Journal of Computational and Applied Mathematics 235 (2011) 2694–2714

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

A rank based particle swarm optimization algorithm with dynamic adaptation

Reza Akbari [*], Koorush Ziarati

Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran

ARTICLE INFO

Article history:
Received 15 June 2008
Received in revised form 17 April 2010

Keywords:
Particle swarm optimization
Rank based particle swarm optimization
Neural networks

ABSTRACT

The particle swarm optimization (PSO) technique is a powerful stochastic evolutionary algorithm that can be used to find the global optimum solution in a complex search space. This paper presents a variation on the standard PSO algorithm called the rank based particle swarm optimizer, or PSO$_{\text{rank}}$, employing cooperative behavior of the particles to significantly improve the performance of the original algorithm. In this method, in order to efficiently control the local search and convergence to global optimum solution, the γ best particles are taken to contribute to the updating of the position of a candidate particle. The contribution of each particle is proportional to its strength. The strength is a function of three parameters: strivness, immediacy and number of contributed particles. All particles are sorted according to their fitness values, and only the γ best particles will be selected. The value of γ decreases linearly as the iteration increases. A time-varying inertia weight decreasing non-linearly is introduced to improve the performance. PSO$_{\text{rank}}$ is tested on a commonly used set of optimization problems and is compared to other variants of the PSO algorithm presented in the literature. As a real application, PSO$_{\text{rank}}$ is used for neural network training. The PSO$_{\text{rank}}$ strategy outperformed all the methods considered in this investigation for most of the functions. Experimental results show the suitability of the proposed algorithm in terms of effectiveness and robustness.

1. Introduction

Optimization has been an active area of research for several decades. As many real-world optimization problems become increasingly complex, better optimization algorithms are always needed. Unconstrained optimization problems can be formulated as D-dimensional minimization (maximization) problems as follows:

$$\text{Min(or Max)} f(\vec{x}), x = (x_1, x_2, \dots, x_D)$$

where D is the number of the parameters to be optimized. In optimization problems, the objective is to find the minimum or maximum of the function under consideration. There are many optimization techniques available for function optimization.

Particle swarm optimization (PSO) is a swarm intelligence technique developed in [1], inspired by the social behavior of bird flocking and fish schooling. PSO has been shown to successfully optimize a wide range of continuous functions [2–7]. The algorithm, which is based on a metaphor of social interaction, searches a space by adjusting the trajectories of individual vectors, called particles. Particles change their state dynamically with both the position of their best past performance and the position of the global best particle in the neighborhood. This iterative process proceeds until an optimal state has been reached or until computation limitations are exceeded. Each particle represents a potential solution which is a point in the search space. The position of each particle is used to compute the value of the function to be optimized. Each particle has a

[*] Corresponding author.
E-mail address: rakbari@cse.shirazu.ac.ir (R. Akbari).

0377-0427/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2010.11.021

fitness value and a velocity for adjusting its flying direction according to the best experiences of the swarm in searching for the global optimum point in the *D*-dimensional solution space.

The PSO algorithm is easy to implement and has been empirically shown to perform well on many optimization problems. However, it may easily get trapped in a local optimum when solving complex multimodal problems. In order to improve PSO's performance on complex multimodal problems, we present the rank based particle swarm optimizer (PSO_{rank}) utilizing experiences of the best particles for escaping from the local optimum. First, we introduce the rank based particle swarm optimizer concept. Second, we introduce the time-varying acceleration coefficient. Third, we apply the proposed algorithm to the well known test functions. Finally, the proposed method is applied for training multilayer neural networks. The trained network is used for image compression and its performance investigated with other training methods presented in the literatures.

This paper is organized as follows. Section 2 introduces the original PSO and some current variants of the original PSO. Section 3 describes the rank based particle swarm optimizer and its extensions. The searching behavior of the proposed algorithm is discussed in Section 4. Section 5 presents the test functions, the experimental setting for each algorithm, and the results. Discussions are presented in Section 6. Finally, conclusions are given in Section 7.

2. Standard PSO

The standard PSO algorithm is an iterative process in which a set of particles are characterized by their position and the velocity with which they move in the solution space of a cost function. Each individual in PSO flies in the parameter space with a velocity which is dynamically adjusted according to its own flying experiences and those of its companions. Therefore, every individual is gravitated toward a stochastically weighted average of the previous best point of its own and that of its neighborhood companions. Mathematically, given a swarm of particles, each particle is associated with a position vector which is a feasible solution for an optimal problem; let the best previous position (the position giving the best objective function value called *pbest*) that the *i*th particle has found in the parameter space be denoted by p^i; the best position that the neighborhood particles of the *i*th particle have ever found, called *gbest*, is denoted using g^i. At the start time all of the positions and velocities are initialized randomly. At each iteration step, the position vector of the particle is updated by adding an increment vector, and denoted using the velocity. In the original PSO algorithm, the particles' positions are updated according to the following equations:

$$v_d^i(k+1) = v_d^i(k) + c_1\text{rand}_1(p_d^i - x_d^i(k)) + c_2\text{rand}_2(g_d^i - x_d^i(k)) \tag{1}$$

$$v_d^i(k+1) = \begin{cases} v_{\max} & \text{if } (v_d^i(k+1) > v_{\max}) \\ -v_{\max} & \text{if } (v_d^i(k+1) < -v_{\max}) \end{cases} \tag{2}$$

$$x_d^i(k+1) = x_d^i(k) + v_d^i\,(k+1) \tag{3}$$

where $v_{\max}$ is a maximum velocity possible for the particle, c_1 and c_2 are two positive constants, and rand_1 and rand_2 are two random parameters of uniform distribution in range [0, 1], which limit the velocity of the particle in the coordinate direction.

As shown in Eq. (1) only the global best particle has an impact on the candidate particle. This iterative process will continue swarm by swarm until a stop criterion is satisfied, and this forms the basic iterative process of a PSO algorithm. It is worth pointing out that in the right-hand side of (1), the second term represents the cognitive part of a PSO algorithm in which the particle changes its velocity on the basis of its own thinking and memory, while the third term is the social part of a PSO algorithm in which the particle modifies its velocity on the basis of the adaptation of the social–psychological knowledge. On the basis of these formulations, only the best particle in the neighborhood has an impact on the candidate particle. Essentially, the PSO algorithm is conceptually very simple, and can be implemented in a few lines of computer code. Also, it requires only primitive mathematical operators and very few algorithmic parameters need to be tuned.

2.1. Some variants of PSO—some previous works

Since the introduction of the PSO method in 1995, there has been a considerable amount of work done in developing the standard version of PSO [5,8,9]. Many researchers have worked on improving its performance in various ways, thereby deriving many interesting variants. Previous works on improvement of the basic PSO can be categorized as follows: (1) introducing inertia weight as a mechanism of balance between exploration and exploitation and proposing a different method for updating this parameter, (2) using the coefficient factor as a mechanism for controlling explosion and increasing the convergence rate, (3) using different neighborhood topologies such as local and global neighborhoods, (4) improving random variables rand_1 and rand_2, (5) using various methods for updating the position of each particle, and (6) improving acceleration factors c_1 and c_2. In population based optimization methods, proper control of global exploration and local exploitation is crucial in finding the optimum solution efficiently [2,10]. The basic PSO has problems in balancing exploration versus exploitation; therefore many of the works focus on this problem and trying to solve it. One of the first variants [11] introduces a parameter called inertia weight. The inertia weight is used to balance the global and local search abilities. A large inertia weight is more appropriate for global search, and a small inertia weight facilitates local search. A linearly decreasing inertia weight, over the course of the search, was proposed in [11]. Setting the parameters and their impacts on PSO algorithms are discussed in [12,11]. Shi and Eberhart designed fuzzy methods for non-linearly changing the inertia weight [13]. Similarly, a dynamic inertia weight was introduced in [6] which varies dynamically on the basis of the run and evolution state. In this strategy the inertia weight is given by a function of the evolution speed factor and the aggregation degree factor, and the value of the inertia weight is dynamically adjusted according to the evolution speed and aggregation degree. In [14], the inertia weight is set at zero except at the time of re-initialization. In addition to the time-varying inertia weight, a linearly decreasing inertia weight was introduced in [15].

Following analyzing the convergence behavior of the PSO, a PSO variant with a constriction factor was introduced in [8]. The constriction factor guarantees the convergence and improves the convergence velocity of the particles in the PSO method.

The performance of an individual in a swarm depends on population topology as well as the algorithm. A comparison between various neighborhood topologies was proposed in [16]. Two major types of neighborhood topology are presented in literature, called global and local neighborhoods. The first type implies that the source of influence on each particle was the best performing individual in the entire population. In the local structure, each individual is connected to the subset of particles in the population—that is, it is influenced by some particles. In the IPSO algorithm presented in [7], the swarm is partitioned into several sub-swarms. Each complex independently executes PSO. After a certain number of generations, the sub-swarms are forced to mix and points are reassigned to ensure information sharing.

An important part of each PSO algorithm is the contribution of each individual in the swarm. A hierarchical version of PSO introduced in [17], where the particles are arranged in a dynamic hierarchy that is used to define a neighborhood structure. Depending on the quality of the solution that is the best found so far, the particles move up or down the hierarchy. In [5], the concept of time delay is introduced into PSO to control the process of information diffusion and maintain the particle diversity. In this method the *gbest* used to update the particle velocity may not be the latest one, and different particles may use different values of *gbest* in the same iteration. Under this method, particle diversity thus increases.

In [9] a center particle is incorporated into the swarm. The center particle has no explicit velocity, and is set to the center of the swarm at each cycle of the algorithm. The center particle generally gets a good fitness value and attracts other particles and guides the search direction of the whole swarm. Some authors used a time-varying acceleration coefficient to overcome stagnation in searching for a global optimal solution. A variant with non-linear time-varying evolution is proposed in [3]. The cognitive parameter c_1 starts with a high value $c_{1\max}$ and non-linearly decreases to $c_{1\min}$. Meanwhile, the social parameter c_2 starts with a low value $c_{2\min}$ and non-linearly increases to $c_{2\max}$.

In the previously mentioned variants of the PSO, a particle with k neighbors selects the best one to be a source of influence and ignores the others. However, all the neighbors of a particle can be sources of influence. This interesting idea has been used by some researchers to improve the performance of the standard PSO [18]. The fully informed particle swarm (or FIPS) was presented by Mendes and Kennedy, in which a particle is not simply influenced by just the best particle among its neighbors [18]; the behavior of each particle in FIPS is affected by its local neighborhood. Hence, the authors presented different topological structures for investigating the performance of the FIPS. The scatter PSO method inspired from the FIPS model and SS/PR template was presented in [19]. Two strategies have been used in scatter PSO to exploit the social knowledge of the swarm. In the first strategy, the social learning of a particle was based on its own best experience and the best experience of another particle. In the second strategy, a set of reference particles have been used as guides for adjusting the trajectory of a particle. The comprehensive learning particle optimizer (or CLPSO) was presented in [4]. In CLPSO, the historical best information of all particles was used to update a particle's velocity. The CLPSO improved the searching ability of a particle by allowing it to learn from different *pbest* values for different dimensions for a few generations. Also, instead of learning from *pbest* and *gbest* at the same time in every generation, every dimension of a particle learns from just one of the *pbest* or *gbest* values for a few generations. As described in [4], the aforementioned extensions improve the performance of PSO by increasing diversity of the algorithm.

3. Rank based particle swarm optimization with dynamic adaptation ($\mathrm{PSO_{rank}}$)

The original PSO algorithm as given in [1] had difficulties in striking a balance between exploration and exploitation. Hence, the global search ability of the PSO algorithm is restricted. Therefore the PSO algorithm gets trapped in a local optimum. To address this problem, some improvements to available PSOs have been made, as described in the previous section. We introduce a novel method for updating the position of an individual based on Latané theory [20].

The sharing of information among particles can be considered a blessing, in that the particles profit from the discoveries and previous experiences of all particles during the search process, resulting in an enhancement of the convergence speed of the solver. The basic idea arose from psychological studies. Psychologists such as Breder and Latané proposed theories about social impacts in human communities.

Bibb Latané's social impact theory, conceived in the 1970s and still evolving, resulted from dozens of laboratory and field experiments with human subjects [20]. As stated by Kennedy, "Latané finds that the impact of a group on an individual is a function of the strivness, immediacy, and number of sources of influence, that is, other people". Here, strivness is a kind of social influence variable, similar to status or persuasiveness or potential, immediacy is the inverse of distance, so it increases as distance decreases, and number is simply the number of people influencing the target individual [21].

Latané had found, for instance, that the size of the tip left by a group of people in a restaurant is a function of the number of people sitting at the table—the more people in a party, the larger the total amount but the smaller the percentage each pays of the bill. The nervousness of participants in a college talent show was shown to be a function of the number of people in the audience [21]. On the basis of these observations, each individual can share knowledge with other individuals in the neighborhood. Therefore, the individuals in a swarm have better interaction mechanisms.

We use this theory for designing a variation of the PSO method as described in Section 3.1. Also, to further improve the performance, dependent random coefficients (Section 3.2) and a new non-linear inertia weight (Section 3.3) are introduced into the algorithm.

3.1. Velocity and position updating

Kennedy found that the effectiveness of the particle swarm algorithms comes from the interactions of particles with their neighbors [16]. In standard PSO only the global best particle is used to update the position of candidate particles, and the particles are attracted towards the best particle. In this way the valuable information provided by neighbor individuals was ignored. Therefore, as the particles become closer together, their search may uncover new regions that are even better and they may trap in a local optimum. By incorporating other neighbor particles as sources of influence in addition to the global best ones, the swarm has the chance to cover new regions.

