Hindawi
Computational Intelligence and Neuroscience
Volume 2017, Article ID 3235720, 15 pages
https://doi.org/10.1155/2017/3235720

Research Article

Dynamic Inertia Weight Binary Bat Algorithm with Neighborhood Search

Xingwang Huang,[1,2] Xuewen Zeng,[1] and Rui Han[1]

[1]*National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China*
[2]*University of Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Rui Han; hanr@dsp.ac.cn

Received 27 February 2017; Accepted 20 April 2017; Published 28 May 2017

Academic Editor: Michael Schmuker

Binary bat algorithm (BBA) is a binary version of the bat algorithm (BA). It has been proven that BBA is competitive compared to other binary heuristic algorithms. Since the update processes of velocity in the algorithm are consistent with BA, in some cases, this algorithm also faces the premature convergence problem. This paper proposes an improved binary bat algorithm (IBBA) to solve this problem. To evaluate the performance of IBBA, standard benchmark functions and zero-one knapsack problems have been employed. The numeric results obtained by benchmark functions experiment prove that the proposed approach greatly outperforms the original BBA and binary particle swarm optimization (BPSO). Compared with several other heuristic algorithms on zero-one knapsack problems, it also verifies that the proposed algorithm is more able to avoid local minima.

1. Introduction

There are many optimization problems with binary search space. And many of them are high dimensional. Thus, it is infeasible to solve them with exhaustive method. So as to optimize these problems, such as unit commitment [1], feature selection [2, 3], task scheduling [4, 5], and 0-1 knapsack problem [6, 7], binary algorithms are proposed to generate binary solutions. For example, Mirjalili et al. [8] adapted the standard continuous BA algorithm to be applied to binary spaces and then BBA combined with k-Nearest Neighbor (KNN, $k = 1$) method was used to solve feature selection problem [2]. BBA can provide competitive performance but, in some cases, it may get stuck to local minima. To solve this issue, an improved binary bat algorithm, named IBBA, is proposed. IBBA will carry out a more diversified search process. The main contributions of the paper can be summarized as follows:

(1) An improved high-performance binary bat algorithm is proposed for binary problems. Using the neighbor bat and dynamic inertia weight strategy, the proposed approach can be more able to avoid being trapped into local minima.

(2) To evaluate its performance, the proposed IBBA and several other algorithms are implemented on benchmark functions and zero-one knapsack problems. The results obtained prove that IBBA outperforms the other algorithms.

The organization of the paper is as follows: a compact overview of BA and BBA is given in Section 2. A literature review on inertia weight strategies is also provided in this section. Section 3 presents the improved structure of IBBA. The experimental results of benchmark functions and zero-one knapsack problems are demonstrated in Sections 4 and 5, respectively. And the reason that the performance of IBBA is better than other algorithms is given in Section 6. Finally, conclusion is made in Section 7.

2. Background

This section provides a brief overview of BA and BBA. A literature review on inertia weight strategies is also presented.

```
Initialize the bat population X_i (i = 1, 2, ..., n) and V_i;
Define pulse frequency F_i;
Initialize pulse rate r_i and the loudness A_i;
while (t < Max number of iterations) do
    Generate new solutions by adjusting frequency, updating
     velocities and positions [Eq. (1) to (3)];
    if (rand > r_i) then
        Select a solution among the best solutions randomly;
        Generate a local solution around the selected best solution;
    end if
    Generate a new solution by flying randomly;
    if (rand < A_i & f(x_i) < f(Gbest)) then
        Accept the new solutions;
        Increase r_i and reduce A_i;
    end if
    Rank the bats and find the current Gbest;
end while
```

Algorithm 1: Pseudocode of bat algorithm.

2.1. The Bat Algorithm. Inspired by the echolocation behavior of bats, Yang proposed the bat algorithm [9]. When bats chase preys, they will decrease the loudness and increase the frequency of emitted ultrasonic sound. These characteristics of real bats have been used in developing the BA. These basic steps of BA have been mathematically described as follows [9].

In the BA, each bat has three vectors, including a frequency vector, a velocity vector, and a position vector that are updated at time step t as (1), (2), and (3):

$$V_i(t+1) = V_i(t) + \left(X_i(t) - Gbest\right) F_i \tag{1}$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \tag{2}$$

where $Gbest$ represents the best position obtained so far and F_i represents the frequency of ith bat which is updated as follows:

$$F_i = F_{\min} + \left(F_{\max} - F_{\min}\right)\beta, \tag{3}$$

where β in the range of $[0, 1]$ is a random vector drawn from a uniform distribution. From (1) and (3), it is obvious that different frequencies promote the exploration capability of bats to the optimal solution.

These equations, to a certain extent, can guarantee the exploitation capability of the BA. However, to perform the intensification better, a random walk operation has also been employed as follows:

$$X_{\text{new}} = X_{\text{old}} + \varepsilon \overline{A}^t, \tag{4}$$

where X_{old} means one solution selected randomly among the current best solutions, ε is a randomly selected number in the range of $[-1, 1]$, and $\overline{A}$ indicates the average loudness of all bats at this time step. The pseudocode of BA algorithm is demonstrated in Algorithm 1. Note that rand is a random number uniformly distributed in the range $[0, 1]$. To an extent, BA can be considered as a balanced combination of global and intensive local search. The pulse emission rate (r) and loudness (A) control the balancing between these two search techniques. As A increases, artificial bats tend to perform a diversification rather than intensification. These two parameters mentioned above are updated as follows:

$$\begin{aligned} A_i(t+1) &= \alpha A_i(t) \\ r_i(t+1) &= r_i(0)\left[1 - \exp\left(-\gamma t\right)\right], \end{aligned} \tag{5}$$

where α and γ are constants and α has the same meaning of the cooling factor in SA [10]. To guarantee that the artificial bats are moving toward the optimal solutions, both loudness and emission rate are updated when the better solutions are found. For any $0 < \alpha, \gamma < 1$,

$$\begin{aligned} A_i(t) &\longrightarrow 0, \\ r_i(t) &\longrightarrow r_i(0), \\ &\text{as } t \longrightarrow \infty. \end{aligned} \tag{6}$$

For simplicity, $\alpha = \gamma$ can be used.

2.2. Binary Bat Algorithm. The binary bat algorithm (BBA) was proposed by Mirjalili et al. [8] to solve optimization problems with binary search space. The structure of BBA is almost the same as the original BA in which the velocity and frequency are defined in continuous space. BBA makes two changes to the original BA:

(i) The vector of position is no longer a continuous-valued vector but a bit string.

(ii) The random operation demonstrated by (4) is no longer suitable to binary search space. Instead, a simpler operation is adopted.

The position update equation for BBA changes to

$$x_i^k(t+1) = \begin{cases} \left(x_i^k(t)\right)^{-1} & \text{rand} \le f\left(v_i^k(t+1)\right) \\ x_i^k(t) & \text{rand} > f\left(v_i^k(t+1)\right), \end{cases} \tag{7}$$

where

$$f\left(v_i^k(t)\right) = \left|\frac{2}{\pi}\arctan\left(\frac{\pi}{2}v_i^k(t)\right)\right| \quad (8)$$

and $x_i^k(t)$ and $v_i^k(t)$ indicate the position and velocity of ith artificial bat at iteration t in kth dimension and $(x_i^k(t))^{-1}$ represents the complement of $x_i^k(t)$.

The operation demonstrated by (4) for BBA changes to

$$X_{\text{new}} = X_{\text{old}}, \quad (9)$$

where X_{old} still denotes a solution selected randomly from the current best solutions.

2.3. Inertia Weight Strategies. In some heuristic algorithms, especially PSO [11], inertia weight strategy plays an important role in the process of keeping balance between global search and local search process. The inertia weight strategy determines the contribution proportion of a particle's old velocity to its new velocity at the current time step. Shi and Eberhart [12] proposed the concept of inertia weight by adopting constant inertia weight strategy and demonstrated that a large inertia weight enhances the exploration while a small inertia weight enhances the exploitation. Further, various dynamic inertia weight strategies have been proposed which can improve the capabilities of PSO and they can be categorized into three classes: constant and random inertia weight strategies, time-varying inertia weight strategies, and adaptive inertia weight strategies. A compact literature review of inertia weight strategies is presented in subsequent paragraphs.

Eberhart and Shi [13] presented a random inertia weight strategy which was demonstrated to be more suitable for dynamic problems. Khan et al. [14] proposed a modified PSO by introducing a mutation mechanism and using dynamic algorithm parameters. To increase the diversity of the particles, the inertia weight of each particle adopts a random updating formula.

