An Improved Brain Storm Optimization with Differential Evolution Strategy for Applications of ANNs

Zijian Cao,1,2 Xinhong Hei,1 Lei Wang,1 Yuhui Shi,3 and Xiaofeng Rong2

1School of Computer Science and Engineering, Xi’an University of Technology, Xi’an 710048, China
2School of Computer Science and Engineering, Xi’an Technological University, Xi’an 710021, China
3Xi’an Jiaotong-Liverpool University, Suzhou 215123, China

Correspondence should be addressed to Xinhong Hei; heixinhong@xaut.edu.cn

Received 3 May 2015; Revised 28 July 2015; Accepted 29 July 2015

Academic Editor: George S. Dulikravich

Copyright © 2015 Zijian Cao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Brain Storm Optimization (BSO) algorithm is a swarm intelligence algorithm inspired by human being’s behavior of brainstorming. The performance of BSO is maintained by the creating process of ideas, but when it cannot find a better solution for some successive iterations, the result will be so inefficient that the population might be trapped into local optima. In this paper, we propose an improved BSO algorithm with differential evolution strategy and new step size method. Firstly, differential evolution strategy is incorporated into the creating operator of ideas to allow BSO jump out of stagnation, owing to its strong searching ability. Secondly, we introduce a new step size control method that can better balance exploration and exploitation at different searching generations. Finally, the proposed algorithm is first tested on 14 benchmark functions of CEC 2005 and then is applied to train artificial neural networks. Comparative experimental results illustrate that the proposed algorithm performs significantly better than the original BSO.

1. Introduction

In the past few decades, swarm intelligence algorithms, which were derived from the concepts, principles, and mechanisms of nature-inspired computation, have attracted more and more attention from researchers. Though swarm intelligence is a new category of stochastic, population-based optimization algorithms compared with evolutionary algorithms (EAs) such as Genetic Algorithms (GA) [1], Evolutionary Programming (EP) [2], Evolutionary Strategies (ES) [3], Genetic Programming (GP) [4], and differential evolution (DE) [5], so far, a lot of swarm intelligence optimization algorithms, such as Particle Swarm Optimization (PSO) [6, 7], have been proposed to tackle many challenging complex computational problems in science and industry.

Brain Storm Optimization (BSO) algorithm [8, 9] is a swarm intelligence algorithm inspired by human being’s behavior of brainstorming. Derived from human creative problem-solving process, BSO has achieved more and more attention and has been applied to the function of optimization, optimal satellite formation reconfiguration [10], the design of DC Brushless Motor [11], economic dispatch considering wind power [12], multiobjective optimization [13, 14], and so forth. Due to the flexibility and scalability of the original BSO algorithm [15], many new versions of BSOS have been designed and implemented. To improve the performance and reduce the computation burden of BSO algorithm, Zhan et al. [16] modified the grouping operator and the creating operator and designed a BSO variant named Modified BSO (MBSO). Cheng et al. [17] used two kinds of partial reinitialization strategies to improve the population diversity in BSO algorithm. Recently, Yang et al. [18] proposed advanced discussion mechanism-based brain storm optimization (ADMBSO), which incorporated intercluster and intracluster discussions into BSO to control global and local searching ability. However, optimization problems have become more and more complex, from simple unimodal functions to hybrid rotated shifted multimodal functions. To the best of our knowledge, all variants of BSO have not been tested on the complex benchmark functions of CEC2005 [19]. Hence, more effective improvement to the original BSO algorithms is always necessary.
Among existing metaheuristics, differential evolution (DE) is a simple yet powerful global optimization technique with successful applications in various areas [5]. To achieve the most satisfactory optimization performance, an improved BSO algorithm with differential evolution strategy and new step size method named BSODE, which can maintain both the exploration ability and the diversity of population, is proposed in the paper. The improved algorithm will be first tested on 14 benchmark functions of CEC 2005 and then will be applied to train artificial neural networks.

The rest of this paper is organized as follows. Section 2 briefly introduces the original BSO algorithm and the basic DE operator. Section 3 describes the improved BSO with differential evolution strategy (BSODE). Section 4 presents the tests on 14 benchmark functions. The applications for artificial neural network are given in Section 5, followed by conclusions in Section 6.

2. Related Work

2.1. Brain Storm Optimization. The BSO algorithm is motivated by the philosophy of brainstorming, and brainstorming is a widely used tool for increasing creativity in organizations which has achieved wide acceptance as a mean of facilitating creative thinking [20]. A potential solution in the solution space represents an idea in BSO. BSO sticks to the rules of interchange of ideas by a team and uses clustering, replacing, and creating operators to produce global optimum generation by generation. In the procedure of BSO, firstly, \( N \) ideas are randomly initialized within the solution space, and then each idea is evaluated according to its fitness function. Next \( m \) points of cluster center are also randomly selected and initialized like \( N \) ideas, where \( m \) is less than \( N \). As presented in [8, 9], the rest of the process in BSO can be described as follows.

2.1.1. Clustering Individuals. Clustering is a process of grouping similar objects together, and, during each generation, all the ideas are clustered into \( m \) clusters according to idea (or individual) features. The best idea in each center is chosen as its cluster center, and the clustering operation can refine a search area. K-means is a popular algorithm used in clustering; herein it is used in clustering operation.

2.1.2. Disrupting Cluster Center. Cluster center disrupting operation randomly chooses a cluster center and replaces it with a newly generated idea with a probability of \( p_{\text{replace}} \), which is also named as the replacing operation. The value of \( p_{\text{replace}} \) is utilized to control the probability to replace a cluster center by a randomly generated solution. This is used to avoid premature convergence and help individuals “jump out” of the local optima.

