Neural Comput & Applic (2015) 26:625–640
DOI 10.1007/s00521-014-1743-5

ORIGINAL ARTICLE

How important is a transfer function in discrete heuristic algorithms

Shahrzad Saremi · Seyedali Mirjalili · Andrew Lewis

Received: 31 May 2014 / Accepted: 22 September 2014 / Published online: 7 October 2014

Abstract Transfer functions are considered the simplest and cheapest operators in designing discrete heuristic algorithms. The main advantage of such operators is the maintenance of the structure and other continuous operators of a continuous algorithm. However, a transfer function may show different behaviour in various heuristic algorithms. This paper investigates the behaviour and importance of transfer functions in improving performance of heuristic algorithms. As case studies, two algorithms with different mechanisms of optimisation were chosen: Gravitational Search Algorithm and Particle Swarm Optimisation. Eight transfer functions were integrated in these two algorithms and compared on a set of test functions. The results show that transfer functions may show diverse behaviours and have different impacts on the performance of algorithms, which should be considered when designing a discrete algorithm. The results also demonstrate the significant role of the transfer function in terms of improved exploration and exploitation of a heuristic algorithm.

Keywords Binary optimisation · Discrete optimisation · Transfer function · Evolutionary algorithm · Heuristic algorithm · Binary algorithm · Discrete algorithm

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

S. Saremi
e-mail: shahrzad.saremi@griffithuni.edu.au

A. Lewis
e-mail: a.lewis@griffith.edu.au

1 Introduction

Heuristic optimisation algorithms have become popular optimisation techniques over the last decade. Generally, heuristic algorithms are suitable substitutions for conventional optimisation techniques due to four main reasons: flexibility, simplicity, derivation-free mechanism, and low computational cost. Flexibility refers to the ability of a heuristic algorithm to adapt to different problems. The literature shows that a heuristic algorithm can be applied to different problems without any special changes. The standard Particle Swarm Optimisation (PSO) algorithm [1], for instance, has been applied to many problem domains such as electromagnetics [2], civil engineering [3], and mechanical engineering [4]. The second reason, simplicity, comes from the simplifying abstraction of the behaviour of animals/insects, natural phenomena, or evolutionary concepts. For example, PSO, Artificial Bee Colony (ABC) algorithm [5], and Ant Colony Optimisation (ACO) [6] mimic social behaviours of birds, bees, and ants, respectively. In addition, Simulated Annealing (SA) [7] and Gravitational Search Algorithm (GSA) [8] were inspired by the annealing process in material engineering and the gravitational forces between particles. Some of the algorithms that have been developed based on the concepts of evolution are Genetics Algorithm (GA) [9], Biogeography-based Optimisation algorithm (BBO) [10], and Evolution Strategy (ES) [11]. From these various inspirations, simple rules have been derived that make heuristic algorithms very understandable for researchers from different fields of study.

The derivation-free mechanism may be considered a major reason for the popularity of heuristic algorithm in real problems. Generally, conventional or mathematical optimisation approaches calculate gradients of the search

space at each step of optimisation. This process needs to have derivative information of search spaces. Derivative information of real problems is often difficult to obtain, so conventional optimisation techniques fail to optimise real problems or are expensive or difficult to implement. Heuristic algorithms do not utilise gradient information because they are generally stochastic algorithms. They start the search process with an initial random set of solutions and enhance it during the optimisation process. Therefore, they can be employed for optimising real problems without concern about the derivability of search spaces. Low computational cost is another advantage of heuristic algorithms. Since heuristic algorithms use simple concepts and there is no need to calculate derivatives of the search space, they generally have very low computational cost. All these advantages make heuristic algorithm readily applicable to real-world problems.

There are two types of real problems in terms of the type of variables (search space): continuous and discrete. A continuous problem has a continuous search space as its name implies. The problem's variables (parameters) have continuous ranges, so the search agents of heuristic algorithms can easily move over the search space. In contrast, the continuity of parameters is no longer meaningful in discrete problems. A discrete search space can be considered as a hypercube. The search agents of heuristic algorithms have to jump to nearer and farther corners of this hypercube. These jumps (sudden movements) are usually achieved by flipping various numbers of bits. It is obvious that the continuous version of a heuristic algorithm cannot be applied directly to binary problems. There are a number of different approaches to convert a continuous heuristic algorithm to a binary algorithm.

For Differential Evolution (DE) [12], Wang et al. [13] proposed a probability estimation operator to make DE capable of solving discrete problems. Harmony search (HS) [14] was converted to Binary HS [15] by a set of HS considerations and pitch adjustment rules. In addition, the binary versions of the Magnetic Optimisation Algorithm (MOA) [16, 17], PSO, GSA, and Bat Algorithm (BA) [18] were adapted to solve binary problems using transfer functions and new position updating rules. The transfer function was the main element of BMOA [19], BPSO [20], BGSA [21], and BBA [22]. Currently, there is insufficient research on transfer functions despite their importance. In fact, one of the few papers in the literature specifically addressing questions relating to the transfer functions themselves was the work of two of the authors in which they proposed two families of transfer functions: s-shaped and v-shaped [23]. However, the impacts of these two families on different algorithms, exploration, and exploitation should be further described to enhance understanding. This is the motivation of this paper, in which we investigate the impact of the proposed transfer functions in [23] on two different algorithms (GSA and PSO) to observe their effectiveness when applied to different heuristic algorithms. In addition, the effectiveness of employing various transfer functions is investigated in terms of improving both exploration and exploitation of the algorithms. The remainder of the paper is organised as follows.

Section 2 presents a brief introduction to GSA and PSO algorithms. Section 3 discusses the basic principles of binary optimisation with emphasis on the mechanisms of BGSA and BPSO. The transfer functions are investigated and discussed in Sect. 4. The experimental results are demonstrated in Sect. 5. Finally, Sect. 6 concludes the work and suggests some directions for future research.

2 Gravitational Search Algorithm and Particle Swarm Optimisation

2.1 Gravitational Search Algorithm

The GSA algorithm has been inspired by gravitational forces between particles in nature [8]. The GSA algorithm solves optimisation problems by a collection of search agents called masses. The fitness function defines the weight of each mass. Masses are allowed to attract each other by gravitational forces between them, and their movements are calculated by the laws of motion (acceleration).

This algorithm has been mathematically formulated as follows [8]:

Every mass has a position in search space as follows:

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

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 with a random set of agents. The gravitational force from agent j on agent i at a specific time t is defined as follows:

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

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 $R_{ij}(t)$ is the Euclidian distance between two agents i and j.

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

$$G(t) = G_0 \times \exp(-\alpha \times iter/maxiter) \tag{2.3}$$

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

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

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

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

where $rand_j$ is a random number in the interval [0, 1].

The law of motion has also been utilised in this algorithm. The acceleration of a mass is proportional to the resultant force and inverse of its mass, so the accelerations of all agents are calculated as follows:

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

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:

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \tag{2.7}$$

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

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

Due to defining the mass of each search agent by the fitness function, the fittest agent is the heaviest agent. This agent has the highest attractive force and slowest movement as can be inferred from Eqs. (2.2) and (2.6), respectively.

Since the fitness function defines the mass of agents, a normalisation method has been adopted to scale masses as follows:

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \tag{2.9}$$

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

where $fit_i(t)$ 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 least fit agent at time t.

In the initial set of agents, the mass of each agent is first defined by the fitness function. The resultant gravitational force is then calculated by Eqs. (2.2) and (2.5). The final velocity and position of each agent are then defined by Eqs. (2.7) and (2.8), respectively. Meanwhile, the other parameters such as the gravitational constant and masses are updated by (2.3) and (2.10). Finally, this process is repeated iteratively until the satisfaction of a termination criterion (e.g. maximum number of iterations).

In spite of the merit of GSA in finding the global optimum, the search process in GSA slows as the number of iterations increases. This is due to the cumulative effect of the fitness function on calculating the masses of search agents, making masses greater over the course of the optimisation process. This prevents the GSA algorithm from rapidly exploiting the best solutions in later iterations. This problem has been addressed directly or indirectly in the literature. 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 [24]. Shaw et al. [25] used an opposition-based learning to improve the convergence speed of GSA. Another improvement in the literature of the GSA algorithm in the hybrid PSO and GSA (PSOGSA) [26, 27]. In 2011, GSA was employed as a global search algorithm, accompanied by a combined multi-type local improvement scheme as a local search operator [28]. A novel Immune Gravitation Optimisation Algorithm (IGOA) was proposed by Zhang et al. [29] in 2012, incorporating the characteristics of antibody diversity and vaccination to solve the slow convergence. Hatamlou et al. [30] 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 2014, a *gbest*-guided version of the GSA algorithm was proposed to improve its exploitation and convergence [31].

