Rkia Fajr* and Abdelaziz Bouroumi

An Improved Particle Swarm Optimization Algorithm for Global Multidimensional Optimization

https://doi.org/10.1515/jisys-2017-0104
Received March 19, 2017; previously published online December 29, 2017.

Abstract: This paper introduces a new variant of the particle swarm optimization (PSO) algorithm, designed for global optimization of multidimensional functions. The goal of this variant, called ImPSO, is to improve the exploration and exploitation abilities of the algorithm by introducing a new operation in the iterative search process. The use of this operation is governed by a stochastic rule that ensures either the exploration of new regions of the search space or the exploitation of good intermediate solutions. The proposed method is inspired by collaborative human learning and uses as a starting point a basic PSO variant with constriction factor and velocity clamping. Simulation results that show the ability of ImPSO to locate the global optima of multidimensional functions are presented for 10 well-know benchmark functions from CEC-2013 and CEC-2005. These results are compared with the PSO variant used as starting point, three other PSO variants, one of which is based on human learning strategies, and three alternative evolutionary computing methods.

Keywords: Particle swarm optimization, swarm intelligence, global optimization, multidimensional functions, collaborative learning.

2010 Mathematics Subject Classification: 90C59.

1 Introduction

Optimization problems are commonly encountered in many real-world situations related to various application domains, including finance, business, transportation, medicine, engineering, etc. Experts in these fields use optimization techniques on a daily basis in order to find the best decisions and tradeoffs that maximize things like profit, revenue, and efficiency, or that minimize others like costs, risks, and losses.

The objective in an optimization problem is to find values for a set of parameters that maximize or minimize an objective function. Concretely, a minimization problem can be defined as follows:

Given f: $S \rightarrow R$, where $S \subseteq R^d$ and d is the dimension of the search space;

find $x^* \in S$ such that $f(x^*) \leq f(x)$, $\forall x \in S$.

By taking the negative of the objective function, any maximization problem can be defined as a minimization problem. Therefore, optimization problems can be referred to as minimization problems without any loss of generality.

Global optimization is the problem of finding the global minimizer, x^*, and the value of the global minimum, $f(x^*)$. It is an NP-hard problem for which many stochastic search algorithms have been developed in order to find near-optimal solutions in reasonable amounts of time. Stochastic search algorithms

***Corresponding author: Rkia Fajr,** Information Processing Laboratory, Ben M'sik Faculty of Sciences, Hassan II University of Casablanca, Avenue Driss El Harti, B.P. 7955 Sidi Othmane, Casablanca, Morocco, e-mail: fajr.rkia@gmail.com
Abdelaziz Bouroumi: Information Processing Laboratory, Ben M'sik Faculty of Sciences, Hassan II University of Casablanca, Avenue Driss El Harti, B.P. 7955 Sidi Othmane, Casablanca, Morocco

are heuristics that start from one or more candidate solutions, which can be randomly chosen, and try to continuously improve them throughout an iterative process using learning rules to guide the search.

Single-solution heuristics proceed by modifying and improving a single-candidate solution. They include simulated annealing [6], iterated local search [30], guided local search [45], and variable neighborhood search [31]. Population-based heuristics maintain multiple candidate solutions and try to improve them using both individual and group characteristics. They include evolutionary computation [1] and swarm intelligence [2], which are the two main families of nature-inspired algorithms. Evolutionary algorithms are inspired from natural evolution. Depending on the problem to be solved, they use more or less complicated operators that simulate the natural processes of selection, reproduction, and mutation in order to evolve the population of candidate solutions. Swarm-based algorithms are relatively new and simpler heuristics that have emerged as nature-inspired, population-based methods that are capable of producing fast and low-cost solutions to complex optimization problems [2]. These algorithms try to model the collective behavior of social swarms in nature, such as bird flocks, fish schools, ant colonies, and honey bees. One of the most known techniques of this category is particle swarm optimization (PSO), which was initially developed as a simulation of social behavior of bird flocks searching for corn [20, 23]. PSO has been successfully applied to many optimization problems such as task assignment [36], reactive power and voltage control [52], and real-time robot planning [34].

Requiring only primitive mathematical operators, PSO can be easily implemented and is computationally inexpensive in terms of both speed and memory requirements [12]. This is why it has received increased interest from researchers who have developed several PSO variants intended to enhance the performance of the original algorithm. One of these variants is a generalized PSO version, developed by the same authors of the original algorithm, where each particle in the swarm interacts with a restrictive number of particles in its neighborhood instead of interacting with the whole swarm as in the original version [12]. However, although it is less sensitive to local optima, this version has proven to be slower, more complex, and has brought new challenges, such as the sensitivity of the algorithm to the way of defining neighborhoods [22]. Many variants based on neighborhood topology are available in the literature [32, 46, 51].

Other PSO variants are based on parameter settings, focusing on improving performance by proper selection of control parameters of the algorithm such as inertia weight and acceleration coefficients. Inertia weight was first introduced by Shi and Eberhart in 1998 [37] as a new parameter aimed at controlling the impact of the velocity of each particle on its movement in the search space. This parameter can be chosen either as a constant or as a function of the number of iterations [37]. Therefore, many authors have proposed different PSO variants with inertia weights that are fixed, random, linearly decreasing, linearly increasing, non-linear, exponential, adaptive, etc. [39, 41]. A comparison of several kinds of inertia weights can be found in Ref. [18]. Acceleration coefficients are parameters that ensure the balance between exploration of new regions of the search space using the own experience of each particle and exploitation of other particles' experience in order to concentrate the search around a promising candidate solution. Many authors have studied the effect of these two parameters on the convergence of PSO algorithms [17, 35, 54] and provided different techniques for selecting them [10, 25, 53]. In Ref. [8], a constriction factor was also introduced in order to control the effects of the two acceleration coefficients on PSO convergence.

A third category of PSO variants is formed by algorithms based on learning strategies. In this framework, Liang et al. [26] have proposed a comprehensive learning PSO variant (CLPSO), which uses a learning strategy that updates the velocity of each particle according to the best historical information of all other particles. Xinchao [50] proposed a perturbed variant of PSO, called PPSO, whose particle-updating strategy is based on the concept of perturbed global best. According to this strategy, the position of the global best particle is replaced by a new vector whose components are generated from a normal distribution depending on the original best position and a non-increasing function of the number of iterations [50]. In 2013, Nepomuceno and Engelbrecht proposed a self-adaptive heterogeneous PSO variant called HPSO, where particles select their next behavior based on the success or failure of different position and velocity update equations [33]. Considering that the best experiences of particles are distributed around the problem's optima, Lim and Mat Isa proposed, in the same year, a two-layer PSO with intelligent division of labor (TLPSO-IDL) based on the use of a new learning strategy to evolve the current swarm and an adaptive task allocation process for the

