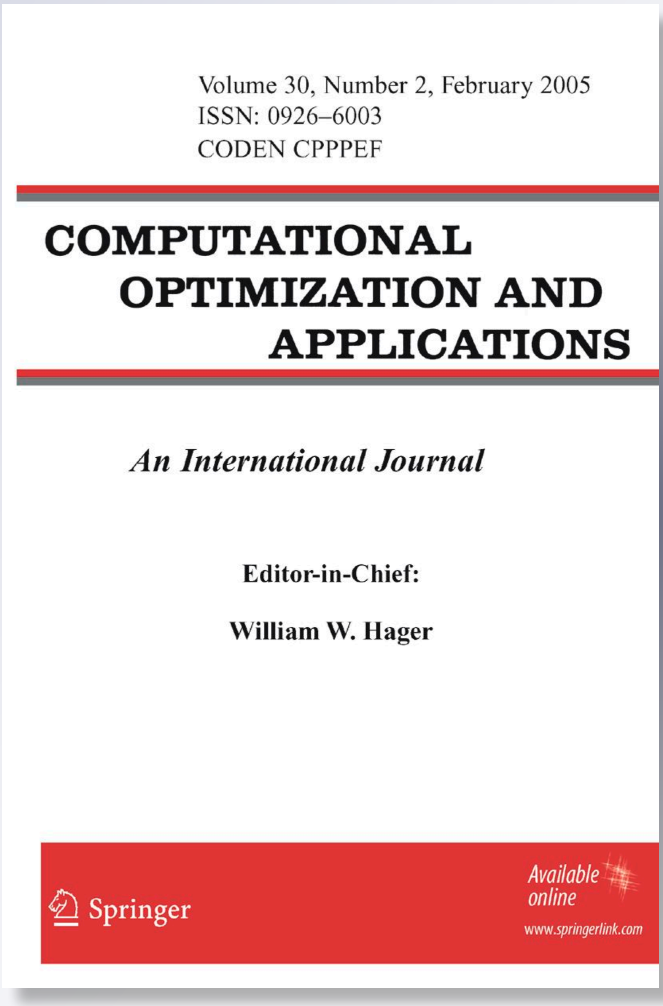
A multi-swarm PSO using charged particles in a partitioned search space for continuous optimization

Abbas El Dor, Maurice Clerc & Patrick Siarry

Computational Optimization and Applications
An International Journal

ISSN 0926-6003
Volume 53
Number 1

Comput Optim Appl (2012) 53:271-295
DOI 10.1007/s10589-011-9449-4



Volume 30, Number 2, February 2005
ISSN: 0926–6003
CODEN CPPPEF

COMPUTATIONAL OPTIMIZATION AND APPLICATIONS

An International Journal

Editor-in-Chief:

William W. Hager

Springer

Available online
www.springerlink.com

Comput Optim Appl (2012) 53:271–295
DOI 10.1007/s10589-011-9449-4

A multi-swarm PSO using charged particles in a partitioned search space for continuous optimization

Abbas El Dor · Maurice Clerc · Patrick Siarry

Received: 28 September 2010 / Published online: 24 November 2011

Abstract Particle swarm optimization (PSO) is characterized by a fast convergence, which can lead the algorithms of this class to stagnate in local optima. In this paper, a variant of the standard PSO algorithm is presented, called PSO-2S, based on several initializations in different zones of the search space, using charged particles. This algorithm uses two kinds of swarms, a main one that gathers the best particles of auxiliary ones, initialized several times. The auxiliary swarms are initialized in different areas, then an electrostatic repulsion heuristic is applied in each area to increase its diversity. We analyse the performance of the proposed approach on a testbed made of unimodal and multimodal test functions with and without coordinate rotation and shift. The Lennard-Jones potential problem is also used. The proposed algorithm is compared to several other PSO algorithms on this benchmark. The obtained results show the efficiency of the proposed algorithm.

