00,\\
&\quad 0.25 \le x_2 \le 1.30, 2.00 \le x_3 \le 15.0,
\end{aligned}
\tag{5.1}
$$

This problem has been tackled by both mathematical and heuristic approaches. Ha and Wang tried to solve this problem using PSO [48]. The evolution strategy (ES) [49], GA [50], harmony search (HS) [51] and DE [52] algorithms have also been employed as heuristic optimisers for this problem. The mathematical approaches that have been adopted to solve this problem are the numerical optimisation technique (constraints correction at constant cost) [45] and mathematical optimisation technique [46]. The comparison of results of these techniques and the proposed method are provided in Table 9. Note that we use a penalty

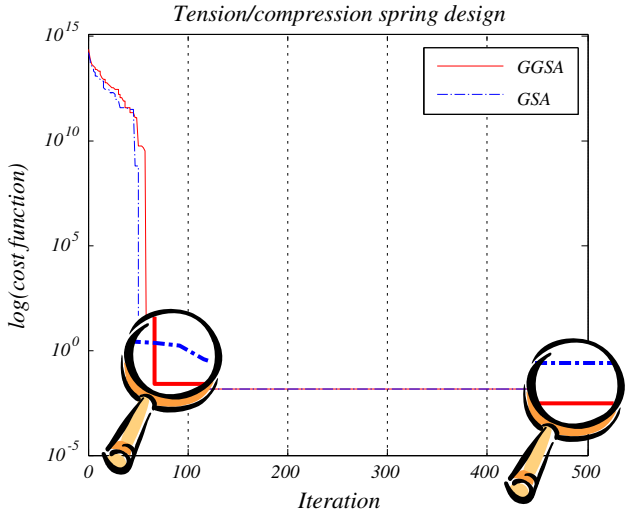

Fig. 7 Convergence curve of GGSA and GSA in tension/compression design problem

function for both GGSA and GSA to perform a fair comparison [53]. As shown in Table 9, GSA only outperforms two of the algorithms, whereas GGSA consistently has the best results. The convergence curves of GGSA and GSA in this problem are illustrated in Fig. 7. The convergence behaviour is entirely consistent with the previous results whereby GGSA is accelerated over the iteration. As the exploitation phase comes up, the GGSA surpasses GSA and exploits a better solution eventually.

5.2 Welded beam design

The objective of this problem was to minimise the fabrication cost of a welded beam [50]. The constraints are shear stress (τ), bending stress in the beam (θ), buckling load on the bar (P_c), end deflection of the beam (δ) and side constraints. This problem has four variables: thickness of weld (h), length of attached part of bar (l), the height of the bar (t) and thickness of the bar (b). The mathematical formulation is as follows:

$$
\begin{aligned}
&\text{Consider} \quad \vec{x} = [x_1\ x_2\ x_3\ x_4] = [h\ l\ t\ b],\\
&\text{Minimize} \quad f(\vec{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2),\\
&\text{Subject to} \quad g_1(\vec{x}) = \tau(\vec{x}) - \tau_{\max} \le 0,\\
&g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{\max} \le 0,\\
&g_3(\vec{x}) = \delta(\vec{x}) - \delta_{\max} \le 0,\\
&g_4(\vec{x}) = x_1 - x_4 \le 0,\\
&g_5(\vec{x}) = P - P_c(\vec{x}) \le 0,\\
&g_6(\vec{x}) = 0.125 - x_1 \le 0\\
&g_7(\vec{x}) = 1.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \le 0,\\
&\text{Variable range} \quad 0.1 \le x_1 \le 2,\\
&0.1 \le x_2 \le 10,\\
&0.1 \le x_3 \le 10,\\
&0.1 \le x_4 \le 2,\\
&\text{where} \quad \tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2},\\
&\tau' = \frac{P}{\sqrt{2}x_1x_2}, \tau'' = \frac{\mathrm{MR}}{J}, M = P\left(L + \frac{x_2}{2}\right),\\
&R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2},\\
&J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\},\\
&\sigma(\vec{x}) = \frac{6\mathrm{PL}}{x_4x_3^2}, \delta(\vec{x}) = \frac{6\mathrm{PL}^3}{Ex_3^2x_4}\\
&P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right),\\
&P = 6000\mathrm{lb}, L = 14\mathrm{in}., \delta_{\max} = 0.25\mathrm{in}.,\\
&E = 30 \times 1^6\,\mathrm{psi}, G = 12 \times 10^6\mathrm{psi},\\
&\tau_{\max} = 13600\,\mathrm{psi}, \sigma_{\max} = 30000\,\mathrm{psi},
\end{aligned}
\tag{5.2}
$$