2.1.3. Creating Individuals. To maintain the diversity of population, a new idea (individual) can be generated based on one idea or two in one cluster or two, respectively. In the creating operation, BSO first randomly chooses one cluster or two according to a probability of \( p_{\text{one}} \). Then, in the basis of choosing one cluster or two, an idea of cluster center or a random idea is selected with a probability of \( p_{\text{one,center}} \) and \( p_{\text{two,center}} \). The selecting operation is defined below as

\[
X_{\text{select}} = \begin{cases} 
X_i & \text{one cluster} \\
\text{rand} \ast X_{i1} + (1 - \text{rand}) \ast X_{i2} & \text{two clusters,}
\end{cases}
\]

where \( \text{rand} \) is a random value between 0 and 1. After choosing one idea or two, the selected idea(s) is updated according to

\[
X_{\text{new}} = X_{\text{select}} + \xi \ast \text{normrnd}(0, 1),
\]

where \( \text{normrnd} \) is the Gaussian random value with mean 0 and variance 1 and \( \xi \) is an adjusting factor slowing the convergence speed down as the evolution goes, which can be expressed as

\[
\xi = \text{rand} \ast \log\left(\frac{0.5 \ast \text{max\_iteration} - \text{current\_iteration}}{k}\right),
\]

where \( \text{rand} \) is a random value between 0 and 1. The \( \text{max\_iteration} \) and \( \text{current\_iteration} \) denote the maximum number of iterations and current number of iterations, respectively. The \( \log\text{sig} \) is a logarithmic sigmoid transfer function, and such form is beneficial to global search ability at the beginning of the evolution and enhances local search ability when the process is approaching to the end. \( k \) is a predefined parameter for changing slopes of the \( \log\text{sig} \) function. The new created idea is evaluated, and if the fitness value is better than the current idea, the old idea will be replaced by the new one.

2.2. Differential Evolution. Differential evolution (DE) [5] is a population-based and stochastic optimization algorithm. Like other EAs, DE begins with an initial population vector containing a number of target individuals. The current generation evolves into the next generation through evolutionary operations repeatedly until the termination condition is attained. For the original DE, the mutation, crossover, and selection operators are defined as follows.

2.2.1. Mutation Operation. The mutation operation of classical DE scheme (DE/rand/bin) can be summarized as follows:

\[
V^k_i = X^k_{r1} + F \ast \left(X^k_{r2} - X^k_{r3}\right),
\]

where \( k \) is the generation number, the indices \( r1, r2, \) and \( r3 \) are mutually exclusive integers randomly chosen from the range between 1 and \( N \), and \( F \) is a mutation scaling factor which affects the differential variation between two individuals.

2.2.2. Crossover Operation. After the process of mutation, crossover operation is used to each pair of the target vector in order to enhance the potential diversity of the population. DE applies a crossover operator on \( X_i \) and \( V_i \) to generate the
offspring individual $U_i$ at the $k$th generation. The crossover operation is defined as

$$U_{ij}^k = \begin{cases} V_{ij}^k & \text{if rand} \leq CR \\ X_{ij}^k & \text{otherwise,} \end{cases} \quad (5)$$

where rand is a random value between 0 and 1 and CR is a parameter of crossover probability.

2.2.3. Selection Operation. The greedy selection is employed by means of comparing a parent and its corresponding offspring. The selection operation at the $k$th generation is described as

$$X_{ij}^{k+1} = \begin{cases} U_{ij}^k & \text{if } f(U_{ij}^k) \leq f(X_{ij}^k) \\ X_{ij}^k & \text{otherwise,} \end{cases} \quad (6)$$

where $f(X)$ is the objective function value of the trial vector $X$.

3. The Improved Algorithm: BSDE

In order to achieve better performance, an improved BSO algorithm needs to make use of both the global search information about the search space and the local search information of solutions found so far. The global search information can guide the search for exploring promising areas, while the local search information of solutions can be helpful for exploitation. In this paper, we effectively integrate the mutation and crossover scheme of DE into BSO in the idea creating operator of intracluster and intercluster. The key reason for integrating differential evolution strategy into BSO is that it can take advantage of DE that is mainly based on the distance and direction information and has the advantage of not being biased towards any prior defined guider.

Although the differential strategy has been utilized in [16] to design a BSO variant named Modified BSO (MBSO) and [10] to develop another BSO variant termed Close Loop BSO (CLBSO), our BSDE algorithm has significant differences with MBSO and CLBSO. On one hand, in the MBSO and CLBSO, a new idea $Y_i$ is created by adding the difference of two distinct random ideas $X_{r1}$ and $X_{r2}$ from all the current ideas to a current idea $X_i$ according to

$$Y_i = X_i + \text{rand} \ast (X_{r1} - X_{r2}), \quad (7)$$

where rand is a random value between 0 and 1 and indices $r1$ and $r2$ are mutually exclusive integers randomly chosen from the range between 1 and $N$. Even though formula (7) is similar to the mutation formula (4) of DE algorithm, they are obviously different in essence. In formula (4), $F$, the mutation scaling factor, is a real constant between 0 and 2, and the idea behind $F$ is to control the amplification of the differential variation. However, in formula (7), the factor, rand, represents a random direction scaling, and the scope of distance is less than $F$ in some cases. In mutation operator, our BSDE will be strictly in accordance with formula (4) of DE algorithm. On the other hand, the MBSO and CLBSO do not take the crossover operator between ideas into account, and our BSDE uses the crossover operator to enhance the performance of local exploitation according to formula (5).

