Research Article

Optimizing High-Dimensional Functions with an Efficient Particle Swarm Optimization Algorithm

Guoliang Li,¹ Jinhong Sun,¹ Mohammad N.A. Rana,¹ Yinglei Song ,¹ Chunmei Liu,² and Zhi-yu Zhu¹

¹School of Electronics and Information Science, Jiangsu University of Science and Technology, Zhenjiang 212003, China

²Department of Electrical Engineering and Computer Science, Howard University, Washington, DC 20059, USA

Correspondence should be addressed to Yinglei Song; syinglei2013@163.com

Received 22 December 2019; Revised 20 May 2020; Accepted 17 June 2020; Published 9 July 2020

Academic Editor: Ricardo Perera

Copyright © 2020 Guoliang Li 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.

The optimization of high-dimensional functions is an important problem in both science and engineering. Particle swarm optimization is a technique often used for computing the global optimum of a multivariable function. In this paper, we develop a new particle swarm optimization algorithm that can accurately compute the optimal value of a high-dimensional function. The iteration process of the algorithm is comprised of a number of large iteration steps, where a large iteration step consists of two stages. In the first stage, an expansion procedure is utilized to effectively explore the high-dimensional variable space. In the second stage, the traditional particle swarm optimization algorithm is employed to compute the global optimal value of the function. A translation step is applied to each particle in the swarm after a large iteration step is completed to start a new large iteration step. Based on this technique, the variable space of a function can be extensively explored. Our analysis and testing results on high-dimensional benchmark functions show that this algorithm can achieve optimization results with significantly improved accuracy, compared with traditional particle swarm optimization algorithms and a few other state-of-the-art optimization algorithms based on particle swarm optimization.

1. Introduction

In both science and engineering, the particle swarm optimization (PSO) algorithm [1] is an important optimization technique that has been extensively used to find the global optima of multivariable functions. The PSO algorithm searches the variable space of a multivariable function by simulating the social behaviors of a group of animals. The animals in the group act cooperatively to find the potential location of the global optimum point, and the search pattern of each particle in the group is adjusted based on the exploring experiences of its own and other particles in the group.

Although the PSO algorithm can effectively search the variable space of a large number of multivariable functions and accurately determine their global optima, it has two disadvantages that may adversely affect its performance on some multivariable functions. Specifically, the speed of

convergence is slow, especially in cases where the function that needs to be optimized is defined on a high-dimensional space. In addition, the algorithm is not guaranteed to find a global optimum and may converge to a local minimum point when the dimensionality of the function is large. A large number of techniques thus have been developed to further improve the performance of the PSO algorithm in the past two decades.

One strategy to improve the performance of the traditional PSO algorithm is to divide the individuals in the swarm into subgroups. In [2], the concept of subpopulation and a reproduction operator is introduced into the PSO algorithm. In [3], a dynamic multiswarm PSO algorithm is developed. Recently, a symbiotic particle swarm optimization (SPSO) algorithm is developed in [4] to optimize the neural fuzzy networks. In [5], the traditional PSO algorithm is modified to solve multimodal function optimization problems. In [6], the particles in the swarm are divided into

mentors, mentees, and independent learner groups based on differences in fitness values and the Euclidian distances from the particle with the best fitness value. As a result, a new dynamic mentoring and self-regulation-based particle swarm optimization (DMeSR-PSO) algorithm is developed. Testing results show that DMeSR-PSO algorithm can achieve excellent and reliable performance on a number of benchmark datasets.

Another technique often used to improve the performance of PSO algorithms is to develop new rules for updating the positions of the particles. In [7], the cognitive and social behaviors of the swarm are randomized based on chaotic sequences and Gaussian distribution. Particles can thus search in a much broader area than the traditional PSO algorithm. In [8], a chaotic PSO algorithm is developed based on a virtual quadratic objective function constructed from the optimum point of an individual particle and the global optimum point. Recently, in [9], an improved PSO algorithm is developed to improve the performance of the traditional PSO by using a new strategy to move each particle. Specifically, particles in the PSO fly to their own predefined target instead of the location of the global optimum that has been found so far.

A third class of methods improves the performance of optimization by designing new strategies to update the velocities of particles. In [10], a self-adaptive PSO algorithm (SAPSO-MVS) that uses a novel velocity updating strategy to balance the exploring and exploiting capabilities of the PSO algorithm is developed to improve the performance. In [11], a new PSO-based optimization algorithm that can adaptively adjust the particle velocities in each step of iteration is developed. The adjustment is based on the current distance of each particle from the location with the best fitness found so far by all particles. Recently, in [12], a mode-dependent velocity updating approach is developed to balance the capabilities of the PSO algorithm for local search and those for the global search, which can effectively reduce the chance to be trapped within local minima.

Recently, in addition to the methods discussed above, a large number of other techniques [2, 3, 6, 8, 13–31] have been developed to further enhance the performance of PSO-based optimization algorithms, and a comprehensive survey of these methods is available in [31]. In general, most of the existing techniques require sophisticated processing of the locations and fitness values of particles to determine the location and velocity of each particle for the next step of iteration [32, 33]. These approaches thus may become computationally inefficient when the function to be optimized is of a large dimensionality [34]. In practice, the optimization of these functions is often important [35]. For example, a protein sequence generally contains a few hundred amino acids. An amino acid usually consists of around 10 atoms. A protein sequence of moderate length is thus comprised of a few thousand atoms. The native folding of a protein sequence is generally predicted by minimizing the free energy of the system formed by the atoms in the sequence. The free energy of a protein sequence is thus often a function of a few thousand variables, and the accuracy of its minimization is crucial for determining the native folding of a protein sequence.

In this paper, we develop a new efficient PSO algorithm for the optimization of high-dimensionality functions. The algorithm performs optimization by a number of large iteration steps. Each large iteration step consists of two stages. In the first stage, an expelling force field is applied to each particle in the swarm to significantly enlarge the area the particles can explore. In the second stage, the standard PSO algorithm is applied to the swarm to converge to a global optimum. After a large iteration step is completed, a translation procedure is applied to each particle in the swarm to further expand the region explored by the swarm. An iteration step in this new algorithm can be efficiently performed, and the algorithm thus can be applied to the optimization of high-dimensional functions. Our analysis and testing results on high-dimensional benchmark functions show that this new algorithm can achieve significantly improved performance on optimization, compared with the traditional PSO algorithm and its a few existing variants.

2. The Proposed Algorithm

The standard PSO algorithm performs the optimization of a function by simulating the social behavior of a swarm of animals. Specifically, a swarm of particles with randomly generated initial locations is created in the variable space of the function. The fitness of a location in the variable space of the function is defined to be the function value of the location. Each particle is also assigned a randomly generated velocity. In each iteration step, the location with the best fitness value that has been found by all individuals in the swarm is determined. In addition, the location with the best fitness in the trajectory of each individual is also determined. The velocity of each particle is adjusted based on the location with the globally best fitness value and the one with the best fitness value in its own trajectory. The location of each particle is then updated based on the current velocity of the particle.