On the basis of Latané theory [20], in order to efficiently control the balance between exploration and exploitation, the concept of ranking can be applied and extended to the particle swarm optimization as follows. At each iteration, after all n particles move to the new positions, the particles are sorted on the basis of their fitness values ($\text{fitness}_1(k) \le \text{fitness}_2(k) \le \cdots \le \text{fitness}_n(k)$), where $\text{fitness}_i(k)$ is the fitness of the ith particle in kth iteration. Only the γ best particles contribute in updating the position of a candidate particle. The value of γ linearly decreases as the iteration increases. The contribution of each particle is proportional to its strength. The strength value is a function of the strivness, immediacy and number of contributed particles. That is, the contribution of an individual in updating the position of a candidate particle is weighted according to the ranking function, the inverse of the distance from the particle and the number of individuals in the neighborhood. Under this configuration the velocity vector of the particles is updated according to the following equation:

$$v_d^i(k+1) = v_d^i(k) + \text{rand}_1(p_d^i - x_d^i(d)) + \text{rand}_2\left(\sum_{j=1}^{n} \psi_i^j(p_d^i(k) - x_d^i(k))\right) \tag{4}$$

$$\psi_i^j(k) = f(\tau_i^j(k), \delta_i^j(k), \xi_i) = \tau_i^j(k) \times \delta_i^j(k) \times \xi_i \tag{5}$$

where $\psi_i^j(k)$ models the influence of the neighbor particle j on the candidate particle i in the kth iteration depending on the three aforementioned parameters. The ranking parameter τ_i^j signifies the strivness of the individual j in the neighborhood of the ith particle. The strivness of the neighbor individual j depends on its relative fitness with neighbor individuals of the ith particle which is defined as

$$\tau_i^j(k) = \frac{\text{fitness}_j(k)}{\sum\limits_{l=0}^{\text{Neighbors}_i} \text{fitness}_l(k)} \tag{6}$$

where $\text{fitness}_j(k)$ is the fitness of the particle j in the neighborhood of the particle i, and Neighbors_i is the number of neighbor particles. The immediacy of individual j from particle i is defined on the basis of the Euclidean distance δ_i^j in the D-dimensional solution space. As the distance of a particle j increases from the candidate particle i, its immediacy decreases. The immediacy is defined as

$$\delta_i^j(k) = \frac{1}{\sqrt{\sum\limits_{d=1}^{D}(x_d^j(k) - x_d^i(k))^2}} \tag{7}$$

where $x_d^j(k)$ and $x_d^i(k)$ respectively represent the positions of the particle j and the candidate particle i in dimension d of the solution space. The effect of the individuals in the neighborhood of ith particle is modeled as ξ_i:

$$\xi_i = \alpha N_i^{\beta} \tag{8}$$

where N_i is the number of individuals in the neighborhood of particle i, and $0 < \alpha < 1$ and $0 < \beta < 1$ are two constant parameters which control the importance of social knowledge provided by the neighbor individuals. The impact of the neighbors on the candidate particle i increases monotonically with the size of the neighbors, but the increase slows as the new neighbor particles are added.

In addition to the aforementioned parameters, it is interesting to note that only the γ best particles are considered for updating the position of the candidate particle i. The size of the best particles, γ, is dynamically adapted throughout iterations based on iteration intervals. The value of γ is determined using the following equation:

$$\gamma = n - \left\lfloor n \times \frac{\text{iter}}{\text{max\_iter}} \right\rfloor \tag{9}$$

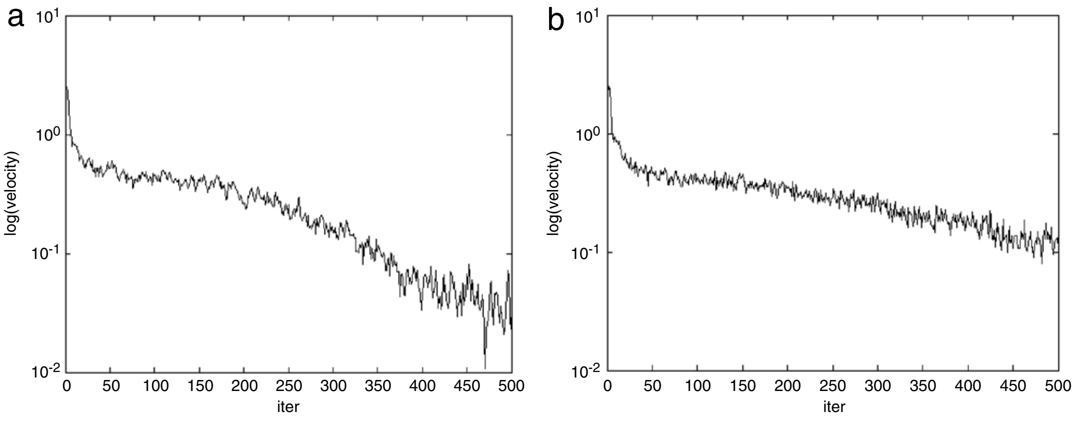


Fig. 1. Average velocity of particles using (a) dependent random coefficients, (b) independent random coefficients.

where n is the number of particles in the swarm, iter is the current iteration, and iter_max is the maximum number of iterations. The number of best particles which contribute in updating the position of a candidate particle linearly decreases as the intervals increase. In the first interval all the individuals in the neighborhood influence the candidate particle. In the last interval the candidate particle is only influenced by the global best particle in the neighborhood. Hence, in the last iteration the algorithm is transformed to the basic PSO algorithm.

Considering all the neighbor individuals as the source of influence results in an effective population based algorithm for optimizing numerical functions. The PSO_{rank} algorithm provides a swarm of particles with an extended flying pattern. The proposed flying pattern in the PSO_{rank} algorithm provides an effective way to balance between exploration and exploitation. This approach alleviates the premature convergence problem which is considered as one of the major weaknesses of the population based methods. Premature convergence occurs when the algorithm reaches its convergence rapidly in the early iterations in which the individuals of the population choose the same position and the population traps to local optima. Stagnation is another major weakness of the population based algorithms. Usually, stagnation may occur due to hard constriction of the movement trajectories of the individuals of a population. The PSO_{rank} algorithm employs shared information from the neighbors to control diversity and stochasticity of the behaviors of the particles in the swarm. This alleviates effectively hard constrictions on the movement trajectories of the particle by considering the solutions which are found by neighbor particles in updating the position of a candidate particle. The performance of PSO_{rank} depends on parameters α and β. Hence, an experiment on tuning these parameters is given in Section 5.3.

3.2. Dependent random coefficients

The PSO_{rank} algorithm enhances the balance between global and local search, but as shown in (4) the two random parameters rand_1 and rand_2 are generated independently, so in some cases the values of the rand_1 and rand_2 parameters are both too large or too small. In the former case, both the personal and social experiences accumulated so far are overused and the particle is driven too far away from the local optimum. For the latter case, both the personal and social experiences are not used fully and the convergence speed of the algorithm is reduced. To alleviate these problems we use dependent random coefficients based on random variables rand_1 and rand_2 as follows:

$$v_d^i(k+1) = v_d^i(k) + \text{rand}_1(1-\text{rand}_2)(p_d^i - x_d^i(k)) + \text{rand}_2(1-\text{rand}_1)\left(\sum_{j=1}^{n} \psi_i^j (p_d^i(k) - x_d^i(k))\right). \tag{10}$$

An experiment was conducted on the Rastrigrin function in order to analyze the impact of the dependent random variables on the evolution of the swarm. Fig. 1 shows how the average velocity of the swarm varies throughout iterations. On one hand, in the early iterations of an optimization algorithm, it is desirable that the particles of the swarm explore large areas in the solution space. The particles with large velocity vectors can wander through solution vectors to discover new regions. From the result, we can see that dependent variables relatively increase the diversity of the swarm in the early iterations. Increasing diversity provides the swarm with the ability to avoid stagnation in the early iterations by wandering through solution space. On the other hand, in the last iterations the particles of the swarm need to move more precisely in the local regions to continuously improve their performances. From Fig. 1, it is apparent that the average velocity of the swarm with dependent random variable has a smaller value than the average velocity of the swarm with independent variables. This phenomenon encourages the particles to pay more attention to finding a better position in the last iterations. This results in a faster convergence speed for the swarm. The velocities of the particles in a swarm with independent variable decreases slowly. So, the swarm needs more iteration to focus on an optimum solution.

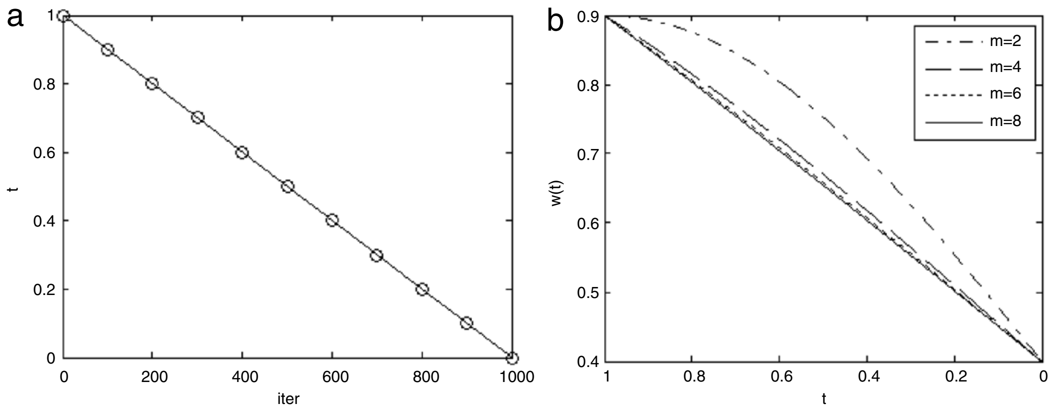


Fig. 2. (a) Evolution of parameter t throughout iterations, (b) non-linear inertia weight $w(t)$.

3.3. Non-linear inertia weight

PSO is a powerful method for finding global optima in the optimization area, but it has some deficiencies which should be resolved in order to achieve better performance. As stated before, premature convergence is one of the main deficiencies of the original PSO. The premature convergence usually occurred due to an improper mechanism of balancing between exploration and exploitation. One solution for mitigating this problem is achieved through inertia weight. The inertia weight may provide a proper balancing mechanism. The inertia weight introduces the preference for the particle to continue moving in the same direction it was going in during the previous iteration. In the first iterations the exploration is preferred and the exploitation is preferred in the last iterations. There have been broad considerations of inertia weight and many improvements presented. Eberhart and Shi [13] found that linearly decreasing the inertia is not very effective in dynamic environments. A non-linear inertia weight provides more flexibility as regards control of the balance between exploration and exploitation throughout iterations. The PSO_{rank} algorithm uses a non-linear and dynamic inertia weight. The inertia weight is based on a non-linear sinusoidal function f:

$$f(t) = \sin\left(\frac{\pi}{m} \times t\right) \tag{11}$$

$$t = \frac{\text{max\_iter} - \text{iter}}{\text{max\_iter}} \tag{12}$$

where the constant factor m can vary between 2 and 10 ($2 \le m \le 10$). The behavior of the inertia weight is controlled by adjusting the parameters m and t. These parameters result in a wide range of behavior from near linear to periodic with multiple minimum and maximum values.

A sinusoidal function generates the data in the range of $[-1, 1]$; so we need to transform them to values in a predefined range. The max–min normalization is used to perform linear transformation on the data produced by the function. By this transformation, the inertia weight factor w is set to change non-linearly according to the following equation:

$$w(t) = \frac{f(t) - f_{\min}}{f_{\max} - f_{\min}} \times (w_{\max} - w_{\min}) + w_{\min} \tag{13}$$

where $w_{\min}$ and $w_{\max}$ respectively represent the minimum and maximum values of the inertia weight. The minimum and maximum values of the function are represented by $f_{\min}$ and $f_{\max}$. In our experiments, $w_{\min}$ is set to 0.4, while $w_{\max}$ is set to 0.9. Adjusting this sinusoidal inertia weight with the previously described values results in a non-linear decreasing coefficient that significantly improves the performance of the original PSO. In this work, the expression $\frac{\pi}{m} \times t$ is constricted to vary only in the range $\left[\frac{\pi}{2}, 0\right)$. For this configuration, the maximum value of the sinusoidal function will be $f_{\max} = \sin\left(\frac{\pi}{m}\right)$, and the minimum value will be zero. Fig. 2 presents parameter t and inertia weight $w(t)$. For large values of m, the inertia weight $w(t)$ acts as a linear inertia weight. As the value of m decreases, the preference for exploration in early iterations increases; in the last iterations, the inertia weight decreases rapidly, encouraging the swarm to exploit local optima.