memory swarm [28]. Later, in 2015, the same authors proposed another variant, called adaptive division of labor PSO (ADOLPSO), aimed at mitigating the high computational cost and slow convergence incurred by TLPSO-IDL [29]. Based on population division, this variant uses both swarm diversity and fitness to adaptively assign the search task of each particle. It uses a stagnation-prevention module to avoid premature convergence and two new operators, which are only applied to the best solution, to improve the convergence speed [29]. In the same year, Tanweer et al. [41] proposed a self-regulating particle swarm optimization algorithm (SRPSO) using two learning strategies inspired by the human learning: a self-regulating inertia weight, for the best particle to improve exploration, and self-perception in global search direction for the rest of the particles, for better exploitation. Combining self-regulation with a dynamic mentoring scheme inspired by human learning psychology, the same authors proposed, in 2016, another variant called dynamic mentoring and self-regulation-based particle swarm optimization (DMeSR-PSO) [43]. Based on their experiences, the particles in DMeSR-PSO are divided into three different groups with dynamically varying sizes and different learning strategies for velocity updates: the mentors that are equipped with a strong self-belief-based search, the mentees that take guidance from mentors, and the independent learners that employ self-perception strategy [43]. In the same year, Tanweer et al. [42] proposed an improved variant of SRPSO referred to as directionally driven self-regulating particle swarm optimization (DD-SRPSO) algorithm. DD-SRPSO incorporates two new learning strategies: a directional update strategy and a rotational invariant strategy. The first strategy is used for the poorly performing particles that are grouped together to get directional updates from the group of elite particles. The best particle uses the same strategy as in SRPSO. All other particles are randomly selected to undergo either the new rotational invariant strategy to explore the rotation variance property of the search space or the SRPSO strategy of self-perception of the global search direction [42]. More recently, in 2017, Dong et al. [11] proposed a new PSO variant based on learning strategies, called opposition-based particle swarm optimization with adaptive mutation strategy (AMOPSO). Combining the generalized opposition-based learning [47] with adaptive mutation, AMOPSO uses two complementary strategies: an adaptive mutation selection that performs local search around the global optimal particle in the current population, and an adaptive nonlinear inertia weight that ensures a balance between exploration and exploitation during the iteration process.

The last category of modified PSO variants are hybridized versions that combine it with other algorithms such as genetic algorithms [15], ant colony [24], artificial bee colony [49], and differential evolution [48]. A survey of the hybrid PSO algorithms can be found in Ref. [44].

Although evident progress and notable achievements have been attained in developing different categories of new PSO variants, most improvements are achieved at the cost of slow convergence or complicated algorithmic structures [4, 5, 9, 29]. This means that the alleviation of PSO drawbacks without significantly impairing its simplicity or its convergence speed is still a challenge [29].

The goal of this paper is to introduce a new PSO variant that tries to improve the performance of PSO algorithm in finding better solutions while preserving both its simplicity and its fast convergence. This variant is based on the introduction of a simple yet effective new operation in the iterative search process in order to enhance the algorithm's ability in both exploring new areas of the search space that may contain better solutions and exploiting intermediate solutions. The starting point for the proposed variant is a modified PSO version based on parameter settings. It is the version introduced by Clerc and Kennedy [8], which uses a constriction factor, acceleration coefficients, and velocity clamping. In the rest of the paper, we will refer to this version as χPSO.

The remainder of this paper is organized as follows. The next section presents a brief review of PSO algorithm, followed by a description of the main steps and a pseudo-code of the χPSO version used as starting point for this work. Then, a detailed description of the PSO modification introduced by this work is presented and explained in Section 3. Simulation results that illustrate the performance of the resulting PSO variant are presented in Section 4 using 10 benchmark multimodal and multidimensional functions with varying dimensions. These results are discussed and compared to those produced by the reference method χPSO, three other PSO variants, and three alternative evolutionary computing methods. Finally, Section 5 provides a conclusion to this work and some directions for further development.

2 Review of Particle Swarm Optimization (PSO)

PSO is a nature-inspired, population-based, stochastic heuristic for solving optimization problems by simulating the collective behavior of bird flocks [20]. A swarm in PSO is a population of interacting agents, called particles, representing each candidate solution. Once initialized, the n particles of a swarm collectively move in the search space, sharing information among them in order to collaboratively find the best solution. For this, at each iteration during the search process, each particle i in the swarm is characterized by four attributes: (1) a position in the search space, $x_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \in S$. This position, whose fitness is evaluated using the function f to minimize, is adjusted using a collaborative learning rule that takes into account both the own experience of i and the global experience of its neighborhood, N_i; (2) a velocity $v_i = (v_{i1}, v_{i2}, \ldots, v_{id})$, which controls the movement of i in the search space. This velocity is used by the learning rule in order to compute the next position of i; (3) the previous best position found so far by i, $p_i = (p_{i1}, p_{i2}, \ldots, p_{id})$. The fitness of this position summarizes the own experience of the particle. It is memorized in a private variable called $pbest_i$; and (4) the best position, p_i^*, found so far within N_i. The fitness of this particle represents the global experience of the neighborhood of i. It is memorized in a global variable called $gbest_i$, which is accessible to all particles within N_i.

In the original version of the algorithm, each particle interacts with the whole swarm; meaning that p_i^* is the same for all particles. The common value of this attribute is noted p_g, where g is the index of the particle in the whole swarm that has found the best solution so far. This attribute is called the global best solution and its fitness is memorized in a global variable denoted *gbest* [20].

This version is a particular case of the generalized version where p_i^* depends on the neighborhood of i, which only consists of a small subset of the swarm [12]. The generalized version presents the advantage of being less vulnerable to converging toward local minima, but it is more complicated and less rapid than the original one.

In the rest of this section, we describe in more detail the main steps of the χPSO algorithm used as starting point for this work.

2.1 Swarm Initialization

Assuming, for simplicity of notation, that the search space is a hypercube of the form $S = [x_{\min}, x_{\max}]^d$ where $x_{\min}$ and $x_{\max}$ are two real numbers, the d components of the initial position x_i and the initial best position p_i of each particle i, $(1 \le i \le n)$, are initialized as follows:

$$x_{ij} = p_{ij} = \text{rand}(x_{\min}, x_{\max}), \text{ for } j = 1, 2, \ldots, d \tag{1}$$

where rand is a function that returns a random real number uniformly generated within the range between its two arguments. Similarly, the d components of the initial velocity v_i are randomly initialized using the relation

$$v_{ij} = \text{rand}(-v_{\max}, v_{\max}), \text{ for } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, d \tag{2}$$

where $v_{\max}$ is a clamping parameter whose role is to prevent the velocities from exploding, which causes premature convergence. The value of this parameter is generally chosen as a fraction of $x_{\max}$ in the form $v_{\max} = k \times x_{\max}$ with $0.1 < k < 1.0$ [21].

The previous best fitness of each particle i is initialized as $pbest_i = f(p_i)$, and the initial global best fitness is set to $gbest = pbest_g$, where g denotes the index of the particle that satisfies the condition $f(p_g) \le f(p_i)\ \forall\, i \ne g$.

2.2 Learning Rule

The learning rule is the relation used by the search process in order to control the movement of each particle by iteratively adjusting both its velocity and its position. It translates the way particles exploit the

information they share about their previous experience in order to guide their collective search for better solutions. For this, the d components of the velocity of each particle are generally adjusted using a relation of the form

$$v_{ij} = wv_{ij} + \text{rand}(0,\, c_1)\times(p_{ij} - x_{ij}) + \text{rand}(0,\, c_2)\times(p_{gj} - x_{ij}),\ \text{for } 1 \le i \le n \text{ and } 1 \le j \le d \tag{3}$$

Then, the d components of the position of this particle are adjusted according to the rule

$$x_{ij} = x_{ij} + v_{ij},\ \text{for } 1 \le j \le d \tag{4}$$

The first term in the right part of equation (3) represents the momentum of the particle. It uses a positive inertia weight parameter w that controls the impact of the previous velocity [37]. The difference $(p_{ij} - x_{ij})$ in the second term represents the particle's own experience. Its influence is weighted by a random number between 0 and c_1, where c_1 is a learning parameter called acceleration coefficient. Likewise, the difference $(p_{gj} - x_{ij})$ in the third term represents the experience of the entire swarm, whose contribution is weighted by rand(0, c_2) where c_2 is the acceleration coefficient toward the global solution.