All these studies show that GSA may face degraded exploration or exploitation in solving problems. These problems occur in the BGSA algorithm as well because BGSA utilises the same concepts as the GSA algorithm. There appears to be no published study on improving exploration and/or exploitation (performance in general) of the BGSA algorithm. Therefore, this paper can also be considered as a first attempt in this regard, although the main focus is on transfer functions.

2.2 Particle Swarm Optimisation

The PSO algorithm was first proposed by Kennedy and Eberhart [32] as a heuristic optimisation algorithm. It mimics the individual and social intelligence (behaviour) of a swarm of birds or school of fish when navigating or foraging. The PSO algorithm employs a set of artificial

particles with two vectors: velocity (v) and position (x). During optimisation, particles update their vectors based on the following two rules:

- Each particle in PSO updates the velocity vector with respect to its previous velocity, the distance to its personal best solution (*pbest*), and the distance to the best solution the swarm has obtained:

$$(gbest)\left(v_i^{t+1} = wv_i^t + c_1 \times rand \times \left(pbest_i - x_i^t\right) + c_2 \times rand \times \left(gbest - x_i^t\right)\right)$$

- Each particle in PSO updates the position vector based on its previous position and the current velocity: $\left(x_i^{t+1} = x_i^t + v_i^{t+1}\right)$.

where v_i^t is the velocity of particle i at iteration t, w is a weighting function, c_j is a weighting factor, *rand* is a random number between 0 and 1, x_i^t is the current position of particle i at iteration t, $pbest_i$ is the *pbest* of agent i at iteration t, and *gbest* is the best solution so far.

The PSO algorithm has been very popular in the literature of heuristic algorithms [33–36]. The binary version of this algorithm has been used in many theoretical and practical studies as well [37–39].

Gravitational Search Algorithm and PSO both belong to the family of heuristic algorithms. However, as can be seen, they perform optimisation with different mechanisms. The GSA algorithm employs a set of masses that interact with each other to reach a stable situation. The PSO algorithm utilises the concepts of swarm intelligence to drive particles towards the optimum. We chose these two algorithms because they have different mechanisms, while both need transfer functions to perform binary optimisation.

3 Discrete (binary) optimisation

The concepts of optimisation in a binary search space are different from those for continuous search space. A binary search space can be considered as a hypercube. The search agents of heuristic algorithms have to jump to nearer and farther corners of this hypercube. A jump is usually performed by flipping various numbers of bits. Without loss of generality, binary optimisation can be formulated as a minimisation problem as follows:

$$\text{Minimise}: \quad f(x_1, x_2, x_3, \ldots, x_{n-1}, x_n) \tag{3.1}$$

$$\text{Subject to}: \quad g_i(x_1, x_2, x_3, \ldots, x_{n-1}, x_n) \geq 0, \quad i = 1, 2, \ldots, m \tag{3.2}$$

$$h_i(x_1, x_2, x_3, \ldots, x_{n-1}, x_n) = 0, \quad i = 1, 2, \ldots, p \tag{3.3}$$

$$x_i \in \{0, 1\} \quad i = 1, 2, \ldots, n \tag{3.4}$$

where n is number of variables, m indicates the number of inequality constraints, and p shows the number of equality constraints

In the velocity-based algorithms such as GSA, PSO, MOA, and BA algorithms, the search agents can move throughout the search space by updating their position vectors. The final position is calculated by the addition of the previous position and the current velocity. However, this process is no longer applicable for a binary search space. The concept of velocity becomes different in binary optimisation, where it defines the probability of flipping the elements of a position vector. The process of mapping velocity values to probability values is performed by a new component called the transfer function. The first transfer function was proposed by Kennedy and Eberhart [20]. This transfer function was as follows:

$$T(x) = \frac{1}{1 + e^{-v_i^k(t)}} \tag{3.5}$$

where $v_i^k(t)$ indicates the velocity of i-th particle at the t-th iteration in the k-th dimension.

It may be observed that $T(x)$ maps all real values of velocity vectors to probability values in the interval [0, 1].

Another change made by Kennedy and Eberhart to design discrete PSO was the position updating formula, as follows:

$$x_i^k(t+1) = \begin{cases} 1 & \text{If } rand < T\left(v_i^k(t+1)\right) \\ 0 & \text{If } rand \geq T\left(v_i^k(t+1)\right) \end{cases} \tag{3.6}$$

In 2009, Rashedi et al. [21] employed a tangent hyperbolic function as the transfer function component for the GSA algorithm:

$$T\left(v_i^k(t)\right) = |\tanh(v_i^k(t))| \tag{3.7}$$

where $v_i^k(t)$ indicates the velocity of the i-th mass at the t-th iteration in the k-th dimension.

After converting velocity values to probability values, position vectors were also modified as follows [21]:

$$x_i^k(t+1) = \begin{cases} \left(x_i^k(t)\right)^{-1} & \text{If } rand < T\left(v_i^k(t+1)\right) \\ x_i^k(t) & \text{If } rand \geq T\left(v_i^k(t+1)\right) \end{cases} \tag{3.8}$$

In both of the transfer functions, high velocity implies high probability of changing position. In other words, high velocity indicates that a search agent is far from the best search agent(s), so it has to flip bits in its position vector with the hope of reaching promising regions of the search space on the way towards the best search agent(s) (exploration). In contrast, small values of velocity show that a search agent needs to stay in its current position and try to search nearby (exploitation) [40].

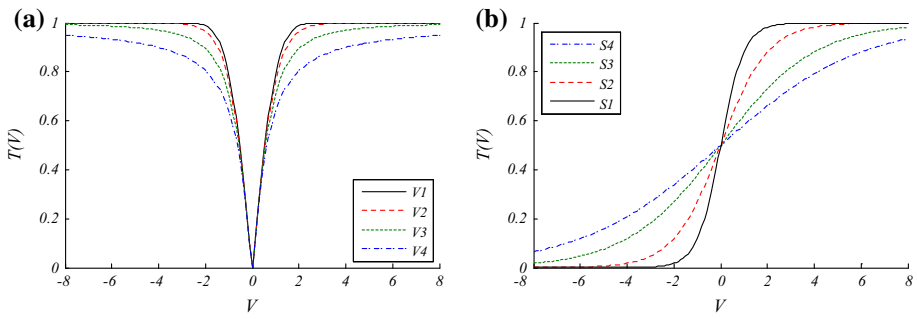

Fig. 1 **a** v-shaped and **b** s-shaped family of transfer functions [23]

The general steps of both BGSA and BPSO are as follows:

A. Initialise a random set of search agents
B. Repeat steps C–E until the satisfaction of the termination criterion
C. Calculate velocities for all search agents
D. Define probabilities by *T(x)*
E. Update position vectors according to the probabilities calculated in step D

As may be seen in the discussion, the transfer function is the main component of these velocity-based algorithms. In the following subsection, the effects of different transfer functions on improving the performance of BGSA and BPSO are investigated.

Table 1 V-shaped and s-shaped transfer functions [23]

	Mathematical formulation
V-shaped transfer functions	
V1 [23]	$T(x) = \left\lvert \operatorname{erf}\left(\frac{\sqrt{\pi}}{2}x\right)\right\rvert = \left\lvert \frac{\sqrt{2}}{\pi}\int_0^{\frac{\sqrt{\pi}}{2}x} e^{-t^2}dt\right\rvert$
V2 [20]	$T(x) = \lvert\tanh(x)\rvert$
V3 [23]	$T(x) = \left\lvert \frac{x}{\sqrt{1+x^2}}\right\rvert$
V4 [23]	$T(x) = \left\lvert \frac{2}{\pi}\arctan\left(\frac{\pi}{2}x\right)\right\rvert$
S-shaped transfer functions	
S1 [23]	$T(x) = \frac{1}{1+e^{-2x}}$
S2 [19]	$T(x) = \frac{1}{1+e^{-x}}$
S3 [23]	$T(x) = \frac{1}{1+e^{\frac{-x}{2}}}$
S4 [23]	$T(x) = \frac{1}{1+e^{\frac{-x}{3}}}$

4 Transfer functions

The transfer function is the main component of the BGSA and BPSO algorithms, so special consideration should be given to this component. In this section, 8 transfer functions [23] are utilised for improving the performance of both algorithms. Moreover, the importance of a transfer function as an extremely cheap operator to improve the exploration and exploitation of BGSA and BPSO will also be demonstrated. According to Rashedi et al. [21], there are some points to consider in choosing a function, as follows:

First, the range of a transfer function should lie in the interval [0, 1] because it provides the probability values. Second, a transfer function should provide a high probability for a large absolute value of the velocity. Those search agents with large absolute values are probably far from the best solution, so they should switch their positions in the next iteration. Third, a transfer function should provide a small probability for a small absolute value of the velocity. Fourth, the return value of a transfer function should increase with increments of velocities. Those search agents that are moving away from the best solution should have a higher probability of changing their position vectors in order to return to their previous positions. Fifth, the return value of a transfer function should decrease as the velocity reduces.