Most of the PSO variants employed time-varying inertia weight strategies in which the value of the inertia weight is adjusted based on the iteration number. In [15], a linear decreasing variant of inertia weight was proposed and was illustrated to be effective in enhancing the fine tuning performance of the PSO. Inspired by the idea of decreasing the inertia weight over time step, a nonlinear decreasing inertia weight strategy was proposed [16]. Gao et al. [17] presented a novel PSO variant which combined chaos mutation operator with the logarithm decreasing inertia weight. Based on the idea of decreasing inertia weight, Chen et al. [18] proposed two natural exponent inertia weight strategies to solve the continuous optimization problems.

Adaptive inertia weight strategies is another research trend of inertia weight strategies which monitor the search situation and adjust the inertia weight value according to one or more feedback parameters. Nickabadi et al. [19] proposed a new adaptive inertia weight approach that employs the success rate of the swarm as the feedback parameter to ascertain the particle's situation in the search space. Zhan et al. [20] presented an adaptive particle swarm optimization (APSO) which provides better search efficiency than classical PSO. Yang et al. [21] used speed factor and aggregation degree factor to adapt the value of inertia weight.

3. Improved Binary Bat Algorithm

BA is an algorithm combined with many merits of previous proposed heuristic algorithms, such as PSO [11] and SA [10]. Therefore, it stands for the reason that the update process of velocity and location in the BA has many similarities with PSO. When the velocity update equation (1) is analyzed, it is obvious that this equation consists of two parts. The first item ($V_i(t)$) denotes the velocity of population and the second item ($(X_i(t) - Gbest)F_i$) controls the velocity of the ith position ($X_i(t)$) with guidance of the global best solution ($Gbest$). The first and second items of the equation affect the algorithm so that it performs global and local search, respectively. It has been proven that the first item of (1) may reduce the convergence rate rapidly and the second item of (1) may result in premature convergence problem [22]. To solve this problem, some improved BA algorithms have been proposed recently [22–25]. Due to the fact that the structure of BBA is effectively the same as the original BA, BBA is not good enough at exploration and exploitation, too.

EBA [22] illustrates that the algorithm could produce better solutions with the guidance of the neighbor bat (kth solution). For this purpose, inspired by [12], the velocity update equation of original BBA is modified as follows:

$$\begin{aligned} V_i(t+1) &= w\left(V_i(t)\right) + \left(X_i(t) - Gbest\right)F_i\delta_1 \\ &\quad + \left(X_i(t) - X_k(t)\right)F_i\delta_2 \\ \delta_1 + \delta_2 &= 1, \end{aligned} \quad (10)$$

where w denotes the inertia weight factor which balances local and global search intensity of the ith solution by controlling the value of old velocity $V_i(t)$, X_k represents one of the best solutions randomly selected from the population ($i \neq k$), δ_1 is self-adaptive learning factor of global best solution ($Gbest$) ranging from 0 to 1, and therefore, δ_2, which is a learning factor of kth solution, ranges from 1 to 0. Since the kth solution information is used to guide the ith solution, the algorithm can be more able to avoid local minima. As δ_1 is increased, the effect of the global best solution ($Gbest$) becomes higher than the kth neighbor solution (X_k) and vice versa. The update equation for δ_1 is shown as follows:

$$\delta_1 = 1 + \left(\delta_{\text{init}} - 1\right)\left(\frac{\text{iter}_{\max} - \text{iter}}{\text{iter}_{\max}}\right)^n, \quad (11)$$

where δ_{init} denotes initial impact factor of δ_1, $\text{iter}_{\max}$ represents the maximum number of iterations, iter indicates the current number of iterations, and n indicates a nonlinear modulation index. As iter is increased, δ_1 will increase from δ_{init} to 1 nonlinearly and δ_2 will decrease from $(1 - \delta_{\text{init}})$ to 0 correspondingly. With a small δ_1 and a large δ_2, bats are allowed to fly around the search space, instead of flying toward the swarm best. On the other hand, a large δ_1 and a small δ_2 allow the bats to converge to the global optimum

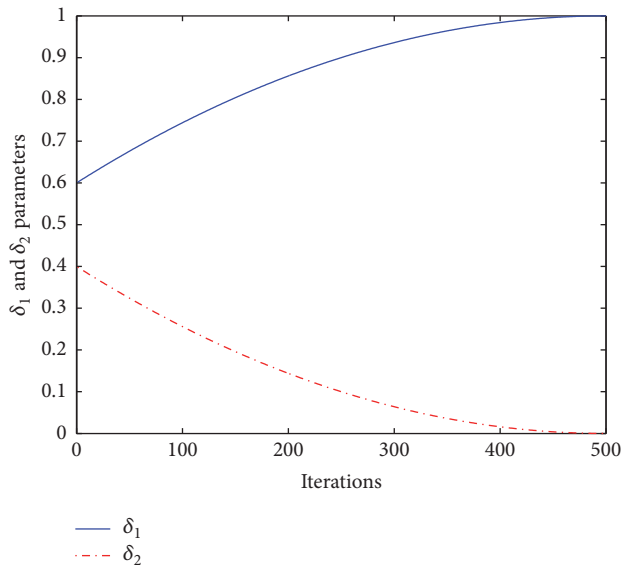

Figure 1: The changes of δ_1 and δ_2 with iterations.

Table 1: Unimodal benchmark functions.

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

solution in the latter stages of the search process. Therefore, the proposed approach can effectively control the global search and enhance convergence to the global best solution during the latter part of the optimization. Thus, the solution can switch from exploration to exploitation. The states of δ_1 and δ_2 have been illustrated in Figure 1 when δ_{init} and n are 0.6 and 2, respectively.

Inspired by [26], dynamic inertia weight strategy is used to control the magnitude of the velocity. This strategy is illustrated as follows:

$$w = w_{\max} * \exp\left(-m * \left(\frac{\text{iter}}{\text{iter}_{\max}}\right)^m\right), \tag{12}$$

where $\text{iter}_{\max}$ indicates the total number of iterations, iter denotes the current number of iterations, maximal inertia values are represented by $w_{\max}$, and m is a constant larger than 1.

It is crucial to balance the large-scale exploration search and exploitation search. Once the algorithm locates the approximate position of global optima, refined local exploitation search capabilities need to be enhanced to get global optimum. To dynamically control the transformation point of global search and local search, an adaptive strategy is designed. If the current global best solution is not improved after q iterations, the algorithm switches to intensive exploitation search with the smaller m or continues exploration search with the current m. The strategy is defined as follows:

$$m = \begin{cases} m & Gbest^{i+q} \ge Gbest^i \\ \delta_1 * \delta_2 * m & \text{otherwise,} \end{cases} \tag{13}$$

where $Gbest^{i+q}$ and $Gbest^i$ represent the $(i+q)$th and ith taken values of $Gbest^t$, respectively, and q indicates an interval of definite iterations.

According to the descriptions mentioned above, the benefit of the proposed improved binary bat algorithm (IBBA) is that it contributes to the dispersion of the solutions into binary search space. In addition, more accurate results can be obtained.

4. Benchmark Experiment

To verify the performance of proposed algorithm, thirteen benchmark functions [27] are employed. These test functions contain two different groups: unimodal and multimodal functions. Tables 1 and 2 demonstrate the benchmark functions used, respectively, where range means the search boundary of the function. The dimensions of all test functions are 5. The global minimum values of all benchmark functions used except $f_8(x)$ are 0 while the global minimum value of $f_8(x)$ is (-418.9829×5).

For this simulation, 15 bits are used to represent each continuous variable in binary. Thus, the dimension of generating bit vector for each benchmark function is calculated as follows:

$$n_b = D_{\text{fun}} \times 15, \tag{14}$$

where n_b is the dimension of each artificial bat in IBBA and D_{fun} represents the dimension of a particular benchmark function. Therefore, the dimensions of the corresponding binary problems are $n_b = 75$ (5×15).

The simulation focused on comparing the performance of the IBBA, BBA [8], and BPSO [28]. Some basic parameters should be initialized before running these algorithms. The initial parameters for these algorithms are demonstrated in Table 3. Tables 4 and 5 present the experimental results. The obtained results of each algorithm are averaged over 30 independent runs, and best results are denoted in bold type. In addition, the mean (Mean), standard deviation value (SD), and medium value (Med) of the optimal solution in the last iteration are presented.

Table 2: Multimodal benchmark functions.

Function	Range
$f_8(x) = \sum_{i=1}^{D} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]$
$f_9(x) = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12, 5.12]$
$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right)$	$[-32, 32]$
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]$
$f_{12}(x) = \frac{\pi}{D}\left\{10\sin^2(\pi y_i) + \sum_{i=1}^{D-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_D - 1)^2\right\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a, \end{cases}$	$[-50, 50]$
$f_{13}(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_D - 1)^2\left[1 + \sin^2(2\pi x_D)\right]\right\} + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	$[-50, 50]$

Table 3: Initial parameters for IBBA, BBA, and BPSO.