2.3 Constriction Factor

The constriction factor was introduced in 2002 by Clerc and Kennedy as a means for appropriately choosing the three parameters w, c_1, and c_2 in order to ensure the convergence of PSO toward a stable point [8]. In the χPSO algorithm, this factor is defined as a function of c_1 and c_2 by the expression

$$\chi = \frac{2}{\varphi - 2 + \sqrt{\varphi^2 - 4\varphi}} \tag{5}$$

where

$$\varphi = c_1 + c_2 > 4$$

Using this factor, the velocity adjustment rule described by equation (3) can be rewritten as:

$$v_{ij} = \chi(v_{ij} + \text{rand}(0,\, c_1)\times(p_{ij} - x_{ij}) + \text{rand}(0,\, c_2)\times(p_{gj} - x_{ij})) \tag{6}$$

By fixing c_1 and c_2 to 2.05, as suggested in Ref. [13], we satisfy the constraint $c_1 + c_2 > 4$, and the resulting constriction factor is $\chi = 0.729$. With these values, it is easy to verify that equation (6) is equivalent to equation (3) with $w = 0.729$ and $c_1 = c_2 = 1.49445$.

2.4 Velocity Clamping

Velocity clamping is a complementary operation, useful for ensuring that the components of the adjusted velocities remain within the limits $-v_{\max}$ and $v_{\max}$. It is performed according to the rule

$$v_{ij} = \min(v_{\max},\, \max(-v_{\max},\, v_{ij})) \tag{7}$$

This rule is applied after the adjustment of each velocity component, v_{ij}, but before adjusting the corresponding position component x_{ij} [13]. However, if the constraint $x_{ij} \in [x_{\min},\, x_{\max}]$ is no longer satisfied after adjusting the d position components, no clamping of x_{ij} is performed; meaning that the particles are allowed to move outside the search space, but as shown by the pseudo-code in Algorithm 1, the fitness evaluation is only performed for particles that remain within this space [13].

Algorithm 1: The χPSO Algorithm.

Given a function $f(x)$ to minimize on a domain $S = [x_{min}, x_{max}]^d \subseteq R^d$
1. Parameter settings
Choose n, c_1, c_2, v_{max}, and FE_{max}.
Calculate χ using (5).
2. Swarm initialization
for $i = 1$ **to** n {
– Initialize x_i and p_i according to (1)
– Initialize v_i according to (2)
– Set $pbest_i = f(p_i)$
}
Find g such that $f(p_g) \leq f(p_k)$ for each $k \neq g$
Set $gbest = f(p_g)$
$FE = n$;//number of function evaluations
3. Search process loop
do {
for $i = 1$ **to** n {
– recalculate v_i according to (6) and (7)
– adjust x_i according to (4)
– **if** ($x_i \in S$) {
– calculate $tmp = f(x_i)$; $FE = FE + 1$;
– **if** ($tmp < pbest_i$) {
– $p_i = x_i$; $pbest_i = tmp$;
– **if** ($pbest_i < gbest$) {$g = i$; $gbest = pbest_i$;}
}
}
}//end for
} **while** ($FE < FE_{max}$ and $|gbest - f(x^*)| > 10^{-08}$);
4. Output
Return $x' = p_g$ and $f(x') = gbest$

2.5 Stopping Criterion

A stochastic algorithm can be stopped any time during its run, generating the best solution found so far. In practice, however, the search process is generally repeated until a stopping criterion is met. In the case of the χPSO algorithm, the stopping condition used is ($FE = FE_{max}$ or $|gbest - f(x^*)| \leq 10^{-08}$), where FE denotes the number of function evaluations and $f(x^*)$ the actual minimum.

3 The Proposed Algorithm, ImPSO

As shown in the previous section, PSO is a very simple algorithm that requires only elementary mathematical operations, which makes it easy to implement in computer code using any programming language. PSO variants based on parameter settings, such as χPSO, ensure convergence of the algorithm by preventing divergent or cyclic behavior of particles, but remain sensitive to premature convergence toward local minima, which is one of the most difficult problems facing any search and optimization algorithm. PSO variants based on neighborhood topology have provided a way to address this problem, but have also proven to be slower and more complex, which limited their use in practice.

The PSO variant we propose in this work uses as starting point: the χPSO algorithm presented in the previous section, and tries to improve its performance in finding better solutions while preserving its fast convergence. The idea behind this variant, called ImPSO, consists in introducing a simple yet effective modification to the search process of χPSO in order to improve both its exploration and its exploitation abilities.

To achieve this goal, a new operation is introduced at the end of each iteration of the do loop of the search process (step 3 in Algorithm 1). This operation consists in randomly choosing a single particle k in the whole swarm, except the best one, and allowing this particle to suddenly and stochastically modify its position x_k. It is inspired by human collaboration in problem solving, particularly the aptitude of human members to intuitively explore new directions in order to improve the collaborative work of all the group for finding the best solution to a difficult problem [3, 7]. The sudden change in x_k simulates the human intuition for exploring a new direction in the quest for a better solution. It is governed by a stochastic rule that functions as follows: each of the d components of x_k has a certain probability of being either randomly reinitialized or set to the same value as the corresponding component of the global best position p_g. The probability of a component to be randomly reinitialized is heuristically determined as a decreasing function of the number of components d so that to ensure a good balance between exploration and exploitation.

Concretely, the implementation of this stochastic rule is performed as follows: for each component j of x_k, $(1 \le j \le d)$, a random number is generated within the range (0, 1). This number represents the chance that x_{kj} would be randomly reinitialized. If it is greater or equal to a certain threshold, whose value is fixed to $1 - 1/d$, then x_{kj} is reinitialized to a random number in the range $(x_{\min}, x_{\max})$; otherwise, x_{kj} is set, as shown below in equation (8), to p_{gj}.

$$x_{kj} = \begin{cases} \text{rand}(x_{\min}, x_{\max}) & \text{if rand}(0, 1) \ge 1 - 1/d \\ p_{gj} & \text{otherwise} \end{cases} \tag{8}$$

The threshold value $1 - 1/d$ is heuristically determined so that the particle k whose position should be suddenly modified does not move too far away from the swarm. The choice of this value guarantees that, on average, only one of the d components of x_k will be randomly initialized, which is sufficient to explore new regions of the search space without moving too far away from the swarm. Indeed, in the particular case of $d = 1$, for example, $1 - 1/d = 0$ and the unique component of x_k is surely set to rand$(x_{\min}, x_{\max})$, which prevents the particle k from entering in collision with the particle that occupies the best position p_g. For $d = 2$, $1 - 1/d = 0.5$ and each of the two components has a probability of 50% of being randomly reinitialized, meaning that, on average, one of these components will be set to rand$(x_{\min}, x_{\max})$. More generally, equation (8) means that the probability each component x_{kj} has to be randomly reinitialized is $1/d$ and that the probability of the same component to be set to p_g is $1 - 1/d$.