Furthermore, in our BSDE, we mimic Osborn’s rule that different unusual ideas generating from different perspectives and suspending assumptions by participants are welcome in the brainstorming of human being. According to the inspiration, we add the differential evolution strategy into the normal idea to generate more different ideas. According to the above analysis, the differential evolution operators of BSDE algorithm include intracluster differential evolution operator and intercluster differential evolution operator. The detailed differential evolution operators of the BSDE algorithm are described below.

3.1. Differential Evolution Operator

3.1.1. Intracluster Differential Evolution Operator. In creating operator of BSO, if the selected idea is a normal one in one cluster according to the probability of $p_{one}$ and (1, $p_{one}$, center), we will let the idea to learn from the differential value of two random selected ideas and the cluster center. The differential evolution operator of a normal idea is defined as

$$X_{new} = X_{center} + F \ast (X_{r1} - X_{r2}), \quad (8)$$

where $F$ is the mutation scaling factor which affects the differential variation between two ideas and the indices $r1$ and $r2$ are mutually exclusive integers randomly chosen in the selected cluster. Then, according to formula (5), the crossover operation is used to generate new solutions by shuffling competing vectors and also to increase the diversity of the population.

Figure 1 shows the working process of intracluster differential evolution operator of a normal idea. The search space of the normal idea includes the area between two random ideas $X_{r1}$ and $X_{r2}$ and the cluster center $X_{center}$, and it can be seen that the local exploitation extends within the cluster space.

3.1.2. Intercluster Differential Evolution Operator. In creating operator of BSO, if the selected ideas are two normal ones in two clusters according to the probability of (1, $p_{one}$) and (1, $p_{two}$, center), and we will let the idea to learn from the differential value of two randomly selected normal ideas in two clusters and the best idea in all clusters, respectively.

![Figure 1: Flowchart showing the working of intracluster differential evolution operator.](image-url)
The differential evolution operator of two normal ideas is defined as

\[ X_{\text{new}} = \text{GlobalIdea} + F \cdot (X_{r1} - X_{r2}), \]

where \( F \) is the mutation scaling factor, GlobalIdea is the best idea in all clusters, and \( X_{r1} \) and \( X_{r2} \) denote the normal idea of cluster 1 and cluster 2. Then, according to formula (5), the crossover operation is used to generate new solutions.

Figure 2 shows the working process of two normal ideas in intercluster differential evolution operator. The search space of the two normal ideas includes the area between global best idea point and two normal ideas, and it can be seen that the global exploration extends between the two clusters.

According to the above analysis, the pseudocode of differential evolution operator about intracluster and intercluster is summarized in Algorithm 1.

3.2. A New Step Size Method. To adjust the convergence speed as the evolution goes in idea generation, the original BSO algorithm defines an adjusting factor \( \xi \) described by formula (3). Figure 3(a) shows the adjusting factor which controls the scale of step. We can observe that at first the adjusting factor keeps around 1, while when half the number of generations has been reached, it rapidly turns to be near 0. This method to control the size of step can also balance exploration and exploitation at different searching generations. However, it just takes effect only for very short interval. Hence, we introduce a simple new step size method which is shown in Figure 3(b). The dynamic mutation function is described as follows:

\[ \xi = \text{rand} \cdot \exp \left( 1 - \frac{\text{max\_iteration}}{\text{max\_iteration} - \text{current\_iteration} + 1} \right), \]

where \( \text{rand} \) is a random value between 0 and 1. The \( \text{max\_iteration} \) and \( \text{current\_iteration} \) denote the maximum number of iterations and current number of iterations, respectively.

3.3. Flowchart and Pseudocode of the BSODE. As previously analyzed, the complete flowchart of the BSODE algorithm is shown in Figure 4.

This efficient differential evolution operator in intercluster improves the global search capability and avoids convergence to local minima. At the same time, the differential evolution operator in intracluster extends the capability of local exploitation and accelerates the convergence. The pseudocode of the BSODE is summarized in Algorithm 2.

4. Benchmark Functions

Optimization Problem

The numerical benchmark functions are used for comparing the performance and accuracy of our proposed BSODE algorithm with the original BSO algorithm and other variants of DE in this section. For a fair comparison, all the experiments are conducted on the same machine with an Intel 3.4 GHz
Figure 3: The comparison chart of adjusting factor between BSO and BSODE.

Figure 4: Flow chart of BSODE algorithm.
Begin
Randomly initialize \( N \) ideas and evaluate their fitness
Initialize \( m \) cluster centers (\( m < N \))
while (stopping condition not met)
Cluster \( N \) ideas into \( m \) clusters %clustering operation
Record the best idea in each cluster as the cluster center
for \( (i = 1 \) to \( N \)) %creating operation
Execute Algorithm 1: differential evolution operator()
Create \( X_{\text{new}} \) using \( X_{\text{new}} \) according formula (9)
Accept \( X_{\text{new}} \) if \( f(X_{\text{new}}) \) is better than \( f(X_{i}) \)
end for
end while
end

Algorithm 2: Pseudocode of the BSODE().

Table 1: Numerical benchmark function definition (CEC2005 \( F1 \)–\( F14 \)).