Given a swarm of m particles, let $x_{i,1}, x_{i,2}, \dots, x_{i,m}$ be the locations of the particles in the i th step of iteration and $v_{i,1}, v_{i,2}, \dots, v_{i,m}$ be their velocities, respectively. $g_{b,i}$ denotes the location with the globally best fitness value that has been found by the swarm after i steps of iteration and b_k is the location with the best fitness value in the trajectory of particle k ($1 \leq k \leq m$). The velocity of particle k in the $i + 1$ th step of iteration can be computed as follows:

$$v_{i+1,k} = \omega v_{i,k} - c_1 r_1 (x_{i,k} - g_{b,i}) - c_2 r_2 (x_{i,k} - b_k), \quad (1)$$

where c_1 and c_2 are two positive constants and r_1 and r_2 are two randomly generated numbers between 0 and 1; ω is a positive constant between 0 and 1. The location of particle k in the $i + 1$ th step of iteration can be computed from $v_{i+1,k}$ as follows:

$$x_{i+1,k} = x_{i,k} + v_{i+1,k}. \quad (2)$$

To effectively enhance the ability of a PSO algorithm to explore within the variable space of a high-dimensional function, we propose to perform the procedure of optimization by a number of *large iteration steps*. Each large

iteration step contains two stages. In the first stage, the particles in the swarm are expelled from the center of mass for the particles in the swarm to explore a wider area in the variable space. This stage is the *exploring stage* of the optimization procedure. After the exploring stage is completed, the standard PSO algorithm is then applied to compute the global optimum of the function. The second stage is thus the *converging stage* of the optimization procedure.

In the exploring stage, the velocity of particle k in the $i + 1$ th step of iteration can be computed from its location $x_{i,k}$ and velocity $v_{i,k}$ in the i th step of iteration as follows:

$$v_{i+1,k} = \omega v_{i,k} - c_1 r_1 (x_{i,k} - g_{b,i}) - c_2 r_2 (x_{i,k} - b_i) + c_3 r_3 (\beta x_{i,k} - y_i), \quad (3)$$

where c_1, c_2, r_1, r_2 , and ω are the same as those shown in equation (1); c_3 and β are positive numbers and r_3 is a random number between 0 and 1. y_i is the center of mass for the particles in the swarm for the i th iteration step and can be conveniently computed as follows:

$$y_i = \frac{\sum_{h=1}^m x_{i,h}}{m}. \quad (4)$$

The exploring stage executes for a certain number of iteration steps to perform an extensive exploration of the variable space. The converging stage starts execution after the exploring stage is completed. In the converging stage, equations (1) and (2) are used to update the velocity and location of each particle in the swarm until the specified number of iterations has been executed.

Before a new large iteration step starts, a translation procedure is applied to the particles in the swarm such that each particle is relocated to a different position. The velocity of each particle remains unchanged after the translation occurs. The translation of particle i is performed as follows:

$$z_i = p_i + c_4 r_4, \quad (5)$$

where p_i is the location of particle i before the translation is applied and z_i is its location after the translation; c_4 is a positive constant and r_4 is a random vector of the same dimensionality as p_i , and each component of r_4 is a random number between -1.0 and 1.0 .

3. Analysis of the Algorithm

In this section, we show that the algorithm can effectively explore the variable space of a high-dimensional function in the exploring stage when appropriate values are selected for parameters c_1, c_2 , and c_3 and the population size. From equations (3)–(5), no coupling exists between any two different dimensions in the coordinate of a particle in the swarm. Each individual dimension can thus be analyzed independent of others. The analysis of the algorithm can be simplified by choosing one particular dimension in the space and analyzing the behavior of the swarm in this dimension. In the rest of this paper, we assume that r_1, r_2 , and r_3 all follow a uniform distribution in interval $[0, 1]$.

Definition 1. Given positive constants c_1, c_2, c_3, ω , and β and a positive integer m , let I_m be the $m \times m$ identity matrix and A_m be an $m \times m$ matrix where every element in A_m is 1. The *partial transition matrix* $P(c_1, c_2, c_3, \omega, \beta, m)$ is defined as follows:

$$P(c_1, c_2, c_3, \omega, \beta, m) = \left(1 + \omega + \frac{1}{2}(c_3\beta - c_1 - c_2)\right)I_m - \frac{c_3}{2m}A_m. \quad (6)$$

Definition 2. Given positive constants c_1, c_2, c_3, ω , and β and a positive integer m , let I_m be the $m \times m$ identity matrix. The *complete transition matrix* $C(c_1, c_2, c_3, \omega, \beta, m)$ is a $2m \times 2m$ square matrix and it can be determined from the partial transition matrix $P = P(c_1, c_2, c_3, \omega, \beta, m)$ as follows:

$$C(c_1, c_2, c_3, \omega, \beta, m) = \begin{bmatrix} P & -\omega I_m \\ I_m & 0 \end{bmatrix}. \quad (7)$$

We assume that the variable space of a high-dimensional function is a “cubic” region in a multidimensional space, where the value of the l th dimension is within interval $[L_l, U_l]$ and both L_l and U_l are positive numbers. The following theorem shows that, for certain values of $c_1, c_2, c_3, \omega, \beta$, and m , the expected value of the l th dimension of each particle explores the corresponding dimension of the cubic region with a function of exponential order.

Theorem 1. Given a “cubic” region in an m -dimensional space, where the value of the l th dimension is within interval $[L_l, U_l]$ and both L_l and U_l are positive numbers, $x_l(i, k)$ is the l th dimension of particle k in iteration step i , and $e_l(i, k)$ is the expected value of $x_l(i, k)$. Let $\lambda_1, \lambda_2, \dots, \lambda_{2m}$ be the eigenvalues of the complete transition matrix constructed from c_1, c_2, c_3, ω , and β and the population size m , $\lambda_1, \lambda_2, \dots, \lambda_{2m}$ are mutually different, and both $|\lambda_1| \leq |\lambda_2| \leq |\lambda_3| \leq \dots \leq |\lambda_{2m}|$ and $|\lambda_{2m}| > 1$ hold for $\lambda_1, \lambda_2, \dots, \lambda_{2m}$. There exist nonnegative constants M_1 and N_1 such that $N_1 |\lambda_{2m}|^i \leq |e_l(i, k) - e_l(i-1, k)| \leq M_1 |\lambda_{2m}|^i$ holds for large enough i .

