Improved particle swarm algorithms for global optimization

M.M. Ali [a,*], P. Kaelo [b]

[a] School of Computational and Applied Mathematics, Witwatersrand University, 1 Jan Smut Avenue, Wits 2050, Johannesburg, Gauteng, South Africa
[b] Department of Mathematics, University of Botswana, Private Bag UB 00704, Gaborone, Botswana

Abstract

Particle swarm optimization algorithm has recently gained much attention in the global optimization research community. As a result, a few variants of the algorithm have been suggested. In this paper, we study the efficiency and robustness of a number of particle swarm optimization algorithms and identify the cause for their slow convergence. We then propose some modifications in the position update rule of particle swarm optimization algorithm in order to make the convergence faster. These modifications result in two new versions of the particle swarm optimization algorithm. A numerical study is carried out using a set of 54 test problems some of which are inspired by practical applications. Results show that the new algorithms are much more robust and efficient than some existing particle swarm optimization algorithms. A comparison of the new algorithms with the differential evolution algorithm is also made.

Keywords: Particle swarm; Global optimization; Population set; Differential evolution

1. Introduction

In this paper, we consider the problem of finding the global minimum of the bound constrained optimization problem (without loss of generality we consider only minimization problems)

$$\min_{x\in\Omega} f(x), \tag{1}$$

where $f: \Omega \subset R^n \to R$ is a continuous real-valued function and the search region Ω is a multidimensional interval specified by lower and upper bounds $x^l, x^u \in R^n$, respectively. A point x_{opt} is said to be a global minimizer of f if

$$f_{\text{opt}} = f(x_{\text{opt}}) \leqslant f(x), \quad \forall x \in \Omega. \tag{2}$$

Global optimization problems are frequently encountered in many practical applications such as physics, engineering design, molecular biology and other scientific applications. In global optimization, it is often

* Corresponding author.
E-mail address: mali@cam.wits.ac.za (M.M. Ali).

required to locate the global optimum among many local optima. In many cases, global optimization problems involve nonlinear functions of many variables, often with attributes, such as discontinuity and non-differentiability, that are difficult to solve using traditional steepest descent methods. Consequently, stochastic search methods which do not require any properties of the objective function have been developed. They include, amongst others, genetic algorithms (GA) [1,2], differential evolution (DE) [3,4], particle swarm optimization (PSO) [5,6] and ant colony optimization (ACO) [7,8]. These methods evaluate the objective function in a random sample of points from the search space and subsequently manipulate the sample. Thus, they are applicable to a wide range of problems. These methods are often referred to as population set-based global optimization methods. This paper deals with solving (1) using PSO.

The PSO method was suggested by Kennedy and Eberhart [5]. It was developed based on the observations of the social behaviour of animals, such as bird flocking and fish schooling, and the swarm theory. Like other population set-based methods [1,4], PSO uses a population of candidate solutions, called particles, with their positions initialized randomly from the search space Ω. It progresses in an epoch or iteration base. At each iteration, each particle is assigned with a velocity, also initialized randomly in Ω, according to its own experiences and those of its companions, i.e., other members of the population set. The velocity of each particle is updated using the best position it visited so far and the overall best position visited by its companions. Then the position of each particle is updated using its updated velocity per iteration. Thus, as compared to other population set-based methods, e.g., genetic algorithm or differential evolution, PSO has memory. Previously visited best positions in PSO are remembered, while in GA and DE, these are forgotten once the current population changes.

PSO has gained popularity lately and has been applied to wide applications in different fields [9–13]. However, it has a number of drawbacks, one of which is the presence of problem dependent parameters. A number of variations to the original PSO have been proposed to make PSO faster and reliable. Some of these variants [14–17] have been based on the choice of parameter selection while the other have been based on hybridization [18–21]. However, all of these variations have been justified using small sets of low dimensional problems. In this paper, we investigate the efficiency and reliability of some variants of PSO that were proved to be superior in an earlier study [17]. In our study, we first identify the slow convergence rate of PSO, and demonstrate the reasons for the slow convergence, specially at the vicinity of the global minimizer. We then propose some remedies in order to improve its convergence rate. Specifically, we suggest some modifications to the velocity update of PSO based on the knowledge of previous swarm success. In addition, we hybridize PSO by embedding the point generation scheme of DE in PSO.

We test the performance of the modified PSO algorithms with their original version as well as with a potential competitor, namely the DE algorithm, using 54 test problems with dimension ranging from 2 to 30. In our view such a comparison is necessary as, to the best of our knowledge, this has not been thoroughly done before.

The remainder of this paper is organized as follows. Section 2 describes the original PSO model and some of its variants. Section 3 presents motivations for our modifications. Then our proposed modifications to PSO are described in Section 4. Numerical results and discussions are presented in Section 5. Finally, Section 6 provides some concluding remarks based on the results in Section 5.

2. Brief introduction of particle swarm optimization

Particle swarm optimization maintains a group of particles. At each iteration k, the ith particle is represented by a vector x_i^k in multidimensional space to characterize its position. The vector v_i^k is used to characterize its velocity. Thus, PSO maintains a set of positions:

$$S = \{x_1^k, x_2^k, \ldots, v_N^k\}$$

and a set of corresponding velocities

$$V = \{v_1^k, v_2^k, \ldots, v_N^k\}.$$

Initially, the iteration counter $k = 0$, and the positions x_i^0 and their corresponding velocities v_i^0 $(i = 1,2,\ldots,N)$, are generated randomly from the search space Ω. Each particle changes its position x_i^k,

per iteration. The new position x_i^{k+1} of the ith particle $(i = 1, 2, \ldots, N)$ is biased towards its best position p_i^k with best function value, referred to as personal best or *pbest*, found by the particle so far, and the very best position p_g^k, referred to as the global best or *gbest*, found by its companions. The *gbest* is the best position in the set

$$P = \{p_1^k, p_2^k, \ldots, p_N^k\},$$

where $p_i^0 = x_i^0, \ \forall i$.

We regard a particle in S as good or bad depending on its personal best being a good or bad point in P. Consequently, we call the ith particle (jth particle) in S the worst (the best) if p_i^k (p_j^k) is the least (best) fitted, with respect to function value, in P. We denote the *pbest* of the worst particle and the best particle in S as p_h^k and p_g^k, respectively. Hence $p_g^k = \arg\min_{i\in\{1,2,\ldots,N\}} f(p_i^k)$ and $p_h^k = \arg\max_{i\in\{1,2,\ldots,N\}} f(p_i^k)$.

At each iteration k, the position x_i^k of the ith particle is updated by a velocity v_i^{k+1} which depends on three components: its current velocity v_i^k and the weighted difference vectors $(p_i^k - x_i^k)$ and $(p_g^k - x_i^k)$. Specifically, the set S is updated for the next iteration using

$$x_i^{k+1} = x_i^k + v_i^{k+1}, \tag{3}$$

where

$$v_i^{k+1} = v_i^k + r_1 c_1 (p_i^k - x_i^k) + r_2 c_2 (p_g^k - x_i^k). \tag{4}$$

The parameters r_1 and r_2 are uniformly distributed random numbers in $[0, 1]$ and c_1 and c_2, known as the cognitive and social parameters respectively, are popularly chosen to be $c_1 = c_2 = 2$ [5]. Thus the values $r_1 c_1$ and $r_2 c_2$ introduce some stochastic weighting in the difference vectors $(p_i^k - x_i^k)$ and $(p_g^k - x_i^k)$, respectively. The set P is updated, as the new positions x_i^{k+1} are created, using the following rule:

$$p_i^{k+1} = \begin{cases} x_i^{k+1} & \text{if } f(x_i^{k+1}) < f(p_i^k) \\ p_i^k & \text{otherwise.} \end{cases} \tag{5}$$

This process of updating the velocities v_i^k, positions x_i^k, *pbest* p_i^k and the *gbest* p_g^k is repeated until a user-defined stopping condition is met. This standard version of PSO is referred to as PSO-S. Below we give a pseudo-code of PSO-S by Kennedy and Eberhart [5].