4. Searching behavior of PSO_{rank}

As a convenient observation, the searching behavior of the PSO_{rank} algorithm for a group of $n = 20$ particles on a two-dimensional Rastrigrin function is presented. Fig. 3 presents the distribution of particles with different roles at the 100th, 200th, 300th, and 400th iterations. The initial state of the particles is chosen at random. It is apparent that after some iteration the particles converge to a local optimum. As described before and shown in this figure, we can see that the distribution of particles in the first iterations depicts that the PSO_{rank} algorithm prefers exploration, while the shrinking particles in the

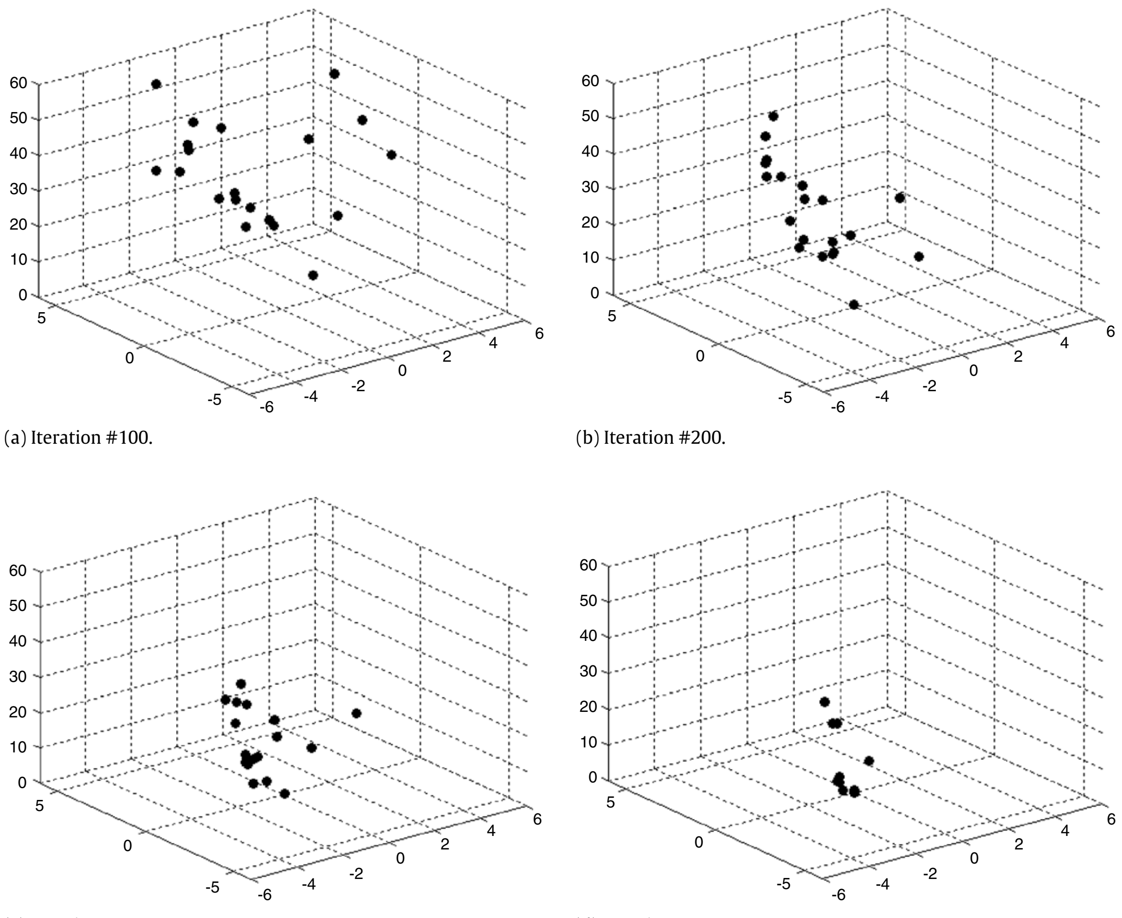

(a) Iteration #100.
(b) Iteration #200.
(c) Iteration #300.
(d) Iteration #400.

Fig. 3. 3D snapshot of the $\mathrm{PSO_{rank}}$ algorithm for a two-dimensional Rastrigrin function.

last iterations represent that the $\mathrm{PSO_{rank}}$ algorithm tends to exploit around local optima. This process has occurred due to stochastic behaviors of particles which modify the topology of the swarm constantly. At each cycle of the algorithm, a particle selects the γ best neighbors as the source of influence. Using this influence approach provides the ability for the swarm to explore a wide region in the early iterations. This occurs due to the contributions of large numbers of neighbors in the early iterations. In the last iterations, on decreasing the value of γ, the swarm prefers exploitation. In particular, in the last interval, the swarm gravitates toward the global best particle. This process provides a powerful way to maintain balance between exploration and exploitation. It controls the global search while maintaining the convergence towards the global optimum.

5. Experiments

In this section, the experiments that have been done to evaluate the performance of the proposed PSO algorithm for a number of analytical benchmark functions are described. Three sets of experiments were conducted. In first set of experiments, we tune the parameters of the $\mathrm{PSO_{rank}}$ algorithm. In a second experiment, the performance of the $\mathrm{PSO_{rank}}$ algorithm is evaluated in comparison with seven other variants of the PSO algorithm. The continuous test functions in this experiment have been extensively used to compare PSO algorithms. In a third algorithm, to evaluate the performance of the proposed algorithm in a real-world application, we have trained a feed forward neural network using $\mathrm{PSO_{rank}}$ and then used the trained network for image compression, and the compression results are compared against those from two other methods called BP and PSO-BP.

5.1. Benchmarks

To test the performance of $\mathrm{PSO_{rank}}$, six well known benchmark functions are used here for comparison, both in terms of the optimum solution after a predefined number of iterations and the rate of convergence to the optimum solution. These benchmarks are widely used in evaluating the performance of population based methods [3,5,7,9].

Table 1
Optimization test functions.

Function name	Formula	Opt. position	Opt. value	Trait
f_{Sph}, Sphere	$f_1(x) = \sum_{i=1}^{n} x_i^2$	$(0, 0, \ldots, 0)$	0	Unimodal
f_{Ros}, Rosenbrock	$f_2(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	$(1, 1, \ldots, 1)$	0	Unimodal
f_{Ras}, Rastrigin	$f_3(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	$(0, 0, \ldots, 0)$	0	Multimodal
f_{Ack}, Ackley	$f_4(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_1^n x_i^2}\right) - \exp\left(\frac{1}{n}\sum_1^n \cos(2\pi x_i)\right) + 20 + e$	$(0, 0, \ldots, 0)$	0	Multimodal
f_{Gri}, Griewank	$f_5(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$(0, 0, \ldots, 0)$	0	Multimodal
f_{Sch}, Schaffer's f6	$f_6 = 0.5 + \frac{(\sin\sqrt{(x_1^2 + x_2^2)})^2 - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}$	$(0, 0, \ldots, 0)$	0	Multimodal

Table 2
Parameter settings for the test functions.

Function	Domain	Asymmetric initialization range	$X_{\max}$	$V_{\max}$
f_{Sph}	$[-100, 100]$	$(50, 100)^n$	100	100
f_{Ros}	$[-30, 30]$	$(15, 30)^n$	30	30
f_{Ras}	$[-5.12, 5.12]$	$(2.56, 5.12)^n$	5.12	5.12
f_{Ack}	$[-30, 30]$	$(15, 30)^n$	30	30
f_{Gri}	$[-600, 600]$	$(300, 600)^n$	600	600
f_{Sch}	$[-100, 100]$	$(50, 100)^n$	100	100

Table 1 gives the test function, the mathematical expression, its optimum value and the function trait. The first two functions are simple unimodal functions (they have a single local optimum that is a global optimum) while the other four functions are multimodal with many local optima. We select these test functions as each of them is a candidate for a different class of real-world problems. They have different characteristics (e.g. they are unimodal or multimodal, or have dependent or independent variables). A robust optimization algorithm maintains balance between exploration and exploitation, controlling diversity, and mitigating premature convergence and stagnation to cope with problems of different types.

The "Sphere" function has independent variables, contains no local optima, and has a smooth gradient toward the global optimum. It represents an easy problem which is successfully solved by many population based optimization algorithms. The Rosenbrock function has smooth slope around its global optimum position, its global optimum lies inside a long, narrow, and parabolic shaped flat valley, its variables are strongly dependent, and the gradient does not point towards its optimum position. All of these features mean that achieving convergence toward the global optimum with the Rosenbrock function is relatively difficult. This function has been frequently used to test optimization algorithms. The algorithms with hard constrictions on movement trajectories of their individuals may easily encounter stagnation.

The most common initialization ranges used in the literature for the benchmarks are considered in this paper. Each particle has been initialized with a random position and a random velocity where in both cases the values have been randomly chosen in the range $\left[\frac{X_{\max}}{2}\right]$. We set $X_{\max} = V_{\max}$. To control the explosion of each particle, Eberhart and Shi [21] suggested that it is good to limit excessive searching outside the predefined search space. During a run of each algorithm, the values of the position of a particle are limited to the interval $[X_{\min}; X_{\max}]$ and the maximum velocity restricted to $V_{\max}$. Table 2 presents the function interval, asymmetric initialization range, and maximum velocity and maximum position of a particle for each test function.

Schaffer's f6 is a multimodal function with dependent variables. It has smooth slope around the global optimum, so methods with poor flying patterns encounter problems in regions near the global optimum. An optimization algorithm should increase diversity of the population to cope with problems of this type.

Ackley's function is a widely used multimodal test function. This function has one narrow global optimum basin and numerous local optima. In comparison with other multimodal functions, it represents a relatively easy problem as it has shallow local optima.

There is no dependency between variables of the Rastrigin test function. The cosine modulation produces frequent local optima. So, this test function is highly multimodal which makes it a complex problem. An optimization algorithm should provide an efficient balance between exploration and exploitation and have good diversity in order to overcome the problems of this type.

Griewank's function is based on the Sphere function. Like the Rastrigin function it has many widespread local optima, regularly distributed. Its second component represents a linkage between variables which make it a difficult multimodal problem. The local optima are located in the direction of the gradient, so an optimization algorithm should provide an efficient balance between global and local search in order to solve this type of problem. Griewank's function with high dimensionality seems unimodal.

5.2. Settings of the algorithms

Simulations were performed in order to observe the performance of the proposed algorithm for finding optimum solutions. The performance of the new method is compared with those of the TPSO [5], IPSO [7], PSO-NTVE [3], CenterPSO [9], FIPS [18], ScatterPSO [19], and CLPSO [4] methods. These algorithms have a few parameters; parts of them are common while other parts are specific to each algorithm.

Common parameters are the number of dimensions for the search space, maximum generation, population size, and total number of trials. For all test functions with the exception of the 2D function Schaffer's f6, three different dimension sizes, 10, 20 and 30, are tested. The corresponding maximum generations are 1000, 1500 and 2000 respectively. For Schaffer's f6 function, the maximum generation is set to 1000. For all the test functions, the population size is set to 30, and a total of 100 runs for each experimental setting are conducted.

Besides the common parameters, the same settings as presented in [3,5,9,18,19,22] are used for the algorithms investigated in this consideration. Four variants of TPSO were proposed in [5]. The best version of the TPSO algorithm (i.e. TPSO-2) is used. In TPSO, the parameter $T_{\max}$ is chosen as 5.

For the IPSO algorithm, the linear version of the inertia weight is used, which decreases from 0.9 to 0.4, and the acceleration coefficients c_1 and c_2 are set to 2.0. The number of sub-swarms, p, is chosen as 4, and the number of points in each sub-swarm is used as $m = 2q$, where q is the population size of particles selected from the points in the sub-swarm. Other parameters in IPSO are set as $G = KT$, and $T = 2n$, where G is the maximum number of iterations, T is the time for which each sub-swarm executes PSO independently and K is the number of sub-swarms shuffling.

The performance of the PSO-NTVE algorithm strongly depends on three parameters: α, β, and γ. As described in [3] all these parameters are within the set $\{0, 1, 1.5, 2\}$. For each of the benchmark functions, different values of the α, β, and γ parameters are used. The value of α is chosen as 0.5 for all the benchmark functions. The value of β is chosen as 1.5 for the Sphere and Schaffer's f6 functions, 1 for Rosenbrock functions, and 0.5 for the remaining benchmark functions. The parameter γ is set to 1.0 for the Sphere and Griewank's functions, 1.5 for the Rosenbrock and Schaffer's f6 functions, and 0.5 for the Rastrigrin and Ackley functions. Finally, the $L_{25}(5^3)$ orthogonal array is used as described in [3].

Like for the IPSO algorithm, in the CenterPSO algorithm, the inertia weight w linearly decreased from 0.9 to 0.4 and the acceleration coefficients are $c_1 = c_2 = 2$. In CenterPSO, the number of center particles is chosen as 1, so we have 29 ordinary particles in the swarm.

In our experiments, the FIPS (fully informed particle swarm) algorithm with the U-ring topology that produced the best results [18] is used.

The linear inertia weight is used in CLPSO. The learning rate $c = 1.5$ is chosen. Different values of the refreshing gap m are used for benchmark functions. The values are selected to produce the best result for each benchmark function. The value of m is selected as 7. The learning probability P_c is set for each particle using the approach presented in [4].

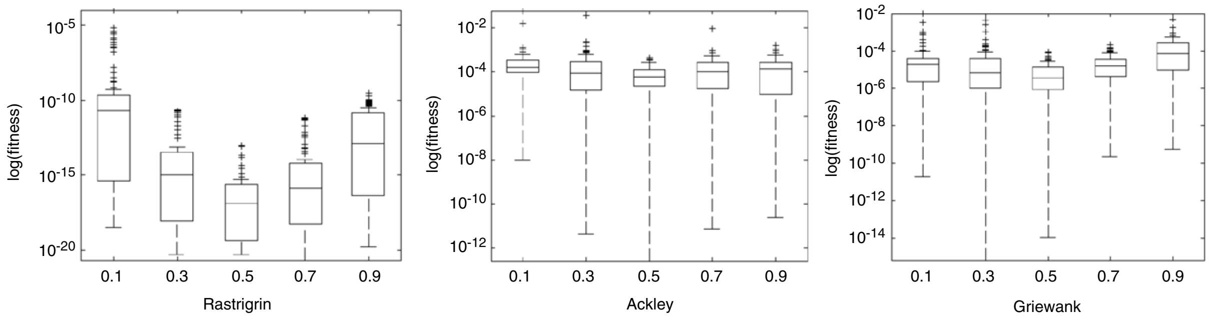
The self-variant A of ScatterPSO is used in this study in which each particle learned from every other member in the swarm. The number of guiding points is chosen as 3.