Once the particle k is randomly chosen and its new position x_k determined using equation (8), the fitness associated with this position is evaluated using the function to minimize f. Then, three different cases can be distinguished depending on the value of $f(x_k)$:

1. $f(x_k) \le gbest$, which necessarily means that $f(x_k) \le pbest_k$
2. $gbest \le f(x_k) \le pbest_k$
3. $f(x_k) \ge pbest_k$

Algorithm 2: The Modified Search Process Loop Used by the Proposed PSO Variant ImPSO.

```
do {
  for i = 1 to n {
    //Processing of particle i as shown in the pseudo code of χPSO (Algorithm 1)
  }
  //The added operation
  Randomly choose a particle k such that k ≠ g
  for j = 1 to d re-initialize x_kj according to (8)
  set tmp = f(x_k);
  FE = FE + 1;
  if (tmp < pbest_k) {
    – p_k = x_k; pbest_k = tmp;
    – if (pbest_k < gbest) {g = k; gbest = pbest_k; }
  }
} while (FE < FE_max and |gbest – f(x*)| > 10^-08);
```

The first case means that the new position x_k improves not only the own experience of particle k but also the experience of the whole swarm. Consequently, both the global best solution of the whole swarm and the local best solution of particle k should be adjusted using the new position x_k and its fitness $f(x_k)$. In the second case, only the best solution of particle k should be adjusted. The third case means that x_k improves neither *gbest* nor $pbest_k$. Accordingly, no adjustment is performed in this case, but if any better solution is located around x_k this solution is most likely to be discovered during the rest of the search process.

In summary, Algorithm 2 describes more precisely the modified version of the search process loop used by the proposed PSO variant, ImPSO.

4 Experimental Results and Discussion

The proposed algorithm, ImPSO, was implemented in software using the C++ language and tested on a variety of multimodal and multidimensional test functions with varying dimensions. In this section, we present and discuss the typical results obtained for 10 test functions chosen from the suites of benchmark functions of the CEC-2005 and CEC-2013 special sessions on real-parameter optimization [27, 38]. The experimental work is divided into three parts using different evaluation criteria. The first part uses five test functions from CEC-2013 and is dedicated to illustrating the improvements brought by ImPSO to the χPSO variant used as starting point for this work. The second part uses five test functions from CEC-2005 and is dedicated to comparing ImPSO with SRPSO, which is a state-of-the-art PSO variant using human learning strategies [41]. The third part uses the same test functions as the first one and concerns the comparison of ImPSO with two other PSO variants and three alternative evolutionary computing methods.

4.1 CEC-2013 Test Functions

The five test functions taken from CEC-2013 are shifted or rotated versions [27] of five well-known benchmark functions: f_1 = Schwefel's function, f_2 = Rastrigin's function, f_3 = Lunacek bi-Rastrigin function, f_4 = Rosenbrock's rotated function, and f_5 = Ackley's rotated function.

The rotations of f_4 and f_5 are performed using two orthogonal matrices, M_1 and M_2, generated from standard normally distributed entries by Gram-Schmidt ortho-normalization [27].

The C++ implementations of these functions, as well as the numerical components of the shift point o and the rotation matrices M_1 and M_2, are taken from Ref. [27], where they are publicly available. The mathematical equations defining the five functions are:

$$f_1(z) = 418.9829 \times d - \sum_{j=1}^{d} g(z_j) + f_1^*$$

where $z = \Lambda^{10}\left(\frac{1000(x-o)}{100}\right) + 4.209687462275036e+002$, Λ^{α} is the diagonal $d \times d$ matrix defined by $\lambda_{ii} = \alpha^{\frac{i-1}{2(d-1)}}$ for $i = 1, 2, \ldots, d$; $f_1^* = -100$ and

$$g(z_j) = \begin{cases} z_j \sin(|z_j|^{1/2}) & \text{if } |z_j| \le 500 \\ (500 - \text{mod}(z_j, 500))\sin\left(\sqrt{|\text{mod}(|z_j|, 500) - 500|}\right) - \dfrac{(z_j - 500)^2}{10{,}000d} & \text{if } z_j > 500 \\ (\text{mod}(|z_j|, 500) - 500)\sin\left(\sqrt{|\text{mod}(|z_j|, 500) - 500|}\right) - \dfrac{(z_j + 500)^2}{10{,}000d} & \text{if } z_j < -500 \end{cases}$$

$$f_2(x)=\sum_{j=1}^{d}(z_j^2-10\cos(2\pi z_j)+10)+f_2^*$$

with $z=\Lambda^{10}T_{asy}^{0.2}\left(T_{osz}\left(\frac{5.12(x-o)}{100}\right)\right)$, T_{asy}^{β}: if $x_j>0$, $x_j=x_j^{1+\beta\frac{j-1}{d-1}\sqrt{x_j}}$, for $j=1,2,\ldots,d$ T_{osz}: $x_j=\text{sign}(x_j)\exp(\hat{x}_j+0.049$ $(\sin(c_1\hat{x}_j)+\sin(c_2\hat{x}_j)))$, for $j=1$ and d, with $\hat{x}_j=\begin{cases}\log(|x_j|) & \text{if } x_j\neq 0\\ 0 & \text{otherwise}\end{cases}$, $\text{sign}(x_j)=\begin{cases}-1 & \text{if } x_j<0\\ 0 & \text{if } x_j=0\\ 1 & \text{otherwise}\end{cases}$

$$c_1=\begin{cases}10 & \text{if } x_j>0\\ 5.5 & \text{otherwise}\end{cases},\quad c_2=\begin{cases}7.9 & \text{if } x_j>0\\ 3.1 & \text{otherwise}\end{cases},\text{ and } f_2^*=-400$$

$$f_3(x)=\min\left(\sum_{j=1}^{d}(\hat{x}_j-\mu_0)^2,\ Dd+s\sum_{j=1}^{d}(\hat{x}_j-\mu_1)^2\right)+10\left(d-\sum_{j=1}^{d}\cos(2\pi z_j)\right)+f_3^*$$

with $\mu_0=2.5$, $\mu_1=-\sqrt{\frac{\mu_0^2-d}{s}}$, $s=1-\frac{1}{2\sqrt{d+20}-8.2}$, $D=1$, $y=\frac{10(x-o)}{100}$, $\hat{x}_j=2\text{sign}(x_j^*)y_j+\mu_0$ for $j=1, 2, \ldots d$, $z=\Lambda^{100}(\hat{x}-\mu_0)$, and $f_3^*=300$

$$f_4(z)=\sum_{j=1}^{d-1}((100(z_j^2-z_{j+1})^2+(z_j-1)^2)+f_4^*$$

with $z=M_1\left(\frac{2.048(x-o)}{100}\right)$ and $f_4^*=-900$

$$f_5(x)=-20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{j=1}^{d}z_j^2}\right)-\exp\left(\frac{1}{d}\sum_{j=1}^{d}\cos(2\pi z_j)\right)+20+e+f_5^*$$

with $z=\Lambda^{10}M_2T_{osy}^{0.5}(M_1(x-o))$ and $f_5^*=-700$

4.2 CEC-2005 Test Functions

The five test functions taken from CEC-2005 are f_6 = Rosenbrock's function, f_7 = Weierstrass's rotated function, f_8 = Schwefel's problem, f_9 = Ackley's rotated function, f_{10} = expanded Griewank's plus Rosenbrock functions. Shifted functions are defined by $z=x-o$ and shifted rotated ones by $z=(x-o)*M$, where $o=(o_1, o_2, \ldots, o_d)$ is the shifted point, and M is the $d\times d$ transformation matrix used for the rotation. All these functions are scalable and non-separable, and their C++ implementations, as well as the numerical components of the shift point o and the rotation matrix M are taken from Ref. [38], where they are publicly available. The mathematical equations defining the five functions are