Pseudo-code of the PSO-S algorithm

Step 1 **Initialization**.
- Step 1a Initialize iteration counter $k = 0$
- Step 1b Initialize N random positions of the particles $(x_i^k, \ i = 1, 2, \ldots, N)$ and store them in S.
- Step 1c Initialize N random velocities $(v_i^k, \ i = 1, 2, \cdots, N)$ and store them in V.
- Step 1d Initialize N *pbest* $(p_i^k, \ i = 1, 2, \ldots, N)$ and store them in P.
- Step 1e Set p_g^k equal the best *pbest* in P.

Step 2 **While not stopping criterion do**

Step 2a For each ith particle:
- **Update** V: Calculate v_i^{k+1} using (4)
- **Update** S: Calculate position x_i^{k+1} using (3)
- **Update** P: Update P using (5).

Step 2b **Update** *gbest* p_g^k: $p_g^{k+1} = \arg\min_{i\in\{1,2,\ldots,N\}} f(p_i^{k+1})$.

Step 2c $k = k + 1$.

We now briefly present a number of modifications to PSO-S that have been introduced in the literature. The first modification was done by Shi and Eberhart [15]. They proposed a constant inertia weight ω which controls how much a particle tends to follow its current direction as compared to the memorized *pbest* p_i^k and the

gbest p_g^k. This version is referred to as PSO with constant inertia or PSO-CI. Hence, in PSO-CI, the velocity update is given as

$$v_i^{k+1} = \omega v_i^k + r_1 c_1 \left(p_i^k - x_i^k\right) + r_2 c_2 \left(p_g^k - x_i^k\right), \tag{6}$$

where the values of r_1 and r_2 are realized component-wise.

Shi and Eberhart [16] also proposed a linearly varying inertia weight during the search. The inertia weight is linearly reduced during the search. This entails a more globally search during the initial stages and a more locally search during the final stages. This version is referred to as PSO with linear inertia or PSO-LI. They also proposed a limitation of each particle's velocity to a specified maximum velocity $v^{\max}$. The maximum velocity was calculated as a fraction γ $(0 < \gamma \leqslant 1)$ of the distance between the bounds of the search space, i.e.,

$$v^{\max} = \gamma(x^u - x^l). \tag{7}$$

The versions PSO-CI and PSO-LI equipped with (7) are referred to as PSO with constant inertia and maximum velocity limitation or PSO-CIV and PSO with linear inertia and velocity limitation or PSO-LIV, respectively.

Fourie and Groenwold [13] suggested a dynamic inertia weight and maximum velocity reduction. In this modification, an initial inertia weight ω_0 and maximum velocity $v^{\max}$, calculated as in (7), are prescribed. The inertia weight and maximum velocity are then reduced by fractions α and β respectively, if no improvement in p_g^k occur after a specified number of iterations h, i.e.,

$$\text{if } f(p_g^k) = f(p_g^{k-h}) \text{ then } \omega_{k+1} = \alpha\omega_k \text{ and } v_k^{\max} = \beta v_k^{\max}, \tag{8}$$

where α and β are such that $0 < \alpha,\beta < 1$. This version is referred to as PSO with dynamic inertia and maximum velocity limitation or PSO-DIV.

Clerc and Kennedy [14] introduced another interesting modification to PSO-S in the form of a constriction coefficient K which controls all the three components in velocity update rule (4). This has an effect of reducing the velocity as the search progresses. In this modification, the velocity update is given as

$$v_i^{k+1} = K\left(v_i^k + r_1 c_1 \left(p_i^k - x_i^k\right) + r_2 c_2 (p_g^k - x_i^k)\right), \tag{9}$$

where

$$K = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}, \quad \phi = c_1 + c_2 > 4. \tag{10}$$

This version is referred to as PSO with constriction or PSO-C. All the above modifications are very simplistic in a sense that they only (gradually) reduce the parameter value(s) without affecting the structure of PSO.

Other researchers have hybridized PSO with other algorithms. For example, Shi et al. [21] hybridized PSO with a variable population-size GA. In this hybrid both GA and PSO are executed separately and then the members from their respective population set are mixed according to their fitness, periodically. The algorithm, however, is tested on six simple and easy test problems.

Liu et al. [18] combined PSO with a chaotic local search (CLS). CLS is a variant of the discrete version of the logistic equation [22] where the chaotic variables are calculated from the members of S, p_g^k and p_h^k. This hybrid is also tested on a set of seven test problems.

Another hybrid is due to Hendtlass [20], where PSO is combined with DE to form a combined swarm differential evolution algorithm (SDEA). In this condensed and short paper, the author updates the particle positions using both PSO scheme and DE scheme, alternatively in a controlled manner. SDEA also modifies the personal best, p_i^k, by some other members of the population. All the swarm members, therefore, influence all other individuals but the magnitude of the influence decreases with both fitness and distance. The algorithm is tested on four test problems only.

Da and Xiurun [19] also modified PSO by introducing a temperature-like control parameter as in the simulated annealing (SA) algorithm. This version of PSO probabilistically selects the overall best, p_g^k, in the updating of v_i^{k+1} for each i. Unlike SA, the temperature here increases from a low to a high value. This ascertains

that PSO randomizes p_g^k in P for the low temperature. At the high temperature, p_g^k is selected deterministically as in the original PSO. Results of this hybrid are reported only for an application from rock engineering.

A recent numerical study, conducted by Schutte and Groenwold [17] using seven variants of PSO, found that three variants were superior to the others. These variants are PSO-CIV, PSO-DIV and PSO-C. The numerical study was carried out using the extended Dixon–Szegö [23] test set. Although only 12 problems were used, this study with PSO has been more thorough than other studies with PSO.

Typically, when an algorithm is developed, it is tested against a set of problems. The set of test problems is often selected haphazardly and differs from that used for the other algorithm. The differences in test problem sets may induce bias towards particular algorithms when comparing their performances. It is therefore important to use a larger and representative set of test problems when comparing different algorithms.

In this paper we propose modifications to PSO-CIV. We then thoroughly assess the strengths of PSO-CIV, PSO-DIV and PSO-C, and the modified PSO-CIV using a much larger test set. We show that the new PSO methods suggested are better than the three variants mentioned above. We present below the reasons for our modifications.

3. Motivation for the proposed modifications

Like the previous variants of PSO, our modifications are inspired by the drawbacks of PSO with respect to reliability and efficiency. When a variant of an algorithm is suggested, often drawbacks are identified with the results obtained. Modifications are then suggested in an attempt to remedy the drawbacks, i.e., to improve the results. The difference made in this paper is that we not only identify the drawbacks with respect to the results obtained, we also investigate the cause of such drawbacks.

Most of the variants of PSO are based on the modification of the velocity update (6). In particular, all the non-hybrid PSO variants reported in this paper have been derived from the original PSO, PSO-S, by introducing the maximum velocity restriction and/or changing the parameter values. These changes were made simply because they produced better results without reporting any (analytical) reasons for such results, except often the intuitive ones. The modifications suggested in this paper are also based on parameter values, but other changes are also incorporated in PSO-CIV that add complementary strengths to the parameter choices.

We have shown posteriori the effects of changing the problem dependent parameters c_1, c_2 and ω in the numerical section, where we have also shown how judicious choices of these parameters can improve PSO-CIV significantly. We have also reported in the numerical section that $c_1 = c_2 = 2$ are good choices as any reduction of these values is detrimental to locating the global minimum. With these parameter values, we investigate, here, the drawbacks of PSO-CIV and suggests modifications in the random weights of the cognitive and social velocity terms of (6). We also suggest a modification in the choice of ω. In particular, we link the choice of ω with the success of particle i at the kth iteration. We present our arguments on why such modifications together with the other changes in the PSO-CIV algorithm need to be made. Modifications will be justified further in the numerical section. The objective in this section, therefore, is to identify the drawbacks and their possible reasons, and then to suggest possible remedies.