Proof. Let $v_l(i, k)$ be the l th dimension of the velocity of particle k in iteration step i , $g_l(i)$ be the l th dimension of the best location found by the swarm in iteration step i , and $b_l(i, k)$ be the l th dimension of the best location in the trajectory of particle k in iteration step i . From equations (4) and (5), when $i > 1$, $x_l(i, k)$ can be written as follows:

$$x_l(i, k) = \left(1 + c_3 r_3 - c_1 r_1 - c_2 r_2 - \frac{c_3 r_3}{m}\right) x_l(i-1, k) - \frac{c_3 r_3}{m} \sum_{h \neq k} x_l(i-1, h) + r_l(i-1, k), \quad (8)$$

where $r_l(i, k)$ is defined to be

$$r_l(i, k) = \omega v_l(i, k) + c_1 r_1 g_l(i) + c_2 r_2 b_l(i, k). \quad (9)$$

Taking the expected value for equation (8), we have

$$e_l(i, k) = \left(1 + \frac{1}{2}(c_3 - c_1 - c_2) - \frac{c_3}{2m}\right)e_l(i-1, k) - \frac{c_3}{2m} \sum_{h \neq k} e_l(i-1, h) + R_l(i-1, k), \quad (10)$$

where $R_l(i-1, k)$ is the expected value of $r_l(i-1, k)$.

Define $S_l(i, k)$, $T_l(i)$, and $H_l(i, k)$ as follows:

$$S_l(i, k) = \sum_{s=1}^i \omega^{i-s} e_l(s, k), \quad (11)$$

$$T_l(i) = \sum_{s=1}^i \omega^{i-s} G_l(s), \quad (12)$$

$$H_l(i, k) = \sum_{s=1}^i \omega^{i-s} B_l(s, k). \quad (13)$$

Equation (10) can then be written as follows:

$$e_l(i, k) = e_l(i-1, k) + C_1 S_l(i-1, k) - \frac{c_3}{2m} \sum_{h \neq k} S_l(i-1, h) + F_l(i-1, k), \quad (14)$$

where C_1 and $F_l(i-1, k)$ are defined as follows:

$$C_1 = \frac{1}{2} \left(c_3 - c_1 - c_2 - \frac{c_3}{m} \right), \quad (15)$$

$$F_l(i-1, k) = \frac{c_1}{2} T_l(i-1) + \frac{c_2}{2} H_l(i-1, k) + \omega^{i-1} v(1, k). \quad (16)$$

From the fact that

$$e_l(i, k) = S_l(i, k) - \omega S_l(i-1, k). \quad (17)$$

Equation (14) can be further written as

$$S_l(i, k) = (1 + \omega + C_1) S_l(i-1, k) - \omega S_l(i-2, k) - \frac{c_3}{2m} \sum_{h \neq k} S_l(i-1, h) + F_l(i-1, k). \quad (18)$$

Let $U_l(i, k) = S_l(i-1, k)$, equation (18) can be written as

$$S_l(i, k) = (1 + \omega + C_1) S_l(i-1, k) - \omega U_l(i-1, k) - \frac{c_3}{2m} \sum_{h \neq k} S_l(i-1, h) + F_l(i-1, k). \quad (19)$$

Define $2m$ -dimensional vectors $X_l(i)$ and $Q_l(i)$ as follows:

$$X_l(i) = [S_l(i, 1), \dots, S_l(i, m), U_l(i, 1), \dots, U_l(i, m)]^T, \quad (20)$$

$$Q_l(i) = [F_l(i, 1), F_l(i, 2), \dots, F_l(i, m), 0, \dots, 0]^T. \quad (21)$$

Equation (19) can be written as follows:

$$X_l(i) = C(c_1, c_2, c_3, \omega, m) X_l(i-1) + Q_l(i-1). \quad (22)$$

Since $C(c_1, c_2, c_3, \omega, m)$ has $2m$ different eigenvalues, it can be decomposed into a form as follows:

$$C(c_1, c_2, c_3, \omega, m) = \Psi \Lambda \Psi^{-1}, \quad (23)$$

where Λ is a $2m \times 2m$ diagonal matrix with $\lambda_1, \lambda_2, \dots, \lambda_{2m}$ on its diagonal positions and the columns of Ψ are the corresponding eigenvectors of $C(c_1, c_2, c_3, \omega, m)$. Based on equation (23), equation (22) can be further written as follows:

$$Y_l(i) = \Lambda Y_l(i-1) + V_l(i-1), \quad (24)$$

where $X_l(i) = \Psi Y_l(i)$ and $Q_l(i) = \Psi V_l(i)$. Let $Y_l(i, j)$ be the j th ($1 \leq j \leq 2m$) dimension of $Y_l(i)$, it is clear that the following can be obtained for $Y_l(i, j)$:

$$Y_l(i, j) = \sum_{s=1}^{i-1} \lambda_j^{i-1-s} V_l(s, j). \quad (25)$$

From equation (25), we are able to obtain the following equation:

$$Y_l(i, j) - Y_l(i-1, j) = \lambda_j^{i-2} V_l(i-1, j) + \sum_{s=2}^{i-1} \lambda_j^{i-1-s} (V_l(s, j) - V_l(s-1, j)). \quad (26)$$

We assume that all particles in the swarm are within the "cubic" region in the iteration steps from 1 to i . From equations (12) and (13), we have

$$\frac{L_l}{1-\omega} \leq |T_l(i)| \leq \frac{U_l}{1-\omega}, \quad (27)$$

$$\frac{L_l}{1-\omega} \leq |B_l(i, k)| \leq \frac{U_l}{1-\omega}. \quad (28)$$

From equation (16), the following equation holds for $F_l(i-1, k)$ when i is sufficiently large:

$$\frac{\max\{c_1 L_l - c_2 U_l, c_2 L_l - c_1 U_l\}}{3(1-\omega)} \leq F_l(i-1, k) \leq \frac{c_1 + c_2 + 1}{2(1-\omega)} U_l, \quad (29)$$

where c_1 and c_2 are selected to guarantee that $c_1 L_l > c_2 U_l$ or $c_2 L_l > c_1 U_l$ holds. Since $|\lambda_{2m}| > 1$, $|\lambda_1| \leq |\lambda_2| \leq |\lambda_3| \leq \dots \leq |\lambda_{2m}|$, both Ψ and Λ are constant matrices and $X_l(i) = \Psi Y_l(i)$ and $Q_l(i) = \Psi V_l(i)$. We immediately obtain from equations (17), (26), and (29) that when i is large enough, there exist positive constants M_1 and N_1 such that $N_1 |\lambda_{2m}|^i \leq |e_l(i, k) - e_l(i-1, k)| \leq M_1 |\lambda_{2m}|^i$ holds for each k ($1 \leq k \leq 2m$). The theorem thus follows.