These characteristics will guarantee a transfer function capable of mapping continuous spaces to binary spaces while preserving similar concepts of the search for a particular evolutionary algorithm. The original BGSA utilises the transfer function presented in Eq. (3.7). This transfer function, called V2, is depicted in Fig. 1a. The mathematical formulation is also available in Table 1. We also employ another three transfer functions with different slopes and saturations compared to V2 in order to investigate the efficiency of these characteristics in improving the performance of heuristic algorithms. Due to the shapes of these curves, they were named v-shaped transfer functions [23].

The proposed transfer function (S2) for BPSO in [20] is also presented in Fig. 1 and Table 1. We again also employ three other transfer functions with different slopes and saturations compared to S2 in order to investigate the efficiency of these characteristics in improving the performance of heuristic algorithms. Due to the shapes of these curves, they were named s-shaped transfer functions [23].

5 Experimental results and discussion

The proposed transfer functions were compared on 17 test functions [41–45]. These benchmark functions are divided into two groups: unimodal and multimodal. We have chosen unimodal and multi-modal test functions to be able to benchmark exploration and exploitation of BGSA and BPSO. Unimodal test functions have single optima, so exploitation is very important. However, multi-modal test functions have many local solutions, and an algorithm needs more exploration capability to be effective on this type of test functions. Tables 2 and 3 list these functions, where *Dim* indicates the dimension of the function, *Range* shows the bounds of the function's search space, and $f_{\min}$ is the global optimum of the function. Figures 2 and 3 illustrate the shape of these test functions.

To represent each continuous variable in binary, 15 bits were used. It should be noted that one bit is reserved for the sign. Therefore, the dimension of the particles is calculated as $Dim_{\text{Particle}} = Dim_{\text{Function}} \times 15$ where Dim_{particle} indicates the dimension of each search agent and Dim_{function} is the dimension of the test functions. Thus, the dimensionality of the particles is 75 (i.e. 5×15) for all test functions.

The initial parameters of BGSA were defined as follows:

The number of masses was 30; α was set to 20; the gravitational constant (G_0) was set to 1; the initial velocities are randomly generated in the interval [0, 1]; the initial values of acceleration and mass were set to 0 for each mass; the maximum number of iterations was set to 500; and the stopping condition was satisfied when the maximum number of iterations was reached.

The initial parameters of BPSO were defined as follows:

The number of particles was 30, C_1 and C_2 were set to 2, W was linearly decreased from 0.9 to 0.4, maximum iterations was set to 500, and the stopping criteria were met when the maximum number of iterations was achieved.

Note that the maximum velocities were set to 6 to prevent both algorithms from overshooting the boundaries and the source codes can be found in http://www.alimirjalili.com/Projects.html or http://www.mathworks.com.au/matlabcentral/fileexchange/authors/184390.

5.1 Results of the BGSA algorithm

We developed BGSA with all the transfer functions in Table 1 and named them BGSA1 to BGSA8. Note that BGSA1 to BGSA4 use S1 to S4, whereas BGSA5 to BGSA8 use V1 to V4. The algorithms were run 30 times. The average (AVE) and standard deviation (STD) of the best-so-far solution in the last iteration are reported in Tables 4 and 5. Note that the best results are highlighted in bold face.

Table 4 and Fig. 4 show the statistical results for unimodal test functions. Table 4 shows that the results of BGSA5 to BGSA8 are better than BGSA1 to BGSA4 in general. Since unimodal test functions do not have local minima, it can be stated that v-shaped transfer functions are better than s-shaped transfer functions in terms of exploitation. Among v-shaped transfer functions, V1 (BGSA5) shows the best results in 43 % of the test functions as shown in Fig. 4. Considering the slope and saturation of V1 in Fig. 1, V1 provides the highest probability values meaning that this algorithm updates its masses more frequently compared to V2, V3, and V4. Although this mechanism theoretically promotes exploration, these results show that it can also improve exploitation.

The convergence curves in Fig. 5 also support this statement. It can be observed that BGSA algorithms with v-shaped transfer functions have the fastest convergence rates compared to s-shaped transfer functions. The discrepancy is considerable between the two groups, due to the improved exploitation of BGSA utilising v-shaped transfer functions. VGSA5 is the best in terms of convergence rate for unimodal test functions.

The experimental results for multimodal test functions are provided in Table 5, and Figs. 6, 7. According to Table 5 and Fig. 6, the results of v-shaped transfer functions are much better than s-shaped transfer functions. Note that multimodal benchmark functions have a massive

Table 2 Unimodal test functions

Function	Dim	Range	$f_{\min}$
$f_1(x) = \sum_{i=1}^{n} x_i^2$	5	[−100, 100]	0
$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	5	[−10, 10]	0
$f_3(x) = \sum_{i=1}^{n} \left(\sum_{j-1}^{i} x_j \right)^2$	5	[−100, 100]	0
$f_4(x) = \max_i \{ \lvert x_i \rvert, 1 \le i \le n \}$	5	[−100, 100]	0
$f_5(x) = \sum_{i=1}^{n-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right]$	5	[−30, 30]	0
$f_6(x) = \sum_{i=1}^{n} (\lvert x_i + 0.5 \rvert)^2$	5	[−100, 100]	0
$f_7(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$	5	[−1.28, 1.28]	0

Table 3 Multimodal test functions

Function	Dim	Range	f_{min}
$f_8(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	5	$[-500, 500]$	-418.9829×5
$f_9(x) = \sum_{i=1}^{m} \left[x_i^2 - 10cos(2\pi x_i) + 10\right]$	5	$[-5.12, 5.12]$	0
$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	5	$[-32, 32]$	0
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	5	$[-600, 600]$	0
$f_{12}(x) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{x_i+1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	5	$[-50, 50]$	0
$f_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	5	$[-50, 50]$	0
$f_{14}(x) = -\sum_{i=1}^{n}\sin(x_i).\left(\sin\left(\frac{i.x_i^2}{\pi}\right)\right)^{2m}, \quad m = 10$	5	$[0, \pi]$	-4.687
$f_{15}(x) = \left[e^{-\sum_{i=1}^{n}(x_i/\beta)^{2m}} - 2e^{-\sum_{i=1}^{n} x_i^2}\right].\prod_{i=1}^{n}\cos^2 x_i, \quad m = 5$	5	$[-20, 20]$	-1
$f_{16}(x) = \left\{\left[\sum_{i=1}^{n}\sin^2(x_i)\right] - \exp\left(-\sum_{i=1}^{n} x_i^2\right)\right\}.\exp\left[-\sum_{i=1}^{n}\sin^2\sqrt{\lvert x_i \rvert}\right]$	5	$[-10, 10]$	-1
$f_{17}(x) = -\sum_{n=0}^{10} 0.5^n\cos(3^n(x + 0.5))$	5	$[-1, 1]$	0