Drawbacks are often associated with the low success rate in obtaining the global minimum (reliability) and the high number of function calls (efficiency). Recent studies [1], however, have shown that the population-based direct search methods achieve high success rates only at the cost of high function calls. Hence any remedy of such drawbacks must hold a balance in the bi-objective criteria. A reasonable balance that is often sought for is to diversify the search at the early stages of an algorithm for greater success, and to intensify the search at the later stages for faster convergence-requiring low function calls.

We begin our study with the search intensification capability of PSO-CIV. We first present numerical motivations and then provide some mathematical reasoning. For this, we run PSO-CIV on a simple function such as

$$f(x, y) = x^2 + y^2 \tag{11}$$

and count the total number of particle positions, x_i^k, generated and how often these positions are able to improve their corresponding *pbest*, p_i^k, during the course of a run, i.e., for all i and k. We study this within the vicinity of the minimizer. In particular, we use $\Omega_1 = [-0.06, 0.06]^2$ and $\Omega_2 = [-0.12, 0.12]^2$ and stop the

PSO-CIV algorithm as soon as $p_g^k \leqslant 10^{-5}$. Average results over 100 runs show that the *pbest* replacement rates are about 22% and 21% on Ω_1 and Ω_2 respectively, irrespective of the size of the population, $N = 10, 20$. Results from 200 individual run on Ω_1 and Ω_2 also show that the maximum replacement rate is only 26%. These results clearly prove that the search intensification capability of PSO-CIV is very low. We, therefore, look into the reasons for such low replacement rates of PSO-CIV on the same function. An advantage of using function (11) is that one can easily visualize the function value without evaluating the function at a point-it is the square of the distance from the origin to the point. If we look for reasons without affecting the operations used by PSO-CIV then we are forced to look at the effects of the parameters on drawbacks.

We run again PSO-CIV in Ω_1 and observe only the particle positions, x_i^k, that take 10 or more iterations to replace their personal best, p_i^k. An average result over 5 runs, based on iterations starting at 10, shows that about 18% particles take more than 10 iterations to replace their *pbest*, p_i^k, i.e., $f(p_i^k) = f(p_i^{k+h_i})$, $h_i > 10$. This is certainly a computational burden at the later stages of PSO-CIV. We carefully looked at the individual components, $V_1 = \omega v_i^k$, $V_2 = 2r_1(p_i^k - x_i^k)$ and $V_3 = 2r_2(p_g^k - x_i^k)$ of the velocity update

$$v_i^{k+1} = \omega v_i^k + 2r_1(p_i^k - x_i^k) + 2r_2\left(p_g^k - x_i^k\right) = 0.6v_i^k + C_1(p_i^k - x_i^k) + C_2\left(p_g^k - x_i^k\right), \tag{12}$$

for these particle positions. When $V_2 = 0$, i.e., immediate after the replacement of the *pbest* p_i^k, the calculation of v_i^{k+1} incorporates the velocity ωv_i^k which is derived from a completely different set of vectors, $x_i^{k-1} \neq x_i^k = p_i^k \neq p_i^{k-1}$. We observed that the position and the length of ωv_i^k coupled with the positions of p_i^k and p_g^k cause the slow replacement of *pbest*. We demonstrate this with the following example for the function (11). We assume that $x_i^k = p_i^k$ and that $f(x_i^{k+1}) > f(p_i^{k+1}) = f(p_i^k)$, i.e., $h_i = 1$. This is quite possible since the position x_i^{k+1} has been calculated using v_i^{k+1} which uses ωv_i^k. We now calculate the next position as follows:

$$x_i^{k+2} = x_i^{k+1} + v_i^{k+2} = x_i^{k+1} + r_1(2p_i^{k+1} - 2x_i^{k+1}) + r_2\left(2p_g^{k+1} - 2x_i^{k+1}\right) + \omega v_i^{k+1}. \tag{13}$$

We now choose $x_i^{k+1} = (x_1, y_1)$ and $p_i^{k+1} = (x_2, y_2)$ in any two adjacent quadrants such that $f(x_i^{k+1}) > f(p_i^{k+1})$ and $p_g^{k+1} = \left(-\frac{x_2}{1+\epsilon}, -y_2\right)$, where ϵ is a small positive number. For example, if we choose $x_i^{k+1} \in [0,2]^2$ and $p_i^{k+1} \in [-1,0]^2$ then it follows from (13) that

$$x_i^{k+2} = (x_1, y_1) + \omega v_i^{k+1} + \left[-2x_1(r_1 + r_2) + 2x_2\left(r_1 - \frac{r_2}{1+\epsilon}\right), -2y_1(r_1 + r_2) + 2y_2(r_1 - r_2)\right]. \tag{14}$$

It can be shown that x_i^{k+2} given by (14) cannot improve its personal best p_i^{k+1} for all $r_1 \geqslant 0.5$ and $r_2 \geqslant 0.5(1 + 2\epsilon)$, even when $\omega = 0$. Indeed, our numerical simulation using one billion positions of x_i^{k+2}, generated using a billion random pair (r_1, r_2), $r_1 \geqslant 0.5, r_2 \geqslant 0.5(1 + 2\epsilon)$, confirmed this. The situation, depending upon the size of ωv_i^{k+1}, further deteriorates when $\omega \neq 0$ if the coordinates of ωv_i^{k+1} do not lie in the same quadrant as x_i^{k+1}. In particular, if ωv_i^{k+1} lies in the third quadrant, i.e. if the coordinates of ωv_i^{k+1} are negative and $\|\omega v_i^{k+1}\| > 2\|x_i^{k+1}\|$ then the probability that x_i^{k+2} improves its personal best is very small. For example, our numerical testing with another one billion simulated positions of x_i^{k+2} were unable to produce a better *pbest*, p_i^{k+1}, for all $r_1, r_2 \geqslant 0.15$. Each of this particle position was obtained by uniformly generating $\omega v_i^{k+1} \in [-4,-2]^2$, $x_i^{k+1} \in [0,2]^2$, $\|\omega v_i^{k+1}\| > 2\|x_i^{k+1}\|$, $p_i^{k+1} \in [-1,0]^2$ and using $p_g^{k+1} = \left(-\frac{x_2}{1+\epsilon}, -y_2\right)$. Notice that the shortcomings remain even if p_g^{k+1} is approximated in the vicinity of its location. Although the situation will improve with iterations, this will require unnecessary function calls at the later stages of any PSO that uses (6).

3.1. Proposed remedies of slow convergence

The derivation (14) of the particle positions, x_i^{k+2}, using a simple function clearly demonstrates the shortcoming of the velocity update (6) of PSO. This shortcoming could easily be worse with respect to search intensification for a badly scaled function in the neighbourhood of the global minimizer. We suggest a number of remedies for the above shortcoming of PSO-CIV. We will study these remedies numerically in the numerical section and show which one(s) of these remedies make(s) the convergence faster without compromising the reliability or the search diversification of PSO-CIV. The remedies are as follows.

($R1$) Randomize the personal best positions of m worst particles in (6) or (12) by choosing them uniformly from the personal best positions of m best particles. This is done to introduce some flexibility in particle movements at each iteration. Our experiments have shown that this adds explorations at early stages and exploitations at later stages.