<table>
<thead>
<tr>
<th>Number</th>
<th>Function name</th>
<th>Type</th>
<th>Low range</th>
<th>Up range</th>
<th>Optima</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Shifted sphere</td>
<td>Unimodal</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>F2</td>
<td>Shifted Schwefel</td>
<td>Unimodal</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>F3</td>
<td>Shifted rotated high conditioned elliptic</td>
<td>Unimodal</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>F4</td>
<td>Shifted Schwefel's problem 1.2 with noise</td>
<td>Unimodal</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>F5</td>
<td>Schwefel's problem 2.6</td>
<td>Unimodal</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>F6</td>
<td>Shifted Rosenbrock</td>
<td>Multimodal</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>F7</td>
<td>Shifted rotated Griewank</td>
<td>Multimodal</td>
<td>0</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td>F8</td>
<td>Shifted rotated Ackley</td>
<td>Multimodal</td>
<td>-32</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>F9</td>
<td>Shifted Rastrigin</td>
<td>Multimodal</td>
<td>-5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>F10</td>
<td>Shifted rotated Rastrigin</td>
<td>Multimodal</td>
<td>-5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>F11</td>
<td>Shifted rotated Weierstrass</td>
<td>Multimodal</td>
<td>-0.5</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>F12</td>
<td>Schwefel's problem 2.13</td>
<td>Multimodal</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>F13</td>
<td>Expanded extended Griewank and Rosenbrock</td>
<td>Expanded</td>
<td>-3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F14</td>
<td>Expanded rotated extended Scaffes</td>
<td>Expanded</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>

CPU, 4 GB memory. The operating system is Windows 7 with MATLAB 8.0 (R2012b). For the purpose of reducing statistical errors, each function is independently run 25 times.

4.1. Benchmark Functions. We choose 14 widely known rotated or shifted benchmark functions used in CEC2005, which is given in Table 1 [19]. All functions are tested on 30 dimensions. Results obtained are compared with some other variants of DE algorithms defined in the next subsection. The searching range and true optima for all functions are also given in Table 1. Among 14 benchmark functions, \( F1 \) to \( F5 \) are shifted unimodal functions, \( F6 \) to \( F12 \) are shifted or rotated multimodal functions, and \( F13 \) to \( F14 \) are expanded shifted or rotated functions.

4.2. Parameter Settings of Comparative Algorithms. Because the BSODE algorithm combines the original BSO and the differential evolution strategy, we specially compare the performance of BSODE with the DE, the original BSO, and the PSO. Besides, we also compare the performance of BSODE with other variants of DE. We choose two DE variants, that is, CoDE [21] and SaDE [22]. To eliminate influences of statistical errors, each problem function is independently run 25 times which is a prescribed evaluation criterion in CEC2005 [19]. For all approaches, the population size is set to 30. The same stopping criterion is used in all algorithms, that is, reaching certain number of iterations or function evaluations (FEs). In our experiments, we have run all
Table 3: The results of solution accuracy for new step size.

<table>
<thead>
<tr>
<th>Function</th>
<th>BSODE</th>
<th>BSO</th>
<th>BSO-newstep</th>
<th>BSODE-nonewstep</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>$4.12E-20 \pm 5.77E-20$</td>
<td>$2.65E-03 \pm 3.03E-03$</td>
<td>$2.60E-16 \pm 4.09E-16$</td>
<td>$3.14E-11 \pm 6.35E-11$</td>
</tr>
<tr>
<td>$F_2$</td>
<td>$5.19E-07 \pm 3.87E-07$</td>
<td>$2.76E-00 \pm 8.16E-01$</td>
<td>$4.08E-05 \pm 1.34E-05$</td>
<td>$1.58E+00 \pm 5.86E-01$</td>
</tr>
<tr>
<td>$F_3$</td>
<td>$5.73E+05 \pm 2.37E+05$</td>
<td>$1.33E+06 \pm 3.58E+05$</td>
<td>$8.39E+05 \pm 3.03E+05$</td>
<td>$1.67E+06 \pm 5.61E+05$</td>
</tr>
<tr>
<td>$F_4$</td>
<td>$2.09E-01 \pm 1.73E-01$</td>
<td>$1.29E+04 \pm 9.06E+03$</td>
<td>$1.29E-01 \pm 1.15E-01$</td>
<td>$1.98E+03 \pm 4.05E+03$</td>
</tr>
<tr>
<td>$F_5$</td>
<td>$2.92E+04 \pm 3.48E+02$</td>
<td>$3.12E+04 \pm 6.02E+02$</td>
<td>$3.02E+04 \pm 5.04E+02$</td>
<td>$2.93E+04 \pm 2.90E+02$</td>
</tr>
<tr>
<td>$F_6$</td>
<td>$1.55E+03 \pm 2.64E+03$</td>
<td>$1.52E+03 \pm 2.71E+03$</td>
<td>$1.86E+03 \pm 2.66E+03$</td>
<td>$1.77E+03 \pm 3.78E+03$</td>
</tr>
<tr>
<td>$F_7$</td>
<td>$8.36E-03 \pm 1.05E-02$</td>
<td>$5.35E+03 \pm 2.32E+02$</td>
<td>$5.07E+03 \pm 1.18E+02$</td>
<td>$5.45E-02 \pm 3.39E-02$</td>
</tr>
<tr>
<td>$F_8$</td>
<td>$2.02E+01 \pm 6.69E-02$</td>
<td>$2.05E+01 \pm 1.17E-01$</td>
<td>$2.02E+01 \pm 2.74E-02$</td>
<td>$2.04E+01 \pm 1.10E-01$</td>
</tr>
<tr>
<td>$F_9$</td>
<td>$4.26E+01 \pm 1.37E+01$</td>
<td>$4.97E+01 \pm 1.02E+01$</td>
<td>$2.45E+01 \pm 7.20E+00$</td>
<td>$8.67E+01 \pm 1.23E+01$</td>
</tr>
<tr>
<td>$F_10$</td>
<td>$3.47E+01 \pm 1.21E+01$</td>
<td>$4.59E+01 \pm 1.19E+01$</td>
<td>$3.49E+01 \pm 5.54E+00$</td>
<td>$8.50E+01 \pm 2.19E+01$</td>
</tr>
<tr>
<td>$F_11$</td>
<td>$1.40E+01 \pm 2.00E+00$</td>
<td>$2.43E+01 \pm 2.58E+00$</td>
<td>$1.60E+01 \pm 1.98E+00$</td>
<td>$2.69E+01 \pm 2.72E+00$</td>
</tr>
<tr>
<td>$F_12$</td>
<td>$7.34E+03 \pm 7.32E+03$</td>
<td>$3.16E+04 \pm 1.70E+04$</td>
<td>$6.81E+03 \pm 3.50E+03$</td>
<td>$3.61E+04 \pm 2.33E+04$</td>
</tr>
<tr>
<td>$F_13$</td>
<td>$3.88E+00 \pm 8.18E-01$</td>
<td>$5.56E+00 \pm 1.35E+00$</td>
<td>$4.70E+00 \pm 4.33E-01$</td>
<td>$5.30E+00 \pm 2.05E+00$</td>
</tr>
<tr>
<td>$F_14$</td>
<td>$1.29E+01 \pm 4.32E-01$</td>
<td>$1.32E+01 \pm 3.92E-01$</td>
<td>$1.26E+01 \pm 4.44E-01$</td>
<td>$1.33E+01 \pm 3.11E-01$</td>
</tr>
</tbody>
</table>