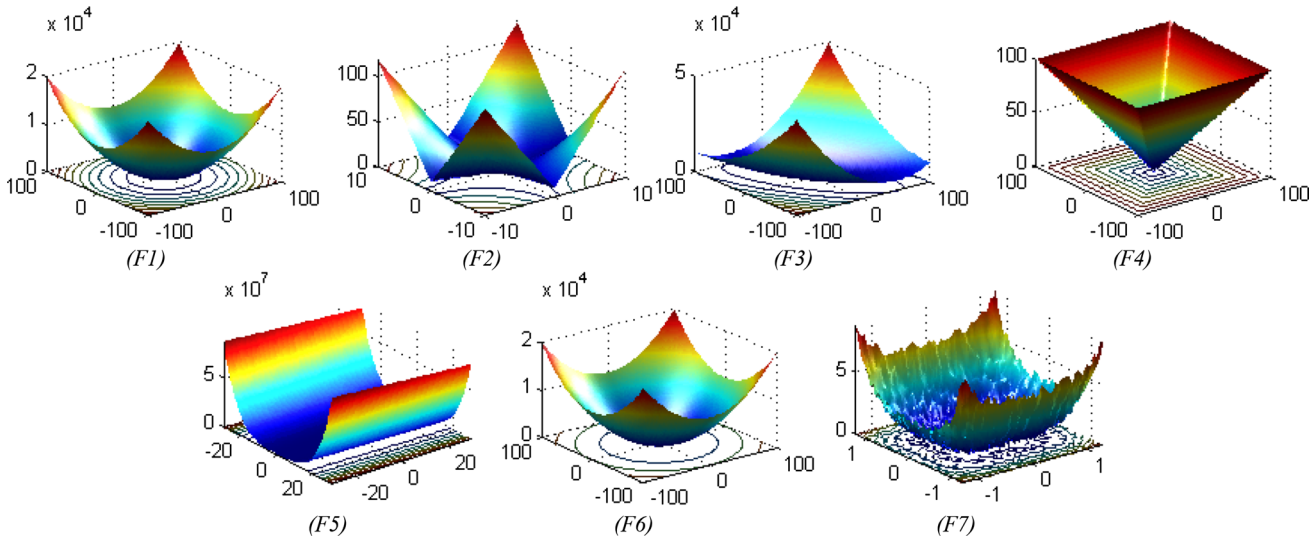

Fig. 2 Search space of unimodal benchmark functions

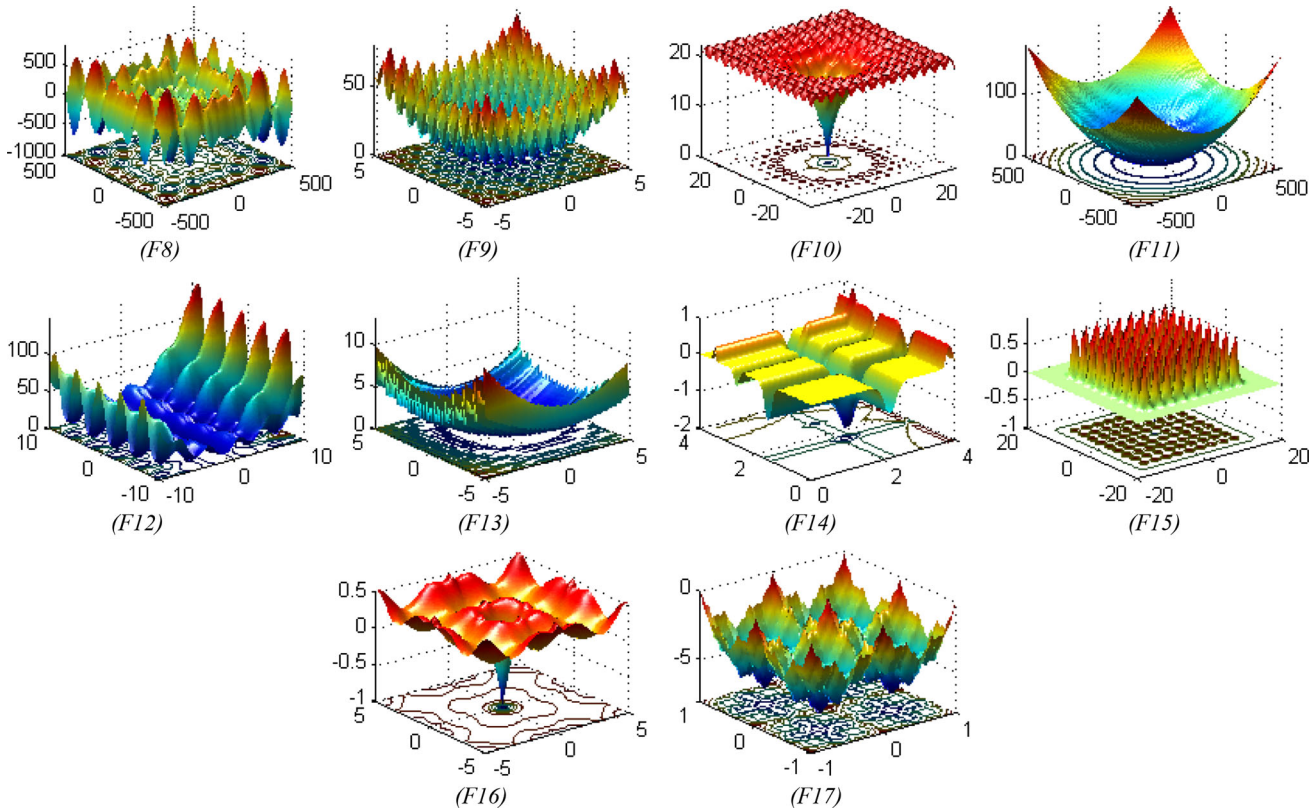

Fig. 3 Search space of multimodal benchmark functions

number of local optima, so they are suitable to examine the exploration of an algorithm. Therefore, the results show that v-shaped transfer functions boost exploration. Among v-shaped transfer functions, V1 again provides the best results on 50 % of multimodal test functions, as Fig. 6 shows, followed by V2. This is due to the nature of multimodal benchmark functions in that we need to emphasise exploration rather than exploitation to avoid local optima.

Table 4 Minimisation results of unimodal test functions over 30 independent runs

Test function		BGSA1	BGSA 2	BGSA 3	BGSA 4	BGSA 5	BGSA 6	BGSA 7	BGSA 8
F1	AVE	554.5374	562.86	575.2497	773.7391	**74.0849**	102.0933	100.3393	113.9429
	STD	218.7025	161.3689	289.4971	294.3383	**48.72593**	74.91857	59.99097	80.68731
F2	AVE	2.829860	2.992659	2.590461	2.77131	**1.210225**	1.325269	1.328534	1.335169
	STD	0.716840	0.615841	0.583805	0.676465	**0.455553**	0.672770	0.470895	0.615285
F3	AVE	596.1014	615.4183	527.3935	643.1609	507.6472	509.0988	569.3689	**376.2232**
	STD	209.9154	201.0591	264.3618	316.2335	395.3808	266.3714	410.7530	**165.4917**
F4	AVE	15.90938	14.26094	16.83281	15.48984	8.465625	**7.999219**	8.038281	8.718750
	STD	2.442651	3.279725	3.186179	5.571737	4.109440	**3.450794**	3.658126	3.991298
F5	AVE	13,761.82	14,400.39	20,452.46	13,155.96	5,226.022	**2,620.465**	4,755.673	3,131.852
	STD	11,569.57	12,652.49	21,377.46	14,024.48	9,094.784	**1,735.901**	3,747.255	2,119.071
F6	AVE	542.2519	637.9138	629.3702	532.2097	92.09465	126.6157	**67.86088**	119.6531
	STD	152.6036	326.5747	288.6595	270.3292	48.46113	88.03597	**66.59637**	97.69142
F7	AVE	0.116317	0.170230	0.212952	0.223447	**0.016811**	0.023861	0.028402	0.025697
	STD	0.058258	0.076526	0.132488	0.090189	0.013225	0.026880	0.031109	0.027532

Table 5 Minimisation results of multimodal test functions over 30 independent runs