($R2$) Make the inertia weight ω dependent of the particle positions and iterations, i.e., ω_i^k. Update the velocity v_i^{k+1} by setting the inertia weight to some smaller value, e.g., $\omega_i^k = 0.3$, in $\omega_i^k v_i^k$ if $h_i > 10$ else $\omega_i^k = 0.6$ for the ith particle, x_i^k, in kth iteration. In particular, update v_i^{k+1} using

$$v_i^{k+1} = \omega_i^k v_i^k + 2r_1(p_i^k - x_i^k) + 2r_2(p_g^k - x_i^k), \tag{15}$$

where ω_i^k may change with particle positions and iterations. This diminishes the effect of v_i^k fast by choosing smaller ω_i^k for higher h_i.

($R3$) Use the information of kth iteration in the updating of v_i^{k+1} for the $(k+1)$th iteration as follows. If the percentage of improvement of the *pbest* in the kth iteration is more than 50%, i.e., if more than $\frac{1}{2}N$ particles improve their personal best, then calculate $C_1 = \max\{2r_1, 2r_2\}$ and $C_2 = \min\{2r_1, 2r_2\}$, otherwise calculate $C_1 = \min\{2r_1, 2r_2\}$ and $C_2 = \max\{2r_1, 2r_2\}$, where C_1 and C_2 are the random weights in the cognitive and social term in the velocity update, e.g., (12), respectively. This will allow the updating rule to incorporate some learning components which will then influence the particle x_i^k to move longer in the direction of *gbest* than that of *pbest* if the improvement is less than 50% and vice versa.

To see the effect of the above changes we test PSO-CIV on problem (11) in Ω_1. Results suggest that the personal best replacement rate, on average, increases from 22% to 27%, 25% and 29% for the remedies $R1$ with $m = 10\%N$, $R2$ and $R3$, respectively. Any combination of the above changes increases the rate even further. For example an implementation of all three remedies increases the rate to 37% from 22%. We will justify some of the above remedies further in the next section while the numerical evidences will be shown in the numerical section.

4. Modifications to PSO-CIV

In this section, we propose two new versions of PSO. The first version modifies the velocity update rule of PSO-CIV. The second version modifies the position update rule of PSO-CIV.

To motivate our first version, we study the velocity update rule (6). At the kth iteration, the ith velocity update v_i^{k+1} is used in the updating of x_i^k $(i = 1, 2, \ldots, N)$. Then the ith particle moves from x_i^k to x_i^{k+1} in the direction of v_i^{k+1}. In an optimization context, it is expected that PSO should be exploratory (search diversification) at earlier stages of the iterations and exploitationary (search intensification) at final stages. Search diversification is needed for the identification of the region of attraction of the global minimizer. On the other hand, search intensification is needed for rapid convergence once the particles are clustered around the global minimizer. The velocity updating rule (6) of PSO-CIV has limitations with regards to both search diversification at the early stages and intensification at the later stages of PSO-CIV.

We have discussed the limitation of (6) in search intensification at the later stages of PSO-CIV in the previous section. We now discuss the search diversification at the early stages of PSO-CIV. For this, we take a likely case where p_i^k for the ith particle and p_g^k do not change for a number of iterations. The velocity update rule (6) for these iterations will use fixed p_i^k and p_g^k, making the corresponding position update x_i^{k+1} less exploratory. In particular, these iterations may create positions x_i^{k+1} that are close to p_i^k, making the difference vector $(p_i^k - x_i^k)$ short, thus limiting the exploration.

We address the above limitation of rule (6) by introducing the remedy $R1$ in PSO-CIV. The incorporation of $R1$ will be further justified in the numerical section. We argue that $R1$ enhances the search diversification at early stages and it enhances the intensification at later stages. The remedy $R1$ randomizes *pbest* or p_i^k in the cognitive component of (6). In particular, we replace $(p_i^k - x_i^k)$ for the ith particle with $(p_l^k - x_i^k)$ in (6), where p_l^k is a randomly chosen *pbest* from a number of fitter *pbests* in P. We incorporate the random vector $(p_l^k - x_i^k)$ in (6) for a number of worse particles in S. At the earlier stages, the positions of p_l^k are scattered. In addition, the random p_l^k makes v_i^{k+1} random and thereby making x_i^{k+1} exploratory. Moreover, for worse particles, at the later stages, when the particles in S are in the region of attraction of the global minimizer, both vectors

$(p_l^k - x_i^k)$ and $(p_g^k - x_i^k)$ will be directed towards locations with lower function values. This makes the algorithm conducive to convergence. This version of PSO is referred to as PSO with randomized *pbest* in the cognitive component or PSO-RPB.

Our second modification is inspired by two studies involving DE and PSO. The study carried out by Paterlini and Krink [9] has shown that DE is much better than PSO in terms of giving accurate solutions to numerical optimization problems. On the other hand, the study by Angeline [24] has shown that PSO is much faster in identifying the promising region of the global minimizer but encounters problems in reaching the global minimizer. This is exactly what we have demonstrated in the previous section for a simple function. Therefore, we integrate the complementary strengths of the two algorithms to give an efficient and reliable PSO hybrid algorithm. In particular, we embed the DE point generation scheme (mutation and crossover rules) in PSO-CIV so that the new positions in S are either generated by the PSO-CIV position update scheme or by the DE point generation scheme, per iteration. This version is referred to as PSO with hybrid scheme for position update or PSO-HS. In the next subsections, we present details of our proposed modifications.

4.1. PSO with randomized pbest in the cognitive component

This modified version, PSO-RPB, of PSO-CIV is based on velocity updates of the m ($m \ll N$) worst particles (with worst *pbests* in P) in the population set S. At the beginning of the kth iteration, we identify m worst particles and m best particles (with best *pbests* in P) in S. We then update the velocities of the m worst particles using

$$v_i^{k+1} = \omega v_i^k + 2r_1\left(p_l^k - x_i^k\right) + 2r_2\left(p_g^k - x_i^k\right), \tag{16}$$

where p_l^k is randomly chosen from the pool of m best *pbest* in P. The vector p_l^k is different from the *gbest* p_g^k. This feature tends to be exploratory at earlier stages and also expedite the convergence of PSO by directing particles with worst *pbest* towards more promising search areas, especially when the particles are clustered around the global minimizer. The velocity updates for the remaining particles use $V_2 = 2r_1(p_i^k - x_i^k)$ in (16). Notice that PSO-RPB becomes PSO-CIV when p_i^k replaces p_l^k in V_2. Therefore, PSO-CIV equipped with $R1$ is PSO-RPB. The algorithm of PSO-RPB is, therefore, easy to implement.

4.2. PSO with hybrid scheme for position update

This version hybridizes the position update rules (3) and (6) of PSO-CIV with the point generation scheme of DE. Roughly speaking, PSO-HS uses the rules (3) and

$$v_i^{k+1} = \omega v_i^k + C_1\left(p_i^k - x_i^k\right) + C_2\left(p_g^k - x_i^k\right), \tag{17}$$

at the initial stages of the search and then switches to the DE point generation scheme. The values of C_1 and C_2 in (17) are calculated for each component of v_i^{k+1}. In particular, we use $C_1 > C_2$ for each component of each particle at the $(k+1)$th iteration if more than 50% successes in replacement of *pbest* have been achieved at the kth iteration, otherwise we use $C_2 > C_1$. Clearly, PSO-CIV implements the remedy $R3$ of previous section. The incorporation of $R3$ will be further justified in the numerical section. We note here that switching to the DE scheme does not mean switching to the DE algorithm. The new algorithm still works as PSO-CIV, but the only difference is that the positions x_i^k of the particles are updated using the operators of DE. In particular, updating of the position x_i^k of the ith particle is carried out by first creating a vector $\hat{x}_i^k$ corresponding to the position x_i^k using the rule

$$\hat{x}_i^k = p_{r1}^k + F_i\left(x_{r2}^k - x_{r3}^k\right), \tag{18}$$

where x_{r2}^k and x_{r3}^k are two randomly chosen distinct positions of two particles from S and also different from the position x_i^k of the ith particle. The *pbest* p_{r1}^k is randomly chosen from P. The parameter F_i is a random scaling factor, different for each i. The rule (18) is known as the mutation rule. The rule (18) differs from the mutation of DE in that the point p_{r1}^k is chosen from P whereas in DE all three points in (18) are chosen randomly from S. Notice that the vector $\hat{x}_i$ is created by shifting a random *pbest* in P in the direction of