Next, we show that $c_1, c_2, c_3, \omega, \beta$, and m can be selected to guarantee that $|\lambda_{2m}| > 1$ holds. The proof is based on the following well-known Gershgorin circle theorem. $\square$

Theorem 2. Let A be an $n \times n$ matrix and $R_i (1 \leq i \leq n)$ denotes the circle in the complex plane with center a_{ii} and radius $\sum_{j \neq i, 1 \leq j \leq n} |a_{ij}|$, that is,

$$R_i = \left\{ z \in \mathbb{C} \mid |z - a_{ii}| \leq \sum_{j \neq i, 1 \leq j \leq n} |a_{ij}| \right\}, \quad (30)$$

where $\mathbb{C}$ denotes the set of complex numbers. The eigenvalues of A are contained within $R = \bigcup_{1 \leq i \leq n} R_i$. Moreover, the union of any k of these circles that do not intersect the remaining contains precisely k (counting multiplicities) of the eigenvalues.

Theorem 3. Let λ_{2m} be the eigenvalue of $C(c_1, c_2, c_3, \omega, \beta, m)$ with the largest magnitude, $|\lambda_{2m}| > 1$, if $c_3(\beta - 1) > c_1 + c_2$.

Proof. Let $C = C(c_1, c_2, c_3, \omega, \beta, m)$ and $r_0 = 1 + \omega + (1/2)(\beta c_3 - c_1 - c_2) - (c_3/2m)$, from Definition 2 and Theorem 2, there exist two circles R_1 and R_2 such that all eigenvalues of C are contained in $R = R_1 \cup R_2$. R_1 is centered at $(r_0, 0)$ in the complex plane and its radius is $r_1 = \omega + ((m - 1)c_3/2m)$ and R_2 is centered at $(0, 0)$ in the complex plane and its radius is 1.

It is clear that when $c_3(\beta - 1) > c_1 + c_2$, $r_0 > r_1 + 1$ holds and R_1 and R_2 are thus disjoint and there is thus at least one eigenvalue λ_r contained in R_1 . We have

$$|\lambda_r| \geq |r_0| - r_1 = 1 + \frac{1}{2}((\beta - 1)c_3 - c_1 - c_2) > 1. \quad (31)$$

The lemma thus follows.

From Theorems 1 and 3, it is clear that when appropriate values are selected for c_1, c_2, c_3, ω , and β , each particle in the swarm is able to enlarge its searched area with the order of an exponential function and the entire “cubic” region can thus be extensively explored by the swarm in a logarithmic number of steps. In practice, the value of $|\lambda_{2m}|$ should be appropriate to guarantee a both efficient and extensive search in a desired region. Theorem 3 only provides a sufficient condition for $|\lambda_{2m}| > 1$ to hold, and experiments are often needed in practice to determine the appropriate values for $c_1, c_2, c_3, \omega, \beta$, and m . $\square$

4. Testing Results

We have implemented this new PSO-based optimization algorithm into a computer program PSOTS in MATLAB and evaluated its performance on the minimization of a few high-dimensional functions in the Benchmark Dataset for PSO functions, which can be downloaded for free from the website at <https://www.cil.pku.edu.cn/resources/somso/271214.htm>. Its performance is also compared with that of a few other PSO-based algorithms, including the traditional PSO algorithm, the AsynLnPSO algorithm [14], the LinWPSO algorithm [13], the GPSO algorithm [19], the CCAS algorithm [21], and the VPSO algorithm [30]. GPSO, CCAS, and VPSO are state-of-the-art PSO-based algorithms particularly designed for the optimization of high-dimensional functions. A large amount of testing has been performed to determine the parameters that can optimize the

performance of each of the four algorithms. For PSOTS, the population size is determined to be 200, the values of c_1, c_2 , and c_3 are set to be 0.7, 0.5, and 0.9, respectively. β is 1.0 and ω is 0.8.

The performance of all four algorithms is then evaluated and compared. To evaluate the performance of a given algorithm on a function, the algorithm is executed 10 times on a randomly rotated version of the function. For a fair comparison, the number of iterations in each execution of an algorithm is set to be 3000. The minimum function value obtained in all 10 executions and the distribution of the minimized function values in all 10 executions is obtained. Tables 1 and 2 show the minimum function values, the mean values, and standard deviations of the minimized function values obtained with the seven algorithms on 9 functions when the dimensionality of each function is 100 and 150, respectively. It is clear from both tables that PSOTS outperforms the other six algorithms in both the best and mean of the minimized function values on 8 out of the 9 functions. The minimized function values obtained with PSOTS on Ackley, Cigar, Rastrigin, and Noncon-rastrigin are significantly lower than those obtained with the other six algorithms. On the other hand, all other algorithms outperform PSOTS on function Schwefel.

It is also interesting to observe that the best minimized function value obtained with PSOTS on Ellipse is less than that obtained with the traditional PSO algorithm when the dimensionality is 100. However, the mean of the minimized function values obtained with PSOTS is larger than that obtained with the traditional PSO algorithm in the same dimensionality. When the dimensionality increases from 100 to 150, PSOTS outperforms the traditional PSO algorithm in both the best and mean of minimized function values. This particular example suggests that the exploration ability of PSOTS is stronger than that of the traditional PSO, and the advantage of its exploring ability over the traditional PSO algorithm is further enhanced when the dimensionality of the function that needs to be optimized increases.

We then increase the dimensionality of the functions to higher values and tested the performance of all seven algorithms on the same set of functions when the dimensionality is 200, 300, and 500, respectively. Tables 3–6 show the performance of the seven algorithms on 10 functions from the benchmark dataset when the dimensionality is selected to be 200, 300, 500, and 1000, respectively. It is clear from the tables that PSOTS significantly outperforms the other six algorithms in 9 tested functions in both the best and mean of minimized function values. Although the other six algorithms still outperform PSOTS on function Schwefel, the relative performance of PSOTS is improved significantly when the dimensionality increases. Table 5 shows that PSOTS outperforms the traditional PSO and AsynLnPSO in the best of the minimized function values of Schwefel when the dimensionality of the function reaches a value of 500.

Compared with other PSO-based optimization algorithms, a certain portion of the iteration steps of PSOTS is dedicated to the exploration of the variable space of a function. This exploring process may not always lead to

TABLE 1: The performance of PSOTS, the traditional PSO algorithm (PSO), the AsynLnPSO algorithm (APSO), the LinWPSO algorithm (LPSO), GPSO, CCAS, and VPSO on 9 functions in the benchmark dataset when the dimensionality is 100.