$$f_6(x)=\sum_{i=1}^{d-1}(100(z_i^2-z_{i+1})^2+(z_i-1)^2)+f_6^*$$

with $z=x-o+1$, $f_6^*=390$ and $x\in[-100, 100]^d$

$$f_7(x)=\sum_{i=1}^{d}\left(\sum_{k=0}^{k_{\max}}[a^k\cos(2\pi b^k(z_i+0.5))]\right)-d\sum_{k=0}^{k_{\max}}[a^k\cos(2\pi b^k\cdot 0.5)]+f_7^*$$

with $z=(x-o)*M$, $a=0.5$, $b=3$, $k_{max}=20$, $f_7^*=90$ and $x\in[-0.5, 0.5]^d$

$$f_8(x)=\sum_{i=1}^{d}(A_i-B_i(x))^2+f_8^*$$

with $A_i=\sum_{j=1}^{d}(a_{ij}\sin\alpha_j+b_{ij}\cos\alpha_j)$, $B_j(x)=\sum_{j=1}^{d}(a_{ij}\sin x_j+b_{ij}\cos x_j)$ are two $d\times d$ matrices, a_{ij} and b_{ij} are integer random numbers in the range [–100, 100], $\alpha=[\alpha_1, \alpha_2, \ldots, \alpha_d]$ where α_j are random numbers in the range $[-\pi, \pi]$, $f_8^*=-460$ and $x\in[-\pi, \pi]^d$.

$$f_9(x)=-20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}z_i^2}\right)-\exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi z_i)\right)+20+\exp(1)+f_9^*$$

with $z=(x-o)*M$, $f_9^*=-140$ and $x\in[-32, 32]^d$.

$$f_{10}(x)=h_1(h_2(z_1, z_2))+h_1(h_2(z_2, z_3))+\ldots+h_1(h_2(z_{d-1}, z_d))+h_1(h_2(z_d, z_1))+f_{10}^*$$

with $h_1(x)=\sum_{i=1}^{d}\frac{x_i^2}{4000}-\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$, $h_2(x)=\sum_{i=1}^{d-1}(100(x_i^2-x_{i+1})^2+(x_i-1)^2)$, $z=x-o+1$, $f_{10}^*=-130$ and $x\in[-3, 1]^d$.

4.3 Experimental Results: Part 1

This part uses the five test functions from CEC-2013. For each of these functions, four d values are considered: 10, 30, 50, and 100; and for each dimension d, the algorithm is repeated 51 times for each function f, recording the minimal value, $f(x')$, found by the algorithm after each run. The final results of all these experiments are presented in terms of the best, the worst, the mean, the median, and the standard deviation values of $f(x')$, calculated over the 51 runs. These results are compared to those produced by the χPSO variant using the same experimental protocol with the following algorithmic parameters: $n=50$, $v_{max}=x_{max}$, $c_1=c_2=2.05$, $FE_{max}=10{,}000\times d$, search space $=S=[-100, 100]^d$, and the stopping criterion ($FE=FE_{max}$ or $|f(x')-f(x^*)|<10^{-8}$), where $f(x^*)$ denotes the actual global minimum f^*.

For $d=10$, the experimental results obtained by repeating 51 times both ImPSO and χPSO algorithms for each function are summarized in Table 1. Highlighted in bold on this table are numerical results that correspond to the best-performing algorithm for each tested function.

Table 1: Best, Worst, Median, Mean, and Standard Deviation Obtained After 51 Runs of Both ImPSO and χPSO Algorithms for CEC-2013 Benchmark Functions f_1 to f_5 with $d=10$.

f	f^*	Algorithm	$f(x')$				
			Best	Worst	Median	Mean	SD
f_1	−100	ImPSO	**–100**	**–88.17**	**–99.62**	**–97.88**	**2.44**
		χPSO	−96.46	340.49	45.25	60.14	127.08
f_2	−400	ImPSO	**–400**	**–400**	**–400**	**–400**	**0**
		χPSO	−399.00	−379.00	−396.02	−394.84	3.79
f_3	300	ImPSO	310.12	**310.63**	**310.15**	**310.21**	**0.13**
		χPSO	**304.01**	318.47	313.56	313.45	2.57
f_4	−900	ImPSO	**–899.99**	**–890.18**	**–890.18**	−894.35	4.78
		χPSO	**–899.99**	**–890.18**	**–890.18**	**–894.45**	**4.75**
f_5	−700	ImPSO	**–680.00**	**–679.53**	**–679.68**	**–679.68**	**0.08**
		χPSO	−679.85	−679.52	−679.66	679.67	**0.08**

Table 2: Best, Worst, Median, Mean, and Standard Deviation Obtained After 51 Runs of Both ImPSO and χPSO Algorithms for CEC-2013 Benchmark Functions f_1 to f_5 with $d=30$.

f	f^*	Algorithm	$f(x')$				
			Best	Worst	Median	Mean	SD
f_1	−100	ImPSO	**−99.95**	**21.34**	**−97.53**	**−94.41**	**16.62**
		χPSO	541.99	3417.18	2015.42	1910.31	529.67
f_2	−400	ImPSO	**−400**	**−400**	**−400**	**−400**	**0**
		χPSO	−371.14	−308.46	−340.30	−338.19	15.71
f_3	300	ImPSO	**330.43**	**330.65**	**330.46**	**330.48**	**0.04**
		χPSO	361.65	499.71	391.32	395.40	22.18
f_4	−900	ImPSO	−899.67	**−824.58**	−887.26	−869.41	**26.75**
		χPSO	**−899.84**	−806.23	**−888.97**	**−869.97**	30.41
f_5	−700	ImPSO	**−679.30**	**−678.98**	**−679.09**	**−679.10**	0.06
		χPSO	−679.17	**−678.98**	−679.07	−679.07	**0.04**

Table 3: Best, Worst, Median, Mean, and Standard Deviation Obtained After 51 Runs of Both ImPSO and χPSO Algorithms for CEC-2013 Benchmark Functions f_1 to f_5 with $d=50$.

f	f^*	Algorithm	$f(x')$				
			Best	Worst	Median	Mean	SD
f_1	−100	ImPSO	**−98.43**	**−77.51**	**−94.16**	**−93.58**	**3.80**
		χPSO	1949.94	6605.39	3967.66	3976.40	838.77
f_2	−400	ImPSO	**−400**	**−400**	**−400**	**−400**	**0**
		χPSO	−314.43	−105.49	−218.91	−222.44	47.31
f_3	300	ImPSO	**350.78**	**351.16**	**350.88**	**350.90**	**0.08**
		χPSO	463.30	640.53	536.10	541.27	41.68
f_4	−900	ImPSO	−873.42	−804.93	**−856.55**	−851.52	15.72
		χPSO	**−890.98**	**−810.23**	**−856.55**	**−857.56**	**15.31**
f_5	−700	ImPSO	**−679.01**	**−678.82**	**−678.91**	**−678.91**	**0.04**
		χPSO	−678.99	−678.80	−678.86	−678.87	**0.04**

Table 1 shows that, globally, ImPSO performed better than χPSO for the first three functions and similarly to χPSO for the remaining two. Indeed, for the first function, for example, the best solution found by ImPSO coincides exactly with the actual minimum, $f_1^* = -100$, and its median, mean (and even worst) solutions are very close to f_1^*, by opposition to their equivalent for χPSO, which are relatively far away from f_1^*.