5.3. Sensitivity to parameters

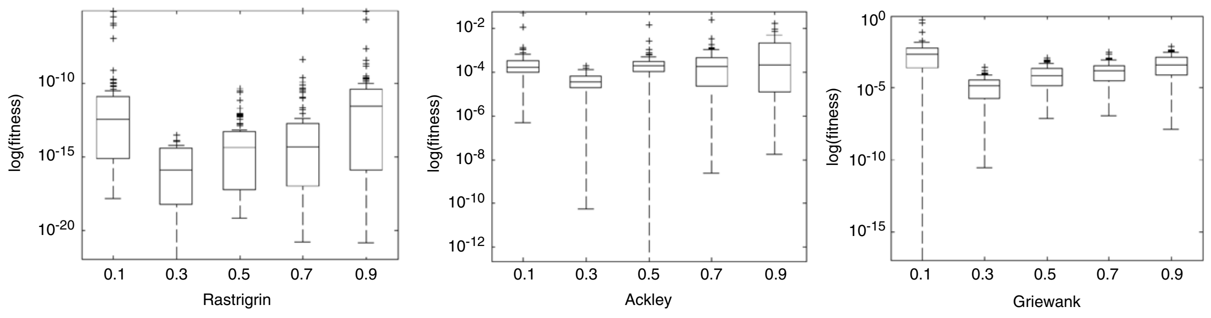
The method presented seems to be a good compromise. The method indicates that exploitation and exploration are at a very high level and well balanced. However, the performance of the PSO_{rank} algorithm depends on some parameters. The optimal combination of α and β should be determined in order to obtain best solutions by using the PSO_{rank} algorithm. Also the parameter m of the inertia weight should be tuned. The PSO_{rank} algorithm was analyzed in terms of its performance and stability in order to calibrate best values for α and β parameters. A large number of experiments have been conducted in which the number of particles used was 20.

(1) *Performance analysis*: We have used three well known multimodal Rastrigrin, Ackley, and Griewank functions in ten dimensions to investigate how the algorithm is affected by these parameters. The maximum number of iterations was set to 1000. Each function was tested with different values of α and β, 50 times. The parameters α and β have many possible values in the range of $(0, 1)$. It may not be possible to evaluate all the combinations of these parameters. Hence, the possible values for the parameters are discretized into five buckets of length 0.2. Ten distinct points selected randomly from each bucket and all the combinations of these points with points in another bucket are considered. So, for each pair of buckets one hundred combinations of α and β parameters were considered and the distribution of solutions for 50 runs was recorded for each combination. To provide a comprehensive representation of the results, the best, worst, and median of the fitness for each pair of buckets are presented using a box plot. The box plots of the fitness values are presented in Fig. 4(a)–(b). Each box plot in Fig. 4(a) represents the distribution of fitness values for one bucket of α and all the buckets of β parameter, and each box plot in Fig. 4(b) represents the distribution of fitness values for one bucket of β and all the buckets of α parameter. The horizontal line within a box encodes the median, while the top and bottom of a box encode the Q1 and Q3 quartiles. The dashed lines describe the shape of the distribution for fitness values, while two small horizontal lines at the ends of dashed lines represent minimum and maximum values. Finally, plus signs represent outliers.

From the result, it is apparent that the performance of the PSO_{rank} algorithm is influenced by α and β parameters. Using marginal values for both of the parameters is not a good idea in optimizing test functions using the PSO_{rank} algorithm. In other words, the performance of the PSO_{rank} algorithm decreases in quality if both of the parameters have small values (i.e. close



(a) Horizontal axis is the ranges of α buckets.



(b) Horizontal axis is the ranges of β buckets.

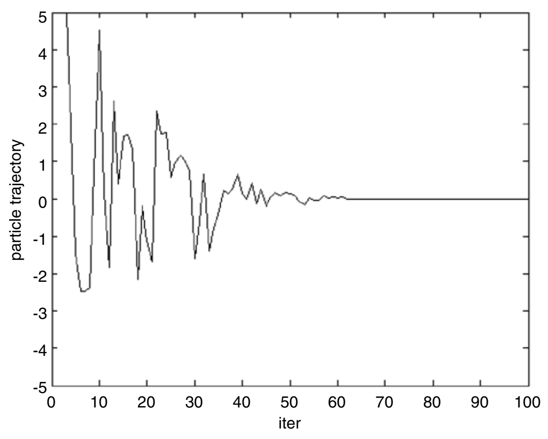
Fig. 4. Results from the PSO_{rank} algorithm for three test functions for different values of α and β. Each bucket is presented at the center of its corresponding range.

to 0) or large values (i.e. close to 1). For small α and β values ($\alpha < 0.1$ and $\beta < 0.1$) the influence of neighbor individuals on the candidate particle decreases and the particle has more reliance on its own thinking. The neighbor individuals have considerable influence on the candidate particle when both α and β have large values ($\alpha > 0.9$ and $\beta > 0.9$). For this configuration, the velocity of particles increases. Hence, the diversity of the algorithm is highly increased and the algorithm needs more time to converge towards the optimum solution. It is interesting to note that for small values of α and relatively large values of β ($\alpha < 0.1$ and $\beta > 0.8$) as well as for relatively large values of α and small values of β ($\alpha > 0.8$ and $\beta < 0.1$) the PSO_{rank} algorithm obtained relatively good results. However, it is clear from Fig. 4 that the best results were obtained when $0.4 < \alpha < 0.6$ and $0.2 \le \beta < 0.4$.

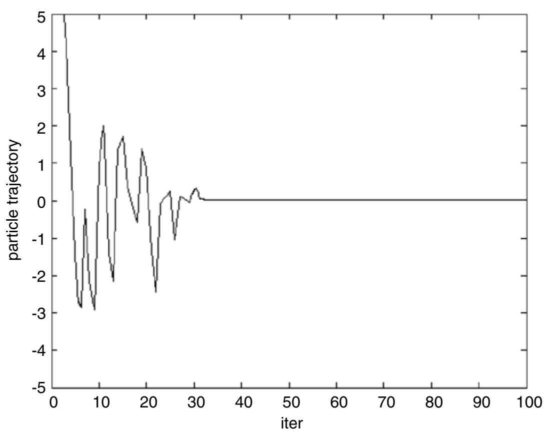
(2) *Stability analysis*: The PSO_{rank} algorithm works by adjusting the trajectories of each of the particles towards its own best position and towards the γ best particles. More precisely, trajectories of each particle are adjusted by considering social knowledge provided by the best particles. The importance of social knowledge is affected by the α and β parameters. Hence, in order to analyze the behavior of the swarm under different conditions, the trajectories of the particles are considered. The trajectories of the particles are used to study the stability of the PSO_{rank} algorithm. We assume that stability occurs when the particles are positioned in the search area defined by a threshold around the optimum position. The search region is defined as $S_{\text{stab}} = \{x : -\phi < x < \phi\}$, where ϕ is the threshold. We have used ϕ as 0.01. The trajectories of the particles in optimizing a one-dimensional Rastrigrin function are give in Fig. 5(a)–(d), demonstrating the stability of the particle dynamics. It should be noted that due to the large number of possible combinations of α and β parameters, a small but representative subset of experiments are presented in Fig. 5.

The particle trajectories show that the PSO_{rank} algorithm guarantees stability under different conditions. The interesting property of the figures is the time at which the algorithm stabilizes. It is clear from Fig. 5 that the time of stability differs under different conditions. Again, the experiments show that the best results are obtained when the α and β parameters are in the ranges of $0.4 < \alpha < 0.6$ and $0.2 \le \beta < 0.4$. For this configuration, the PSO_{rank} algorithm achieves the fast speed of convergence to the success criteria described in Section 4. Hence, we have selected α and β parameters from the aforementioned ranges for the experiments conducted in this study. Our empirical study showed that the best results were obtained around $\alpha = 0.45$ and $\beta = 0.385$, as presented in Fig. 5(b). When large values are used for the α and β parameters ($\alpha > 0.9$ and $\beta > 0.9$), the diversity of the algorithm increases and stability occurs in the last iterations, as presented in Fig. 5(d). It seems that for a high dimensional test function, the PSO_{rank} algorithm needs more time to converge to the global optimum. For small values of the α and β parameters ($\alpha < 0.1$ and $\beta < 0.1$), the convergence speed of the algorithm decreases, as presented in Fig. 5(a).

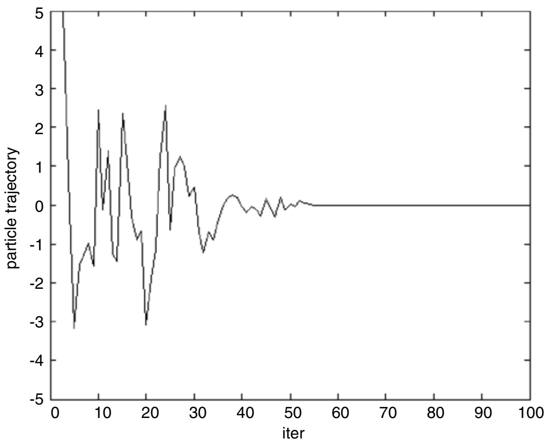
(3) *Conclusions from performance and stability analysis*: In general, by considering the results from performance and stability analysis, we see that the PSO_{rank} algorithm obtained a faster convergence speed and better performance when α was drawn



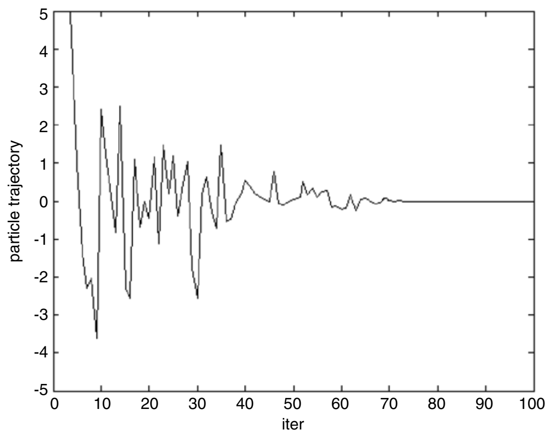
(a) $\alpha = 0.05$ and $\beta = 0.05$.



(b) $\alpha = 0.45$ and $\beta = 0.385$.



(c) $\alpha = 0.3$ and $\beta = 0.6$.



(d) $\alpha = 0.95$ and $\beta = 0.95$.

Fig. 5. Trajectories of the particles for different settings of α and β parameters.

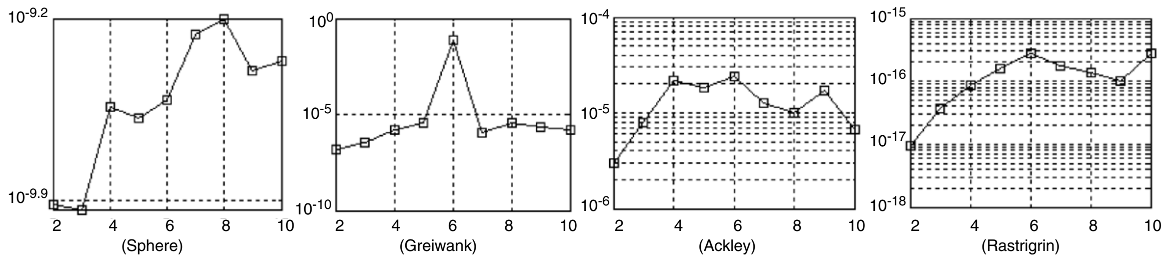


Fig. 6. Results from the PSO$_{\text{rank}}$ algorithm for four test functions for different values of the parameter m.

from the third bucket and β was drawn from the second bucket. The marginal values for both of the parameters are not good choices for the PSO$_{\text{rank}}$ algorithm.

(4) *Tuning the non-linear inertia weight*: The parameter m should be tuned. We used Sphere, Griewank, Ackley, and Rastrigrin test functions in ten dimensions to investigate the impact of this parameter. Each function was tested with different values of m, 50 times. The average fitness of the PSO$_{\text{rank}}$ algorithm is represented in Fig. 6. It is clear from the results that for the Sphere function, $m_1 = 3$ produces the best results. For other test functions, the best results were obtained for $m = 2$. From Fig. 6, we can see that the inertia weight $w(t)$, which has interesting properties and provides a better balance between exploration and exploitation throughout the iterations, depends on the parameter m. This property of inertia weight empowers the PSO$_{\text{rank}}$ algorithm to produce better results in comparison with the linear inertia weight. In our experiments the PSO$_{\text{rank}}$ algorithm uses the parameter $m = 2$.

Table 3
Mean and standard deviation for ten-dimensional test functions.