Alg.	Parameters	Values
IBBA	Number of bats	30
	$F_{\min}$	0
	$F_{\max}$	2
	A	0.25
	r	0.5
	ε	$[-1, 1]$
	α	0.9
	γ	0.9
	$w_{\max}$	0.9
	Modulation index, n	2
	m	50
	q	50
	δ_{init}	0.6
	Max iteration	500
	Stopping criterion	Max iteration
BBA	Number of particles	30
	$F_{\min}$	0
	$F_{\max}$	2
	A	0.25
	r	0.5
	ε	$[-1, 1]$
	α	0.9
	γ	0.9
	Max iteration	500
	Stopping criterion	Max iteration
BPSO	Number of particles	30
	C_1, C_2	2, 2
	W	Decreased linearly from 0.9 to 0.4
	Max iterations	500
	Max velocity	6
	Stopping criterion	Max iteration

Table 4: Performance comparison on unimodal benchmark functions.

f	Metric	IBBA	BBA	BPSO
f_1	Mean	**8.38241e − 05**	5.80234e + 00	1.99866e + 01
	SD	**1.40763e − 04**	6.36667e + 00	1.10249e + 01
	Med	**4.6569e − 05**	3.774e + 00	1.92831e + 01
f_2	Mean	**2.95012e − 03**	3.24087e − 01	6.4235e − 01
	SD	**7.12409e − 03**	1.59684e − 01	2.02785e − 01
	Med	**1.52593e − 03**	2.94809e − 01	6.5554e − 01
f_3	Mean	**2.30664e − 01**	1.3097e + 01	2.63465e + 01
	SD	**9.28325e − 01**	1.25882e + 01	1.72224e + 01
	Med	**1.39707e − 04**	8.60949e + 00	2.29727e + 01
f_4	Mean	**4.88296e − 03**	2.37292e + 00	3.5139e + 00
	SD	**8.94619e − 03**	1.33978e + 00	9.97208e − 01
	Med	**3.05185e − 03**	2.03558e + 00	3.59813e + 00
f_5	Mean	**4.38632e + 00**	1.31671e + 02	2.9431e + 02
	SD	**4.43842e + 00**	1.50937e + 02	2.24461e + 02
	Med	**3.98383e + 00**	6.204e + 01	2.56548e + 02
f_6	Mean	**7.05071e − 01**	6.87477e + 00	1.70907e + 01
	SD	**4.25751e − 01**	6.73081e + 00	1.03668e + 01
	Med	**7.59196e − 01**	4.16702e + 00	1.6497e + 01
f_7	Mean	**1.27594e − 03**	9.92062e − 03	2.35084e − 02
	SD	**9.97156e − 04**	5.56257e − 03	1.30689e − 02
	Med	**1.23028e − 03**	9.12516e − 03	1.97515e − 02

To judge whether IBBA is significantly different from BBA and BPSO or not, we carried out the statistical Wilcoxon's rank-sum test [29] at a 5% significance level. The p values calculated in Wilcoxon's rank-sum test comparing IBBA and other algorithms over all the benchmark functions are summarized in Tables 6 and 7. In the demonstrated

Table 5: Performance comparison on multimodal benchmark functions.

f	Metric	IBBA	BBA	BPSO
f_8	Mean	**−2.08441e + 03**	−1.99819e + 03	−2.03321e + 03
	SD	**1.83611e + 01**	7.09513e + 01	5.53951e + 01
	Med	**−2.09464e + 03**	−2.00568e + 03	−2.05689e + 03
f_9	Mean	**1.30379e + 00**	4.46724e + 00	8.28313e + 00
	SD	**1.44133e + 00**	2.75498e + 00	1.83015e + 00
	Med	**1.0004e + 00**	3.63582e + 00	8.39933e + 00
f_{10}	Mean	**5.35486e − 03**	2.35002e + 00	3.44122e + 00
	SD	**5.71445e − 03**	9.84327e − 01	6.31882e − 01
	Med	**3.95716e − 03**	2.41859e + 00	3.46388e + 00
f_{11}	Mean	**4.8303e − 02**	7.41724e − 01	1.05605e + 00
	SD	**3.35797e − 02**	2.47447e − 01	2.35421e − 01
	Med	**3.98203e − 02**	7.20032e − 01	1.05621e + 00
f_{12}	Mean	**8.11717e − 01**	2.02809e + 00	2.2475e + 00
	SD	**7.04812e − 01**	1.89214e + 00	1.86339e + 00
	Med	**6.61436e − 01**	1.50807e + 00	1.63553e + 00
f_{13}	Mean	**2.29539e − 01**	6.26045e − 01	1.33228e + 00
	SD	**1.47623e − 01**	3.4657e − 01	9.26226e − 01
	Med	**2.54401e − 01**	5.8134e − 01	1.01025e + 00

Table 6: p values on unimodal benchmark functions.

p value	IBBA	BBA	BPSO
f_1	N.A.	2.36567e − 12	2.36567e − 12
f_2	N.A.	4.11097e − 12	4.11097e − 12
f_3	N.A.	2.58369e − 11	2.33455e − 11
f_4	N.A.	3.16021e − 12	3.16021e − 12
f_5	N.A.	1.28366e − 09	1.09044e − 10
f_6	N.A.	3.65093e − 11	2.98783e − 11
f_7	N.A.	1.28704e − 09	1.61323e − 10

Table 7: p values on multimodal benchmark functions.

p value	IBBA	BBA	BPSO
f_8	N.A.	1.35348e − 07	6.02212e − 07
f_9	N.A.	3.82641e − 08	1.04174e − 10
f_{10}	N.A.	1.20954e − 11	4.11097e − 12
f_{11}	N.A.	3.68743e − 11	3.01797e − 11
f_{12}	N.A.	1.72846e − 06	6.00786e − 08
f_{13}	N.A.	8.34855e − 08	3.33631e − 11

tables, NA means "Not Applicable" which indicates that the corresponding approach could not statistically compare with itself in the Wilcoxon's rank-sum test. According to Derrac et al. [30], if a p value is < 0.05, it can be considered as strong evidence against the null hypothesis. In other words, it means that the compared algorithms have significant difference from each other.

4.1. Unimodal Benchmark Functions. Unimodal benchmark functions are effective to quantify the convergence speed. Tables 4 and 6 summarize the results for these benchmark functions. It can be observed from Table 4 that IBBA could get the most optimal values. p values in Table 6 also point out that IBBA outperforms the other algorithms significantly. The results obtained prove that the proposed IBBA has excellent performance in finding the global optimal solution of unimodal functions.

Figures 2–8 illustrate the averaged convergence curves of algorithms on unimodal benchmark functions. As these curves show, the accuracy of the IBBA is much better than

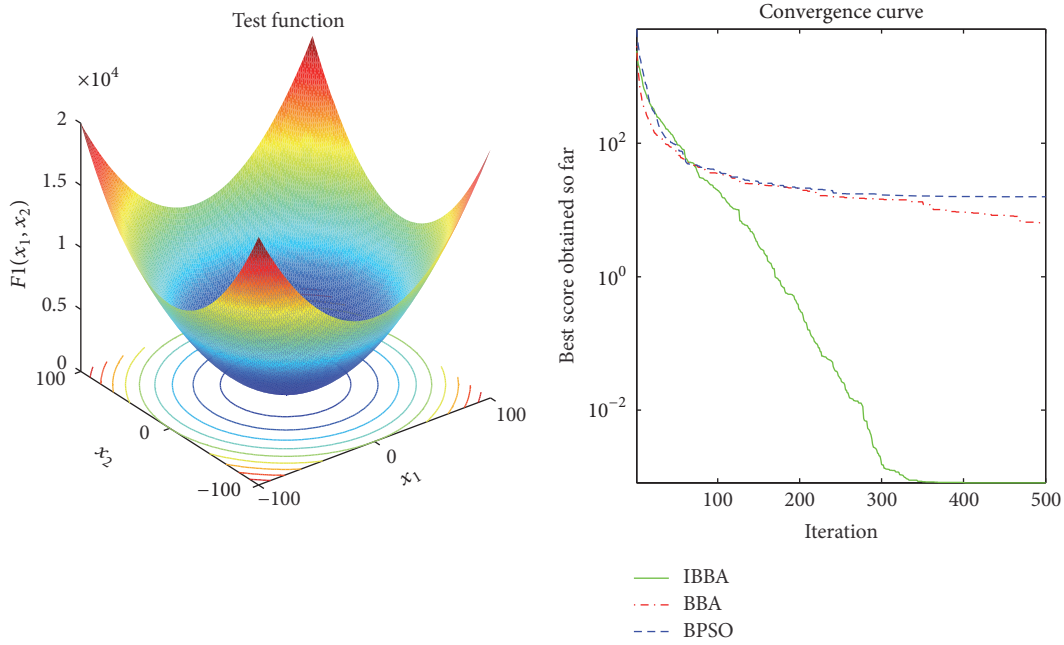

Figure 2: The averaged convergence curve of f_1.