For the second function, f_2, which is the only example not satisfying the non-separability criterion, the salient result from Table 1 is that ImPSO converges to the actual minimum f_2^* in all the 51 runs. This is due to the modification introduced in ImPSO, which independently acts on each variable of the randomly chosen position. For the third function, although the best solution found by χPSO is slightly closer to f_3^* than the one found by ImPSO, the worst, the median, and the mean solutions provided by ImPSO are much more closely related to each other and to f_3^* than their equivalent for χPSO. Finally, as shown in the last four lines of Table 1, the two algorithms give very similar results for the two rotated functions f_4 and f_5.

For the remaining three dimensions, $d=30$, $d=50$, and $d=100$, the experimental results obtained for the five test functions are, respectively, reported in Tables 2–4, using the same form as in Table 1.

A brief examination of these results shows that the proposed algorithm sustains its performance for the five analyzed functions, while a significant deterioration in the performance of χPSO is observed for the three dimensions in the case of the first three functions, especially for f_1, where even the best solutions provided by χPSO for $d=30$, $d=50$, and $d=100$ are hundreds of times greater than the actual minimum f_1^*.

Table 4: Best, Worst, Median, Mean, and Standard Deviation Obtained After 51 Runs of Both ImPSO and χPSO Algorithms for CEC-2013 Benchmark Functions f_1 to f_5 with $d=100$.

f	f^*	Algorithm	$f(x')$				
			Best	Worst	Median	Mean	SD
f_1	−100	ImPSO	**−95.90**	**33.18**	**−87.46**	**−83.46**	**23.78**
		χPSO	9095.61	14435.97	11471.08	11475.51	1270.17
f_2	−400	ImPSO	**−400**	**−400**	**−400**	**−400**	**0**
		χPSO	−40.82	533.26	284.52	283.64	137.09
f_3	300	ImPSO	**401.62**	**401.97**	**401.76**	**401.78**	**0.078**
		χPSO	943.31	1658.33	1222.98	1227.97	178.15
f_4	−900	ImPSO	−874.28	−624.06	−727.29	−736.81	**52.50**
		χPSO	**−880.33**	**−639.96**	**−733.29**	**−745.17**	55.86
f_5	−700	ImPSO	**−678.80**	**−678.66**	**−678.72**	**−678.71**	0.034
		χPSO	−678.78	−678.65	−678.70	−678.70	**0.026**

For f_1, f_2, and f_3, these results also show an increase with d in the improvement brought by ImPSO to the χPSO algorithm, whose performance in higher dimensions $d=30$, $d=50$, and $d=100$ is sustained only for the two rotated functions f_4 and f_5.

4.4 Experimental Results: Part 2

This part concerns the comparison of ImPSO with SRPSO, which is a start of the art PSO variant based on concepts from human learning strategies [41]. It uses the five CEC-2005 benchmark functions f_6 to f_{10}, for which SRPSO results are available in Ref. [41].

For each of these functions, two dimensions are considered: $d=30$ and $d=50$, and for each function and each dimension, the execution of ImPSO is repeated 100 times, which is the same number of repetitions used for SRPSO. The results of these executions are summarized in Table 5 in the form of the median, mean, and standard deviation of the error $f(x')-f(x^*)$, calculated over 100 runs.

Table 5 shows that ImPSO performs better for two shifted functions: f_6 and f_{10}. For f_9, which is both shifted and rotated, the results are slightly in favor of ImPSO. For the remaining two functions, f_7, which is shifted and rotated, and f_8, which is shifted but not rotated, the performance of SRPSO is better than ImPSO.

Table 5: Median, Mean, and Standard Deviation of the Error $f(x')-f(x^*)$ Calculated Over 100 Runs of ImPSO and SRPSO Algorithms for CEC-2005 Benchmark Functions f_6 to f_{10} with $d=30$ and $d=50$.

f	Algorithm	$d=30$			$d=50$		
		Median	Mean	SD	Median	Mean	SD
f_6	ImPSO	**8.24**	**14.76**	26.45	**28.01**	**47.51**	46.71
	SRPSO	14.72	39.78	57.08	30.5	50.08	44.40
f_7	ImPSO	27.68	27.67	4.01	56.89	56.36	6.34
	SRPSO	**9.08**	**9.58**	4.27	**21.72**	**21.95**	5.65
f_8	ImPSO	2.53E+3	4.21E+3	4.79E+3	1.24E+4	1.64E+4	1.36E+4
	SRPSO	**1.64E+3**	**2.49E+3**	2.80E+3	**6.99E+3**	**1.18E+4**	1.21E+4
f_9	ImPSO	**20.06**	**20.07**	0.038	**20.11**	**20.13**	0.047
	SRPSO	20.95	20.94	0.042	21.15	21.14	0.037
f_{10}	ImPSO	**1.22**	**1.26**	0.26	**2.10**	**2.15**	0.34
	SRPSO	2.50	2.63	0.61	5.22	5.23	1.01

4.5 Experimental Results: Part 3

This part is aimed at comparing the proposed algorithm with two other PSO variants as well as with three alternative evolutionary computing methods. It uses another series of experiments conducted on the five CEC-2013 test functions for the three dimensions 10, 30, and 50. The results of this series are presented in terms of the mean and the standard deviation of the error $f(x') - f(x^*)$, calculated over 51 runs of the algorithm for each function and each dimension. These results are compared with those produced by two other PSO variants: CLPSO [26] and self-adaptive HPSO [33] and three evolutionary algorithms: a differential evolution method, called Rank-DE [16], a genetic algorithm, GL-25 [14], and an evolution strategy algorithm, CMA-ES [19].

Table 6: Mean and Standard Deviation of the Error $f(x') - f(x^*)$ Calculated for 51 Runs of ImPSO, CLPSO, and Self-adaptive HPSO Algorithms for CEC-2013 Benchmark Functions f_1 to f_5 with $d = 10$, 30, and 50.

f	d	ImPSO		Self adaptive HPSO		CLPSO	
		Mean	SD	Mean	SD	Mean	SD
f_1	10	**2.11**	**2.44**	37.8	42.1	8.87	29.5
	30	**5.58**	16.62	704	238	18.2	**6.33**
	50	**6.41**	**3.80**	1960	460	135	20.5
f_2	10	**0**	**0**	0.17	0.37	0.058	0.23
	30	**0**	**0**	23.6	8.76	**0**	**0**
	50	**0**	**0**	86.1	19.9	4.26e − 5	2.22e − 5
f_3	10	**10.2**	0.13	11	3.28	**10.2**	**0.07**
	30	**30.48**	**0.04**	52.6	7.11	31.5	0.27
	50	**50.90**	**0.08**	116	15.8	62.2	1.31
f_4	10	5.64	4.78	2.64	**2.05**	**2.52**	3.72
	30	30.58	26.75	29.9	17.6	**26.4**	**7.17**
	50	48.47	15.72	55.1	22.5	**47.3**	**0.46**
f_5	10	**20.3**	0.08	**20.3**	0.08	20.4	**0.07**
	30	**20.89**	0.06	20.9	0.06	21.0	**0.05**
	50	**21.08**	0.04	21.1	0.04	21.1	**0.03**