Function		Ackley	Cigar	Griewank	Rastrigin	Rosenbrock	Noncon-rast	Schwefel	Tablet	Ellipse
PSOTS	Best	0.267	1514.120	0.003	46.419	115.979	27.432	46.644	0.510	43.013
	Mean	0.365	2926.385	0.005	54.378	127.799	53.147	54.476	0.720	93.883
	STD	0.078	828.050	0.001	6.993	12.423	14.221	4.598	0.161	28.614
PSO	Best	0.329	2592.790	0.005	186.276	119.860	379.094	36.258	0.732	53.693
	Mean	0.470	3733.502	0.007	250.535	138.929	491.466	42.399	1.033	89.141
	STD	0.068	664.997	0.002	32.151	10.426	82.755	4.212	0.169	24.346
APSO	Best	0.411	3565.981	0.006	175.738	133.197	413.469	35.950	0.902	65.058
	Mean	0.478	4494.736	0.007	241.698	143.726	497.665	44.798	1.137	104.126
	STD	0.040	679.345	0.001	44.640	7.900	72.039	4.644	0.238	26.315
LPSO	Best	0.333	2522.980	0.005	161.634	127.660	385.249	35.380	0.798	73.928
	Mean	0.449	3742.696	0.007	252.637	138.899	467.999	43.031	1.131	107.747
	STD	0.070	1030.457	0.001	43.480	9.272	67.804	5.211	0.194	29.712
GPSO	Best	0.312	2221.012	0.005	121.513	131.571	227.312	44.372	0.735	55.721
	Mean	0.427	3251.760	0.008	203.624	168.632	359.351	53.921	1.129	95.463
	STD	0.071	943.279	0.002	41.531	11.212	65.231	5.137	0.134	23.461
CCAS	Best	0.308	2013.537	0.004	124.249	121.326	195.272	42.681	0.683	52.381
	Mean	0.392	3121.583	0.007	227.515	173.625	293.422	51.732	1.127	91.272
	STD	0.068	856.863	0.001	52.234	10.521	74.321	4.928	0.183	25.641
VPSO	Best	0.281	1923.361	0.004	93.627	126.731	143.368	33.291	0.654	49.728
	Mean	0.353	3251.672	0.008	182.676	165.472	252.431	43.562	0.972	88.362
	STD	0.042	943.612	0.002	45.572	9.837	78.635	5.138	0.136	26.732

TABLE 2: The performance of PSOTS, the traditional PSO algorithm (PSO), the AsynLnPSO algorithm (APSO), the LinWPSO algorithm (LPSO), GPSO, CCAS, and VPSO on 9 functions in the benchmark dataset when the dimensionality is 150.

Function		Ackley	Cigar	Griewank	Rastrigin	Rosenbrock	Noncon-rast	Schwefel	Tablet	Ellipse
PSOTS	Best	0.425	5116.750	0.005	90.681	190.554	92.819	91.398	0.717	73.556
	Mean	0.481	6324.494	0.007	116.478	205.230	121.122	99.691	1.200	220.456
	STD	0.040	1068.823	0.001	20.753	12.054	34.780	3.756	0.336	77.121
PSO	Best	0.511	7021.059	0.008	469.011	224.680	695.866	78.957	1.479	137.010
	Mean	0.618	9136.416	0.011	546.090	255.356	872.599	89.771	1.782	250.101
	STD	0.057	1282.803	0.001	39.747	22.179	101.196	6.417	0.230	66.171
APSO	Best	0.605	7502.959	0.008	479.003	230.769	686.900	82.957	1.367	127.008
	Mean	0.679	9991.976	0.011	551.005	251.660	818.986	90.327	1.839	310.903
	STD	0.061	1541.439	0.002	48.771	14.570	81.069	5.341	0.321	93.523
LPSO	Best	0.521	5367.866	0.008	472.005	217.039	788.823	75.947	1.367	106.011
	Mean	0.598	8151.604	0.011	542.003	231.982	872.196	83.736	1.711	275.302
	STD	0.052	1557.269	0.001	56.972	10.096	56.059	4.234	0.240	91.931
GPSO	Best	0.474	6025.012	0.008	295.561	239.459	577.883	89.198	1.145	96.957
	Mean	0.577	7641.640	0.012	450.332	300.165	743.809	110.752	1.708	247.262
	STD	0.064	1539.431	0.002	67.280	14.744	74.040	5.309	0.241	80.445
CCAS	Best	0.469	5396.279	0.007	337.274	211.787	522.905	78.234	1.173	125.268
	Mean	0.534	7310.747	0.010	510.999	292.134	630.278	94.943	1.712	257.922
	STD	0.073	1607.475	0.002	99.349	17.176	89.913	6.630	0.272	88.161
VPSO	Best	0.438	5077.673	0.007	211.627	207.912	322.436	67.878	1.073	126.744
	Mean	0.490	7205.705	0.013	389.462	241.185	549.318	89.587	1.568	238.459
	STD	0.043	1766.442	0.003	83.077	17.642	80.935	7.567	0.213	74.187

improved performance of optimization. As a direct result of the exploration, fewer iteration steps are available for the convergence process. The convergence rates of PSOTS thus may be lower than those of other PSO-based algorithms on functions that do not have a large number of local optima. Function Schwefel in the benchmark set is an example of such functions. In addition, the global minimum of Schwefel is located in a straight line with small gradient values in the

variable space. The existence of this straight line may further slow down the convergence process of PSOTS. Specifically, in the converging process of any PSO-based algorithm, particles tend to move toward the straight line rapidly while converging slowly along the direction of the straight line to the global minimum. In PSOTS, since particles are expelled apart from one another during the exploring process, the converging process first moves them to points that have

TABLE 3: The performance of PSOTS, the traditional PSO algorithm (PSO), the AsynLnPSO algorithm (APSO), the LinWPSO algorithm (LPSO), GPSO, CCAS, and VPSO on 9 functions in the benchmark dataset when the dimensionality is 200.

Function		Ackley	Cigar	Griewank	Rastrigin	Rosenbrock	Noncon-rast	Schwefel	Tablet	Ellipse
PSOTS	Best	0.431	8313.319	0.006	125.642	266.531	142.388	135.922	1.145	213.868
	Mean	0.510	10573.300	0.009	190.698	291.663	180.543	148.872	1.749	426.983
	STD	0.059	1297.349	0.001	44.402	15.715	28.069	8.772	0.394	157.209
PSO	Best	0.623	13592.310	0.011	828.157	330.324	1162.641	129.083	1.924	442.334
	Mean	0.733	15953.110	0.013	908.032	360.406	1258.925	137.839	2.415	567.524
	STD	0.095	2141.955	0.001	65.745	25.679	57.524	6.567	0.290	87.083
APSO	Best	0.628	13927.090	0.012	761.458	333.122	1168.319	125.760	2.115	374.350
	Mean	0.728	16153.560	0.016	911.018	363.264	1284.167	137.659	2.590	615.184
	STD	0.063	1431.241	0.002	69.228	22.187	51.557	8.961	0.287	153.836
LPSO	Best	0.540	11784.830	0.010	766.147	287.388	1230.984	119.961	2.078	411.546
	Mean	0.651	13929.390	0.012	880.708	337.612	1375.003	130.168	2.392	548.331
	STD	0.055	2044.389	0.002	73.132	25.340	110.808	8.568	0.212	92.605
GPSO	Best	0.502	10276.490	0.011	513.656	345.566	922.450	142.323	1.765	320.551
	Mean	0.620	13015.940	0.017	746.526	432.668	1113.325	166.486	2.458	549.419
	STD	0.087	1986.996	0.002	109.377	20.7677	101.249	11.501	0.283	108.289
CCAS	Best	0.483	9755.112	0.010	523.218	302.989	782.621	126.101	1.574	386.631
	Mean	0.627	12079.051	0.013	841.210	422.458	944.007	144.237	2.459	523.554
	STD	0.114	1665.608	0.002	206.596	26.121	120.584	13.762	0.292	102.498
VPSO	Best	0.464	9264.162	0.010	353.963	298.692	541.078	102.796	1.436	370.183
	Mean	0.552	12345.613	0.017	641.174	347.596	829.347	138.759	2.130	502.851
	STD	0.045	1925.664	0.003	156.169	34.864	138.683	10.812	0.246	97.771