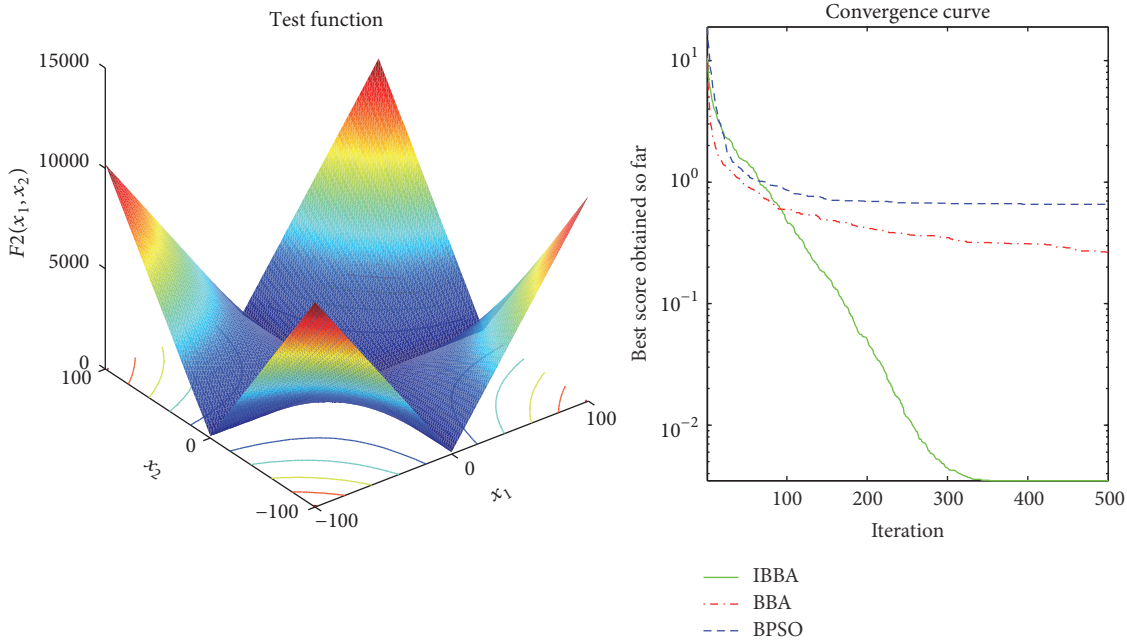

Figure 3: The averaged convergence curve of f_2.

other algorithms on these unimodal benchmark functions in terms of the mean, standard deviation value, and median value of the results. According to Tables 4 and 6 and Figures 2–8, it can be concluded that the IBBA is capable of finding the optimal solution with considerable fast convergence rate in unimodal benchmark functions.

4.2. Multimodal Benchmark Functions. Multimodal benchmark functions are usually used to detect whether the algorithm faces premature convergence problem. Tables 5 and 7 provide the results of the multimodal benchmark functions. Table 5 shows that IBBA could get the best results in all cases. And p values in Table 7 indicate that, compared

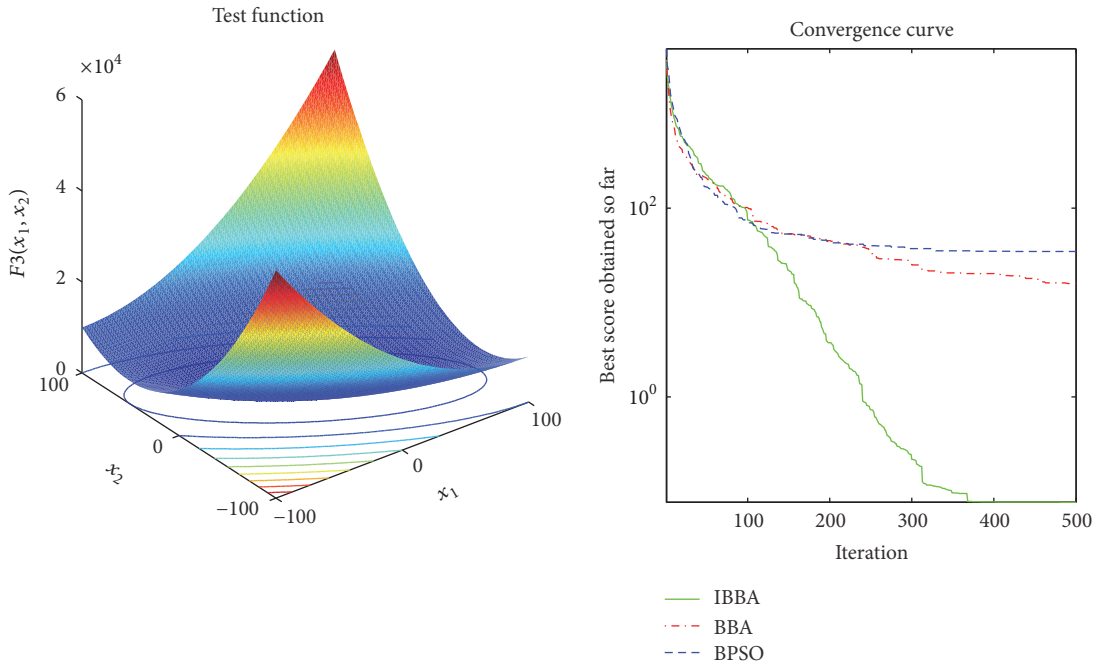

Figure 4: The averaged convergence curve of f_3.

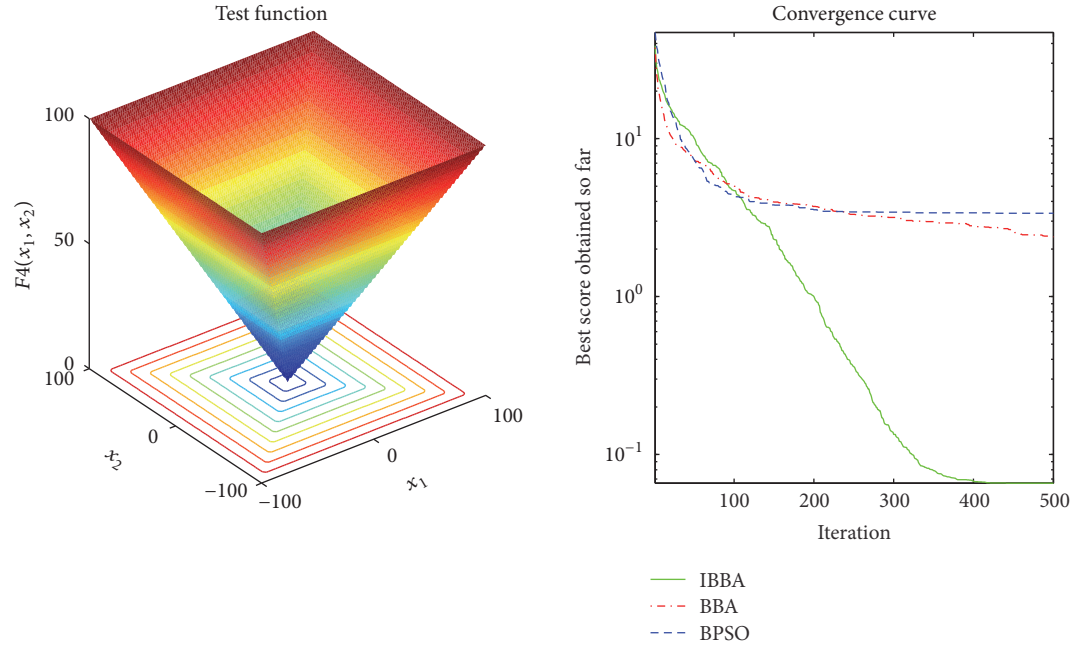

Figure 5: The averaged convergence curve of f_4.

to other algorithms, IBBA achieves significant performance boost in all the multimodal benchmark functions. Therefore, this is evidence that IBBA has the best capability of avoiding to be trapped in local minima.

The averaged convergence curves of all algorithms when they deal with multimodal benchmark functions are demonstrated in Figures 9–14. It can be observed from these figures that IBBA also has a significant improvement on the accuracy than other algorithms on multimodal benchmark functions. According to Tables 5 and 7 and Figures 9–14, results obtained prove that the IBBA is able to avoid local minima.

According to the comparison and discussion mentioned above, it can be concluded that the proposed algorithm has huge merit among the binary algorithms. The next section presents the performance of IBBA in solving the zero-one knapsack problems.