Table 7: Mean and Standard Deviation of the Error $f(x') - f(x^*)$ Calculated for 51 Runs of ImPSO, Rank-DE, GL-25, and CMA-ES Algorithms for CEC-2013 Benchmark Functions f_1 to f_5 with $d = 10$, 30, and 50.

f	d	ImPSO		Rank-DE		GL-25		CMA-ES	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD
f_1	10	**2.11**	**2.44**	38.6	29.1	279	200	1770	429
	30	**5.58**	**16.62**	4560	229	3350	961	5540	604
	50	**6.41**	**3.80**	10,400	331	6480	2740	8580	1110
f_2	10	**0**	**0**	**0**	**0**	3.81	2.10	191	261
	30	**0**	**0**	103	7.28	25.5	8.85	48.3	14.1
	50	**0**	**0**	283	11.7	49.7	13.4	137	284
f_3	10	10.2	0.13	**10.1**	**0.01**	17.1	4.58	1040	409
	30	**30.48**	**0.04**	134	7.47	110	34.7	4280	811
	50	**50.90**	**0.08**	342	12.4	183	50.9	7100	1310
f_4	10	**5.64**	4.78	7.84	**3.80**	6.06	4.72	6.3	4.7
	30	30.58	26.75	15.2	**0.26**	29.6	22.4	**2.07**	7.17
	50	48.47	15.72	45.1	**0.18**	44.7	0.74	**43.6**	0.79
f_5	10	**20.3**	0.08	**20.3**	**0.05**	20.4	0.06	21.1	0.44
	30	**20.89**	0.06	20.9	0.06	21.0	**0.04**	21.5	0.08
	50	**21.08**	0.04	21.1	0.04	21.1	**0.03**	21.5	0.07

The results of this part are summarized in Tables 6 and 7. Table 6 shows, for each function and each of the three dimensions, 10, 30, and 50, the mean and the standard deviation of the error measure $f(x') - f(x^*)$, calculated over 51 runs of ImPSO, as well as the published results of CLPSO and self-adaptive HPSO taken, respectively, from Refs. [33] and [40]. In Table 7, the same ImPSO results are compared with those obtained using three evolutionary algorithms: Rank-DE [16], GL-25 [14], and CMA-ES [19].

Table 6 shows that ImPSO performed better than the self-adaptive HPSO and CLPSO for the first three functions and the three dimensions, except for f_2 and $d = 30$, where CLPSO produced the same results as ImPSO. For f_4 and $d = 50$, ImPSO performed better than the self-adaptive HPSO and similarly to CLPSO. For the last function, f_5, the three compared algorithms provide similar results for the three tested dimensions.

For f_1, f_2, and f_3, Table 7 shows that ImPSO performed better than the three evolutionary algorithms for the three studied dimensions. The same table shows that ImPSO performed better than these algorithms for f_4 and $d = 10$ and produced similar results for the last function, f_5, in the case of the three studied dimensions.

5 Conclusion

This paper introduced a new variant of the PSO algorithm, aimed at improving the ability of the algorithm to both exploring new regions of the search space and exploiting intermediate solutions during its search for global optima of multidimensional and multimodal functions. The improvement brought by this variant, called ImPSO, is due to the introduction of a new operation into the search process. This operation is performed at the end of each iteration of the search process loop. It is inspired by human collaborative learning and consists in randomly choosing a single particle of the swarm and allowing it to suddenly change its position according to a stochastic learning rule that guarantees either the exploration of a new region of the search space or the exploitation of the global best solution found so far. To achieve this goal, each of the d components of the position to be changed has a probability of $1/d$ to be randomly initialized and a probability of $1 - 1/d$ to be set to the corresponding component in the best global position. Thus, for $d = 1$, the position is randomly initialized with a probability of 1, which avoids any collision with the global best particle; and for $d > 1$, only one component is, on average, randomly initialized, which is sufficient for exploring a new region of the search space without going too far from the whole swarm.

The proposed method uses as starting point the basic χPSO variant with constriction factor and velocity clamping. Its effectiveness was tested on 10 well-known multimodal benchmark functions from CEC-2013 and CEC-2005 test suites, with varying dimensions. The results of these tests are compared with χPSO, self-regulating PSO, which is a state-of-the-art variant based on human learning strategies, two other PSO variants, and three alternative evolutionary computing methods. Comparison results indicate that ImPSO has shown significant improvement on most functions, especially for shifted ones.

For two CEC2005 functions, one of which is both shifted and rotated, the performance of SRPSO is better than ImPSO indicating the need for further refinement of the proposed method. This encourages the continuation of this work in order, for example, to investigate the possibility of using other ways for selecting the particles whose positions should suddenly be changed or other rules for changing these positions.

Bibliography

[1] A. Abraham, N. Nedjah and L. de M. Mourelle, *Evolutionary computation: from genetic algorithms to genetic programming*, pp. 1–20, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[2] H. Ahmed and J. Glasgow, Swarm intelligence: concepts, models and applications, *School of Computing, Queens University Technical Report* (2012).

[3] D. Anderson, E. Anderson, N. Lesh, J. Marks, B. Mirtich, D. Ratajczak and K. Ryall, *Human-guided simple search*, AAAI/IAAI, pp. 209–216, Austin, TX, USA, 2000.

[4] A. Banks, J. Vincent and C. Anyakoha, A review of particle swarm optimization. Part I: background and development, *Nat. Comput.* **6** (2007), 467–484.