Test function		BGSA1	BGSA 2	BGSA 3	BGSA 4	BGSA 5	BGSA 6	BGSA 7	BGSA 8
F8	AVE	−813.835	−806.584	−842.984	−813.547	**−871.659**	−828.602	−856.619	−819.402
	STD	44.5023	41.32977	54.41304	28.13047	**52.20139**	42.63769	78.06458	69.59389
F9	AVE	24.1654	22.45271	21.3143	24.84155	9.694211	**5.999694**	10.57055	9.589724
	STD	3.61862	5.591833	4.825795	3.633672	2.565838	**2.963102**	4.815394	4.458082
F10	AVE	10.18744	9.729899	9.835851	10.30257	**2.338205**	2.947044	3.184335	3.054951
	STD	0.902716	1.178572	2.216132	1.040921	**0.855805**	1.481999	0.77957	0.557956
F11	AVE	1.476758	1.538274	1.475263	1.747154	0.771398	**0.647846**	0.702132	0.753869
	STD	0.250373	0.331764	0.229607	0.275921	0.233634	**0.228547**	0.195358	0.177457
F12	AVE	114.6318	40.90543	70.65441	829.5176	**7.850254**	12.66839	8.936504	11.26534
	STD	184.8846	31.30786	103.1468	1701.529	**3.629425**	8.774814	4.113277	12.21871
F13	AVE	26,171.49	34,522.57	42,406.55	52,085.44	**124.039**	922.4907	4,463.524	857.3561
	STD	33,495.67	52,011.32	62,908.07	84,165.35	**234.5106**	2,458.653	12,522.49	2,274.657
F14	AVE	**−3.08159**	−2.98982	−2.70403	−2.81342	−2.65212	−2.494	−2.40796	−2.67405
	TD	**0.41485**	0.287082	0.184736	0.254568	0.257559	0.126732	0.165367	0.276908
F15	AVE	−5E−117	**−7E−110**	−5E−118	−7E−120	−3E−112	−1E−112	−4E−113	−1E−112
	STD	1.2E−116	**2.1E−109**	1.6E−117	1.9E−119	4.5E−112	3.1E−112	1.2E−112	2E−112
F16	AVE	**−0.02573**	0.003504	0.002343	0.002569	0.001401	0.001947	0.001965	0.002258
	STD	**0.092386**	0.001559	0.000809	0.001186	0.000692	0.00113	0.000997	0.000922
F17	AVE	−17.0267	−16.7994	−16.8472	−16.7358	**−17.3621**	−17.1638	−17.2648	−17.2042
	STD	0.441468	0.20924	0.435992	0.344902	0.583757	0.594186	0.712621	0.547962

Figure 7 shows the convergence of the algorithms dealing with multimodal test functions. The curves of this figure demonstrate the fast convergence behaviour of v-shaped transfer functions, especially V1. It can clearly be seen that there are significant discrepancies between s-shaped and v-shaped transfer functions in most of the test functions. The V1 transfer function has the fastest convergence rate in most of the cases, showing the ability of this transfer function in terms of improved exploitation.

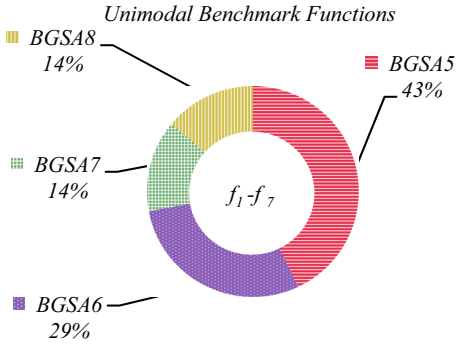

Fig. 4 Statistical results for unimodal benchmark functions

To summarise, statistically speaking the BGSA5 algorithm has the best results in 46 % of the benchmark functions as shown in Fig. 8. Therefore, these results show that the proposed V1 transfer function could successfully improve exploration and exploitation of BGSA. It is worth mentioning that this success has been achieved without any extra computational burden for the BGSA algorithm. Moreover, these results indicate how important the role of the transfer function is in BGSA, since by selecting a suitable function the performance of BGSA can be increased remarkably.

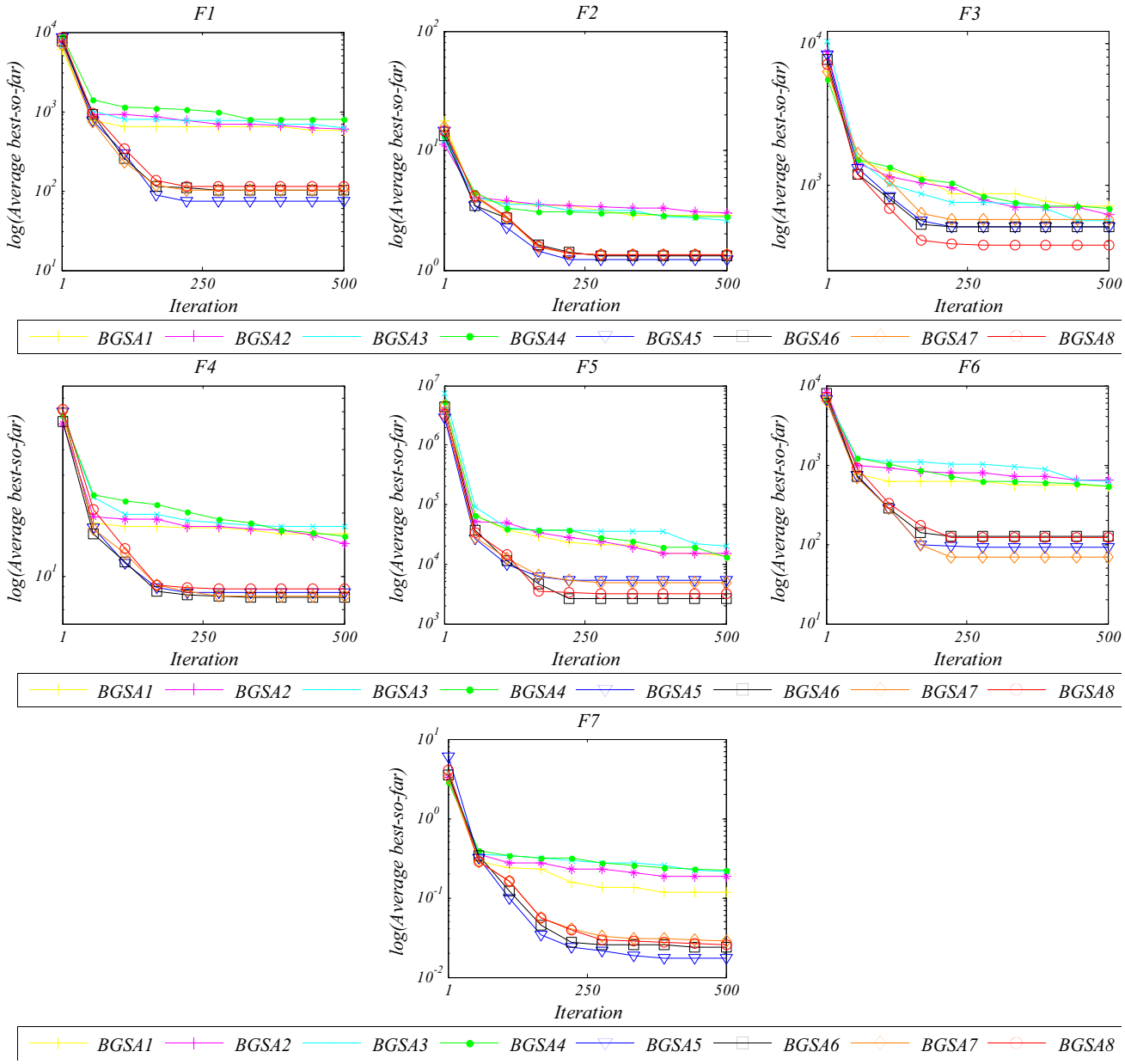

Fig. 5 Convergence curves of BGSA algorithms on unimodal functions

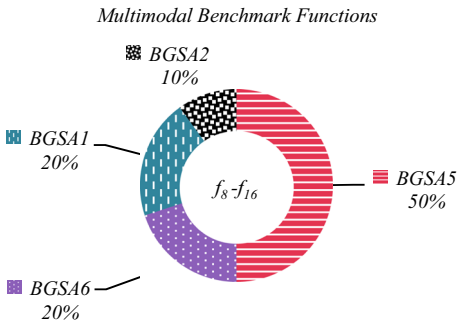

Fig. 6 Statistical results for multimodal benchmark functions

5.2 Results of the BPSO algorithm

We also developed BPSO with all the transfer functions in Table 1 and named them BPSO1 to BPSO8. Note that BPSO1 to BPSO4 use S1 to S4, whereas BPSO5 to BPSO8 use V1 to V4. The algorithms were run 30 times. The average and standard deviation of the best-so-far solution in the last iteration are reported in Tables 6 and 7. Note that the best results are highlighted in bold face.

The results of the BPSO algorithms on the unimodal test functions in Table 6 show that BPSO with v-shaped transfer functions mostly outperforms those with s-shaped transfer functions. The same behaviour can be observed in the results of multi-modal test functions in Table 7, in which BPSO 8 outperforms other algorithms in the majority of test functions. Among the BPSO algorithms with s-shaped transfer functions, S1 (in BPSO1) shows the best results, but other s-shaped transfer functions show the worst results.

These results are consistent with those of BGSA, showing the v-shaped transfer functions are more suitable for heuristic algorithms. However, it is interesting that V1 shows the best results in BGSA, whereas V4 provides the best results in BPSO. This strongly suggests that transfer functions may have different impacts on different heuristic algorithms. The convergence curves of BPSO algorithms in Figs. 9 and 10 show that v-shaped transfer functions accelerate the convergence as well. Comparing convergence of BPSO and BGSA, it may be seen that transfer functions can also provide different convergence behaviour for various algorithms.