Method	Function				
	f_{Sph}	f_{Ros}	f_{Ras}	f_{Gri}	f_{Ack}
Standard PSO	4.38E−05 ± (2.57E−05)	1.64E+01 ± (2.93E+00)	1.34E+00 ± (8.26E−01)	6.20E−02 ± (8.17E−02)	5.31E−02 ± (7.72E−02)
PSO + RC	3.50E−07 ± (6.63E−07)	5.59E+00 ± (8.18E−01)	1.95E−02 ± (3.52E−02)	5.20E−02 ± (3.17E−02)	4.66E−02 ± (8.67E−02)
PSO + IW	9.23E−05 ± (1.94E−04)	1.72E+00 ± (6.47E−01)	7.46E−03 ± (9.30E−03)	2.75E−02 ± (4.56E−02)	8.92E−03 ± (1.34E−02)
PSO + RC + IW	1.68E−06 ± (4.32E−06)	9.84E−01 ± (5.14E−01)	3.29E−03 ± (7.44E−03)	3.42E−03 ± (7.05E−03)	1.57E−03 ± (3.12E−03)
PSO$_{rank}$−RC−IW	7.80E−11 ± (9.26E−11)	4.63E−02 ± (6.34E−02)	1.35E−13 ± (4.64E−13)	1.15E−04 ± (1.72E−04)	2.33E−05 ± (7.63E−05)
PSO$_{rank}$ − IW	**4.12E−11** ± (**4.84E−11**)	2.91E−01 ± (6.74E−01)	5.82E−15 ± (1.03E−14)	4.83E−05 ± (5.49E−05)	9.29E−06 ± (2.47E−05)
PSO$_{rank}$ − RC	3.04E−10 ± (6.83E−10)	1.69E−02 ± (2.93E−02)	**0.00E+00** ± (**0.00E+00**)	**1.42E−05** ± (**4.75E−05**)	9.26E−06 ± (2.38E−05)
PSO$_{rank}$	3.77E−10 ± (1.22E−09)	**5.02E−03** ± (**9.77E−03**)	**0.00E+00** ± (**0.00E+00**)	2.83E−05 ± (**6.29E−05**)	**1.17E−06** ± (**4.92E−06**)

5.4. *The effects of three mechanisms*

The proposed algorithm is designed on the basis of three different mechanisms: those of Latané theory, dependent random coefficients, and non-linear inertia weight. Hence, its performance may be influenced by each of these mechanisms. To provide a comprehensive study on the effects of these mechanisms, an experiment was conducted by incorporating different combinations of these mechanisms into the standard PSO algorithm [21]. On the basis of this configuration, eight variants of standard PSO will emerge:

(1) PSO: standard PSO.
(2) PSO + RC: PSO with dependent random coefficients.
(3) PSO + IW: PSO with non-linear inertia weight.
(4) PSO + RC + IW: PSO with both dependent random coefficients and non-linear inertia weight.
(5) PSO$_{rank}$ − RC − IW: PSO with only Latané theory.
(6) PSO$_{rank}$ − IW: PSO with Latané theory and dependent random coefficients.
(7) PSO$_{rank}$ − RC: PSO with Latané theory and non-linear inertia weight.
(8) PSO$_{rank}$: PSO with Latané theory, dependent random coefficients, and non-linear inertia weight.

In this experiment, five test functions in ten dimensions are considered. The maximum number of iterations was set to 1000 and a total of 50 runs for each experimental setting were conducted. The effects of the proposed mechanism are given in Table 3 in terms of mean and standard deviation. It can be seen from the results that the dependent random coefficients and the non-linear inertia have positive effects on the performance of the standard PSO algorithm. Incorporating these two mechanisms results in three variants of standard PSO (i.e. PSO + RC, PSO + IW, PSO + RC + IW) which produce competitive results as compared to standard PSO. Although PSO with these two mechanisms produced relatively good results, better performances can been obtained by the other variants of the standard PSO algorithm employing Latané theory.

Incorporating only Latané theory into the standard PSO, PSO$_{rank}$ − RC − IW will emerge. The results show that the Latané theory has a positive effect on standard PSO. Significant improvements can be obtained by incorporating Latané theory in treating Sphere and Rastrigrin functions. Also, better performances were obtained for the remaining test functions compared to the PSO variants without Latané theory. However, more improvement has been obtained by adding non-linear inertia weight into the Latané theory (i.e. PSO$_{rank}$ − RC), dependent random coefficients (PSO$_{rank}$ − IW), or both of them (PSO$_{rank}$). The results show that the PSO algorithm with three mechanisms surpasses other variants for most of the test functions.

Conclusion: In general, PSO$_{rank}$ performs better than other variants of standard PSO on most of test functions. We can say that, although each of the proposed mechanisms has a positive effect on the performance of the standard PSO algorithm, Latané theory provides more efficiency. Hence, we suggest the PSO variant which employs the three proposed mechanisms.

5.5. *Comparative study*

We present different analyses based on the some dependent measures. The measures provide the ability to consider algorithms from different perspectives. Following the experiments we present the overall performance of the proposed method compared to other methods.

5.5.1. *Performance evaluation*

The numerical results for each test function are recorded in Tables 4–6. In the experiments the number of iterations for reaching a predefined threshold was specified for each function. Different success criteria for different functions are presented in the literature. For Schaffer's f6, the success criterion is set to 0.000001, whereas for the other functions, the success criteria are set to 0.01. After the maximum iteration, if the minimum value reached by the algorithm was not below the threshold, the run was considered unsuccessful. Fitness smaller than E−15 was considered as zero.

(1) *Results for* 10-*dimensional functions*: Table 4 represents three measures (mean, standard deviation and success ratio) for 100 runs of the seven algorithms for five test functions in 10 dimensions as well as Schaffer's f6 in two dimensions. The mean and standard deviation represent the quality of the results obtained by each algorithm, and the success ratio shows

Table 4
Results for Schaffer's f6 in two dimensions and other benchmark functions in ten dimensions.

Method	TPSO		IPSO		PSO-NTVE		CenterPSO	
Function	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.
f_{Sph}	2.18E−08 ± (3.98E−08)	**1**	1.74E−09 ± (5.41E−08)	**1**	9.13E−03 ± (1.07E−02)	**1**	7.50E−3 ± (9.81E−03)	**1**
f_{Ros}	4.62E+01 ± (1.51E+01)	XX	1.05E+01 ± (6.79E+00)	XX	3.26E−01 ± (7.31E−01)	0.27	1.47E+00 ± (1.95E+00)	XX
f_{Ras}	4.17E+01 ± (1.72E+01)	XX	3.29E+00 ± (1.31E+00)	0.09	9.56E−01 ± (1.02E+00)	0.35	2.30E+00 ± (1.59E+00)	0.13
f_{Ack}	2.31E+00 ± (1.02E+00)	0.02	7.42E−02 ± (1.57E−01)	0.05	5.37E−02 ± (9.41E−02)	0.16	1.70E+00 ± (9.17E−01)	0.06
f_{Gri}	9.16E−02 ± (1.80E−01)	0.21	7.84E−02 ± (1.63E−01)	0.27	2.39E−02 ± (1.21E−02)	0.34	3.65E−02 ± (7.25E−02)	0.29
f_{Sch}	1.17E−03 ± (2.83E−02)	0.09	6.80E−03 ± (1.08E−02)	0.02	1.61E−03 ± (3.76E−04)	0.07	2.72E−03 ± (1.42E−02)	0.03
Method	FIPS		Scatter PSO		CLPSO		PSO_{rank}	
Function	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.
f_{Sph}	7.28E−07 ± (3.23E−06)	**1**	9.62E−08 ± (2.58E−07)	**1**	3.96E−06 ± (4.73E−05)	**1**	**1.21E−10** ± (**8.36E−10**)	**1**
f_{Ros}	7.43E+00 ± (4.79E+00)	XX	3.72E−02 ± (2.48E−01)	0.79	5.22E+00 ± (2.63E+00)	XX	**9.14E−3** ± (**1.42E−02**)	**0.96**
f_{Ras}	3.54E+00 ± (2.64E+00)	XX	1.69E+00 ± (7.31E−01)	XX	**0.00** ± (**0.00**)	**1**	**0.00** ± (**0.00**)	**1**
f_{Ack}	7.31E−05 ± (2.26E−04)	**1**	3.58E−03 ± (8.14E−03)	0.19	8.50E−06 ± (3.84E−05)	**1**	**1.31E−06** ± (**6.54E−06**)	**1**
f_{Gri}	8.73E+00 ± (5.28E+00)	XX	5.35E−01 ± (1.18E+00)	0.08	7.48E+00 ± (4.07E+00)	XX	**2.53E−05** ± (**3.47E−05**)	**1**
f_{Sch}	5.71E−06 ± (1.14E−05)	0.38	7.36E−04 ± (1.37E−03)	0.18	1.62E−06 ± (5.08E−06)	0.93	**0.00** ± (**0.00**)	**1**

Table 5
Results for the benchmark functions in 20 dimensions.

Method	TPSO		IPSO		PSO-NTVE		CenterPSO	
Function	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.
f_{Sph}	1.70E−05 ± (3.86E−05)	**1**	2.25E−07 ± (3.19E−06)	**1**	8.71E−03 ± (1.14E−02)	**1**	4.81E−03 ± (7.39E−03)	**1**
f_{Ros}	1.18E+02 ± (3.01E+01)	XX	7.57E+01 ± (5.69E+01)	XX	3.80E+00 ± (3.62E+00)	0.15	4.13E+01 ± (3.95E+00)	XX
f_{Ras}	1.27E+02 ± (3.71E+01)	XX	1.64E+01 ± (9.14E+00)	XX	1.15E+01 ± (9.63E+00)	XX	1.09E+01 ± (3.98E+00)	XX
f_{Ack}	2.48E+00 ± (9.30E−01)	XX	9.70E−02 ± (1.39E−01)	0.03	1.07E−01 ± (1.96E−01)	XX	2.74E+00 ± (1.04E+00)	XX
f_{Gri}	5.90E−02 ± (1.21E−01)	0.37	2.36E−02 ± (3.93E−02)	0.43	9.76E−03 ± (2.03E−02)	0.71	7.30E−03 ± (1.40E−02)	0.78
Method	FIPS		Scatter PSO		CLPSO		PSO_{rank}	
Function	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.
f_{Sph}	3.27E−6 ± (1.13E−05)	**1**	7.43E−07 ± (3.64E−06)	**1**	5.34E−5 ± (9.73E−5)	**1**	**1.08E−09** ± (**3.76E−09**)	**1**
f_{Ros}	1.54E+01 ± (8.61E+00)	XX	**1.53E+00** ± (**3.74E+00**)	**0.59**	1.23E+01 ± (6.94E+00)	XX	1.61E+00 ± (2.04E+00)	0.56
f_{Ras}	3.91E+01 ± (1.12E+01)	XX	8.83E+00 ± (3.95E+00)	XX	8.52E−12 ± (3.27E−11)	**1**	**0.00** ± (**0.00**)	**1**
f_{Ack}	4.92E−03 ± (9.63E−03)	0.83	2.89E−01 ± (6.35E−01)	0.09	1.86E−05 ± (6.49E−05)	**1**	**4.22E−6** ± (**9.11E−06**)	**1**
f_{Gri}	2.91E+00 ± (1.83E+00)	0.05	2.49E−02 ± (6.76E−02)	0.73	1.36E+00 ± (9.62E−01)	0.21	**4.47E−7** ± (**7.69E−7**)	**1**

Table 6
Results for the benchmark functions in 30 dimensions.

Method	TPSO		IPSO		PSO-NTVE		CenterPSO	
Function	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.
f_{Sph}	5.30E−04 ± (8.72E−04)	**1**	3.48E−05 ± (2.94E−04)	**1**	1.04E−02 ± (1.07E−02)	0.86	2.67E−03 ± (4.51E−03)	**1**
f_{Ros}	2.80E+02 ± (9.12E+01)	XX	9.98E+01 ± (7.43E+01)	XX	1.73E+01 ± (2.11E+01)	0.09	6.14E+01 ± (5.32E+01)	XX
f_{Ras}	2.72E+02 ± (8.14E+01)	XX	3.50E+01 ± (2.47E+01)	XX	3.11E+01 ± (1.89E+01)	XX	2.16E+01 ± (6.49E+00)	XX
f_{Ack}	2.37E+00 ± (8.32E−01)	XX	1.06E−01 ± (1.78E−01)	XX	2.37E−01 ± (2.42E−01)	XX	3.18E+00 ± (1.45E+00)	XX
f_{Gri}	3.72E−02 ± (9.51E−02)	0.46	1.65E−02 ± (2.64E−02)	0.54	9.43E−03 ± (1.97E−02)	0.63	4.74E−03 ± (8.36E−03)	0.84
Method	FIPS		Scatter PSO		CLPSO		PSO_{rank}	
Function	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.	Mean (std. dev.)	S.R.
f_{Sph}	2.69E−04 ± (6.84E−04)	**1**	4.83E−06 ± (9.14E−06)	**1**	7.46E−4 ± (1.73E−3)	**1**	**2.05E−08** ± (**6.41E−08**)	**1**
f_{Ros}	3.12E+01 ± (1.76E+00)	XX	1.45E+01 ± (1.08E+00)	0.07	2.72E+01 ± (1.86E+01)	XX	**1.27E+01** ± (**1.39E+01**)	**0.19**
f_{Ras}	8.30E+01 ± (2.15E+01)	XX	2.04E+01 ± (9.61E+00)	XX	7.64E−8 ± (2.95E−7)	**1**	**0.00** ± (**0.00**)	**1**
f_{Ack}	4.81E−02 ± (9.17E−01)	0.62	3.26E+00 ± (1.92E+00)	XX	8.17E−05 ± (2.19E−04)	**1**	**3.12E−05** ± (**8.35E−05**)	**1**
f_{Gri}	8.23E−02 ± (1.13E−01)	0.31	8.93E−03 ± (2.76E−02)	0.78	2.11E−02 ± (5.51E−02)	0.45	**2.73E−8** ± (**5.24E−8**)	**1**

the robustness of the algorithms in solving optimization problems. For ease of observation, the best results obtained by the algorithms are shown in bold. The XX sign shows that an algorithm never converged to the success criteria.