+/ = /− \hfill 10/4/0 \hfill 7/6/1 \hfill 9/5/0

algorithms on the benchmark functions using the same FEs of 10000 * D for a fair comparison, where D is the size of the problem dimension. The parameter settings of all the algorithms are given in Table 2.

4.3. The Contribution of the New Step Size Method to BSODE. To verify the rationality and effectiveness of the new step size in BSODE and to fully understand the effect of the new step size method, herein, we investigate the contribution of the new step size to BSODE. Considering the consistency of the paper, we compare all 14 benchmark functions with 25 independent runs. We test the BSO, BSO with new step size (BSO-newstep), BSODE, and the original adjusting factor, that is, without new step size (BSODE-nonewstep). The experimental results are presented in Table 3.

Table 3 shows that the BSO-newstep (BSO with new step size) has better performance than the original BSO and BSODE has better performance than the BSODE-nonewstep (BSODE without new step size), and it can be concluded that the new step size plays a significant role in BSODE algorithm. This is in good agreement with our analysis in Section 3.2 that the control of the new step size can balance exploration and exploitation at different searching generations in Section 3.2.

4.4. Comparison of Results

4.4.1. Comparisons on Solution Accuracy. The results of solution accuracy are given in Table 4 in terms of the mean optimum solution and the standard deviation of the solutions obtained in the 25 independent runs by each algorithm over 300,000 FEs on 14 benchmark functions. In all experiments, the dimensions of all problems are 30. In each row of the table, the mean values are listed in the first part, and the standard deviations are listed in the last part, and the two parts are divided with a symbol "+/−." The best results among the algorithms are displayed in bold.

From Table 4, it can be observed that the mean value and the standard deviation value of the original BSO algorithm performs well only for function $F_6$ compared with the BSODE. The BSODE algorithm performs better than the other algorithms for 4 functions $F_4$, $F_7$, $F_8$, and $F_{10}$, especially for $F_4$, and the mean value and the standard deviation value are far better than the other five algorithms. PSO outperforms the other algorithms in terms of solution accuracy only for function $F_{12}$, and DE outperforms the other algorithms in terms of solution accuracy only for functions $F_3$ and $F_5$. Although CoDE performs better for 5 functions $F_1$, $F_6$, $F_9$, $F_{11}$, and $F_{14}$ and SaDE performs better for 3 functions $F_1$, $F_2$, and $F_3$, we can conclude that the BSODE algorithm remains still a good performance of the solution accuracy for complex shifted and rotated benchmark functions.

4.4.2. The Comparison Results of Convergence Speed. Figure 5 shows the convergence graphs of the compared 6 algorithms in terms of the mean fitness values of 25 runs. Due to page limitation, only graphs for 3 functions $F_1$, $F_7$, and $F_{14}$ which include shifted, shifted rotated, and expanded rotated extended functions are shown. From Figure 5, we can observe that BSODE has fast convergence speed compared to the original BSO or similar convergence speed compared to the other four algorithms.

4.4.3. The Comparison Results of t-Tests. To statistically compare BSODE with the other five algorithms, t-test [23, 24] is carried out to show the differences between BSODE and the other five algorithms. The $t$ values on every function by two-tailed test with a significance level of 0.05 between BSODE and other algorithms are given in Table 5. Better, same, and worse give the number of experiments that BSODE performs significantly better than, almost the same as, and significantly worse than the compared algorithm under the same conditions, respectively.

Table 5 gives information about the $t$ value, $p$ value, and $h$ value of 25 independent runs of 6 algorithms over 300,000 FEs on 14 test functions. The best results among the algorithms are shown in bold.
Table 4: The results of solution accuracy.