[5] A. Banks, J. Vincent and C. Anyakoha, A review of particle swarm optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications, *Nat. Comput.* **7** (2008), 109–124.
[6] D. Bertsimas and J. Tsitsiklis, Simulated annealing, *Stat. Sci.* **8** (1993), 10–15.
[7] A. Bouroumi and R. Fajr, Collaborative and cooperative e-learning in higher education in Morocco: a case study, *Int. J. Emerg. Technol. Learn.* **9** (2014), 66–72.
[8] M. Clerc and J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, *IEEE Trans. Evol. Comput.* **6** (2002), 58–73.
[9] Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez and R. G Harley, Particle swarm optimization: basic concepts, variants and applications in power systems, *IEEE Trans. Evol. Comput.* **12** (2008), 171–195.
[10] J. Ding, J. Liu, K. R. Chowdhury, W. Zhang, Q. Hu and J. Lei, A particle swarm optimization using local stochastic search and enhancing diversity for continuous optimization, *Neurocomputing* **137** (2014), 261–267.
[11] W. Dong, L. Kang and W. Zhang, Opposition-based particle swarm optimization with adaptive mutation strategy, *Soft Comput.* **21** (2017), 5081–5090.
[12] R. C. Eberhart and J. Kennedy, A new optimizer using particle swarm theory, in: *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1, pp. 39–43, New York, NY, 1995.
[13] R. C. Eberhart and Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, in: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, 1, vol. 1, pp. 84–88, La Jolla, CA, USA, 2000.
[14] C. Garca-Martnez, M. Lozano, F. Herrera, D. Molina and A. M Sánchez, Global and local real-coded genetic algorithms based on parent-centric crossover operators, *Eur. J. Oper. Res.* **185** (2008), 1088–1113.
[15] H. Garg, A hybrid PSO-GA algorithm for constrained optimization problems, *Appl. Math. Comput.* **274** (2016), 292–305.
[16] W. Gong and Z. Cai, Differential evolution with ranking-based mutation operators, *IEEE Trans. Cybern.* **43** (2013), 2066–2081.
[17] L. Guo and X. Chen, A novel particle swarm optimization based on the self-adaptation strategy of acceleration coefficients, in: *International Conference on Computational Intelligence and Security, 2009. CIS'09.*, 1, pp. 277–281, IEEE, Beijing, China, 2009.
[18] W. Han, P. Yang, H. Ren and J. Sun, Comparison study of several kinds of inertia weights for PSO, in: *2010 IEEE International Conference on Progress in Informatics and Computing (PIC)*, 1, pp. 280–284, IEEE, Shanghai, China, 2010.
[19] N. Hansen and A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, *Evol. Comput.* **9** (2001), 159–195.
[20] J. Kennedy and R. C. Eberhart, Particle swarm optimisation, in: *IEEE International Conference on Neural Networks*, pp. 1942–1948, IEEE, Perth, WA, Australia, 1995.
[21] J. Kennedy and R. C. Eberhart, in: D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli and K. V. Price, eds., The Particle Swarm: Social Adaptation in Information-Processing Systems, in: *New Ideas in Optimization*, pp. 379–388, McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.
[22] J. Kennedy and R. Mendes, Population structure and particle swarm performance, in: *Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC'02.*, 2, pp. 1671–1676, IEEE, Honolulu, HI, USA, 2002.
[23] J. Kennedy, R. C. Eberhart and Y. Shi, *Swarm intelligence*, Morgan Kaufmann, 2001.
[24] S. K. Lahiri and N. M. Khalfe, Hybrid particle swarm optimization and ant colony optimization technique for the optimal design of shell and tube heat exchangers, *Chem. Prod. Process Model.* **10** (2015), 81–96.
[25] M.-S. Leu and M.-F. Yeh, Grey particle swarm optimization, *Appl. Soft Comput.* **12** (2012), 2985–2996.
[26] J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Trans. Evol. Comput.* **10** (2006), 281–295.
[27] J. J. Liang, B. Y. Qu, P. N. Suganthan and A. G. Hernández-Daz, Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization, *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report* **201212**, Cancún, México, (2013).
[28] W. H. Lim and N. A. M. Isa, Two-layer particle swarm optimization with intelligent division of labor, *Eng. Appl. Artif. Intell.* **26** (2013), 2327–2348.
[29] W. H. Lim and N. A. M. Isa, Adaptive division of labor particle swarm optimization, *Expert Syst. Appl.* **42** (2015), 5887–5903.
[30] H. R. Lourenço, O. C. Martin and T. Stützle, Iterated local search, in: Glover, F. and Kochenberger G. (eds.), *Handbook of metaheuristics*, pp. 320–353, Springer, US, 2003.
[31] N. Mladenović and P. Hansen, Variable neighborhood search, *Comput. Oper. Res.* **24** (1997), 1097–1100.
[32] G. Nápoles, I. Grau and R. Bello, Particle swarm optimization with random sampling in variable neighbourhoods for solving global minimization problems, in: *International Conference on Swarm Intelligence*, pp. 352–353, Springer, Berlin, Heidelberg, 2012.
[33] F. V. Nepomuceno and A. P. Engelbrecht, A self-adaptive heterogeneous PSO for real-parameter optimization, in: *IEEE Congress on Evolutionary Computation (CEC2013)*, pp. 361–368, IEEE, Cancun, Mexico, 2013.
[34] Y. Q. Qin, D. B. Sun, N. Li and Q. Ma, Path planning for mobile robot based on particle swarm optimization, *Robot* **26** (2004), 222–225.
[35] A. Ratnaweera, S. K Halgamuge and H. C Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, *IEEE Trans. Evol. Comput.* **8** (2004), 240–255.

[36] A. Salman, I. Ahmad and S. Al-Madani, Particle swarm optimization for task assignment problem, *Microprocess. Microsyst.* **26** (2002), 363–371.
[37] Y. Shi and R. C. Eberhart, A modified particle swarm optimizer, in: *IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on Evolutionary Computation Proceedings, 1998*, pp. 69–73, IEEE, Anchorage, AK, USA, 1998.
[38] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger and S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, *KanGAL report* **2005005** (2005), 2005.
[39] M. Taherkhani and R. Safabakhsh, A novel stability-based adaptive inertia weight for particle swarm optimization, *Appl. Soft Comput.* **38** (2016), 281–295.
[40] L. Tang, Y. Dong and J. Liu, Differential evolution with an individual-dependent mechanism, *IEEE Trans. Evol. Comput.* **19** (2015), 560–574.
[41] M. R. Tanweer, S. Suresh and N. Sundararajan, Self regulating particle swarm optimization algorithm, *Inf. Sci.* **294** (2015), 182–202.
[42] M. R. Tanweer, R. Auditya, S. Suresh, N. Sundararajan and N. Srikanth, Directionally driven self-regulating particle swarm optimization algorithm, *Swarm Evol. Comput.* **28** (2016), 98–116.
[43] M. R. Tanweer, S. Suresh and N. Sundararajan, Dynamic mentoring and self-regulation based particle swarm optimization algorithm for solving complex real-world optimization problems, *Inf. Sci.* **326** (2016), 1–24.
[44] R. Thangaraj, M. Pant, A. Abraham and P. Bouvry, Particle swarm optimization: hybridization perspectives and experimental illustrations, *Appl. Math. Comput.* **217** (2011), 5208–5226.
[45] C. Voudouris, *Guided Local Search*, University of Esses, UK, Report no. CSM-247, 1995.
[46] H. Wang, Z. Wu, S. Rahnamayan, C. Li, S. Zeng and D. Jiang, Particle swarm optimisation with simple and efficient neighbourhood search strategies, *Int. J. Innovative Comput. Appl.* **3** (2011), 97–104.
[47] H. Wang, Z. Wu, S. Rahnamayan, Y. Liu and M. Ventresca, Enhancing particle swarm optimization using generalized opposition-based learning, *Inf. Sci.* **181** (2011), 4699–4714.
[48] B. Xin, J. Chen, J. Zhang, H. Fang and Z.-H. Peng, Hybridizing differential evolution and particle swarm optimization to design powerful optimizers: a review and taxonomy, *IEEE Trans. Syst. Man Cybern. Part C (Applications and Reviews)* **42** (2012), 744–767.
[49] B. Xin, Y. Wang, L. Chen, T. Cai and W. Chen, A review on hybridization of particle swarm optimization with artificial bee colony, pp. 242–249, Springer International Publishing, Cham, 2017.
[50] Z. Xinchao, A perturbed particle swarm algorithm for numerical optimization, *Appl. Soft Comput.* **10** (2010), 119–124.
[51] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam and M. R. Meybodi, A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization, *Appl. Soft Comput.* **13** (2013), 2144–2158.
[52] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama and Y. Nakanishi, A particle swarm optimization for reactive power and voltage control considering voltage security assessment, *IEEE Trans. Power Syst.* **15** (2000), 1232–1239.
[53] Z.-H. Zhan, J. Zhang, Y. Li and H. S.-H. Chung, Adaptive particle swarm optimization, *IEEE Trans. Syst. Man Cybern. Part B (Cybernetics)* **39** (2009), 1362–1381.
[54] T. Ziyu and Z. Dingxue, A modified particle swarm optimization with an adaptive acceleration coefficients, in: *Asia-Pacific Conference on Information Processing, 2009, APCIP 2009*, 2, pp. 330–332, IEEE, Shenzhen, China, 2009.