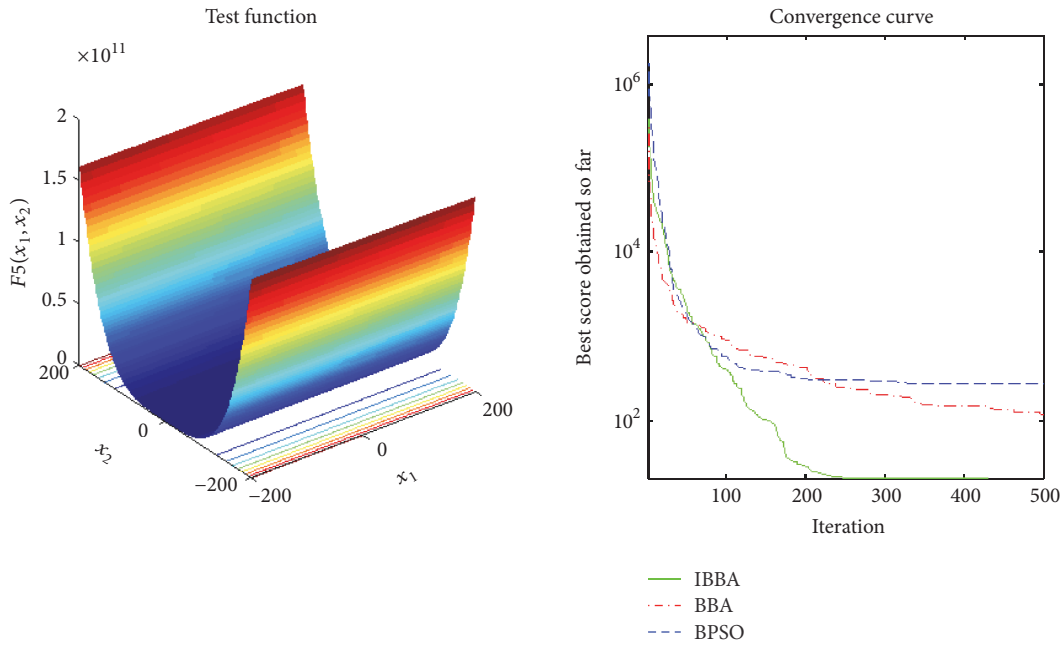

Figure 6: The averaged convergence curve of f_5.

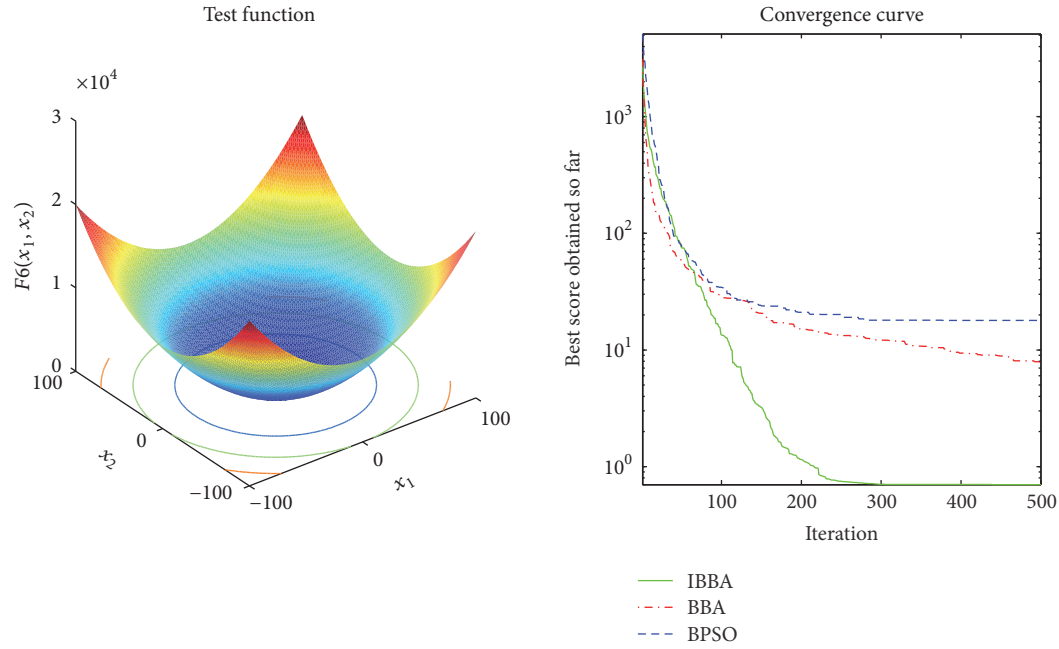

Figure 7: The averaged convergence curve of f_6.

5. Zero-One Knapsack Problem

Zero-one knapsack problem is a classical combinatorial optimization problem [31] and it has plenty of immediate applications in industry and financial management. It is assumed that, in this problem, there are many items waiting to be carried in knapsack and each item is with a specific benefit and weight. The aim of this problem is to maximize the total profit in the knapsack while the total weight in the knapsack is not more than the given limit. The zero-one knapsack problem can be modeled as follows [31]:

$$\begin{aligned} \max \quad & f(x) = \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \le C \\ & x_i = 0, 1 \quad (i = 1, 2, \ldots, n), \end{aligned} \tag{15}$$

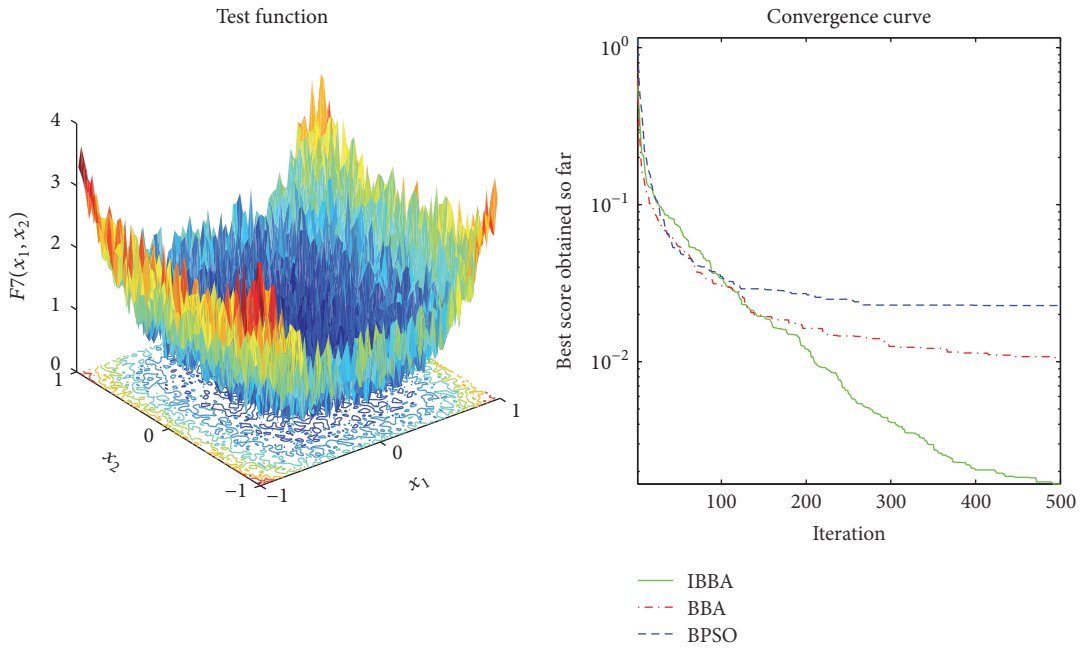

Figure 8: The averaged convergence curve of f_7.