<table>
<thead>
<tr>
<th>Function</th>
<th>BSODE</th>
<th>BSO</th>
<th>PSO</th>
<th>DE</th>
<th>CoDE</th>
<th>SaDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>$4.12E-20$ ± $5.77E-20$</td>
<td>$2.65E-03$ ± $3.03E-03$</td>
<td>$4.67E+03$ ± $2.86E+03$</td>
<td>$5.46E-01$ ± $2.59E+00$</td>
<td>$0.00E+00$ ± $0.00E+00$</td>
<td>$0.00E+00$ ± $0.00E+00$</td>
</tr>
<tr>
<td>F2</td>
<td>$5.19E-07$ ± $3.87E-07$</td>
<td>$2.76E+00$ ± $8.16E-01$</td>
<td>$6.51E+03$ ± $8.15E+03$</td>
<td>$1.19E-01$ ± $5.88E-01$</td>
<td>$9.11E-06$ ± $1.11E-05$</td>
<td>$5.55E-10$ ± $6.99E-10$</td>
</tr>
<tr>
<td>F3</td>
<td>$5.73E+05$ ± $2.37E+05$</td>
<td>$1.33E+06$ ± $3.58E+05$</td>
<td>$2.29E+07$ ± $4.93E+07$</td>
<td>$3.65E+05$ ± $1.69E+05$</td>
<td>$3.65E+05$ ± $1.69E+05$</td>
<td>$3.65E+05$ ± $1.69E+05$</td>
</tr>
<tr>
<td>F4</td>
<td>$2.09E-01$ ± $1.73E-01$</td>
<td>$1.29E+04$ ± $9.06E+03$</td>
<td>$4.56E+03$ ± $6.54E+03$</td>
<td>$1.25E+02$ ± $5.87E+02$</td>
<td>$4.25E+01$ ± $1.25E+01$</td>
<td>$4.25E+01$ ± $1.25E+01$</td>
</tr>
<tr>
<td>F5</td>
<td>$2.92E+04$ ± $3.48E+02$</td>
<td>$3.12E+04$ ± $6.02E+02$</td>
<td>$3.43E+04$ ± $2.28E+03$</td>
<td>$2.93E+04$ ± $2.11E+04$</td>
<td>$2.93E+04$ ± $2.11E+04$</td>
<td>$2.93E+04$ ± $2.11E+04$</td>
</tr>
<tr>
<td>F6</td>
<td>$8.36E-03$ ± $1.05E-02$</td>
<td>$5.35E+03$ ± $2.32E+02$</td>
<td>$5.36E+03$ ± $5.34E+02$</td>
<td>$4.70E+03$ ± $9.47E+03$</td>
<td>$4.70E+03$ ± $9.47E+03$</td>
<td>$4.70E+03$ ± $9.47E+03$</td>
</tr>
<tr>
<td>F7</td>
<td>$2.02E+01$ ± $6.69E-02$</td>
<td>$2.05E+01$ ± $1.17E-01$</td>
<td>$2.09E+01$ ± $6.56E-02$</td>
<td>$2.09E+01$ ± $7.37E-02$</td>
<td>$2.09E+01$ ± $7.37E-02$</td>
<td>$2.09E+01$ ± $7.37E-02$</td>
</tr>
<tr>
<td>F8</td>
<td>$4.26E+01$ ± $1.37E+01$</td>
<td>$4.97E+01$ ± $1.02E+01$</td>
<td>$6.90E+01$ ± $2.28E+01$</td>
<td>$2.18E+01$ ± $5.63E+00$</td>
<td>$1.95E+00$ ± $1.27E+00$</td>
<td>$1.95E+00$ ± $1.27E+00$</td>
</tr>
<tr>
<td>F9</td>
<td>$3.47E+01$ ± $1.21E+01$</td>
<td>$4.59E+01$ ± $1.19E+01$</td>
<td>$1.26E+01$ ± $3.82E+01$</td>
<td>$4.28E+01$ ± $1.59E+01$</td>
<td>$3.98E+01$ ± $1.55E+01$</td>
<td>$4.75E+01$ ± $1.13E+01$</td>
</tr>
<tr>
<td>F10</td>
<td>$1.40E+01$ ± $2.00E+00$</td>
<td>$2.43E+01$ ± $2.58E+00$</td>
<td>$2.02E+01$ ± $2.59E+00$</td>
<td>$2.30E+01$ ± $6.15E+00$</td>
<td>$2.30E+01$ ± $6.15E+00$</td>
<td>$2.30E+01$ ± $6.15E+00$</td>
</tr>
<tr>
<td>F11</td>
<td>$7.34E+03$ ± $7.32E+03$</td>
<td>$3.16E+04$ ± $1.70E+04$</td>
<td>$4.83E+03$ ± $5.14E+03$</td>
<td>$2.08E+05$ ± $8.19E+04$</td>
<td>$1.31E+05$ ± $2.54E+04$</td>
<td>$9.67E+04$ ± $1.47E+04$</td>
</tr>
<tr>
<td>F12</td>
<td>$3.88E+00$ ± $8.18E-01$</td>
<td>$5.56E+00$ ± $1.35E+00$</td>
<td>$3.65E+00$ ± $1.14E+00$</td>
<td>$3.24E+00$ ± $5.77E-01$</td>
<td>$4.02E+00$ ± $1.22E+00$</td>
<td>$2.76E+00$ ± $3.60E-01$</td>
</tr>
<tr>
<td>F13</td>
<td>$1.29E+01$ ± $4.23E-01$</td>
<td>$1.32E+01$ ± $3.92E-01$</td>
<td>$1.21E+01$ ± $5.27E-01$</td>
<td>$1.30E+01$ ± $2.66E-01$</td>
<td>$1.25E+01$ ± $3.82E-01$</td>
<td>$1.27E+01$ ± $1.83E-01$</td>
</tr>
</tbody>
</table>
For example, comparing BSODE and DE, the former significantly outperformed the latter in 7 functions (F2, F4, F7, F8, F10, F11, and F12), does the same as the latter in 3 functions (F6, F13, and F14), and does worse in 4 function (F1, F3, F5, and F9), yielding a "General Merit" value $7 - 4 = 3$, indicating that the BSODE generally outperforms the DE. Although it performed slightly weaker in some functions, the BSODE in general offered much improved performance than other compared algorithms, as confirmed in Table 5.