All of the algorithms successfully optimize the Sphere function. They have success ratio 1 for that function. The mean result from PSO_{rank} is better than those from the other algorithms for the Sphere function. For two-dimensional Schaffer's f6, PSO_{rank} significantly outperforms all the other algorithms. Also, the results show that methods which consider more sources of influence such as FIPS, ScatterPSO and CLPSO surpass other algorithms such as IPSO, TPSO, PSO-NTVE, and CenterPSO.

From the result, we can see that PSO_{rank} and CLPSO produce the same result for the Rastrigrin function. Both of them are much better than all the other algorithms for this function. The Rastrigrin function with numerous local optima represents a complex problem for which all the algorithms except PSO_{rank} and CLPSO become trapped in local optima.

The importance of considering social knowledge provided by all the particles in the swarm can be seen much better in optimizing the Ackley function with frequent local optima. It is apparent from Table 4 that considering all the neighbors or part of them in algorithms such as PSO_{rank}, CLPSO, and FIPS renders them capable of optimizing the Ackley function with success ratio 1. Usually, an optimization algorithm tends to move in the direction of the gradient. In the Griewank function, the local optima are placed in the direction of the gradient. So, an optimization algorithm should provide an appropriate level of global search for escaping from the local optima and moving toward global optima. As can be seen from Table 4, the PSO_{rank} algorithm surpasses all the other algorithms. FIPS and CLPSO never converged to the criteria.

The Rosenbrock function is a unimodal function that can be used to evaluate the ability of an algorithm in mitigating the stagnation problem. Due to a long, narrow, and parabolic shaped flat valley around global optima in the Rosenbrock function, stagnation may occur in an optimization algorithm. An algorithm may mitigate such problem by increasing the diversity of the swarm. Parts of algorithms containing TPSO, IPSO, CenterPSO, CLPSO, and FIPS fail in converging to the criteria. One reason for this problem is that useful information provided by the neighbor particles beyond *gbest* is ignored by algorithms such as IPSO or TPSO. It seems that an optimization algorithm in which a particle is influenced by more than one neighbor such as PSO_{rank} and ScatterPSO provide appropriate levels of diversity, guaranteeing convergence towards the criteria.

(2) *Results for* 20-*dimensional functions*: The same set of experiments was carried out for benchmark functions in 20 dimensions. The results for 20-dimensional test functions are presented in Table 5. The complexity of the benchmark functions except the Griewank one increases as the dimensions increase. Increasing dimensions in the Griewank function causes the quality of the result of ScatterPSO to significantly increase. PSO_{rank} surpasses other algorithms for the Griewank function. The numerical results show that PSO_{rank} and CLPSO significantly outperform other algorithms for Rastrigrin functions. The remaining algorithms never converge to the criteria for the Griewank function. This implies that the optimized multimodal function has characteristics similar to the Rastrigrin function optimized by PSO_{rank}, independently of their dimensions. All the algorithms converge successfully to the criteria for the Sphere function; however the best results were obtained by the PSO_{rank} and IPSO algorithms. Like for the 10-dimensional functions, PSO_{rank}, FIPS and CLPSO outperform other algorithms for the Ackley function. All the algorithms except PSO_{rank} and ScatterPSO never converge to the criteria for the Rosenbrock function in 20 dimensions. The best result for the Rosenbrock function was obtained by ScatterPSO algorithm. As a consequence, the results in Tables 3 and 4 show that the multimodal functions with numerous local optima were successfully optimized by the PSO_{rank} algorithm.

(3) *Results for* 30-*dimensional functions*: Table 6 represents the experimental results of the eight algorithms for the 30-dimensional benchmark functions. Also, the evolution of the algorithms is presented in Fig. 7(a)–(f). This experiment was conducted for 30-dimensional functions except for Schaffer's f6 function, with two dimensions. The straight line represents the success criteria for each benchmark function. The algorithms with success rate 1 cross the straight line. The evolving of the algorithms represents its convergence behavior throughout the iterations. It implies that an algorithm may or may not provide better performance if the number of iterations is extended. Also it shows the convergence speed of the algorithms.

The unimodal Sphere function represents an easy problem to solve. Fig. 7(a) shows that all the algorithms except PSO-NTVE successfully solve this problem. It can be seen from Fig. 7(a) that each algorithm achieves better performance if the iterations are extended. However, the PSO_{rank} method converges rapidly to the criteria. The unimodal Rosenbrock function is hard to optimize. Its complexity increases as the dimensions increase. TPSO, IPSO, PSO-NTVE, CenterPSO, FIPS, and CLPSO fail to reach the criteria for the Rosenbrock function in 30 dimensions. ScatterPSO and PSO_{rank} outperform the other algorithms in optimizing the 30-dimensional Rosenbrock function. The best results are obtained by PSO_{rank}.

The Rastrigrin function is a highly multimodal with frequent local optima. An algorithm with poor balance between exploration and exploitation was simply trapped in local optima in early iterations. It is apparent from Fig. 7(c) that the all the algorithms except PSO_{rank} and CLPSO were rapidly trapped in local optima. Table 6 shows that PSO_{rank} significantly outperforms other algorithms. In spite of the shallow local optima in the Ackley function, compared to the Rastrigrin function, some of the algorithms were trapped in local optima in early iterations. It seems that the algorithms with flying patterns similar to TPSO, IPSO, ScatterPSO and PSO-NTVE have difficulties in solving applications with numerous local optima. All the aforementioned algorithms never converge to the criteria. It is clear from Fig. 7(c), (e) that PSO_{rank} and CLPSO successfully optimize the multimodal functions of these types.

Half of the algorithms have relatively similar evolving patterns in optimizing Griewank functions. These algorithms (i.e. PSO_{rank}, ScatterPSO, IPSO, CLPSO, and PSO-NTVE) constantly optimize the function throughout the iterations. So, we expect better performance to be obtained by extending the number of iterations. TPSO, FIPS, and CenterPSO stagnate in the last iterations. The stagnation occurs due to hard constriction on the movement trajectories of the particles. The stagnation problem can be mitigated by incorporation of the knowledge of all the particles in the swarm, as employed by PSO_{rank}, ScatterPSO, and CLPSO. The PSO_{rank} algorithm significantly outperforms all the other algorithms for the Griewank function.

Schaffer's f6 function has smooth slope near to global optima; hence the stagnation problem may occur in an optimization algorithm when the swarm reaches the region around the global optima. It is apparent from Fig. 7(f) that a stagnation problem occurs in all the algorithms except PSO_{rank}. The main difference in optimizing this function is the time at which the stagnation occurs. IPSO, PSO-NTVE, CenterPSO, and ScatterPSO stagnate in the early iterations, while TPSO, FIPS, and CLPSO stagnate in the intermediate or last iterations. PSO_{rank} significantly outperforms other algorithms in optimizing Schaffer's f6 function.

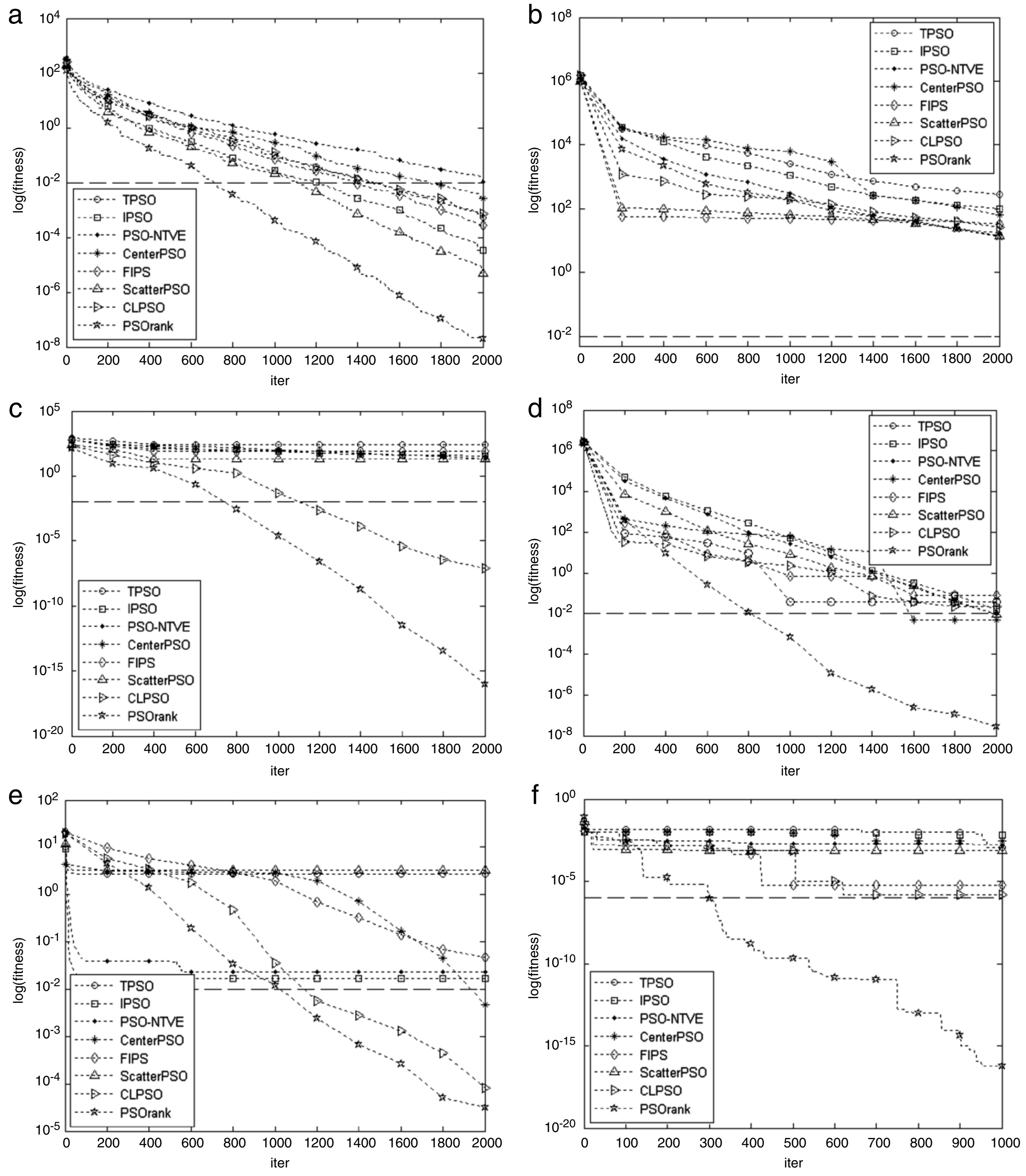


Fig. 7. Evolution of the average fitness for the algorithms; (a) Sphere, (b) Rosenbrock, (c) Rastrigrin, (d) Griewank, (e) Ackley, (f) Schaffer's f6.

5.5.2. Statistical analysis of trials on the basis of the success rate

When two algorithms have to be compared for a given set of problem instances, one can use the success for an algorithm to determine whether a solution has better quality than the solution produced by the other method for the same problem instance. Hence, the success ratio is an important measure in optimization problems. It determines the success probability of an algorithm. One would like to use the algorithm having the highest success probability.