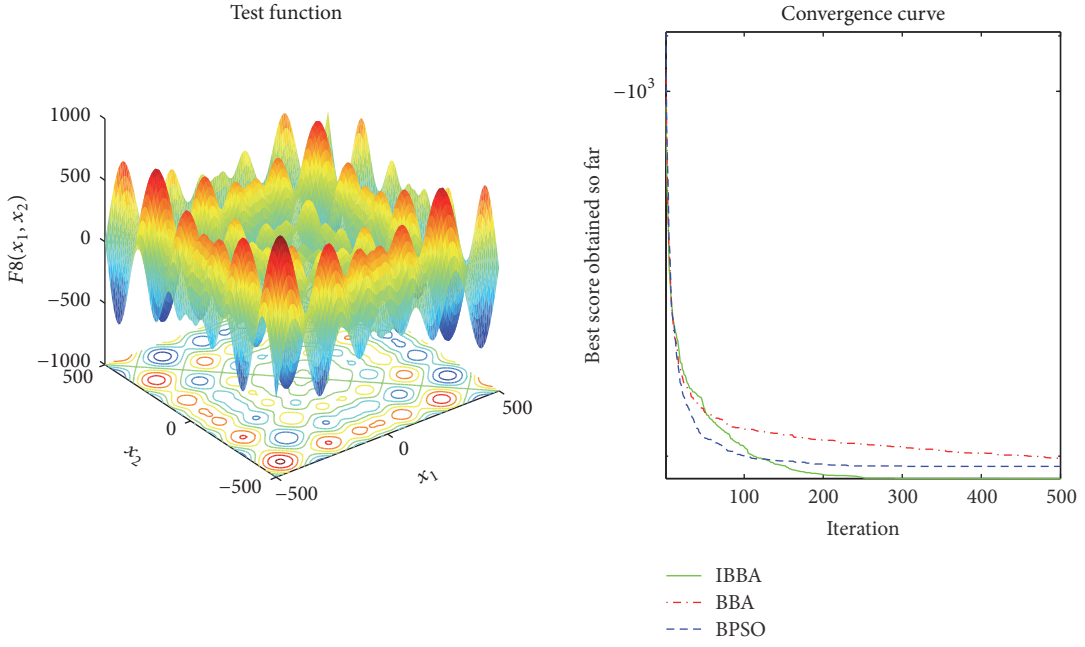

Figure 9: The averaged convergence curve of f_8.

where n is the number of items and x_1 to x_n have a benefit p_i and weight w_i. The maximum weight that this knapsack can carry is C. Assume that all benefits and weights are nonnegative. In this study, five typical knapsack problem datasets are used [7]. These test datasets are listed in Table 8, where Dim represents the dimension of knapsack problems, parameters w, p, C present the information of weight, price, and weight limit, and Opt indicates the optimal value of corresponding knapsack problem. Table 3 is used to set the initial parameters of each algorithm, too. The averaged results obtained are listed in Table 9 and optimal results are denoted in bold type.

According to Table 9, it can be observed that, in all cases, IBBA performs better than the other two algorithms. The maximum results of IBBA in four out of five test datasets are equal to the optimal values of corresponding knapsack

Table 8: Five typical test datasets of zero-one knapsack problem.

Number	Dim	Parameters (w, p, C)	Opt
$k1$	10	w = (95, 4, 60, 32, 23, 72, 80, 62, 65, 46), p = (55, 10, 47, 5, 4, 50, 8, 61, 85, 87), C = 269	295
$k2$	20	w = (92, 4, 43, 83, 84, 68, 92, 82, 6, 44, 32, 18, 56, 83, 25, 96, 70, 48, 14, 58), p = (44, 46, 90, 72, 91, 40, 75, 35, 8, 54, 78, 40, 77, 15, 61, 17, 75, 29, 75, 63), C = 878	1024
$k3$	50	w = (80, 82, 85, 70, 72, 70, 66, 50, 55, 25, 50, 55, 40, 48, 59, 32, 22, 60, 30, 32, 40, 38, 35, 32, 25, 28, 30, 22, 50, 30, 45, 30, 60, 50, 20, 65, 20, 25, 30, 10, 20, 25, 15, 10, 10, 10, 4, 4, 2, 1), p = (220, 208, 198, 192, 180, 180, 165, 162, 160, 158, 155, 130, 125, 122, 120, 118, 115, 110, 105, 101, 100, 100, 98, 96, 95, 90, 88, 82, 80, 77, 75, 73, 72, 70, 69, 66, 65, 63, 60, 58, 56, 50, 30, 20, 15, 10, 8, 5, 3, 1), C = 1000	3103
$k4$	80	w = (199, 194, 193, 191, 189, 178, 174, 169, 164, 164, 161, 158, 157, 154, 152, 152, 149, 142, 131, 125, 124, 124, 124, 122, 119, 116, 114, 113, 111, 110, 109, 100, 97, 94, 91, 82, 82, 81, 80, 80, 80, 79, 77, 76, 74, 72, 71, 70, 69, 68, 65, 65, 61, 56, 55, 54, 53, 47, 47, 46, 41, 36, 34, 32, 32, 30, 29, 29, 26, 25, 23, 22, 20, 11, 10, 9, 5, 4, 3, 1), p = (40, 27, 5, 21, 51, 16, 42, 18, 52, 28, 57, 34, 44, 43, 52, 55, 53, 42, 47, 56, 57, 44, 16, 2, 12, 9, 40, 23, 56, 3, 39, 16, 54, 36, 52, 5, 53, 48, 23, 47, 41, 49, 22, 42, 10, 16, 53, 58, 40, 1, 43, 56, 40, 32, 44, 35, 37, 45, 52, 56, 40, 2, 23, 49, 50, 26, 11, 35, 32, 34, 58, 6, 52, 26, 31, 23, 4, 52, 53, 19), C = 1173	5183
$k5$	100	w = (54, 95, 36, 18, 4, 71, 83, 16, 27, 84, 88, 45, 94, 64, 14, 80, 4, 23, 75, 36, 90, 20, 77, 32, 58, 6, 14, 86, 84, 59, 71, 21, 30, 22, 96, 49, 81, 48, 37, 28, 6, 84, 19, 55, 88, 38, 51, 52, 79, 55, 70, 53, 64, 99, 61, 86, 1, 64, 32, 60, 42, 45, 34, 22, 49, 37, 33, 1, 78, 43, 85, 24, 96, 32, 99, 57, 23, 8, 10, 74, 59, 89, 95, 40, 46, 65, 6, 89, 84, 83, 6, 19, 45, 59, 26, 13, 8, 26, 5, 9), p = (297, 295, 293, 292, 291, 289, 284, 284, 283, 283, 281, 280, 279, 277, 276, 275, 273, 264, 260, 257, 250, 236, 236, 235, 235, 233, 232, 232, 228, 218, 217, 214, 211, 208, 205, 204, 203, 201, 196, 194, 193, 193, 192, 191, 190, 187, 187, 184, 184, 184, 181, 179, 176, 173, 172, 171, 160, 128, 123, 114, 113, 107, 105, 101, 100, 100, 99, 98, 97, 94, 94, 93, 91, 80, 74, 73, 72, 63, 63, 62, 61, 60, 56, 53, 52, 50, 48, 46, 40, 40, 35, 28, 22, 22, 18, 15, 12, 11, 6, 5), C = 3818	15170

Table 9: Performance comparison on zero-one knapsack problems over 30 independent runs.

Alg.	Metric	0-1 knapsack problem datasets				
		$k1$	$k2$	$k3$	$k4$	$k5$
IBBA	Avg	**295**	**1024**	**3085.7**	**5134.27**	**15051.6**
	Std	**0**	**0**	**7.32097**	**34.7453**	**35.6122**
	Max	**295**	**1024**	**3103**	**5178**	**15170**
	Min	**295**	**1024**	**3073**	**5063**	**15046**
BBA	Avg	**295**	**1024**	3003.2	4410.67	12942.7
	Std	**0**	**0**	27.3526	95.5894	256.52
	Max	**295**	**1024**	3049	4663	13445
	Min	**295**	**1024**	2944	4251	12489
BPSO	Avg	**295**	959.167	2851.13	3987.77	10089.2
	Std	**0**	23.0563	37.0328	99.9616	238.369
	Max	**295**	1016	2956	4164	10491
	Min	**295**	913	2774	3812	9671

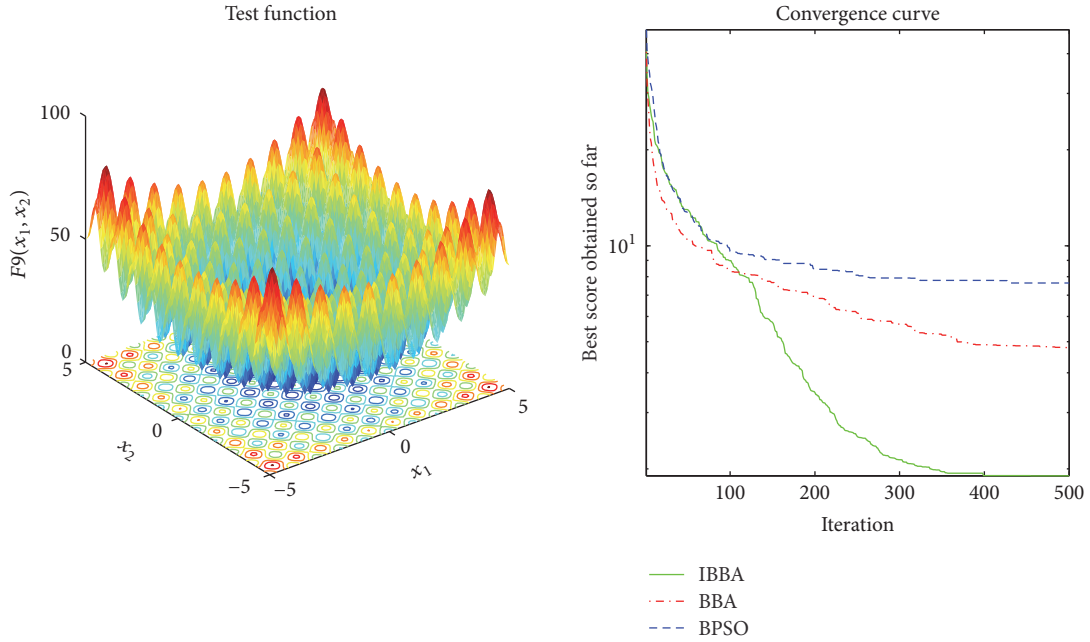

Figure 10: The averaged convergence curve of f_9.

problems. In the remaining test datasets ($k4$), the maximum result of IBBA is also almost equal to the optimal values. What is more, the standard deviation values of IBBA are smaller than those of other algorithms, which means that IBBA can search the optimal value more stably. Thus, it could be concluded that IBBA could provide better capability to solve the zero-one knapsack problem than original BBA and BPSO.