Coello [54] and Deb [55, 56] employed GA, whereas Lee and Geem [57] utilised HS to solve this problem. Richardson's random method, simplex method, Davidon–Fletcher–Powell, Griffith and Stewart's successive linear approximation are the mathematical approaches that have been adopted by Radgsdell and Philips [58] for this problem. The comparison results are provided in Table 10. The results

Table 10 Comparison results for welded beam design problem

Algorithm	Optimum variables				Optimum cost
	h	l	t	b	
GGSA	**0.21591707125**	**3.3149551004**	**8.8961948919**	**0.21591707126**	**1.770829299**
GSA	0.1821296746	3.8569797232	10.00000000	0.2023764710	1.879952224
GA (Coello)	N/A	N/A	N/A	N/A	1.8245
GA (Deb)	N/A	N/A	N/A	N/A	2.3800
GA (Deb)	0.2489	6.1730	8.1789	0.2533	2.4331
HS (Lee and Geem)	0.2442	6.2231	8.2915	0.2443	2.3807
Random	0.4575	4.7313	5.0853	0.6600	4.1185
Simplex	0.2792	5.6256	7.7512	0.2796	2.5307
David	0.2434	6.2552	8.2915	0.2444	2.3841
APPROX	0.2444	6.2189	8.2915	0.2444	2.3815

show that GGSA and GSA both have high performance in this problem. The result of GGSA, however, is better than GSA. Moreover, Fig. 8 demonstrates that GGSA has a slightly better convergence rate with the acceleration noted in previous benchmarks.

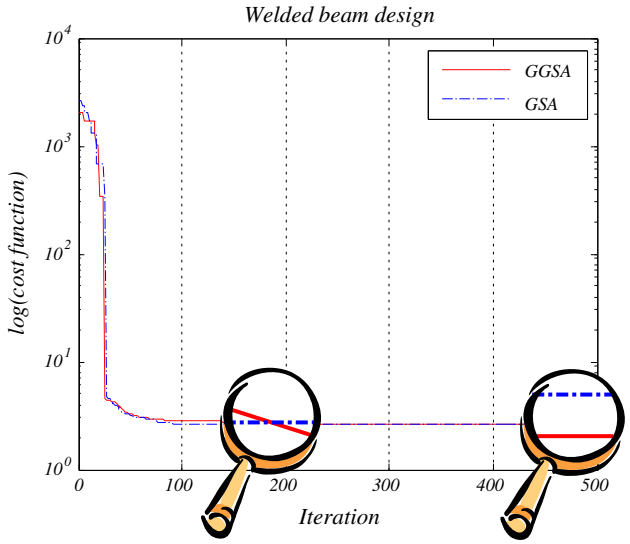

Fig. 8 Convergence curve of GGSA and GSA in welded beam design problem

The possible reason for this is that GSA is not able to discover new feasible areas in the search space when masses should cross an infeasible area to reach a new feasible area. This phenomenon is visualised in Fig. 9. It can be observed in this figure that the masses are not able to move to the second feasible zone since there is no mass there to attract them. Moreover, the infeasible masses are extremely light when using a penalty function or assigning large fitness function values to handle constraints. Thus, they get attracted back immediately they exit feasible areas. However, as is clearly shown in Fig. 10, the proposed method helps masses cross infeasible areas of search space. Therefore, the proposed method helps masses not only to accelerate towards the best solution but also discover new feasible areas in some special situations.

In summary, the results show that the proposed method successfully outperforms GSA in a majority of benchmark functions. Furthermore, tests using classical engineering problems show that GGSA is capable of providing very competitive results, indicating the superior performance of this algorithm in solving constrained problems compared to GSA. It therefore appears from this comparative study that the proposed method has merit in the field of evolutionary algorithm and optimisation.