$(x_{r2}^k - x_{r3}^k)$. This mutation rule induces some local effects around p_{r1}^k at later stages when the particles in S are clustered around the global minimizer. If the vector $\hat{x}_i^k$ is not in the domain then (18) is repeated. The new position x_i^{k+1} is then created using the rule

$$x_{ij}^{k+1} = \begin{cases} \hat{x}_{ij}^k & \text{if } R^j \leqslant C_R \text{ or } j = I_i \\ x_{ij}^k & \text{if } R^j > C_R \quad \text{and } j \neq I_i, \end{cases} \tag{19}$$

where $\hat{x}_{ij}^k$, x_{ij}^k and x_{ij}^{k+1} are the jth components of $\hat{x}_i^k$, x_i^k and x_i^{k+1}, respectively. The integer I_i is randomly chosen from the set

$$I = \{1, 2, \ldots, n\},$$

and $R^j \in (0,1)$ is drawn randomly for each j. The rule (19) is known as the crossover rule. The parameter $C_R \in [0,1]$, known as the crossover parameter, together with I_i ensure that the position x_i^{k+1} has components coming from the components of x_i^k and the components of $\hat{x}_i^k$.

The criterion which determines when to use the DE operators (18) and (19) is based on the component-wise standard deviation of the positions x_i^k in S. At the beginning of the kth iteration ($k = 1, 2, \ldots$) we calculate the component-wise standard deviation of positions in S. That is, we calculate the standard deviation σ_j^k of the jth components of positions in S. Intuitively, as the iterations proceed, $\|\sigma^k\| \to 0$, where $\sigma^k = (\sigma_1^k, \sigma_2^k, \ldots, \sigma_n^k)$, n is the dimension of the problem being solved. We use the condition $\|\sigma^k\| \geqslant \epsilon_1 \|\sigma^0\|$ to switch between the position update rules (3) and (17) of PSO-CIV and position update rules (18) and (19) of DE, where $\epsilon_1 \in (0,1)$ is a switching parameter and is user provided. Notice that $\|\sigma^0\|$ depends on the size of the search space Ω. Notice also that PSO-HS becomes PSO-CIV when C_1 and C_2 are replaced with $2r_1$ and $2r_2$ respectively in (17), and when $\epsilon_1 = 0$. Below we present the pseudo-code of the PSO-HS algorithm.

Pseudo-code of the PSO-HS algorithm

Step 1 **Initialization**.
- Step 1a Initialize iteration counter $k = 0$
- Step 1b Initialize N random particles (x_i^k, $i = 1, 2, \ldots, N$) and store them in S.
- Step 1c Initialize N random velocities (v_i^k, $i = 1, 2, \ldots, N$) and store them in V.
- Step 1d Initialize N *pbest* (p_i^k, $i = 1, 2, \ldots, N$) and store them in P.
- Step 1e Set p_g^k equal the best *pbest* in P.

Step 2 **While not stopping criterion do**
- Step 2a Evaluate $\sigma^k = (\sigma_1^k, \sigma_2^k, \ldots, \sigma_n^k)$. If $\|\sigma^k\| \geqslant \epsilon_1 |\sigma^0\|$ then go to Step 2b else go to Step 2c.
- Step 2b For each ith particle:
  - **Update** V: Calculate v_i^{k+1} using (17)
  - **Update** S: Calculate position x_i^{k+1} using (3)
  - **Update** P: Update P using (5).
  - **Go to Step 2d**
- Step 2c For each ith particle:
  - **Update** S: Calculate position x_i^{k+1} using (18) and (19).
  - **Update** P: Update P using (5).
- Step 2d **Update** *gbest* p_g^k: $p_g^{k+1} = \arg\min_{i \in \{1,2,\ldots,N\}} f(p_i^{k+1})$.
- Step 2e $k = k + 1$.

5. Numerical results and discussion

In this section, numerical results of all the new PSO algorithms along with the results of PSO-CIV, PSO-C and PSO-DIV are presented. The new PSO algorithms are also compared with their potential competitors.

We use 54 test problems from Ali et al. [25] and Yao et al. [26]. The number of problems is 50, with several variations in dimension for some problems, bringing the final number up to 54. All the problems are of continuous variables and a detailed description of the problems can be found in Ali et al. [25]. We compare the new algorithms with the PSO variants PSO-CIV, PSO-C and PSO-DIV to assess their robustness in finding the global minimum and their efficiency in terms of the number of function calls. The algorithms were run 100 times on each of the test problems to determine the success rate (sr) (or percentage success) of each algorithm. We calculated the average number of function calls (fe) for those runs for which the global minima were found. We used sr and fe as the criteria for comparison. In all cases, a success was counted when the condition

$$f_{\min} - f_{\text{opt}} \leqslant 0.001, \tag{20}$$

was met, where $f_{\min}$ is the best solution found when an algorithm terminates and f_{opt} is the known global minimum of the problem considered. The tolerance used in condition (20) was found suitable as any value larger than 0.001 would mean counting local minimizers as the global minimizer for a number problems.

5.1. Parameter selection

All the algorithms have some parameters that have to be provided by the user. These parameters are presented in Table 1. The first six parameters are common to all algorithms tested in this sections. These are the size N of the population; ω, c_1 and c_2 are the parameters of the velocity updates (6), (16) and (17); T and ϵ are tolerances used in the stopping rule. In the literature, different values of N have been used for population size. In this paper, we set $N = 10n$, where n is the dimension of the problem under consideration. The inertia weight ω is set to 0.6 [17] for all variants of PSO. The cognitive and social parameters c_1 and c_2 were both set to 2 as suggested in the literature for all variants except PSO-C. For PSO-C, we set $c_1 = 2.8$ and $c_2 = 1.3$ as suggested in [17].

One of the important questions in PSO is when to stop a run. Many researchers have used either the maximum number of iterations (T) or maximum number of function calls as stopping conditions in their numerical experiments [17,18]. Liu et al. [18] also used $|f_{\min} - f_{\text{opt}}| \leqslant \epsilon$, where $f_{\min}$ is the best solution found so far. The stopping condition $|f_{\min} - f_{\text{opt}}| \leqslant \epsilon$ only applies if the optimal value of the problem under consideration is known. However, in many practical applications, the optimal value is not known. On the other hand, the maximum number of iterations (or function calls) cannot be judged for an arbitrary function. This may lead to unnecessary function calls when the minimum is reached long before the maximum number of iterations (or function calls)–thus increasing on computational costs. In this paper, we use a combination of the maximum number of iterations (T) and the condition $|f_{\max} - f_{\min}| \leqslant \epsilon$, where $f_{max} = f(p_h^k)$ is the function value of the current worst *pbest* p_h^k in P and $f_{\min} = f(p_g^k)$ is the function value of the current best *pbest* p_g^k in P. Since each *pbest* in P is always updated with an improving point (see (5)) at each iteration, the set P will gradually contract. Hence, we stop a run when the points in P are identical to an accuracy of four decimal places, i.e.,

Table 1
Parameters used in the PSO algorithms

N	Swarm size
ω	Inertia weight
c_1	Cognitive parameter
c_2	Social parameter
T	Maximum iterations for stopping an algorithm
ϵ	Tolerance used in the stopping rule of an algorithm
α	Dynamic inertia reduction
β	Dynamic velocity reduction
h	Number of iterations between any two consecutive replacement of p_g^k
h_i	Number of iterations between any two consecutive replacement of the ith *pbest*, p_i^k
m	Number of worst particles considered in PSO-RPB
ϵ_1	Switching parameter in PSO-HS
F_i	Scaling factor in PSO-HS
C_R	Crossover parameter in PSO-HS

$$|f_{\max} - f_{\min}| \leqslant \epsilon = 10^{-4}, \tag{21}$$

or the maximum number of iterations, $T = 5000$, is reached.