TABLE 4: The performance of PSOTS, the traditional PSO algorithm (PSO), the AsynLnPSO algorithm (APSO), the LinWPSO algorithm (LPSO), GPSO, CCAS, and VPSO on 9 functions in the benchmark dataset when the dimensionality is 300.

Function		Ackley	Cigar	Griewank	Rastrigin	Rosenbrock	Noncon-rast	Schwefel	Tablet	Ellipse
PSOTS	Best	0.480	11239.310	0.007	270.220	435.735	243.118	236.858	1.757	768.211
	Mean	0.551	16700.500	0.010	304.659	460.780	309.250	245.861	2.642	1044.533
	STD	0.052	2864.634	0.002	25.149	20.493	40.135	8.772	0.626	199.579
PSO	Best	0.725	26182.680	0.016	797.438	558.017	1958.180	218.271	3.607	1087.085
	Mean	0.837	29781.450	0.017	1558.475	581.011	2144.747	230.463	4.063	1383.916
	STD	0.071	3263.52	0.002	277.037	19.795	93.044	9.453	0.347	199.579
APSO	Best	0.792	26675.420	0.015	506.867	569.283	622.970	225.835	3.506	1164.937
	Mean	0.872	30096.930	0.019	1530.612	604.836	2103.667	234.035	4.481	1444.022
	STD	0.054	2666.445	0.002	363.651	23.546	527.097	3.823	0.511	196.814
LPSO	Best	0.676	21480.700	0.014	499.562	510.421	1961.123	210.576	3.263	1058.076
	Mean	0.751	24369.240	0.017	1575.189	559.471	2243.721	223.646	4.070	1214.656
	STD	0.053	2254.595	0.002	454.681	33.480	167.328	9.551	0.558	105.178
GPSO	Best	0.610	14351.613	0.013	386.601	578.391	290.975	248.015	2.805	837.559
	Mean	0.730	21013.471	0.020	1204.562	702.838	735.345	276.637	4.017	1290.958
	STD	0.083	2958.214	0.002	295.832	24.869	83.056	9.544	0.529	188.628
CCAS	Best	0.564	13233.792	0.011	566.908	512.184	179.650	215.826	2.632	941.745
	Mean	0.690	21699.961	0.016	1365.832	671.090	721.552	242.225	3.930	1207.535
	STD	0.103	2854.312	0.003	121.696	27.090	193.967	8.806	0.481	173.677
VPSO	Best	0.574	13157.77	0.014	550.812	507.583	164.3968	180.937	2.272	888.696
	Mean	0.652	20398.05	0.019	1024.967	578.245	677.9356	234.522	3.389	1159.823
	STD	0.042	2905.144	0.004	181.691	21.837	189.815	5.169	0.557	220.522

longer distances from the global minimum along the direction of the straight line and then let them converge to the global minimum. The longer distances to the global minimum clearly lead to optimization performance inferior to that of other PSO-based algorithms.

The performance of PSOTS on functions with a dimensionality of a few thousand is also tested and evaluated.

The dimensionality of two functions, including Ellipse and Sphere, is selected to be 5000, and all seven algorithms are used to minimize them. Each algorithm is executed 10 times. Table 7 shows the result of minimization obtained with the seven algorithms. It is clear from the table that PSOTS significantly outperforms the other six algorithms on both functions when the dimensionality is 5000.

TABLE 5: The performance of PSOTS, the traditional PSO algorithm (PSO), the AsynLnPSO algorithm (APSO), the LinWPSO algorithm (LPSO), GPSO, CCAS, and VPSO on 9 functions in the benchmark dataset when the dimensionality is 500.

Function		Ackley	Cigar	Griewank	Rastrigin	Rosenbrock	Noncon-rast	Schwefel	Tablet	Ellipse
PSOTS	Best	0.492	25412.630	0.009	478.891	706.558	493.205	414.928	3.475	1496.384
	Mean	0.581	29691.450	0.012	553.646	803.327	557.334	435.652	4.947	2092.524
	STD	0.051	3513.855	0.002	70.573	51.248	50.482	10.066	1.145	320.057
PSO	Best	0.829	45422.870	0.019	999.299	1022.024	3829.275	423.314	5.766	2244.141
	Mean	0.886	56384.080	0.022	2780.208	1111.354	4051.348	433.986	7.550	3255.934
	STD	0.052	5545.354	0.002	916.323	47.555	167.972	7.313	0.963	199.579
APSO	Best	0.833	50024.720	0.019	1126.165	1000.456	3746.506	418.279	6.807	2937.273
	Mean	0.929	56943.480	0.023	3075.081	1084.608	4065.724	433.210	8.100	3627.655
	STD	0.049	4846.618	0.003	691.812	48.341	217.894	8.016	1.265	422.285
LPSO	Best	0.742	43541.120	0.017	920.020	943.407	3905.713	411.002	5.975	2373.675
	Mean	0.806	48230.900	0.020	2685.457	985.057	4112.780	422.941	7.051	3115.352
	STD	0.043	3195.987	0.002	933.487	44.286	135.469	10.654	0.603	443.890
GPSO	Best	0.697	28284.351	0.015	768.416	1048.064	924.820	441.750	4.485	1907.092
	Mean	0.775	38520.982	0.024	2072.027	1253.445	1278.837	503.868	7.090	3093.015
	STD	0.061	4961.206	0.002	740.987	36.754	112.2568	15.367	1.076	769.6676
CCAS	Best	0.602	26503.243	0.013	1169.143	852.805	890.7993	423.028	4.560	1809.055
	Mean	0.738	39164.056	0.020	2645.188	1270.997	1264.482	462.079	6.907	2947.759
	STD	0.076	4060.912	0.004	301.950	50.363	132.752	12.689	0.606	274.5918
VPSO	Best	0.599	27384.475	0.012	776.843	901.369	860.366	434.518	4.089	1731.14
	Mean	0.691	38449.613	0.022	1961.521	1062.914	1363.843	455.697	6.049	2571.514
	STD	0.038	4278.437	0.005	357.148	46.818	136.64	4.991	1.399	720.7601

TABLE 6: The performance of PSOTS, the traditional PSO algorithm (PSO), the AsynLnPSO algorithm (APSO), the LinWPSO algorithm (LPSO), GPSO, CCAS, and VPSO on 9 functions in the benchmark dataset when the dimensionality is 1000.