In summary, the results show that s-shaped transfer functions are much less effective than the v-shaped transfer functions. This is due to the mechanism for updating position when using an s-shaped transfer function in Eq. (3.6), in which the search agents are assigned 0 or 1. This mandatory assignment may not be effective in different problems. For instance, if a variable of a search agent is 1 and the velocity is high, there should be a high possibility for the variable to become 0. However, Eq. (3.6) requires this variable stays in 1 with a very high probability. This can significantly degrade exploration of a heuristic algorithm.

In contrast to s-shaped test functions, v-shaped transfer functions tend more often to form the complement of variables using Eq. (3.8). This mechanism promotes and guarantees changing the position of search agents proportional to their velocities. This is the main reason for the superiority of v-shaped test functions.

The results also show that V1 is the best transfer function for BGSA, while V4 better improves the performance of BPSO. The V1 transfer function is saturated earlier than V2, V3, and V4. This causes it to return more values close to 1, so the first condition in Eq. (3.8) is satisfied more often than the second condition, and the search agents are moved more frequently. This promotes exploration of the search space, which is the main reason for the superior results of BGSA8. Considering the performance of algorithms BGSA5 to BGSA8, the results are progressively less degraded from BGSA5 to BGSA8. Considering the shape of V1 to V4 and this behaviour, it can be concluded the v-shaped transfer functions show lower exploration from V1 to V4.

The results of the BPSO algorithm are progressively more degraded from BPSO4 to BPSO8, which is exactly the opposite to those of BGSA algorithms. Considering Eq. (3.8), it may be concluded that the transfer functions that are saturated earlier are more suitable for BPSO algorithms. This is due to the high probability of satisfying the second condition in Eq. (3.8), especially when using V4, in which case the variables tend to maintain their positions more frequently. This promotes exploitation, which assists BPSO8 to outperform BPSO5 to BPSO7. The accelerated convergence observed of v-shaped BPSO algorithms is also due to this reason.

Considering the effect of the slope and saturation of s-shaped transfer function on exploration and exploitation, the results show that S1 performs better for both BGSA and BPSO compared to other s-shaped transfer functions. However, specifics of exploration and exploitation cannot be discussed due to the mechanism of position updating in Eq. (3.6). For the test problems investigated, it seems that a transfer function performs well when it assigns 1 to variables. However, this cannot be extended since such a behaviour may be reversed if we swap the conditions of Eq. (3.6). This can also be considered as another drawback of s-shaped transfer functions.

In contrast, the v-shaped transfer functions show high exploration if they become saturated early because they tend to change the variables of search agents more

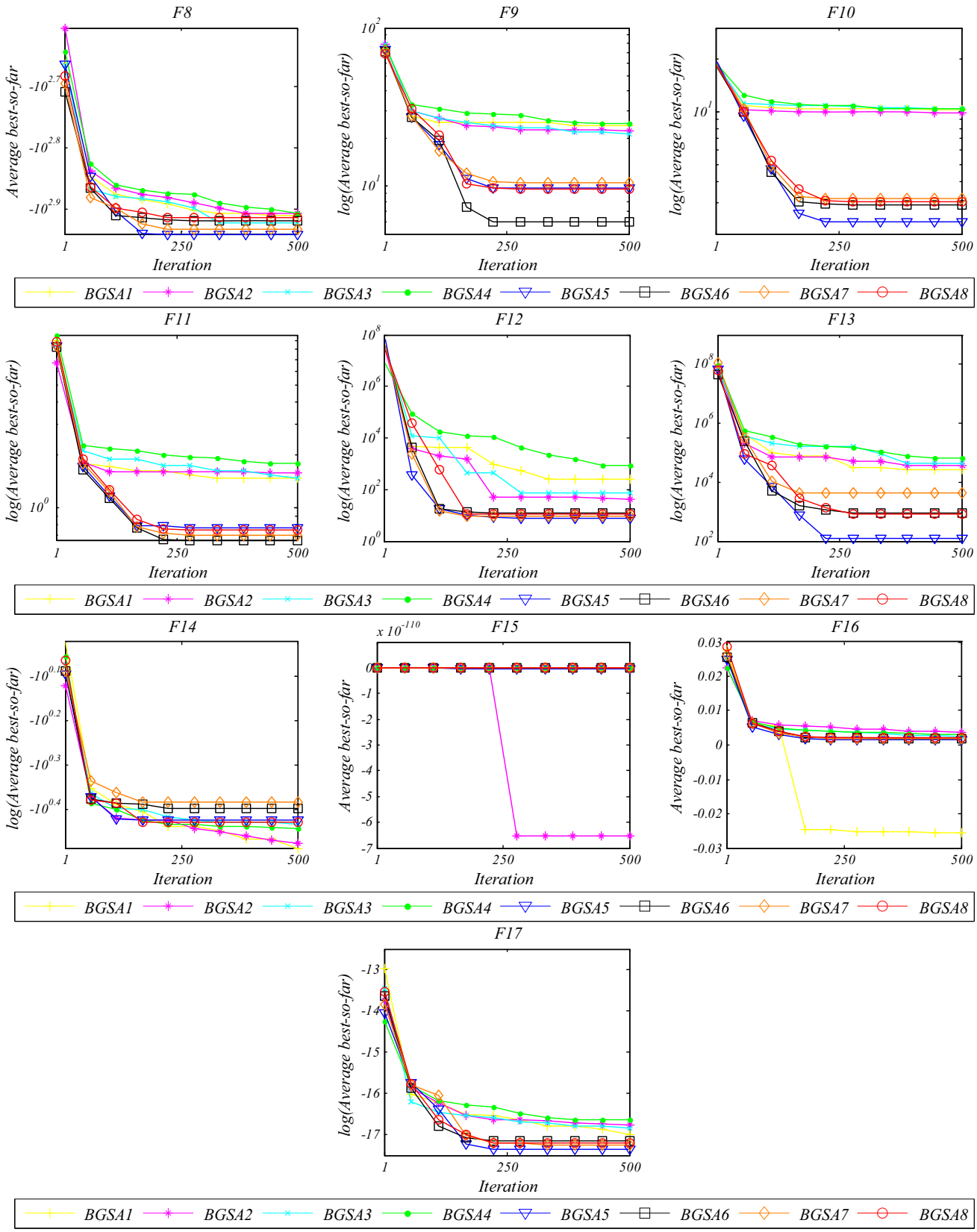

Fig. 7 Convergence curves of BGSA algorithms on multimodal benchmark functions

frequently. Exploration is decreased proportional to the saturation of the v-shaped test functions; V1 shows the highest exploration and V4 provides the highest exploitation. The results of both algorithms show that the selection of transfer functions should be based on the mechanism of a heuristic algorithm and should be considered carefully when designing a discrete heuristic algorithm.

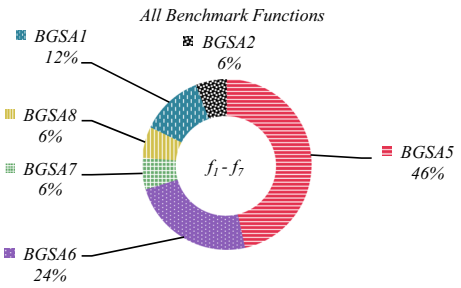

Fig. 8 Statistical results for all benchmark functions

6 Conclusion

This work specifically investigated transfer functions as the main component of velocity-based heuristic algorithms. We employed eight different mathematical functions to investigate their effectiveness as transfer functions for BGSA and BPSO, two algorithms with different mechanisms. The transfer functions were chosen with diverse saturations and classified into two groups: s-shaped and v-shaped. A test bed consisting of 17 test functions benchmarked the performance of BGSA and BPSO algorithms equipped with the new transfer functions. The results showed that v-shaped transfer functions (especially with the earliest saturation) was able to promote

Table 6 Statistical results of the BPSO algorithms on unimodal benchmark functions over 30 independent runs