5. Applications of ANN Using BSODE

Artificial neural network (ANN) [25] simulates the learning pattern of human brain for solving many difficult problems in the field of applications such as prediction of principal ground-motion parameters [26] and prediction of peak ground acceleration [27]. ANN is a flexible mathematical scheme which is capable of identifying complex nonlinear relationships between input data and output data. The task to train ANN needs vigorous optimization techniques because the search space is highly dimensional and multimodal which suffers from noises and data in general. The usually used training method is Back-Propagation (BP) algorithm [28]. However, BP algorithm has some potential problems. On one hand, for the complex pattern classification problems [29], BP algorithm may lightly get trapped into local minima. On the other hand, the training time of the BP algorithm is very long for complex nonlinear problems.
In recent decades, in order to deal with the complicated training problem of the ANN, many metaheuristic optimization algorithms, such as Simulated Annealing (SA) [26, 27], GA [30], and PSO [31], have been utilized to optimize the weights and biases of ANN. Besides, the recently proposed EvoNN [32, 33] algorithm, which utilizes the multiobjective optimization technique in the training process of a feed-forward neural network, ensures correct neural training by
working out a Pareto tradeoff between the accuracy of the training and the complexity of the network. In this section, we use our BSODE algorithm to adjust connection weights and biases of ANN. To validate the usefulness and the efficiency of our BSODE algorithm, two types of applications including time series prediction and System Identification are illustrated.

5.1. Training ANN with BSODE Algorithm. The structure of training ANN using our BSODE algorithm is a three-layer feed-forward network. The basic structure of the proposed scheme is given in Figure 6.

In our experiments, activation function used in the input layer is a linear combination of input variables, activation function used in the hidden layer is the sigma function, and activation function used in the output layer is a linear combination of output variables of hidden layer. In training ANN, the aim is to find a set of weights and biases with the smallest error value. Herein, the objective function is the mean sum of squared errors (MSE) which is given as follows:

\[
MSE = \frac{1}{Q \times K} \sum_{i=1}^{Q} \sum_{j=1}^{K} \frac{1}{2} (d_{ij} - y_{ij})^2,
\]

where \(Q\) is the number of training data sets, \(K\) is the number of output units, \(d_{ij}\) is desired output, and \(y_{ij}\) is output inferred from ANN.

5.2. Time Series Prediction of Lorenz Chaotic System. The first application is the time series prediction of Lorenz chaotic system described by the following equation [34]:

\[
\begin{align*}
\dot{x}(t) &= a(y(t) - x(t)) \quad (12a) \\
\dot{y}(t) &= bx(t) - x(t)z(t) - y(t) \quad (12b) \\
\dot{z}(t) &= x(t)y(t) - cz(t), \quad (12c)
\end{align*}
\]

where \(a, b, \) and \(c\) are parameters of the Lorenz chaotic system. The system is chaotic when \(a = 10, b = 28,\) and \(c = 8/3.\) The differential equations were solved numerically using 4th order Runge-Kutta integration with a step size 0.01 and initial values \(x(0) = 0, y(0) = -1,\) and \(z(0) = 0.\) The\( x\)-coordinate of the Lorenz time series system is chosen for prediction. The goal is to construct the prediction model of chaotic time series according to the following formula:

\[
y(t + 1) = f(y(t - 18), y(t - 12), \ldots, y(t - 6), y(t)).
\]

In order to train the ANN, we choose 10000 data sets according to the real model, where the first 8000 data sets are discarded; the rest normalized 2000 data sets are considered as experiment data. The former 1500 points are selected as the training data points and the rest 500 points are selected as the testing data points to test the validity of the model.

In this experiment, the parameter settings of the six algorithms are the same as those in Section 4.2, and the maximal generation used as ended condition of algorithm is 100 for BSODE, BSO, PSO, DE, CoDE, and SaDE. The results are shown in Table 6 in terms of the mean optimum solution, the standard deviation of the solutions, and run time independent runs by each algorithm.

As is seen in Table 6, SaDE has the worst MSE in both training and testing cases. BSODE performs better than BSO, PSO, DE, CoDE, and SaDE in the mean of the MSE. The model trained by six algorithms follows the chaotic behavior of the Lorenz chaotic time series which is demonstrated in Figure 7. It can be concluded that BSODE is the most effective learning algorithm for Lorenz chaotic time series even though BSODE needs more running time than PSO algorithm.