6. Discussion

This section briefly presents the reason that the performance of IBBA was better than the other two algorithms:

(1) Inspired by several heuristic algorithms, BA combines the merit of previously proposed heuristic algorithms, such as PSO and SA.

(2) With the guidance of neighbor bat, the algorithm could produce more diversified solutions.

(3) To balance the global search and local search, dynamic inertia weight strategy is employed to control the magnitude of the velocity. Thus, it could avoid premature convergence.

(4) The adaptive strategy designed to dynamically control the transformation point of global search and local search promotes the accuracy of results.

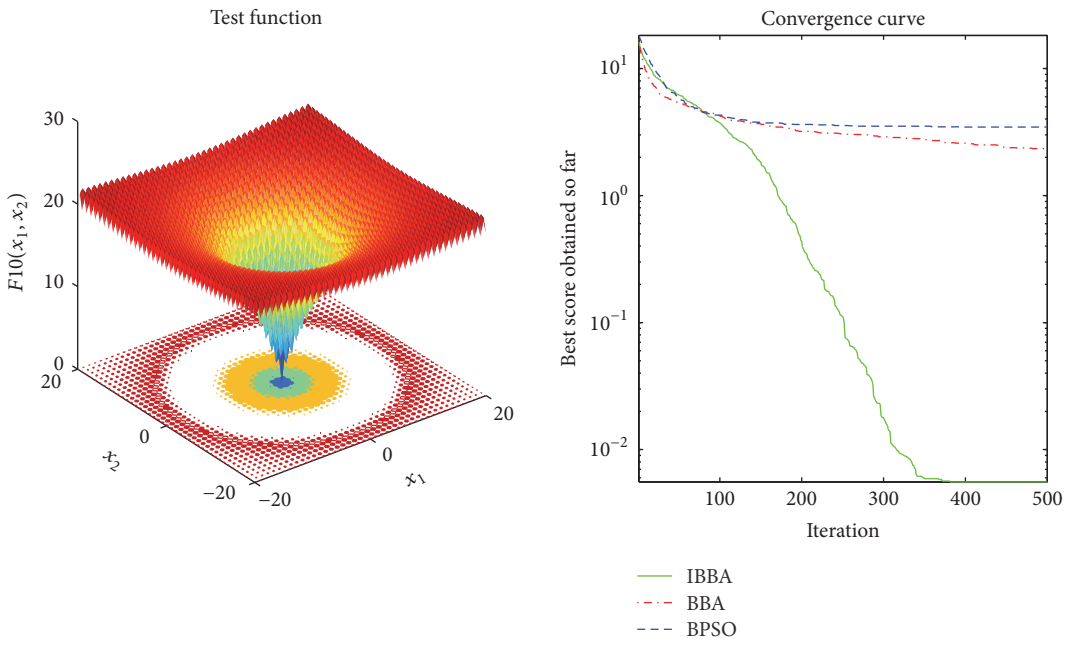

Figure 11: The averaged convergence curve of f_{10}.

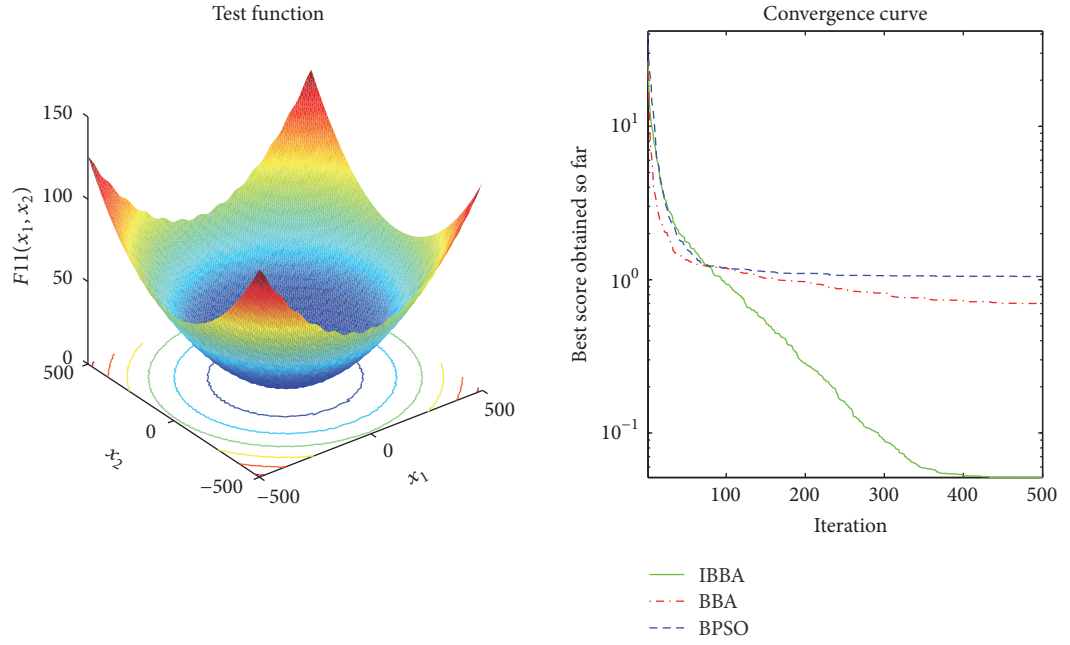

Figure 12: The averaged convergence curve of f_{11}.

7. Conclusions

In this paper, an improved binary bat algorithm is presented. Simple but effective modifications are employed to enhance the BBA. Then the proposed IBBA is implemented and tested on thirteen well-known benchmark functions and three typical zero-one knapsack problems. The obtained results are compared with those of BBA and BPSO. The experiment results show that the proposed approach could get much better performance than the other two algorithms mentioned above in terms of accuracy of the results.

For further studies, we intend to apply the proposed algorithm to practical applications, such as application switch problem [32] and task scheduling problem [4].

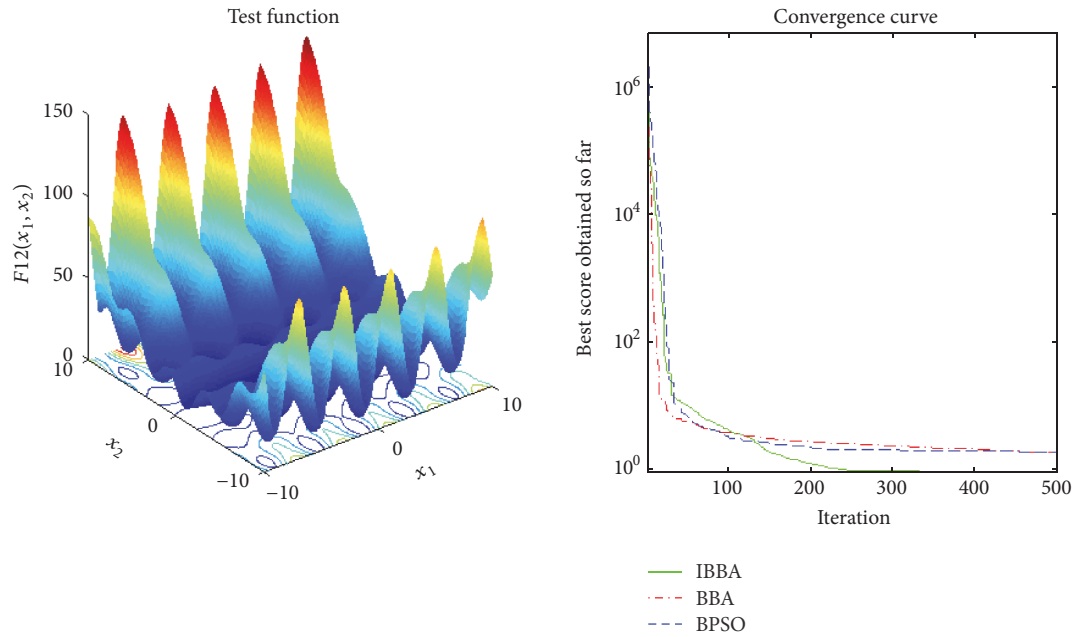

Figure 13: The averaged convergence curve of f_{12}.