Function		Ackley	Cigar	Griewank	Rastrigin	Rosenbrock	Noncon-rast	Sphere	Tablet	Ellipse
PSOTS	Best	0.551	58336.090	0.013	1049.727	1583.446	1036.363	5.691	7.021	4561.731
	Mean	0.654	66590.620	0.017	1206.143	1695.683	1224.481	6.641	8.866	6034.033
	STD	0.066	8530.537	0.003	108.670	80.271	122.191	0.872	1.767	1089.670
PSO	Best	0.883	101581.400	0.022	1856.572	2091.178	2176.873	10.588	12.760	8299.737
	Mean	0.959	119206.001	0.030	3752.308	2202.001	7460.916	12.395	13.751	9268.079
	STD	0.052	10297.240	0.004	2558.991	100.112	2777.144	1.500	1.345	1063.982
APSO	Best	0.903	114610.100	0.019	2051.027	2102.363	2168.402	11.284	12.864	8528.893
	Mean	0.976	126324.800	0.029	3395.736	2268.229	8025.089	12.318	16.518	10217.47
	STD	0.044	5550.201	0.005	2225.827	119.697	2069.662	0.623	1.783	1131.887
LPSO	Best	0.840	96167.060	0.022	1844.305	1959.617	8590.621	9.892	12.613	6859.916
	Mean	0.864	105884.300	0.028	4760.095	2053.572	8861.428	10.804	15.071	8193.032
	STD	0.029	7576.268	0.003	3024.811	75.338	135.469	0.814	1.484	916.128
GPSO	Best	0.671	63817.053	0.018	1630.047	2305.846	1779.136	6.781	9.207	6002.318
	Mean	0.827	84967.952	0.031	4267.252	2552.418	2407.985	8.051	13.506	8425.598
	STD	0.052	10801.034	0.003	1479.187	85.12814	132.854	0.833	2.204	1820.534
CCAS	Best	0.676	58706.483	0.013	1363.015	1847.001	1681.628	6.632	9.931	6127.127
	Mean	0.820	82986.121	0.027	4717.921	2528.346	2605.835	7.767	15.745	7943.676
	STD	0.092	9234.537	0.007	843.3347	113.5974	1637.341	0.825	1.039	755.427
VPSO	Best	0.657	57804.753	0.014	1330.581	1915.181	1661.348	6.568	8.420	5118.947
	Mean	0.741	85502.262	0.029	3187.537	2146.646	2969.365	7.753	12.515	7077.191
	STD	0.027	8076.055	0.008	676.163	93.69879	1248.472	0.848	1.956	672.217

5. Conclusions

In this paper, a new efficient PSO algorithm is developed for the optimization of high-dimensional functions. Based on a simple expelling force field and a translational procedure, the algorithm can explore in a significantly enlarged area in the variable space of a high-dimensional function and the performance of optimization can thus be significantly

improved for functions that have a large number of local minima. Our analysis and testing results on high-dimensional benchmark functions also show that this new algorithm can achieve significantly improved performance on the optimization of high-dimensional functions, compared with the traditional PSO algorithm, a few of its variants, and a number of state-of-art algorithms for the optimization of high-dimensional functions.

TABLE 7: The performance of PSOTS, the traditional PSO algorithm (PSO), the AsynLnPSO algorithm (APSO), the LinWPSO algorithm (LPSO), GPSO, CCAS, and VPSO on 9 functions in the benchmark dataset when the dimensionality is 5000.

Functions		PSOTS	PSO	APSO	LPSO	GPSO	CCAS	VPSO
Sphere	Best	30.556	57.098	62.388	49.728	35.24098	36.06005	35.6112
	Mean	36.499	64.915	68.973	55.641	41.47434	41.16323	42.53189
	STD	4.759	4.427	3.928	3.263	5.111826	3.354321	3.27586
Ellipse	Best	29088.000	58646.700	61238.400	48136.800	39403.572	41069.376	34520.315
	Mean	35375.900	65562.900	72245.200	58339.700	54707.362	54713.734	42669.342
	STD	3657.300	4712.800	6179.100	5080.900	3856.751	3736.693	3716.105

Since a large number of PSO-based algorithms have been developed for the optimization of multivariable functions, it remains unknown whether this approach can outperform the PSO-based algorithms that have not been tested in our experiments on high-dimensional functions or not. A much more comprehensive testing and comparison is thus needed to evaluate the performance of this algorithm.

Recently, due to the rapid growth in the amounts of datasets with dimensionalities larger than ten thousand, the extraction of crucial features from such datasets has become an important problem in the area of data mining. Such datasets are often called ultrahigh-dimensional datasets. The exploring ability of the proposed approach may deteriorate significantly in such ultrahigh-dimensional spaces, which may adversely affect the application of this approach to the accurate processing of ultrahigh-dimensional datasets. Improving the optimization performance of this approach on ultrahigh-dimensional functions thus constitutes an important part of our future work.

Data Availability

The source code and testing data of this work are freely available upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work of G. Li, J. Sun, M.N.A. Rana, and Y. Song was fully supported by the Fund of Specially Appointed Professor of Jiangsu Province, China, under grant number 1034901501. The work of C. Liu was fully supported by the US Science & Technology Center Grant under grant number CCF-0939370.