The other parameters are α, β and h in PSO-DIV. We set $\alpha = \beta = 0.99$ and $h = 10$ as suggested in [17].

A parameter of PSO-RPB is m and its value was determined after numerical testings. The parameter m was set to the closest integer value to $0.1N$. We have conducted a numerical study using a small range of values of m, e.g., using $m \in [0.05N, 0.2N]$. The study suggested that $m = 0.1N$ is a good value to choose.

The parameters of PSO-HS are ϵ_1, F_i and C_R. The parameters F_i and C_R are due to the DE algorithm. The parameter ϵ_1 is used to switch between the particle generation rules of PSO-CIV and DE. A numerical study was carried out for the parameter ϵ_1 in PSO-HS. The study suggested that a smaller value of ϵ_1 was better, especially for high dimensional problems. The most suitable values of ϵ_1 were empirically found to lie roughly in $[0.001, 0.005]$ with the overall best value being 0.003. The other parameters for PSO-HS are the scaling factor F_i in (18) and the crossover parameter in C_R in (19). In our numerical experiments, variable values of the parameters were used. The scaling factor F_i was chosen randomly from $[0.4, 1]$ for each i and C_R was chosen randomly from $[0.5, 0.7]$ for each kth iteration. Experiments have shown that these choices are better than the fixed values. Our numerical experiments with the original DE have also shown that these parameter values produces better results [27]. In the original DE algorithm, F_i and C_R are fixed. A better version of DE is therefore used when comparing the original DE with other algorithms in this section.

The maximum velocity limitation for PSO-CIV, PSO-RPB and PSO-HS was set as $v^{\max} = 0.5(x^u - x^l)$. However, the initial maximum velocity limitation for PSO-DIV was set as $v^{\max} = (x^u - x^l)$ as suggested in [17].

When updating positions in PSO it may happen that the new positions leave the search region Ω. Whenever this happens, we employ the strategy suggested in Paterlini and Krink [9]. That is, if the component x_{ij}^{k+1}, $j = 1, 2, \ldots, n$, of the new position x_i^{k+1} leaves the domain of the search space, it is reflected back into the domain by

$$x_{ij}^{k+1} = x_{ij}^{k+1} - 2\left(x_{ij}^{k+1} - x_j^u\right), \tag{22}$$

if $x_{ij}^{k+1} > x_j^u$ and

$$x_{ij}^{k+1} = x_{ij}^{k+1} + 2\left(x_j^l - x_{ij}^{k+1}\right), \tag{23}$$

if $x_{ij}^{k+1} < x_j^l$, where x_j^u and x_j^l are the upper and lower bounds of each jth component, respectively. In both cases, the jth velocity component is reversed, i.e.,

$$v_{ij}^{k+1} = -v_{ij}^{k+1}. \tag{24}$$

5.2. Numerical studies of parameter values

We study the effect of changing parameters values of PSO-CIV in this section. This study is necessary to justify the proposed changes to the parameter values which resulted in two new variants of PSO-CIV. Results of the new variants, PSO-HS and PSO-RPB, will follow this study. In this study, we use 50 test problems with dimension up to 20. Each problem was run 100 times and therefore there were 5000 runs in total. Summarised results are presented in Table 2 where the results in the columns under fe and sr are the total results; *np* is the number of problems solved out of 50 problems; sr/39 is the total number of successes out of 3900 runs on 39 problems and sr/np is the total sr in np problems. For a fair comparison we use the total fe and sr on 39 problems where all algorithms were successful.

In order to facilitate the understanding and to make the difference between various parameter setting more explicit we append to PSO-CIV the parameter or its value, e.g., PSO-CIV(ω,c_1,c_2). Using this notation we thus write PSO-CIV as PSO-CIV(0.6,2,2), and hence the parameter values used in versions 1–6 of PSO-CIV in Table 2 are very clear. The version 7 implements PSO-CIV with C_1 and C_2 as suggested in remedy $R3$ in Section 3.1. The use of $R3$ has been indicated with the superscript. Similarly, the version 8 implements PSO-CIV with a variable inertia weight, ω_i^k, as suggested in remedy $R2$ in Section 3.1. The version 9 implements PSO-CIV with remedies $R2$ and $R3$ of Section 3.1 and these have also been indicated in superscripts.

Table 2
Comparison of PSO-CIV on 50 problems

Version	PSO-CIV	fe	sr/39	sr/np	np
1	PSO-CIV(0, 2, 2)	1521839	3341	3431	44
2	PSO-CIV(0.3, 2, 2)	1578211	3337	3494	44
3	PSO-CIV(0.6, 2, 2) = PSO-CIV	1798196	3572	3699	44
4	PSO-CIV(0.6, 1, 2)	283281	3424	3616	41
5	PSO-CIV(0.6, 2, 1)	1134914	3510	3623	43
6	PSO-CIV(0.6, 1, 1)	281552	3220	3220	39
7	PSO-CIV$(0.6, C_1, C_2)^{R3}$	485669	3528	3708	44
8	PSO-CIV $(\omega_i^k, 2, 2)^{R2}$	1292168	3580	3710	44
9	PSO-CIV $(\omega_i^k, C_1, C_2)^{R2,R3}$	406168	3463	3662	45

The versions 1–3 measure the effects of lowering ω. Results show that total sr/np are much lower for lower values of ω although each of the three versions solves the same number of problems. The implementation of PSO-CIV with $\omega > 0.6$, although the results are not shown, are detrimental to fe while achieving about the same sr. On the other hand, a comparison of the versions 3–6, which measure the effects of lowering c_1 and c_2, shows that both sr/np and np decreases with the decrease of either c_1 or c_2 or both. A comparison between the versions 4 and 5 shows that the effect of lowering c_2 is much higher than the effect of lowering c_1, with respect to fe. If the parameter values are chosen based on numerical studies then perhaps the values $\omega = 0.6$, $c_1 = c_2 = 2$ are balanced choices as suggested in previous literature [17].

Having discussed the sensitivities of the three parameters of PSO-CIV, we now study the effects of parameter-based remedies, $R2$ and $R3$, suggested in Section 3.1. The remedy $R1$ is not parameter-based and thus will be studied later in this section. We begin with remedy $R3$. The version 7, which replaces c_1r_1 and c_2r_2 in PSO-CIV with C_1 and C_2, is about 72% superior to PSO-CIV with respect to fe while it has minor differences in sr/np and np. Similarly, the version 8, which implements $R2$ in PSO-CIV, is about 28% superior to PSO-CIV with respect to fe. On the other hand, the version 9, which implements both $R2$ and $R3$, is about 77% superior to PSO-CIV with respect to fe but about 2% inferior to PSO-CIV with respect to sr/np. It also solved an additional problem, namely the 9 dimensional Price Transistor Modelling (PTM) problem. Therefore, Table 2 clearly shows the effectiveness of the remedies $R1 - R3$ suggested in Section 3.1. Next we conduct the numerical studies with the new algorithms, PSO-RPB and PSO-HS, where we will also study the effect of remedy $R1$.

5.3. A numerical study of PSO-RPB and PSO-HS

In this section, we study the effect of the remedies, $R1$–$R3$, in the new algorithms to determine which remedy suites which algorithm. Again we use 50 problems used in Table 2.