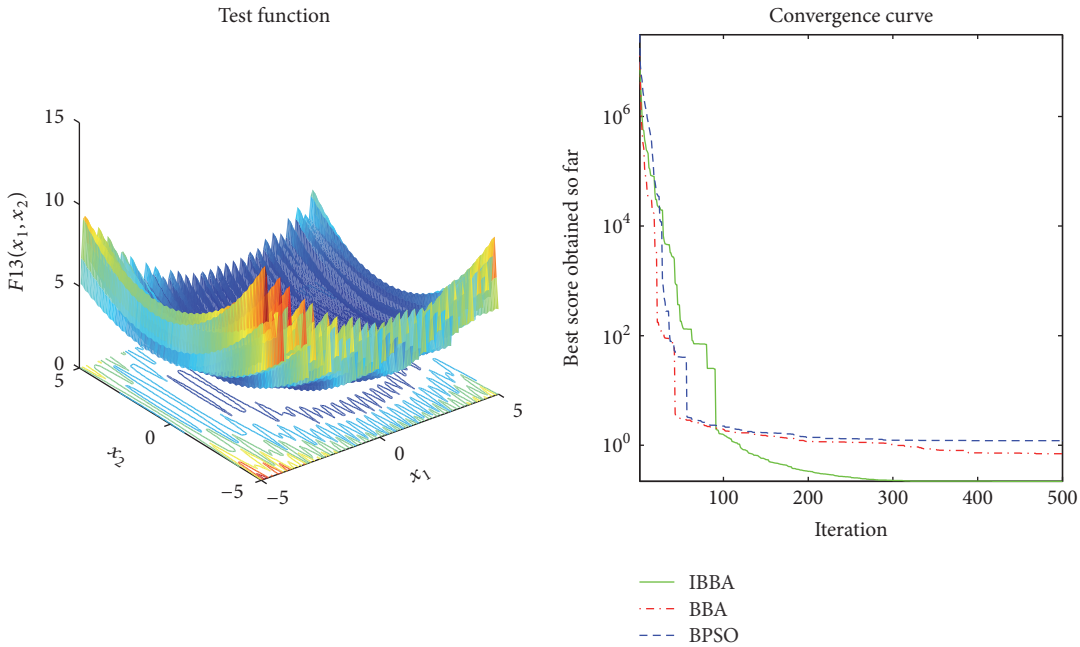

Figure 14: The averaged convergence curve of f_{13}.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National High Technology Research and Development Program of China (863 Program) (Grant no. 2015AA015802).

References

[1] Z.-L. Gaing, "Discrete particle swarm optimization algorithm for unit commitment," in *Proceedings of the IEEE Power Engineering Society General Meeting*, vol. 1, pp. 418–424, Toronto, Canada, 2003.

[2] R. Y. Nakamura, L. A. Pereira, K. A. Costa, D. Rodrigues, J. P. Papa, and X. Yang, "Bba: a binary bat algorithm for feature selection," in *Proceedings of the 25th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI '12)*, pp. 291–297, Ouro Preto, Brazil, 2012.

[3] S. Kashef and H. Nezamabadi-pour, "An advanced ACO algorithm for feature subset selection," *Neurocomputing*, vol. 147, no. 1, pp. 271–279, 2015.

[4] H. S. Al-Olimat, M. Alam, R. Green, and J. K. Lee, "Cloudlet scheduling with particle swarm optimization," in *Preceedings of the 5th International Conference on Communication Systems and Network Technologies (CSNT '15)*, pp. 991–995, April 2015.

[5] L. Xu, K. Wang, Z. Ouyang, and X. Qi, "An improved binary PSO-based task scheduling algorithm in green cloud computing," in *Proceedings of the 9th International Conference on Communications and Networking in China (CHINACOM '14)*, pp. 126–131, August 2014.

[6] J. C. Bansal and K. Deep, "A modified binary particle swarm optimization for knapsack problems," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11042–11061, 2012.

[7] H. Wu, F. Zhang, and R. Zhan, "A binary wolf pack algorithm for solving 0-1 knapsack problem," *Systems Engineering and Electronics*, vol. 36, no. 8, pp. 1660–1667, 2014.

[8] S. Mirjalili, S. M. Mirjalili, and X.-S. Yang, "Binary bat algorithm," *Neural Computing and Applications*, vol. 25, no. 3-4, pp. 663–681, 2014.

[9] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, J. R. Gonzalez, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, Eds., vol. 284 of *Studies in Computational Intelligence*, pp. 65–74, Springer, Berlin, Germany, 2010.

[10] S. P. Brooks and B. J. Morgan, "Optimization using simulated annealing," *The Statistician*, vol. 44, no. 2, pp. 241–257, 1995.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks (ICNN '95)*, vol. 4, pp. 1942–1948, Perth, Western Australia, 1995.

[12] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the IEEE International Conference on Evolutionary Computation and IEEE World Congress on Computational Intelligence*, pp. 69–73, Anchorage, Ala, USA, 1998.

[13] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Proceedings of the Congress on Evolutionary Computation*, vol. 1, pp. 94–100, Seoul, Republic of Korea, May 2001.

[14] S. U. Khan, S. Yang, L. Wang, and L. Liu, "A modified particle swarm optimization algorithm for global optimizations of inverse problems," *IEEE Transactions on Magnetics*, vol. 52, no. 3, 2016.

[15] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the 2000 Congress on Evolutionary Computation (CEC' 00)*, vol. 1, pp. 84–88, La Jolla, Claif, USA, 2000.

[16] A. Chatterjee and P. Siarry, "Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization," *Computers & Operations Research*, vol. 33, no. 3, pp. 859–871, 2006.

[17] Y.-L. Gao, X.-H. An, and J.-M. Liu, "A particle swarm optimization algorithm with logarithm decreasing inertia weight and chaos mutation," in *Proceedings of the 2008 International Conference on Computational Intelligence and Security (CIS '08)*, pp. 61–65, chn, December 2008.

[18] G. Chen, X. Huang, J. Jia, and Z. Min, "Natural exponential inertia weight strategy in particle swarm optimization," in *Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA '06)*, vol. 1, pp. 3672–3675, June 2006.

[19] A. Nickabadi, M. M. Ebadzadeh, and R. Safabakhsh, "A novel particle swarm optimization algorithm with adaptive inertia weight," *Applied Soft Computing Journal*, vol. 11, no. 4, pp. 3658–3670, 2011.

[20] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, vol. 39, no. 6, pp. 1362–1381, 2009.

[21] X. Yang, J. Yuan, H. Yuan, and H. Mao, "A modified particle swarm optimizer with dynamic adaptation," *Applied Mathematics & Computation*, vol. 189, no. 2, pp. 1205–1213, 2007.

[22] S. Yilmaz and E. U. Küçüksille, "A new modification approach on bat algorithm for solving optimization problems," *Applied Soft Computing*, vol. 28, pp. 259–275, 2015.

[23] G. Wang, M. Lu, and X. Zhao, "An improved bat algorithm with variable neighborhood search for global optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '16)*, pp. 1773–1778, Vancouver, Canada, 2016.

[24] X. Cai, L. Wang, Q. Kang, and Q. Wu, "Bat algorithm with gaussian walk," *International Journal of Bio-Inspired Computation*, vol. 6, no. 3, pp. 166–174, 2014.

[25] X. Cai, X. Z. Gao, and Y. Xue, "Improved bat algorithm with optimal forage strategy and random disturbance strategy," *International Journal of Bio-Inspired Computation*, vol. 8, no. 4, pp. 205–214, 2016.

[26] K. Lei and C. Pu, "Complex optimization problems using highly efficient particle swarm optimizer," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 12, no. 4, pp. 1023–1030, 2014.

[27] S. Mirjalili, "Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing & Applications*, vol. 27, no. 4, pp. 1053–1073, 2016.

[28] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation*, vol. 5, pp. 4104–4108, IEEE, Orlando, Fla, USA, October 1997.

[29] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[30] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[31] Y. Sharafi, M. A. Khanesar, and M. Teshnehlab, "Discrete binary cat swarm optimization algorithm," in *Proceedings of the 3rd IEEE International Conference on Computer, Control and Communication (IC4 '13)*, pp. 1–6, 2013.

[32] Y. Jiang, X. Zeng, and P. Sun, "A rapid application switch technique based on resource cache," *Journal of Network New Media*, vol. 2, no. 4, pp. 33–38, 2013.