Test function		BPSO1	BPSO2	BPSO3	BPSO4	BPSO5	BPSO6	BPSO7	BPSO8
F1	AVE	0.4921	5.2965	33.3306	71.0918	0.7844	0.3514	0.2663	**0.0977**
	STD	0.4165	2.7657	17.0770	37.9921	2.9905	0.8997	0.7691	**0.2725**
F2	AVE	0.6973	0.2292	0.5558	0.9278	0.0273	0.0213	0.0166	**0.0081**
	STD	0.0300	0.0938	0.1637	0.2347	0.0537	0.0243	0.0258	**0.0122**
F3	AVE	2.7005	14.4901	66.8813	112.7035	18.6627	**1.9228**	2.3975	2.1282
	STD	1.9069	9.6482	33.9296	61.6797	39.6152	**3.0689**	6.8512	6.5787
F4	AVE	0.6917	2.0422	4.8242	6.9307	0.7956	1.0539	0.4133	**0.3146**
	STD	0.3195	0.7229	1.4989	1.6784	1.1958	1.7647	0.7719	**0.4294**
F5	AVE	20.2467	65.3304	437.3886	744.8748	26.7199	21.1831	14.8538	**7.3941**
	STD	26.1441	79.3444	347.783	623.8992	45.3441	34.2671	33.188	**19.4263**
F6	AVE	0.5518	5.4924	37.6002	71.0657	0.1388	0.3225	0.2508	**0.1186**
	STD	0.3831	3.0702	19.5233	37.3071	0.1939	0.7496	0.5769	**0.4174**
F7	**AVE**	0.0076	0.0109	0.0267	0.0372	0.0039	0.0054	0.0039	**0.0023**
	STD	0.0038	0.0063	0.0163	0.0258	0.0024	0.0052	0.0041	**0.0014**

Table 7 Statistical results of the BPSO algorithms on multimodal benchmark functions over 30 independent runs

Test function		BPSO1	BPSO2	BPSO3	BPSO4	BPSO5	BPSO6	BPSO7	BPSO8
F8	AVE	−996.382	−992.878	−964.879	−934.203	−958.486	−953.118	**−1,000.48**	−991.784
	STD	17.56417	14.8197	29.1603	36.61892	45.1939	51.1077	**16.1427**	30.278
F9	AVE	2.2054	4.1528	8.2714	12.4642	1.9923	2.0495	1.8899	**1.6945**
	STD	1.0149	1.6081	3.3343	3.2046	0.8618	0.8105	1.2806	**0.7212**
F10	AVE	0.5554	2.23	3.6516	5.1836	1.0516	0.9738	0.5097	**0.2393**
	STD	0.2922	0.4583	0.632	1.145	1.1293	0.9998	0.776	**0.5819**
F11	AVE	0.2598	0.3985	0.6403	0.7989	0.1684	0.1762	0.1351	**0.1032**
	STD	0.0868	0.1088	0.1612	0.1465	0.1068	0.1016	0.0904	**0.0655**
F12	AVE	0.0432	0.3603	2.2328	5.3827	0.1058	0.0068	0.0879	**0.02698**
	STD	0.0404	0.3355	1.1696	1.6694	0.2209	0.0161	0.2218	**0.1139**
F13	**AVE**	0.0626	0.3361	1.3827	4.4317	0.0706	0.0349	0.0510	**0.01178**
	STD	0.0311	0.1873	0.8874	2.9733	0.1721	0.0483	0.1699	**0.0321**

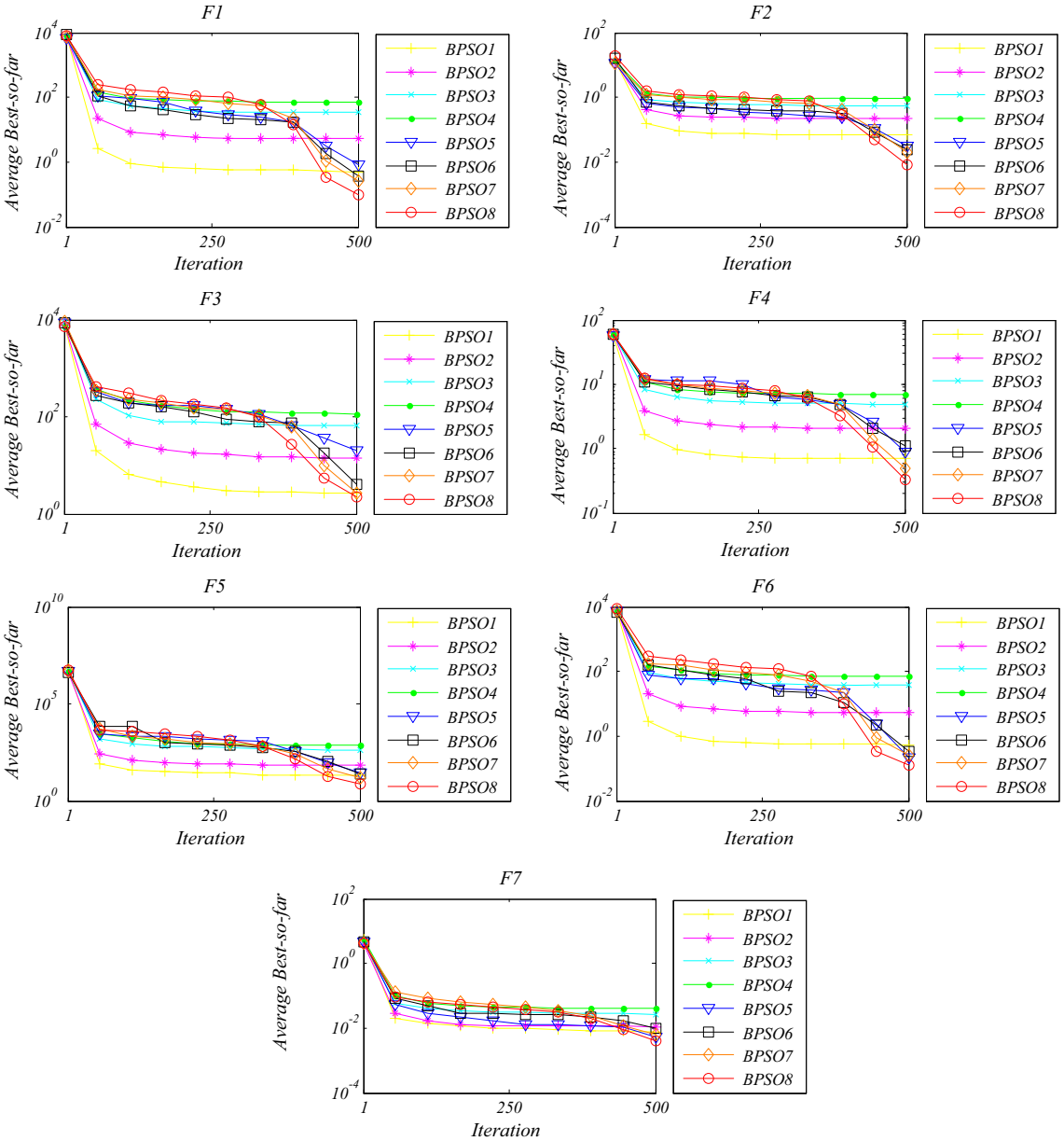

Fig. 9 Convergence curves of BPSO algorithms on unimodal benchmark functions

exploration of the BGSA algorithm. In addition, the results showed that the exploitation of the BPSO algorithm is improved by v-shaped transfer function with the latest saturation. Convergence curves confirmed accelerated convergence behaviour of v-shaped BPSO algorithms as well.

The findings demonstrated the significant impact that a transfer function may have on exploration, exploitation, and convergence behaviour. Therefore, a transfer function should be selected very carefully for binary algorithms because failure to do so may significantly degrade performance.