We begin with PSO-RPB. The objective here is to determine which of the remedies is more effective than the others. For this we summarize the total results of various implementations of PSO-RPB in Table 3, where we have used similar notations as in Table 2. All the algorithms in Table 3 were successful in 44 problems, except

Table 3
Comparison of PSO-RPB on 44 problems

Version	PSO	fe	sr
3	PSO-CIV(0.6, 2, 2)=PSO-CIV	3147039	3699
7	PSO-CIV $(0.6, C_1, C_2)^{R3}$	1246428	3697
8	PSO-CIV $(\omega_i^k, 2, 2)^{R2}$	2557358	3710
9	PSO-CIV $(\omega_i^k, C_1, C_2)^{R2,R3}$	1155143	3660
10	PSO-CIV $(0.6, 2, 2)^{R1}$=PSO-RPB	1033327	3701
11	PSO-CIV $(0.6, C_1, C_2)^{R1,R3}$	607018	3665
12	PSO-CIV $(\omega_i^k, C_1, C_2)^{R1,R2,R3}$	482908	3618
13	PSO-CIV $(\omega_i^k, 2, 2)^{R1,R2}$	1353226	3622

Table 4
Comparison of PSO-HS on 44 problems

Version	PSO-HS	fe	sr
14	PSO-HS $(0.6, C_1, C_2, \epsilon_1)^{R3}$=PSO-HS	1 094 122	3702
15	PSO-HS $(\omega_i^k, C_1, C_2, \epsilon_1)^{R2,R3}$	809 772	3665
16	PSO-HS $(\omega_i^k, 2, 2, \epsilon_1)^{R2}$	2 328 808	3612

version 9 which was able to solve 45 problems. Results in Table 3 are therefore based on 44 problems, where we have also presented the results of the versions 3,7–9 of PSO-CIV of Table 2 on 44 problems for the purpose of comparison.

Results of the versions 10–13 in Table 3 clearly show that PSO-RPB, which implements remedy $R1$, is superior to all other implementations, versions 11–13, that combine $R1$ with other remedies. In particular, these variants are much inferior to PSO-RPB with respect to sr. PSO-RPB is about 67% superior to PSO-CIV with respect to fe while it achieves almost the same sr. Moreover, PSO-RPB is also better than the other implementation of PSO-CIV, namely the implementations of PSO-CIV, namely the implementations in versions 7 and 9. Version 8 achieves slightly better sr than PSO-RPB but this was achieved at a much higher cost in fe. The design of PSO-RPB is therefore clearly justified.

We now study PSO-HS. This is a hybrid of PSO-CIV and DE. PSO-HS implements $R3$ only. We study the effect of $R2$ and $R3$ in PSO-HS via (17). Using the earlier notation we denote PSO-HS by PSO-HS$(0.6, C_1, C_2, \epsilon_1)^{R3}$, where the nonzero parameter ϵ_1 indicates the activation of DE position generation using (18) and (19). All implementations of PSO-HS solved 44 problems. We present the summarised total results on 44 problems in Table 4.

Table 4 shows that PSO-HS with $R2$ is the worse performer, although it is much superior to PSO-CIV, see for example version 8 in Table 2 or Table 3. On the other hand, a joint implementation of $R2$ and $R3$ reduces both fe and sr. Overall best results are therefore obtained by PSO-HS.

Finally, we compare the original PSO-CIV with its new variants PSO-RPB and PSO-HS. Results obtained by versions 3,10 and 14 suggest that all three achieved very similar sr. However, PSO-RPB and PSO-HS are about 67% and 65% superior to PSO-CIV respectively, in terms of fe. While these results are considerable improvements on PSO-CIV, we further compare the new algorithms with other variants of PSO and with DE in the next section.

5.4. Comparisons and discussions

Having shown the superiority of the new algorithms over PSO-CIV we now compare PSO-RPB and PSO-HS with other algorithms with a full set of results. In particular, we compare the new algorithms with PSO-C and PSO-DIV and with a potential competitor, e.g., DE. First we compare the algorithms using 50 problems of dimension up to 20 and then we compare them again with four more problems of dimension 30. In the first test set, there are 5000 runs in total. We note that none of the algorithms succeeded in finding the global minimum for four 10 dimensional functions, namely Epistatic Michalewicz (EM), Salomon (SAL), Price's Transistor Modelling (PTM) and Shekel's Foxholes (SF) and 9, 17 dimensional Storn's Tchebychev (ST) functions. Except for these six problems, all algorithms have a positive sr for all the other 44 problems. Therefore, results for these six problems are not reflected in our presentation in Table 5. The results of PSO-DIV, PSO-C, DE along with the results of the new PSO algorithms using 44 problems are presented in Table 5. The first and second columns of Table 5, respectively, represent the acronym of the problems and the corresponding dimensions. The tr in the last row is the total fe and sr.

The total results show that amongst all PSO variants PSO-RPB and PSO-HS are very much comparable while PSO-DIV is superior to PSO-C. On the other hand, the new variants, PSO-RPB and PSO-HS, are much superior to PSO-C in terms of fe and sr. The new variants are also much more superior to PSO-DIV in terms of fe while PSO-DIV outperforms the new algorithms by about 30 sr. This is because PSO-DIV has 93% success rate on Schwefel problem (SWF) while the new algorithms have only about 5% success. In terms of fe, PSO-RPB and PSO-HS, are superior to PSO-DIV by about 58% and 56%, respectively.

Table 5
Comparison of DE, PSO-RPB, PSO-HS, PSO-C and PSO-DIV on 44 problems

P	n	DE		PSO-RPB		PSO-HS		PSO-C		PSO-DIV	
		fe	sr	fe	sr	fe	sr	fe	sr	fe	sr
H6	6	7053	95	8420	60	8675	59	87216	52	14098	46
ML	10	256730	22	26100	7	47900	8	23873	7	50650	6
MRP	2	41474	56	3046	49	11007	53	56756	61	50026	53
MGP	2	1312	70	2766	81	2526	75	6137	89	4485	82
NF2	4	79249	96	35581	100	74956	100	199035	100	175060	100
PRD	2	3024	92	3066	63	8072	55	21816	84	16733	79
RG	10	93991	100	46100	5	43633	4	453377	7	370786	5
RB	10	500100	1	500100	86	464075	83	495441	13	500100	2
SF1	2	3998	79	6253	27	11091	32	37587	49	92943	37
SWF	10	34875	100	32246	5	27100	6	70814	4	49236	93
S5	4	5824	92	6641	32	6030	31	107246	23	44227	31
S7	4	5346	95	6860	41	6078	53	103739	51	43489	59
S10	4	4822	99	6747	57	5602	47	91496	62	54172	65
WP	4	16286	98	30946	100	68447	100	200040	97	199820	98
ACK	10	25237	100	36116	100	29187	100	31332	100	41313	100
AP	2	723	100	1784	100	1637	100	2620	100	3487	100
BL	2	1012	100	2083	100	1833	100	19681	100	8845	100
B1	2	988	100	3411	100	1806	100	3484	100	3177	100
B2	2	973	100	3317	100	1818	100	2567	100	3202	100
BR	2	1305	100	2652	100	2018	100	43735	100	38871	100
CB3	2	923	100	2226	100	1642	100	2484	100	2195	100
CB6	2	1127	100	2561	100	2390	100	48148	100	27566	100
CM	4	2238	100	5546	100	4024	100	5939	100	5598	100
DA	2	1358	100	4301	100	2115	100	8375	100	13852	100
EP	2	1125	89	2450	88	1619	96	2678	89	2175	75
EXP	10	10234	100	15354	100	13089	100	16269	100	26466	100
GP	2	884	100	2817	100	1698	100	3049	100	3186	100
GW	10	85129	100	19916	100	14475	100	20661	100	31658	100
GRP	3	3548	97	7037	100	11316	100	23855	100	45829	100
H3	3	1238	100	3564	100	2948	100	4080	100	3483	100
HV	3	3835	98	7823	100	5363	100	5464	100	4849	100
HSK	2	738	100	1909	100	1560	100	2324	100	1609	100
KL	4	1925	100	2677	100	2408	100	4691	100	3966	100
LM1	3	1420	100	3402	100	2924	100	6757	100	5534	100
LM2	10	11256	100	16440	100	14115	100	19366	100	26016	100
MC	2	824	100	1701	100	1587	100	2155	100	1038	100
MR	3	3157	78	4413	100	6573	100	10287	100	6629	100
MCP	4	3829	100	4299	100	3964	100	2685	100	1877	100
NF3	10	75949	100	58446	100	100370	100	483343	100	306301	100
PP	10	15411	100	25110	100	16699	100	19493	100	29773	100
PQ	4	5852	100	7484	100	5360	100	8833	100	12883	100
SF2	2	2236	100	5742	100	2930	100	7880	100	8223	100
SBT	2	2430	100	4206	100	6216	100	90476	100	55842	100
SIN	20	64016	100	59668	100	45246	100	49608	100	88218	100
tr		1385004	4057	1033327	3701	1094122	3702	2906892	3688	2479459	3731