5.3. Nonlinear System Identification. The second application is to build a three-layer feed-forward ANN with three input units, five hidden units, and one output unit. The ANN is used to model the curve of a nonlinear function which is given by the following equation:

\[
y(t + 1) = \frac{y(t) y(t - 1) [y(t) + 2.5]}{1 + y^2(t) + y^2(t - 1)} + u(t)
\]

\[
u(t) = \sin \left( \frac{2\pi t}{25} \right).
\]

The simulation target is to construct the approximation model of the nonlinear function according to the following formula:

\[
y(t + 1) = f(u(t), y(t - 1), y(t)).
\]

To train the ANN, we choose 200 pairs of data from the real model. In addition, to further evaluate the performance of BSODE algorithm, we add 20 db additive white Gaussian noise into the 200 pairs of data.

In this experiment, population size of each algorithm is 30, and the maximal generation used as termination
Table 6: Comparisons of MSE on Lorenz chaotic time series over 25 runs.

<table>
<thead>
<tr>
<th>Performance</th>
<th>BSODE</th>
<th>BSO</th>
<th>PSO</th>
<th>DE</th>
<th>CoDE</th>
<th>SaDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training error</td>
<td>3.33E − 03 ± 2.02E − 03</td>
<td>3.81E − 03 ± 1.70E − 03 (+)</td>
<td>4.01E − 03 ± 2.30E − 03 (+)</td>
<td>1.03E − 02 ± 5.95E − 03 (+)</td>
<td>4.84E − 03 ± 1.97E − 03 (+)</td>
<td>9.97E − 02 ± 1.11E − 01 (+)</td>
</tr>
<tr>
<td>Testing error</td>
<td>3.48E − 03 ± 2.15E − 02</td>
<td>3.99E − 03 ± 1.78E − 03 (+)</td>
<td>4.21E − 03 ± 3.20E − 03 (+)</td>
<td>1.06E − 02 ± 6.08E − 03 (+)</td>
<td>4.99E − 03 ± 2.07E − 03 (+)</td>
<td>9.74E − 02 ± 1.13E − 01 (+)</td>
</tr>
<tr>
<td>Run time(s)</td>
<td>7.39E + 01 ± 8.30E − 01</td>
<td>7.41E + 01 ± 6.60E − 01 (+)</td>
<td>7.11E + 01 ± 3.07E + 00 (-)</td>
<td>7.35E + 01 ± 4.85E − 01 (-)</td>
<td>2.13E + 02 ± 3.05E + 00 (+)</td>
<td>7.14E + 01 ± 4.99E − 01 (-)</td>
</tr>
</tbody>
</table>
Figure 7: Comparison of the performance curves for Lorenz chaotic time series prediction.
Table 7: Comparisons between BSODE and other algorithms on MSE over 25 runs.

<table>
<thead>
<tr>
<th>App.</th>
<th>BSODE</th>
<th>BSO</th>
<th>PSO</th>
<th>DE</th>
<th>CoDE</th>
<th>SaDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without noise</td>
<td>6.69E−03 ± 3.89E−03</td>
<td>7.78E−03 ± 3.87E−03</td>
<td>2.90E−03 ± 1.69E−03</td>
<td>9.89E−03 ± 1.69E−02</td>
<td>2.90E−03 ± 1.69E−03</td>
<td>9.89E−03 ± 1.69E−02</td>
</tr>
<tr>
<td>With noise</td>
<td>3.20E−02 ± 3.55E−03</td>
<td>4.08E−02 ± 6.24E−03</td>
<td>3.28E−02 ± 3.66E−03</td>
<td>3.27E−02 ± 1.55E−02</td>
<td>3.63E−02 ± 4.14E−03</td>
<td>3.51E−02 ± 4.40E−03</td>
</tr>
</tbody>
</table>
(a) System Identification without noise using BSODE

(b) System Identification with noise using BSODE

(c) System Identification without noise using BSO

(d) System Identification with noise using BSO

(e) System Identification without noise using PSO

(f) System Identification with noise using PSO

Figure 8: Continued.
Figure 8: Comparison of the performance curves for System Identification.
condition of each algorithm is 1000 for BSODE, BSO, PSO, DE, CoDE, and SaDE. The parameter settings of the six algorithms are also the same as in Section 4.2. The results are given in Table 7 in terms of the mean MSE and the standard deviation obtained in the 25 independent runs for six algorithms.

From Table 7, in case without noise, we observed that BSODE performs worse than SaDE, CoDE, and PSO, but BSODE performs better than BSO and DE in terms of the mean MSE. However, in case of 20 db additive white Gaussian noise, BSODE performs better than all other five algorithms. Figure 8 shows that the output of the identified system tracks the target output for three algorithms. We can conclude that BSODE has the best performance in case without noise and a better performance compared with BSO and DE in building the approximation model.

6. Conclusion

In this paper, an improved BSO with differential evolution strategy and new step size method named BSODE is proposed for solving complex optimization problems and training artificial neural networks. Experiments on the 14 widely used benchmark functions of CEC 2005 and the two real-world applications of artificial neural network are carried out to compare the proposed BSODE with five state-of-the-art algorithms in this paper. From these results, we can draw the conclusion that the BSODE significantly improves the performance of the original BSO. Moreover, the BSODE gives good performance on the training of artificial neural networks compared with other algorithms. We expect that BSODE will be used to solve more real-world global optimization problems. Further work includes research into neighbor search to make the algorithm more efficient. In the near future, we also plan to apply the algorithm to large scale optimization domains.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was partially supported by National Natural Science Foundation of China (under Grants nos. 61272283, 61403304, 61304082, and 61273367), the Project of Department of Education Science Research of Shaanxi Province, China (under Grants nos. 14JK1360 and 14JK1511), and the Project of Science Research Plan in Xi’an of China (nos. CXY1437 (4) and CXY1439 (5)).

References


Submit your manuscripts at
http://www.hindawi.com