The Taillard statistical [22] tests are conducted, for the success ratios of the PSO_{rank} algorithm and the success ratios of the other algorithms. This test is used to determine whether the PSO_{rank} algorithm is significantly more successful than the other algorithms. For this purpose, we used the tools downloadable from [23]. The significance level is fixed at $\alpha = 0.05$ for

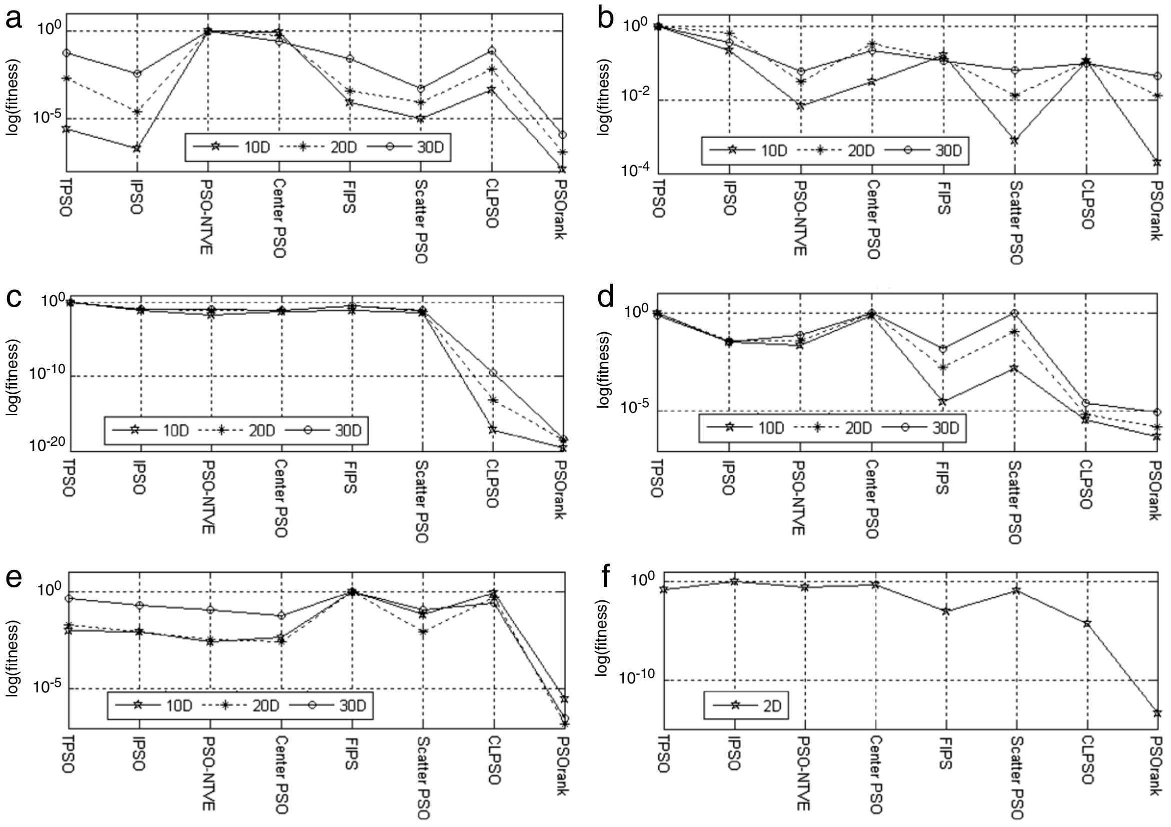


Fig. 8. Comparative performance of the algorithms: (a) Sphere, (b) Rosenbrock, (c) Rastrigrin, (d) Ackley, (e) Griewank, (f) Schaffer's f6 (the horizontal line represents the algorithm, and the vertical line represents merits).

a two-sided test. PSO$_{\text{rank}}$ statistically has better performance than another algorithm if the $\hat{T}$ value for their success ratios is smaller than the significance level α. The results of the Taillard tests are presented in Table 7. The values 1 show that the PSO$_{\text{rank}}$ algorithm has statistically better performance with 95% certainty than another algorithm for a test function of the predefined dimensions. In a comparison of two algorithms, each of which has success ratio 1, the Taillard test returns $\hat{T} = 0 < \alpha$. The values 2 show that two algorithms have success ratio 1 for a test function. The values 0 imply that the PSO$_{\text{rank}}$ algorithm does not have statistically better performance than another algorithm.

It is apparent from the results of the Taillard test that the PSO$_{\text{rank}}$ algorithm has better performance than all the other algorithms for the Griewank (10D, 20D, and 30D), the Rosenbrock (10D, 30D), and Schaffer's f6 (2D) function. The proposed approach also has significantly better performance than other algorithms except ScatterPSO for the Rosenbrock function in 20 dimensions. For the Sphere function, PSO$_{\text{rank}}$ has performance comparable to those of other algorithms. The PSO$_{\text{rank}}$ and CLPSO algorithms significantly outperform other algorithms for the Rastrigrin and Ackley functions.

5.5.3. Overall performance

In this section the overall performance of the proposed algorithm compared with other algorithms is presented. Three measures are used to compare the overall performances of the algorithm investigated in this consideration. As the first measure, relative performances of the eight optimization algorithms for six benchmark functions are considered. For this purpose we used a performance measure m (called the merit). A similar approach has been used in [19]. The merit measure represents the relative fitness of an algorithm for a benchmark function. The merit measure is defined as $m_i = \frac{\text{fit}(a_i, f_j)}{\text{max\_fit}(a, f_j)}$, where fit($a_i$) is the fitness of the solution which is found by algorithm i for the benchmark function f_j, and max_fit(a, f_j) represents the fitness of the worst algorithm (used as a reference algorithm) for the benchmark function f_j. As each algorithm minimizes the benchmark function, a smaller value of the merit represents a better performance. The relative merits of the algorithm tested for benchmark functions in 2, 10, 20, and 30 dimensions are represented in Fig. 8. The algorithm with worst performance has merit 1. As the performance improves the merit decreases. Considering the distance between two merits, we can determine the degree of improvement obtained by an algorithm i in optimizing a function f_j. The results show the significant improvements obtained by the PSO$_{\text{rank}}$ algorithm for the Rastrigrin, Schaffer's f6, the Griewank, and the Ackley functions.

A second measure is used to compare the average performances of the algorithms for all the test results. We used the parameter $\text{avg}(i, D) = \frac{\sum_{j=1}^{N} \text{fit}(a_i, f_j)}{N}$, where avg($i, D$) is the average fitness of the algorithm i for all the functions f_j with D

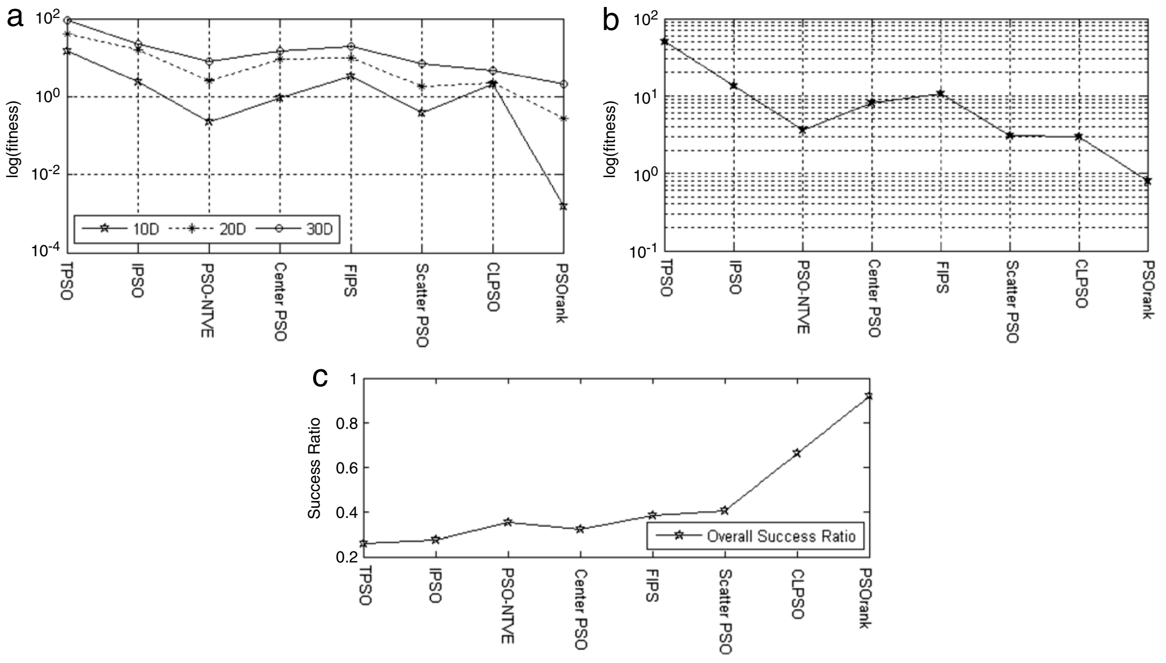


Fig. 9. Overall performance of the algorithms: (a) average performance for *D*-dimensional functions, (b) average performance obtained from all the tests conducted, (c) average success ratio from all the tests conducted.

dimensions, and *N* is the number of benchmark functions optimized in *D* dimensions. Fig. 9(a) represents the avg parameter for eight algorithms tested on the six functions in 2, 10, 20, and 30 dimensions. It is apparent from Fig. 9(a) that the PSO_{rank} algorithm outperforms all the other algorithms. The overall performance of the algorithms in all experiments conducted is shown in Fig. 9(b). The overall performance is considered as the average fitness of an algorithm in optimizing all the functions in 2, 10, 20, and 30 dimensions. This parameter is defined as $\text{overall}(i) = \frac{\sum_{k=1}^{4} \text{avg}(i, D_k)}{4}$, where $D_k \in \{2, 10, 20, 30\}$. From the results, we can see that the PSO_{rank} algorithm has the best overall performance. Also, CLPSO and ScatterPSO have good performance compared to the remaining algorithms.

A third measure, Overall_SR, determines the overall success rate of each of the algorithms tested for six test functions. Fig. 9(c) shows the Overall_SR measure. This measure is computed as the average of all the success rates obtained by each of the algorithms. The result shows that significant improvement is provided by the PSO_{rank} algorithm for the success ratio measure. The performance value of the PSO_{rank} algorithm is greater than 0.92 for the six test functions.

Conclusion: The average fitness and success ratio are the main performance measures. By analyzing the overall performance measures presented in Figs. 8 and 9, we can see that the PSO_{rank} algorithm surpasses all the other algorithms for the average fitness. Considerable improvement in success rate was obtained by the PSO_{rank} algorithm. The best results for the PSO_{rank} algorithm were obtained for the multimodal functions with numerous local optima. This implies that the PSO_{rank} algorithm solves the multimodal functions effectively in a reasonable time with high success rate. Considering social knowledge provided by all the particles along with the non-linear inertia weight results in an algorithm which successfully optimizes well known test functions.

5.6. *Neural network training*

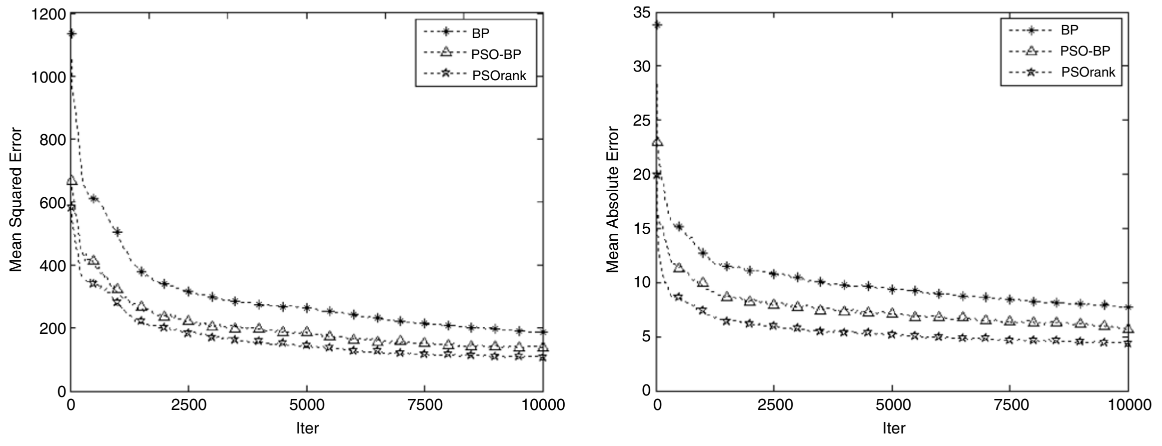
After validating the proposed algorithm using continuous test functions, it is also employed to optimize a real-world application. Specifying the weights of a neural network is an optimization problem with the goal of finding a set of weights that minimizes the mean square error of a network. We used PSO_{rank} for training a feed forward neural network. Standard images such as Lena, Pepper, etc. are used as training and test data sets. The problem is defined as follows.

A feed forward neural network is composed of three tiers called the input, hidden and output layers. The input and output layers both have n neurons, and the hidden layer has m neurons ($m < n$). We used images as training and test sets. The input image was partitioned to $n \times n$ non-overlapping blocks. Each block was used as an input for the neural network. The objective is to compress the image such that the error of compression is minimized. The compression rate for the following experiments was computed as $\frac{n-m}{m} \times 100$.

In this experiment each particle is encoded for a weight vector. For feed forward network training, each particle represents all weights of the network. As the algorithm proceeds, the new position of a candidate particle is updated on the basis of the current position and new velocity. The new position is a set of new weights for neural networks. After a predefined iteration, the algorithm converges to the local optima and the training phase is completed.

Table 7
Results of the Taillard test: the values 1 show that the PSO$_{rank}$ algorithm has statistically better performance than another algorithm, whereas the values 0 imply that PSO$_{rank}$ does not have statistically better than another algorithm.

Function	Dim	TPSO	IPSO	PSO-NTVE	CenterPSO	FIPS	Scatter PSO	CLPSO
f_{Sph}	10	2	2	2	2	2	2	2
	20	2	2	2	2	2	2	2
	30	2	2	1	2	2	2	2
f_{Ros}	10	1	1	1	1	1	1	1
	20	1	1	1	1	1	0	1
	30	1	1	1	1	1	1	1
f_{Ras}	10	1	1	1	1	1	1	2
	20	1	1	1	1	1	1	2
	30	1	1	1	1	1	1	2
f_{Ack}	10	1	1	1	1	2	1	2
	20	1	1	1	1	1	1	2
	30	1	1	1	1	1	1	2
f_{Gri}	10	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1
f_{Sch}	2	1	1	1	1	1	1	1



(a) Mean squared error.

(b) Mean absolute error.

Fig. 10. (a) Evaluation of MSE. (b) Evaluation of MAE.