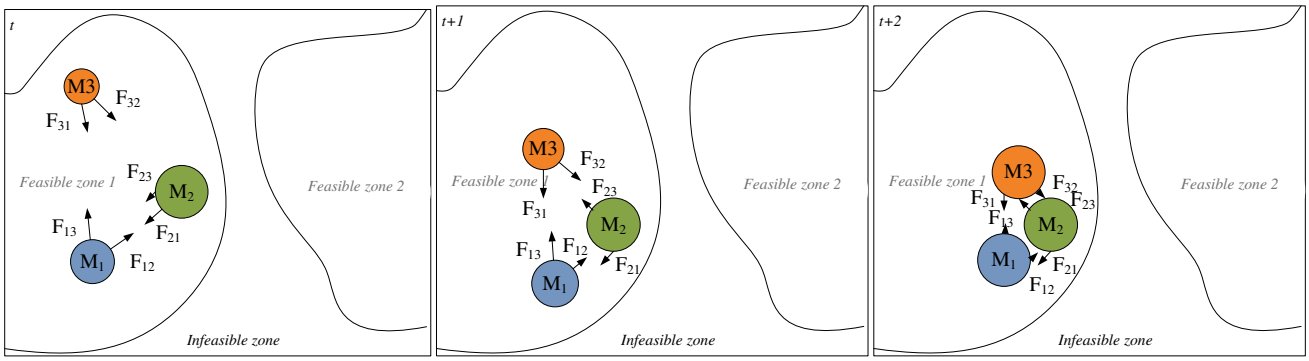

Fig. 9 Behaviour of masses of GSA in constrained search space

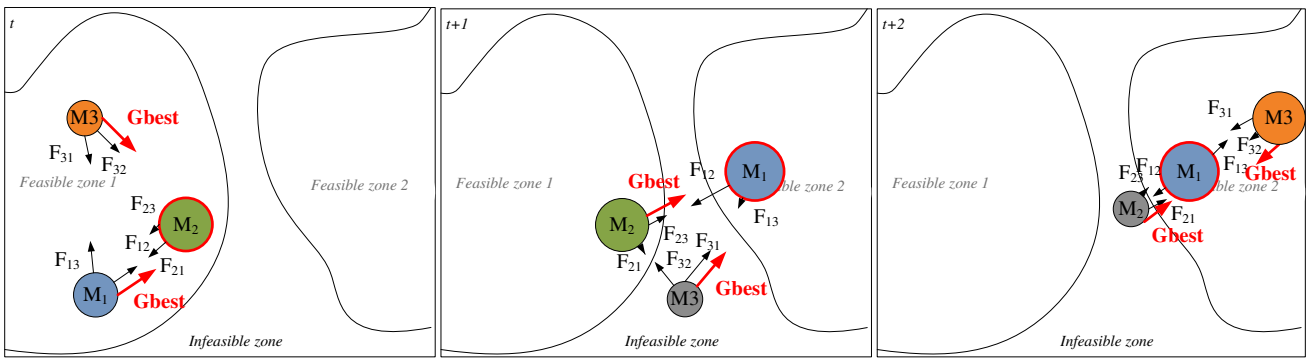

Fig. 10 Behaviour of masses of GGSA in constrained search space

6 Conclusion

In this work, the exploitation of GSAs was improved by accelerating the masses towards the best solution currently obtained. The proposed method proved its superior performance on 25 benchmark functions in terms of improved avoidance of local minima and increased convergence rate.

Furthermore, two classical engineering design problems were employed to benchmark the performance of the proposed method when applied to real problems. The results verified the superior performance of GGSA on optimising constrained problems when compared to GSA, due to its improved ability to discover new, promising feasible areas.

For future studies, it would be interesting to expand the application of GGSA further on other real-world engineering problems.

Appendix

See Fig. 11.