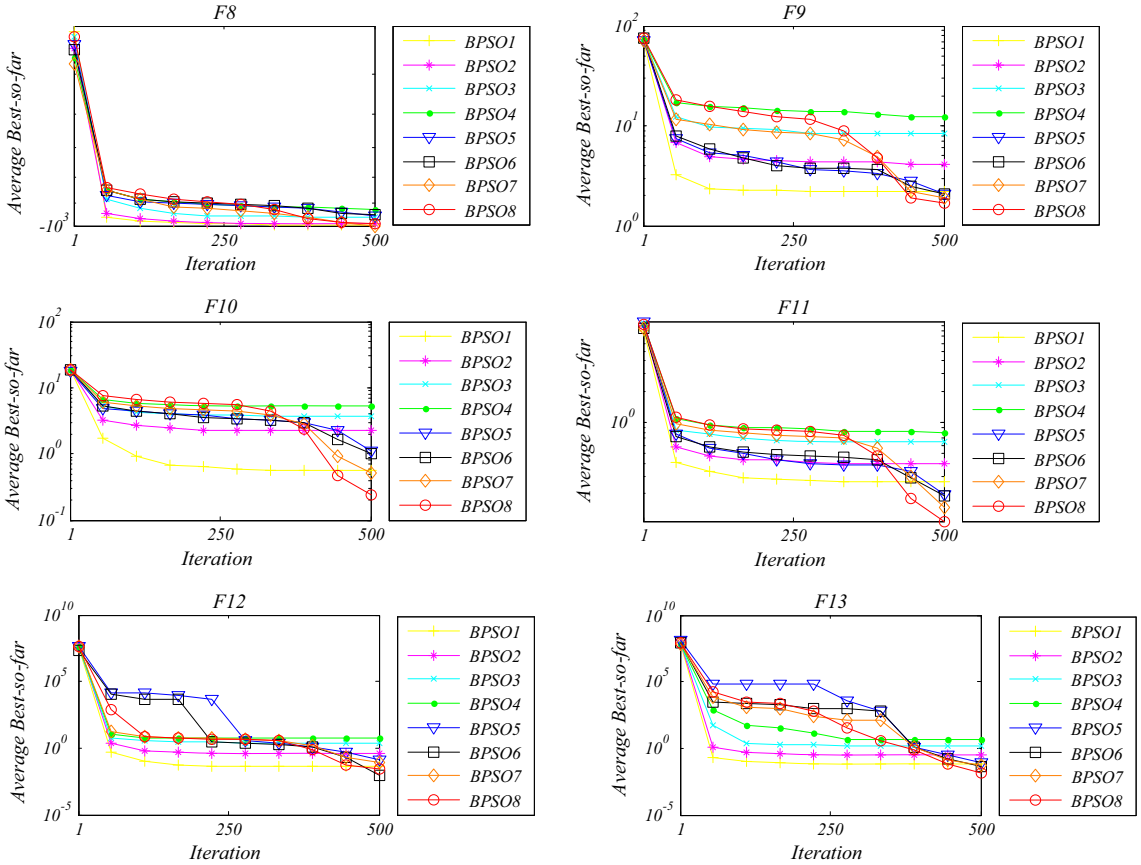

Fig. 10 Convergence curves of BPSO algorithms on multimodal benchmark functions

For future study, v-shaped algorithms could be employed for solving real problems with the purpose of benchmarking the performance of such algorithms in practice. In addition, the insights gained into their impact on algorithm behaviour may allow the choice of other mathematical functions worth researching as transfer functions.

References

1. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of the IEEE international conference on neural networks, Perth, pp 1942–1948
2. Robinson J, Rahmat-Samii Y (2004) Particle swarm optimization in electromagnetics. IEEE Trans Antennas Propagon 52:397–407
3. Nagesh Kumar D, Janga Reddy M (2007) Multipurpose reservoir operation using particle swarm optimization. J Water Resour Plan Manag 133:192–201
4. He S, Prempain E, Wu Q (2004) An improved particle swarm optimizer for mechanical design optimization problems. Eng Optim 36:585–605
5. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39:459–471
6. Dorigo M, Di Caro G (1999) Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation, CEC 99
7. Kirkpatrick S, Gellatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
8. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179:2232–2248
9. Holland JH (1992) Genetic algorithms. Sci Am 267:66–72
10. Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12:702–713
11. Rechenberg I (1994) Evolution strategy. Comput Intellect Imitating Life 1
12. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11:341–359
13. Wang L, Pan Q-K, Suganthan P, Wang W-H, Wang Y-M (2010) A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Comput Oper Res 37:509–520
14. Geem ZW, Kim JH, Loganathan G (2001) A new heuristic optimization algorithm: harmony search. Simulation 76:60–68
15. Wang L, Xu Y, Mao Y, Fei M (2010) A discrete harmony search algorithm. In: Li K, Li X, Ma S, Irwin GW (eds) Life system modeling and intelligent computing. Communications in computer and information science, vol 98. Springer, Heidelberg, pp 37–43
16. Tayarani NM, Akbarzadeh TM (2008) Magnetic Optimization Algorithms a new synthesis. In: IEEE Congress on evolutionary computation, 2008. CEC (IEEE world congress on computational intelligence). 2008, pp 2659–2664
17. Mirjalili S, Sadiq AS (2011) Magnetic optimization algorithm for training multi layer perceptron. In: 2011 IEEE 3rd international

conference on communication software and networks (ICCSN), pp 42–46
18. Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: González JR, Pelta DA, Cruz C, Terrazas G, Krasnogor N (eds) Nature inspired cooperative strategies for optimization (NICSO 2010). Studies in computational intelligence, vol 284. Springer, Heidelberg, pp 65–74
19. Mirjalili S, Hashim SZM (2012) BMOA: binary magnetic optimization algorithm. Int J Mach Learn Comput 2:204–208
20. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: IEEE international conference on systems, man, and cybernetics, 1997. Computational cybernetics and Simulation, pp 4104–4108
21. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2010) BGSA: binary gravitational search algorithm. Nat Comput 9:727–745
22. Mirjalili S, Mirjalili SM, Yang X-S (2014) Binary bat algorithm. Neural Comput Appl 25:663–681
23. Mirjalili S, Lewis A (2013) S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization. Swarm Evol Comput 9:1–14
24. 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
25. 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 Electr Power Energy Syst 35:21–33
26. Mirjalili S, Hashim SZM (2010) A new hybrid PSOGSA algorithm for function optimization. In: 2010 international conference on computer and information application (ICCIA), pp 374–377
27. Mirjalili S, Mohd Hashim SZ, Moradian Sardroudi H (2012) Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Appl Math Comput 218:11125–11137
28. Chen H, Li S, Tang Z (2011) Hybrid gravitational search algorithm with random-key encoding scheme combined with simulated annealing. IJCSNS 11:208
29. Zhang Y, Wu L, Zhang Y, Wang J (2012) Immune gravitation inspired optimization algorithm. In: Huang D-S, Gan Y, Bevilacqua V, Figueroa JC (eds) Advanced intelligent computing. Lecture notes in computer science, vol 6838. Springer, Heidelberg, pp 178–185
30. Hatamlou A, Abdullah S, Othman Z (2011) Gravitational search algorithm with heuristic search for clustering problems. In: 2011 3rd conference on data mining and optimization (DMO), pp 190–193
31. Mirjalili S, Lewis A (2014) Adaptive gbest-guided gravitational search algorithm. Neural Comput Appl 1–16. doi:10.1007/s00521-014-1640-y
32. Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, pp 39–43
33. AlRashidi MR, El-Hawary ME (2009) A survey of particle swarm optimization applications in electric power systems. IEEE Trans Evol Comput 13:913–918
34. Mirjalili S, Lewis A, Sadiq AS (2014) Autonomous particles groups for particle swarm optimization. Arabian J Sci Eng 39(6):4683–4697. doi:10.1007/s13369-014-1156-x
35. Song M-P, Gu G-C (2004) Research on particle swarm optimization: a review. In: Proceedings of 2004 international conference on machine learning and cybernetics, pp 2236–2241
36. Wei Y, Qiqiang L (2004) Survey on Particle Swarm Optimization Algorithm. Eng Sci 5:87–94
37. Khanesar MA, Teshnehlab M, Shoorehdeli MA (2007) A novel binary particle swarm optimization. In: Mediterranean conference on control and automation, 2007. MED'07, pp 1–6
38. Chuang L-Y, Chang H-W, Tu C-J, Yang C-H (2008) Improved binary PSO for feature selection using gene expression data. Comput Biol Chem 32:29–38
39. Sudholt D, Witt C (2008) Runtime analysis of binary PSO. In: Proceedings of the 10th annual conference on genetic and evolutionary computation, pp 135–142
40. Mirjalili S, Wang G-G, Coelho LdS (2014) Binary optimization using hybrid particle swarm optimization and gravitational search algorithm. Neural Comput Appl 1–13. doi:10.1007/s00521-014-1629-6
41. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3:82–102
42. Digalakis J, Margaritis K (2001) On benchmarking functions for genetic algorithms. Int J Comput Math 77:481–506
43. Molga M, Smutnicki C (2005) Test functions for optimization needs. Test Functions for Optimization Needs
44. Yang X-S (2010) Appendix A: test problems in optimization. In: Engineering optimization. Wiley, pp 261–266. doi:10.1002/9780470640425.app1
45. Saremi S, Mirjalili S, Lewis A (2014) Biogeography-based optimisation with chaos. Neural Comput Appl 25(5):1077–1097. doi:10.1007/s00521-014-1597-x