A comparison of the new algorithms and DE shows that DE is superior to the new algorithms with about 350 successes in 4400 runs. A close look at the individual sr in Table 5 shows that this difference was caused by the Shekel family and SWF. In the former DE has about 100% sr while the new algorithms have less than 44% sr and in the latter DE has 100% sr while the new algorithms have barely 5% sr. However, DE is inferior to PSO-RPB and PSO-HS by 25% and 21% respectively, with respect to fe. Moreover, for the difficult problems in Table 5, where success rate is less than 100%, DE, on average, used more fe than the new algorithms. It is therefore quite clear that there are circumstances, e.g., for the case of expensive function calls, where the new algorithms certainly have a role to play.

Table 6
Comparison in higher dimensions

P	PSO-CIV		PSO-RPB		PSO-HS		DE	
	fe	sr	fe	sr	fe	sr	fe	sr
ACK	365 160	100	335 040	100	462 940	100	257 960	100
GW	244 770	100	193 950	100	103 350	100	244 000	100
SIN	232 740	100	201 300	100	96 840	100	158 275	100
tr	842 670		730 290		663 130		660 235	

Finally, we compare the new PSO algorithms with their original counterpart and with DE. For this comparison we took four 30 dimensional problems, namely the Ackley function (ACK), the Griewank function (GW), the sinusoidal function (SIN) and the Rosenbrock function (RB). Except RB, all three problems were solved by all algorithms. Total results of the four algorithms are presented in Table 6. Since all the algorithms have 100% success, we only compare the total fe. Results show that within the PSO variants, PSO-CIV is the worse performer and PSO-HS is the best. Although the overall best is DE the difference between PSO-HS and DE is negligible and therefore they are comparable.

6. Conclusion

In this paper, two new particle swarm optimization (PSO) methods are proposed. Our main contribution is the modification of the velocity update rule and the hybridization of the position update rule of particle swarm optimization. In particular, we incorporated a randomized personal best in the cognitive component of the velocity update rule and hybridized the position update rule with the differential evolution trial point generation operators. The new algorithms are tested on a large set of test problems. The results clearly demonstrate that the incorporation of the new features makes PSO conducive to convergence. A comparison with other variants of PSO suggested in the literature shows that the new PSO are much more superior in efficiency.

A comparison of the new PSO with differential evolution suggests that they both have strengths and weaknesses and therefore they certainly have different roles to play in global optimization.

Further research is underway for designing a PSO algorithm for high dimensional problems with equality and inequality constraints.

Acknowledgements

The authors thank A. Bhowandas of the School of Computational and Applied Mathematics, University of the Witwatersrand, Johannesburg, for his help in writting the code of PSO.

References

[1] M.M. Ali, A. Törn, Population set-based global optimization algorithms: Some modifications and numerical studies, Computers and Operations Research 31 (10) (2004) 1703–1725.
[2] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin, 1996.
[3] R. Storn, K. Price, Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[4] P. Kaelo, M.M. Ali, A numerical study of some modified differential evolution algorithms, European Journal of Operational Research 169 (3) (2006) 1176–1184.
[5] J. Kennedy, R.C. Eberhart, The particle swarm: social adaptation in information-processing systems, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, Cambridge, UK, 1999, pp. 379–387.
[6] I.C. Trelea, The particle swarm optimization algorithm: Convergence analysis and parameter selection, Information Processing Letters 85 (6) (2003) 317–325.
[7] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, Cambridge, UK, 1999, pp. 11–32.
[8] V. Maniezzo, A. Carbonaro, Ant colony optimization: An overview, in: C.C. Ribeiro, P. Hansen (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, Boston, 2002, pp. 469–492.

[9] S. Paterlini, T. Krink, Differential evolution and particle swarm optimization in partitional clustering, Computational Statistics and Data Analysis 50 (1) (2006) 1220–1247.
[10] C. Elegbede, Structural reliability assessment based on particle swarm optimization, Structural Safety 27 (2) (2005) 171–186.
[11] K.E. Parsopoulos, M.N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, Natural Computing 1 (2002) 235–306.
[12] X.-T. Feng, B.-R. Chen, C. Yang, H. Zhou, X. Ding, Identification of visco-elastic models for rocks using genetic programming coupled with the modified particle swarm optimization algorithm, International Journal of Rock Mechanics and Mining Sciences 43 (2006) 789–801.
[13] P.C. Fourie, A.A. Groenwold, The particle swarm optimization algorithm in size and shape optimization, Structural and Multidisciplinary Optimization 23 (4) (2002) 259–267.
[14] 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.
[15] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proceedings of the IEEE International Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ, 1998, pp. 69–73.
[16] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, Evolutionary Programming VII. Lecture Notes in Computer Science, Vol. 1447, Springer, Berlin, 1998, pp. 591–600.
[17] J.F. Schutte, A.A. Groenwold, A study of global optimization using particle swarms, Journal of Global Optimization 31 (1) (2005) 93–108.
[18] B. Liu, L. Wang, Y.-H. Jin, F. Tang, D.-X. Huang, Improved particle swarm optimization combined with chaos, Chaos, Solitons and Fractals 25 (2005) 1261–1271.
[19] Y. Da, G. Xiurun, An improved PSO-based ANN with simulated annealing technique, Neurocomputing 63 (2005) 527–533.
[20] T. Hendtlass, A combined swarm differential evolution algorithm for optimization problems, Lecture Notes in Artificial Intelligence, 2070, Springer-Verlag, Berlin Heidelberg, 2001, pp. 11–18.
[21] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu, L.M. Wang, An improved GA and a novel PSO-GA-based hybrid algorithm, Information Processing Letters 93 (2005) 255–261.
[22] R. May, Simple mathematical models with very complicated dynamics, Nature 261 (1976) 459–467.
[23] L.C.W. Dixon, G.P. Szegö, The global optimization problem: an introduction, in: L.C.W. Dixon, G.P. Szegö (Eds.), Towards Global Optimization, Vol. 2, North-Holland, Amsterdam, 1978, pp. 1–15.
[24] P.J. Angeline, Evolutionary optimization versus particle swarm optimization: philosophy and performance differences, Evolutionary Computation VII. Lecture Notes in Computer Science, 1447, Springer, Berlin, 1998, pp. 601–610.
[25] M.M. Ali, C. Khompatraporn, Z.B. Zabinsky, A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, Journal of Global Optimization 31 (2005) 635–672.
[26] X. Yao, Y. Liu, G. Lin, Evolutionary Programming made faster, IEEE Transactions on Evolutionary Computation 3 (2) (1999) 82–102.
[27] P. Kaelo, Some population set based methods for unconstrained global optimization, Ph.D. thesis, University of the Witwatersrand, South Africa, 2005.