Fig. 11 Convergence behaviour of GGSA and GSA on the benchmark functions with dim = 30

References

1. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, vol 4, pp 1942–1948
2. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: IEEE international conference on evolutionary computation. Anchorage, Alaska, pp 69–73
3. Holland JH (1992) Genetic algorithms. Scientific Am 267:66–72
4. Price K, Storn R (1997) Differential evolution. Dr. Dobb's J 22:18–20
5. Price KV, Storn RM, Lampinen JA (2005) Differential evolution: a practical approach to global optimization. Springer, New York
6. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. Comput Intell Mag, IEEE 1(4):28–39. doi:10.1109/MCI.2006.329691
7. Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12:702–713
8. Mirjalili S, Mirjalili SM, Lewis A (2014) Let a biogeography-based optimizer train your multi-layer perceptron. Inf Sci 269:188–209. doi:10.1016/j.ins.2014.01.038
9. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
10. Gandomi AH, Alavi AH (2012) Krill Herd: a new bio-inspired optimization algorithm. Commun Nonlinear Sci Num Simul 17:4831–4845
11. Guo L, Wang G-G, Gandomi AH, Alavi AH, Duan H (2014) A new improved krill herd algorithm for global numerical optimization. Neurocomputing. doi:10.1016/j.neucom.2014.01.023
12. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179:2232–2248
13. Mirjalili S, Mirjalili SM, Yang X-S (2013) Binary bat algorithm. Neural Comput Appl 1–19. doi:10.1007/s00521-013-1525-5
14. Mirjalili S, Lewis A (2013) S-shaped versus V-shaped transfer functions for binary particle swarm optimization. Swarm Evol Comput 9:1–14
15. Lai X, Zhang M (2009) An efficient ensemble of GA and PSO for real function optimization. In: 2nd IEEE International conference on computer science and information technology, 2009. ICCSIT 2009. pp 651–655
16. Esmin AAA, Lambert-Torres G, Alvarenga GB (2006) Hybrid evolutionary algorithm based on PSO and GA mutation. In: International conference on hybrid intelligent systems. IEEE Computer Society, Los Alamitos, CA, USA, p 57. http://doi.ieeecomputersociety.org/10.1109/HIS.2006.33
17. Zhang WJ, Xie XF (2003) DEPSO: hybrid particle swarm with differential evolution operator, vol. 4, pp. 3816–3821
18. Niu B, Li L (2008) A novel PSO-DE-based hybrid algorithm for global optimization. In: advanced intelligent computing theories and applications. With aspects of artificial intelligence, pp 156–163
19. Holden NP, Freitas AA (2007) A hybrid PSO/ACO algorithm for classification, pp 2745–2750
20. Holden N, Freitas AA (2008) A hybrid PSO/ACO algorithm for discovering classification rules in data mining. J Artif Evol Appl 2008:2
21. Wang G-G, Gandomi AH, Alavi AH, Hao G-S (2013) Hybrid krill herd algorithm with differential evolution for global numerical optimization. Neural Comput Appl 1–12. doi:10.1007/s00521-013-1485-9
22. Wang G-G, Gandomi AH, Alavi AH (2013) An effective krill herd algorithm with migration operator in biogeography-based optimization. Appl Math Model. doi:10.1016/j.apm.2013.10.052
23. Noman N, Iba H (2008) Accelerating differential evolution using an adaptive local search. IEEE Trans Evol Comput 12:107–125
24. Chen J, Qin Z, Liu Y, Lu J (2005) Particle swarm optimization with local search, pp 481–484
25. Chen S, Mei T, Luo M, Yang X (2007) Identification of nonlinear system based on a new hybrid gradient-based PSO algorithm, pp 265–268
26. Meuleau N, Dorigo M (2002) Ant colony optimization and stochastic gradient descent. Artif Life 8:103–121
27. Wang G–G, Gandomi AH, Alavi AH (2013) A chaotic particle-swarm krill herd algorithm for global numerical optimization. Kybernetes 42:9
28. Wang G-G, Guo L, Gandomi AH, Hao G-S, Wang H (2014) Chaotic Krill Herd algorithm. Inf Sci. doi:10.1016/j.ins.2014.02.123
29. Saremi S, Mirjalili SM, Mirjalili S (2014) Chaotic Krill Herd optimization algorithm. Procedia Technol 12:180–185
30. Wang G-G, Gandomi AH, Alavi AH (2014) Stud Krill Herd algorithm. Neurocomputing 128:363–370
31. Wang G, Guo L, Wang H, Duan H, Liu L, Li J (2014) Incorporating mutation scheme into krill herd algorithm for global numerical optimization. Neural Comput Appl 24(3–4):853–871. doi:10.1007/s00521-012-1304-8
32. Zhang Y, Wu L, Zhang Y, Wang J (2012) Immune gravitation inspired optimization algorithm advanced intelligent computing, vol 6838. In: Huang D-S, Gan Y, Bevilacqua V, Figueroa J (eds). Springer, Berlin/Heidelberg, pp. 178–185
33. Sinaie S (2010) Solving shortest path problem using gravitational search algorithm and neural networks. Master, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia
34. Shaw B, Mukherjee V, Ghoshal SP (2012) A novel opposition-based gravitational search algorithm for combined economic and emission dispatch problems of power systems. Int J Electric Power Energy Syst 35:21–33
35. Chen H, Li S, Tang Z (2011) Hybrid gravitational search algorithm with random-key encoding scheme combined with simulated annealing. IJCSNS 11:208
36. Hatamlou A, Abdullah S, Othman Z (2011) Gravitational search algorithm with heuristic search for clustering problems. In: Data mining and optimization (DMO), 2011 3rd conference on 2011, pp 190–193
37. Li C, Zhou J (2011) Parameters identification of hydraulic turbine governing system using improved gravitational search algorithm. Energy Convers Manag 52:374–381
38. Mirjalili S, Mohd Hashim SZ, Moradian Sardroudi H (2012) Training feed forward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Appl Math Comput 218:11125–11137
39. Mirjalili S, Hashim SZM (2010) A new hybrid PSOGSA algorithm for function optimization. In: Computer and information application (ICCIA), 2010 international conference on, 2010, pp 374–377
40. Gauci M, Dodd TJ, Groß R (2012) Why 'GSA: a gravitational search algorithm' is not genuinely based on the law of gravity. Nat Comput 11(4):719–720. doi:10.1007/s11047-012-9322-0
41. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2010) BGSA: binary gravitational search algorithm. Nat Comput 9:727–745
42. Suganthan PN, Hansen N, Liang JJ, Deb K, Chen Y, Auger A, Tiwari S (2005) Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Nanyang Technological University, Singapore, Tech. Rep, vol 2005005
43. 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(1):3–18. doi:10.1016/j.swevo.2011.02.002

44. Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics Bull 1:80–83
45. Arora JS (2004) Introduction to optimum design. Academic Press, London
46. Belegundu AD (1983) Study of mathematical programming methods for structural optimization. Dissertation abstracts international part B: science and engineering [DISS. ABST. INT. PT. B- SCI. & ENG.], vol 43, p 1983
47. Coello Coello CA, Mezura Montes E (2002) Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Adv Eng Inf 16:193–203
48. He Q, Wang L (2007) An effective co-evolutionary particle swarm optimization for constrained engineering design problems. Eng Appl Artif Intell 20:89–99
49. Mezura-Montes E, Coello CAC (2008) An empirical study about the usefulness of evolution strategies to solve constrained optimization problems. Int J General Syst 37:443–473
50. Coello Coello CA (2000) Use of a self-adaptive penalty approach for engineering optimization problems. Comput Ind 41:113–127
51. Mahdavi M, Fesanghary M, Damangir E (2007) An improved harmony search algorithm for solving optimization problems. Appl Math Comput 188:1567–1579
52. Huang F, Wang L, He Q (2007) An effective co-evolutionary differential evolution for constrained optimization. Appl Math Comput 186:340–356
53. Yang XS (2011) Nature-inspired metaheuristic algorithms. Luniver Press, UK
54. Carlos A, Coello C (2000) Constraint-handling using an evolutionary multiobjective optimization technique. Civil Eng Syst 17:319–346
55. Deb K (1991) Optimal design of a welded beam via genetic algorithms. AIAA J 29:2013–2015
56. Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186:311–338
57. Lee KS, Geem ZW (2005) A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. Comput Methods Appl Mech Eng 194:3902–3933
58. Ragsdell K, Phillips D (1976) Optimal design of a class of welded structures using geometric programming. ASME J Eng Ind 98:1021–1025