References

- [1] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pp. 39–43, Nagoya, Japan, October 1995.
- [2] P. N. Suganthan, "Particle swarm optimizer with neighborhood operator," in *Proceedings of the Congress on Evolutionary Computing*, pp. 1958–1962, Washington, DC, USA, July 1999.
- [3] J. J. Liang and P. N. Suganthan, "Dynamic multi-swarm particle swarm optimizer," in *Proceedings of IEEE Swarm Intelligence Symposium*, pp. 124–129, Pasadena, CA, USA, June 2005.
- [4] C.-C. Peng and C.-H. Chen, "Compensatory neural fuzzy network with symbiotic particle swarm optimization for temperature control," *Applied Mathematical Modelling*, vol. 39, no. 1, pp. 383–395, 2015.
- [5] W.-D. Chang, "A modified particle swarm optimization with multiple subpopulations for multimodal function optimization problems," *Applied Soft Computing*, vol. 33, pp. 170–182, 2015.
- [6] M. R. Tanweer, S. Suresh, and N. Sundararajan, "Dynamic mentoring and self-regulation based particle swarm optimization algorithm for solving complex real-world optimization problems," *Information Sciences*, vol. 326, pp. 1–24, 2016.
- [7] L. D. S. Coelho and C.-S. Lee, "Solving economic load dispatch problems in power systems using chaotic and Gaussian particle swarm optimization approaches," *International Journal of Electrical Power & Energy Systems*, vol. 30, no. 5, pp. 297–307, 2008.
- [8] K. Tatsumi, T. Ibuki, and T. Tanino, "Particle swarm optimization with stochastic selection of perturbation-based chaotic updating system," *Applied Mathematics and Computation*, vol. 269, pp. 904–929, 2015.
- [9] T. T. Ngo, A. Sadollah, and J. H. Kim, "A cooperative particle swarm optimizer with stochastic movements for computationally expensive numerical optimization problems," *Journal of Computational Science*, vol. 13, pp. 68–82, 2016.
- [10] Q. Fan and X. Yan, "Self-adaptive particle swarm optimization with multiple velocity strategies and its application for p-xylene oxidation reaction process optimization," *Chemometrics and Intelligent Laboratory Systems*, vol. 139, pp. 15–25, 2014.
- [11] G. Ardizzone, G. Cavazzini, and G. Pavesi, "Adaptive acceleration coefficients for a new search diversification strategy in particle swarm optimization algorithms," *Information Sciences*, vol. 299, pp. 337–378, 2015.
- [12] Y. Lu, N. Zeng, Y. Liu, and N. Zhang, "A hybrid wavelet neural network and switching particle swarm optimization algorithm for face direction recognition," *Neurocomputing*, vol. 155, pp. 219–224, 2015.
- [13] M. A. Arasomwan and A. O. Adewumi, "On the performance of linear decreasing inertial weight particle swarm optimization for global optimization," *The Scientific World Journal*, vol. 2013, pp. 1–12, 2013.
- [14] N. A. A. Aziz and Z. Ibrahim, "Asynchronous particle swarm optimization for swarm robotics," *Procedia Engineering*, vol. 41, pp. 951–957, 2012.
- [15] H. Banka and S. Dara, "A hamming distance based binary particle swarm optimization (HDBPSO) algorithm for high dimensional feature selection, classification and validation," *Pattern Recognition Letters*, vol. 52, pp. 94–100, 2015.
- [16] Z. Beheshti and S. M. Shamsuddin, "Non-parametric particle swarm optimization for global optimization," *Applied Soft Computing*, vol. 28, pp. 345–359, 2015.
- [17] W. Fang, J. Sun, H. Chen, and X. Wu, "A decentralized quantum-inspired particle swarm optimization algorithm

- with cellular structured population," *Information Sciences*, vol. 330, pp. 19–48, 2015.
- [18] J. Geng, M. Li, Z.-H. Dong, and Y.S. Liao, "Port throughput forecasting by MARS-RSVR with chaotic simulated annealing particle swarm optimization algorithm," *Neurocomputing*, vol. 147, pp. 239–250, 2015.
 - [19] T. A. Giuseppe, "A cooperative coevolutionary differential evolution algorithm with adaptive subcomponents," *Procedia Computer Science*, vol. 51, no. 1, pp. 834–844, 2015.
 - [20] I. Gosciniaik, "A new approach to particle swarm optimization algorithm," *Expert Systems with Applications*, vol. 42, no. 2, pp. 844–854, 2015.
 - [21] J. J. Jamian, M. N. Abdullah, H. Mokhlis, M. W. Mustafa, and A. H. A. Bakar, "Global particle swarm optimization for high dimension numerical functions analysis," *Journal of Applied Mathematics*, vol. 2014, pp. 1–14, 2014.
 - [22] Y. Li, Z.-H. Zhan, S. Lin, J. Zhang, and X. Luo, "Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems," *Information Sciences*, vol. 293, pp. 370–382, 2015.
 - [23] Z. Li, T. T. Nguyena, S. Chen, and T. T. Khac, "A hybrid algorithm based on particle swarm and chemical reaction optimization for multi-object problems," *Applied Soft Computing*, vol. 35, pp. 325–340, 2015.
 - [24] W. H. Lim and N. A. Mat Isa, "Adaptive division of labor particle swarm optimization," *Expert Systems with Applications*, vol. 42, no. 14, pp. 5887–5903, 2015.
 - [25] S. Salehian and S. K. Subraminiam, "Unequal clustering by improved particle swarm optimization in wireless sensor network," *Procedia Computer Science*, vol. 62, pp. 403–409, 2015.
 - [26] G. G. Samuel and C. C. A. Rajan, "Hybrid: particle swarm optimization-genetic algorithm and particle swarm optimization-shuffled frog leaping algorithm for long-term generator maintenance scheduling," *International Journal of Electric Power Energy Systems*, vol. 65, pp. 432–442, 2015.
 - [27] A. Sharifi, J. Kazemi Kordestani, M. Mahdavian, and M. R. Meybodi, "A novel hybrid adaptive collaborative approach based on particle swarm optimization and local search for dynamic optimization problems," *Applied Soft Computing*, vol. 32, pp. 432–448, 2015.
 - [28] R. Shirkhani, H. Jazayeri-Rad, and S. J. Hashemi, "Modeling of a solid oxide fuel cell power plant using an ensemble of neural networks based on a combination of the adaptive particle swarm optimization and Levenberg-Marquardt algorithms," *Journal of Natural Gas Science and Engineering*, vol. 21, pp. 1171–1183, 2014.
 - [29] H. Soleimani and G. Kannan, "A hybrid particle swarm optimization and genetic algorithm for closed-loop supply chain network design in large-scale networks," *Applied Mathematical Modelling*, vol. 39, no. 14, pp. 3990–4012, 2015.
 - [30] X. Song, Y. Zhang, N. Guo, X. Sun, and Y. Wang, "Variable-size cooperative coevolutionary particle swarm optimization for feature selection on high-dimensional data," *IEEE Transactions on Evolutionary Computation*, p. 1, 2020.
 - [31] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft Computing*, vol. 22, no. 2, pp. 387–408, 2018.
 - [32] J. J. Jamian, M. W. Mustafa, and H. Mokhlis, "Optimal multiple distributed generation output through rank evolutionary particle swarm optimization," *Neurocomputing*, vol. 152, pp. 190–198, 2015.
 - [33] C. Yang, W. Gao, N. Liu, and C. Song, "Low-discrepancy sequence initialized particle swarm optimization algorithm with high-order nonlinear time-varying inertia weight," *Applied Soft Computing*, vol. 29, pp. 386–394, 2015.
 - [34] S. Zhai and T. Jiang, "A new sense-through-foliage target recognition method based on hybrid differential evolution and self-adaptive particle swarm optimization-based support vector machine," *Neurocomputing*, vol. 149, pp. 573–584, 2015.
 - [35] L. Zhang, Y. Tang, C. Hua, and X. Guan, "A new particle swarm optimization algorithm with adaptive inertia weight based on Bayesian techniques," *Applied Soft Computing*, vol. 28, pp. 138–149, 2015.