Usually, in neural networks, the training objectives are the mean squared error (MSE) and mean absolute error (MAE), over a training set. So, the proposed algorithm should minimize the error by adjusting the weights of the neural network. The MSE and MAE can be calculated using the following formulations:

$$\mathrm{MSE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(u_{ij} - u_{ij}^{*})^2 \tag{14}$$

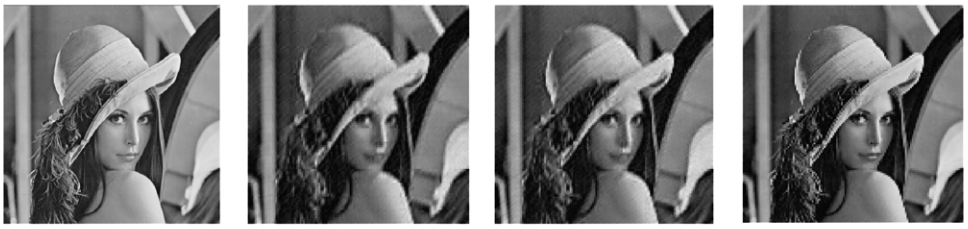
$$\mathrm{MAE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}|u_{ij} - u_{ij}^{*}| \tag{15}$$

where u_{ij} is the intensity of color of pixel (i, j) in the original image and u_{ij}^{*} is the intensity of color of pixel (i, j) in the reconstructed image, N is the height of the image and M is the width of the image.

PSO$_{rank}$ and two other algorithms were compared as neural network training algorithms. For the comparison we use the traditional back propagation (BP) [24] algorithm and a hybrid approach called PSO-BP [25], which combines BP and PSO algorithms for training neural networks. Various experiments have been done. In the first experiment the training image and the test image are identical. A part of an image is randomly selected to train the network and then the whole image is used as a test set. The training and test sets for the second experiment are different. In both of them the network is trained using back propagation, PSO-BP, and PSO$_{rank}$. Table 8 shows the results for both objectives, MSE and MAE. Comparison results show that the PSO$_{rank}$ training shows a better performance than BP and BP-PSO. Therefore PSO$_{rank}$ can preserve the image details very well. Fig. 10 shows the MSE and MAE values for BP, PSO-BP and PSO$_{rank}$ after 100 runs.

Table 8
The comparison of the performances of BP, BP-PSO and PSO_{rank}.

Compression rate	Experiment #1						Experiment #2					
	BP		PSO-BP		PSO_{rank}		BP		PSO-BP		PSO_{rank}	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
1/4	34.96	3.16	19.74	3.21	17.31	2.36	41.31	3.85	22.96	2.68	21.43	2.64
1/8	76.37	5.67	54.37	4.17	48.93	3.81	88.54	5.97	61.58	4.31	56.39	4.18
1/16	186.84	7.92	133.4	6.73	103.42	4.56	216.31	8.59	146.41	7.34	112.67	6.61


(a) Original image. (b) BP. (c) BP-PSO. (d) PSO_{rank}.

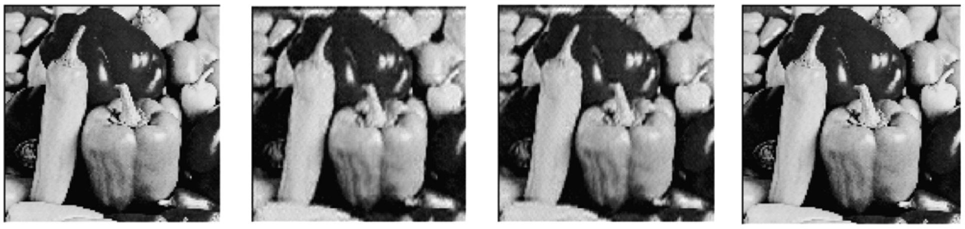

(a) Original image. (b) BP. (c) BP-PSO. (d) PSO_{rank}.

Fig. 11. Results for the images restored by the algorithms (row #1: Lena, row #2: Pepper).

The image compression results for the neural network trained by BP, PSO-BP, and PSO_{rank} are shown in Fig. 11. The figure represents the restored images (e.g. Lena, Pepper) for compression ratio 16:1 of Experiment #2. It can be seen that the performance of the neural network for image compression is improved by using the proposed PSO algorithm. The PSO_{rank} learning algorithm can avoid local optima in compression images and provide better quality for the restored images even for large compression rates. The images restored by means of running the PSO_{rank} algorithm already have similar quality. This shows the ability of the PSO_{rank} learning algorithm, which could be adapted successfully for the compression of other images.

6. Discussion

From the results, we can see that PSO_{rank} provides high searching ability. As described in Section 5.4, the effectiveness of the PSO_{rank} algorithm depends on three different mechanisms: the first mechanism based on Latané theory for updating the velocity and position, the second mechanism (i.e. that of dependent random coefficients) for maintaining the importance of social and cognitive knowledge, and the third mechanism (i.e. that of non-linear inertia weight) for controlling the importance of the previous velocity. However, the first mechanism has the main role in finding good solutions. The next two mechanisms provide further improvements for the algorithm. The effects of these mechanisms are discussed in the following subsections.

6.1. The effect of Latané theory

The searching ability indicates how an algorithm can overcome deficiencies which arise in a searching process. Usually, stagnation and premature convergence are identified as the major weaknesses that a PSO algorithm may have. A PSO algorithm tries to cope with these problems by introducing different approaches such as inertia weight, dynamic acceleration coefficients, time-varying coefficients, etc. These problems can be mitigated by incorporating the knowledge of all the particles in the swarm, as employed by PSO_{rank}. Here, the Latané theory comes into the play. The Latané theory aims to cope with hard constriction on the trajectories of the particles, and provide an extended flying pattern for the particle in which an appropriate level of diversity is maintained. This means that the stagnation problem can be alleviated. Also, diversity provides a high exploration level in the first iterations of the algorithm. Hence, Latané theory empowers the algorithm to avoid premature convergence by exploring large areas of the search space in the first iterations. The effect of just Latané theory in PSO algorithm (i.e. $\text{PSO}_{\text{rank}} - \text{RC} - \text{IW}$) can be seen in Table 3. The results show the effectiveness of Latané theory for the avoidance of stagnation and premature convergence for all the test functions. It can be seen from

Fig. 7 that PSO algorithms with many sources of influence on their particles such as PSO_{rank}, CLPSO and ScatterPSO have more success in alleviating the aforementioned problems compared to the other methods such as PSO-NTVE, IPSO, and CenterPSO methods.

6.2. The effect of dependent random coefficients

As can be seen from Table 3, Latané theory provides more efficiency than non-linear inertia weight and dependent random variables, and acts as the main mechanism in PSO_{rank}. However, more enhancements can be obtained by efficient use of cognitive and social knowledge. To achieve this goal we need to define an efficient way of using the coefficients of social and cognitive knowledge. In previous variants of PSO with independent coefficients, these areas of knowledge may be overused or not used fully. However, dependent random coefficients suggest a way for avoiding inefficient use of social and cognitive knowledge. Assume that random parameters rand_1 and rand_2 have large values, in an independent way; two coefficients will have large values and both social and cognitive knowledge are overused. However, in the dependent approach, the first (second) coefficient is multiplied by 1-rand_2 (1-rand_1), and the large values of the coefficients will be decreased. The result is that the social and cognitive knowledge will be used in a better way. If one parameter has a large value while the other one has a small value, both of them are decreased but the larger one is decreased more. Hence, the algorithm can use the social and cognitive knowledge in a more efficient way. From Table 3, we can see that dependent random coefficients provide efficiency for the standard PSO algorithm. The PSO variants with dependent random coefficients have better performances than the standard PSO algorithm for most of the test functions.

6.3. The effect of non-linear inertia weight

In an optimization algorithm, we need to balance between exploration and exploitation throughout the iterations. In particular, we prefer exploration in the first iterations and exploitation in the last iterations. Inertia was first introduced for this purpose as a fixed value. After that, its linear variant was proposed. The large value of the inertia weight in the first iterations encourages a particle to move along its previous direction, while its small value in the last iterations enforces the particle moving towards the global knowledge. This may result in inefficiency in optimizing highly multimodal functions. In such cases we need more flexibility. For example, we need an appropriate level of exploration even in the last iterations when optimizing benchmarks such as Rastrigrin or Griewank ones with a large number of optima around the optimum position. As can be seen from Fig. 2, using non-linear inertia weight provides this type of flexibility. By decreasing the m parameter, one can extend the preference of exploration throughout iterations. This approach encourages the particles to wander in a larger area around the global best position and try to find a better position. From Table 3, it is apparent that non-linear inertia weight has a positive effect on optimizing multimodal functions. The effect of non-linear inertia weight on the proposed algorithm can be seen in Table 3. PSO variants with non-linear inertia weight successfully optimize multimodal functions with highly local optima. From the results, we see that these variants have better performance for multimodal functions such as Rastrigrin and Griewank.

7. Conclusions

In the standard PSO, only the best particle has an impact on the next position of a candidate particle, and the others have no influence. Sharing information among all particles will improve the performance of a particle swarm optimizer. On the basis of these facts, and taking inspiration from Latané theory, a rank based particle swarm optimization algorithm, called PSO_{rank}, is proposed in this paper. In PSO_{rank}, a set of best particles contribute to adjusting the velocity of the each particle. The contribution of each particle is proportional to its strength. The particle's strength value is a function of the strivness, immediacy and number of the contributing particles. The number of contributing particles dynamically reduces as the algorithm proceeds. A new time-varying inertia weight is introduced which decreases the non-linearly as the algorithm proceeds. This new method can enhance the balance between exploration and exploitation. Six popular test functions are used to evaluate the performance of PSO_{rank} and other algorithms presented in the literature. A large number of experiments were performed for optimizing numerical functions as well as training neural networks. Experimental results show that PSO_{rank} achieves better performance than other optimization methods investigated in this paper.

References

[1] J. Kennedy, R. Eberhart, PSO optimization, in: Proceeding of IEEE International Conference Neural Networks, vol. IV, 1995, pp. 1941–1948.
[2] A. Ratnaweera, S.K. Halgamuge, H.C. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 240–255.
[3] C. Ko, Y. Chang, C. Wu, An orthogonal-array-based particle swarm optimizer with nonlinear time-varying evolution, Journal of Applied Mathematics and Computation 191 (2007) 272–279.
[4] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Transactions on Evolutionary Computation 10 (3) (2006) 281–295.
[5] T. Xiang, K. Wong, X. Liao, A novel particle swarm optimizer with time-delay, Journal of Applied Mathematics and Computation 186 (2007) 789–793.
[6] X. Yang, J. Yuan, J. Yuan, H. Mao, A modified particle swarm optimizer with dynamic adaptation, Journal of Applied Mathematics and Computation 189 (2007) 1205–1213.

[7] Y. Jiang, T. Hu, C. Huang, X. Wu, An improved particle swarm optimization algorithm, Applied Mathematics and Computation 193 (2007) 231–239.
[8] M. Clerc, J. Kennedy, The particle swarm, explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.
[9] Y. Liu, Z. Qin, Z. Shi, J. Lu, Center particle swarm optimization, Neurocomputing 70 (4) (2007) 672–679.
[10] J.F. Schutte, A.A. Groenwold, A study of global optimization using particle swarms, Journal of Global Optimization 31 (2005) 93–108.
[11] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, in: V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben (Eds.), in: Evolutionary Programming, vol. VII, Springer-Verlag, Berlin, Germany, 1998, pp. 591–600.
[12] K.E. Parsopoulos, M.N. Vrahatis, On the computation of all global minimizers through particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 211–224.
[13] Y. Shi, R.C. Eberhart, Particle swarm optimization with fuzzy adaptive inertia weight, in: Proceeding of Workshop on Particle Swarm Optimization, Indianapolis, 2001, pp. 101–106.
[14] W.B. Langdon, R. Poli, Evolving problems to learn about particle swarm optimizers and other search algorithms, IEEE Transactions on Evolutionary Computation 11 (5) (2007) 561–578.
[15] S.T. Hsieh, T.Y. Sun, C.L. Lin, C.C. Liu, Effective learning rate adjustment of blind source separation based on an improved particle swarm optimizer, IEEE Transactions on Evolutionary Computation 12 (2) (2008) 242–251.
[16] J. Kennedy, R. Mendes, Neighborhood topologies in fully informed and best-of-neighborhood particle swarms, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 36 (4) (2006) 515–519.
[17] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer and its adaptive variant, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 35 (6) (2005) 1272–1282.
[18] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler, may be better, IEEE Transactions on Evolutionary Computation 8 (2004) 204–210.
[19] P.Y. Yin, F. Glover, M. Laguna, J.X. Zhu, Scatter PSO—a more effective form of particle swarm optimization, in: Proceedings of IEEE Congress on Evolutionary Computation, 2007, pp. 2289–2296.
[20] B. Latané, The psychology of social impact, American Psychologist 36 (1981) 343–356.
[21] R.C. Eberhart, Y. Shi, J. Kennedy, Swarm Intelligence, Morgan Kaufmann, 1997.
[22] E.D. Taillard, Few guidelines for analyzing methods, in: Proceeding of the Sixth Metaheuristics International Conference, 2005.
[23] http://ina2.eivd.ch/collaborateurs/etd/codes.dir/comparaison.dir/comparaison.htm.
[24] L.Y. Wang, E. Oja, Image compression by neural networks: a comparison study, Proceeding of IEEE Winter Workshop on Nonlinear Digital Signal Processing 7 (2) (1993) 31–36.
[25] J.R. Zhang, J. Zhang, T.M. Lok, M.R. Lyu, A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training, Applied Mathematics and Computation 185 (2007) 1026–1037.