Keywords Particle swarm optimization · Tribes-PSO · Global optimization · Multi-swarm · Repulsion heuristic · Partitioned search space

1 Introduction

Particle swarm optimization (PSO), introduced by Kennedy and Eberhart [1] from a simulation of social behavior, is a class of population-based global optimization

A. El Dor · M. Clerc · P. Siarry (✉)
Université de Paris-Est Créteil, LiSSi (E.A. 3956), 61 avenue du Général de Gaulle, 94010 Créteil, France
e-mail: siarry@univ-paris12.fr

A. El Dor
e-mail: abbas.eldor@univ-paris12.fr

M. Clerc
e-mail: maurice.clerc@writeme.com

algorithms for solving complex problems. It uses coordinated particles to explore the search space. Particles correspond to the search points, and each particle is initialized with a random position and a random initial velocity in the search space. The position and the velocity of each particle are updated along with the search process in an intelligent way, based on its own experience and on the experience of its neighbors. This optimization paradigm has become very popular among swarm intelligence-based algorithms.

PSO has the ability to quickly converge, which can lead to a stagnation on a local optimum [2]. In this paper, an improvement of the Standard PSO algorithm is proposed, named PSO-2S, that takes advantage of this fast convergence. Several techniques are used in PSO-2S to produce an improved initial population: charged particles are used to generate the initial population of each auxiliary swarm. The auxiliary swarms are located in different areas of the partitioned search space. This strategy of PSO-2S requires some evaluations at the beginning of the algorithm in order to locate the best particle of each auxiliary swarm, further used to construct the main swarm.

This paper is organized as follows: Sect. 2, the particle swarm optimization paradigm, and several PSO algorithms available in the literature are briefly presented. In Sect. 3, the proposed algorithm is described. In Sect. 4, an experimental analysis of the proposed algorithm and comparisons with several other PSO algorithms are presented. Then, concluding remarks are given in Sect. 5.

2 Particle swarm optimization

Particle swarm optimization (PSO) is a metaheuristic proposed in 1995 by Kennedy and Eberhart [1]. The original PSO algorithm is inspired by the social behavior of biological organisms. PSO is a population-based stochastic approach for solving continuous and discrete optimization problems. In PSO, simple software agents, called particles, move in the search space of an optimization problem. The position of a particle represents a candidate solution of the optimization problem at hand. The particles fly over the search space, keeping in memory the best solution encountered. At each iteration, each particle adjusts its velocity vector, based on its momentum, influences of its best solution and of the best solution of its neighbors, then computes a new point to be evaluated. The displacement of a particle is influenced by three components:

1. *Physical component*: the particle tends to keep its current direction of displacement;
2. *Cognitive component*: the particle tends to move towards the best site that it has explored until now;
3. *Social component*: the particle tends to rely on the experience of its congeners, then moves towards the best site already explored by its neighbors.

2.1 Overview of the standard PSO

In this paper, the swarm size is denoted by s, and the search space is n-dimensional. In general, the particles have three attributes: the current position $X_i = (x_{i,1}, x_{i,2}, \dots, x_{i,n})$, the current velocity vector $V_i = (v_{i,1}, v_{i,2}, \dots, v_{i,n})$ and the past best position $Pbest_i = (p_{i,1}, p_{i,2}, \dots, p_{i,n})$. These attributes are used with the global best position $Gbest = (g_1, g_2, \dots, g_n)$ of the swarm, to update iteratively the state of each particle in the swarm. The objective function to be minimized is denoted by f. The new velocity vector V_i of each particle is updated as follows:

$$\begin{aligned} v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1_{i,j}}(t)\big[pbest_{i,j}(t) - x_{i,j}(t)\big] \\ + c_2 r_{2_{i,j}}(t)\big[gbest_j(t) - x_{i,j}(t)\big] \end{aligned} \tag{1}$$

$v_{i,j}$ is the velocity of the ith particle ($i \in 1, 2, \dots, s$) of the jth dimension ($j \in 1, 2, \dots, n$) where: c_1, c_2 are the learning factors that will be fixed throughout the whole process, called *acceleration coefficients*, r_1, r_2 are two random numbers in the range $[0, 1]$ selected uniformly for each dimension at each iteration, $v_{i,j}(t)$ is the *physical* component, $c_1 r_{1_{i,j}}(t)$ $[pbest_{i,j}(t) - x_{i,j}(t)]$ is the *cognitive* component, where c_1 controls the cognitive behavior of the particle, and $c_2 r_{2_{i,j}}(t)$ $[gbest_j(t) - x_{i,j}(t)]$ is the *social* component, where c_2 controls the social behavior of the particle.

The new position X_i of each particle is calculated as follows:

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \tag{2}$$

$x_{i,j}$ is the position of the ith particle ($i \in 1, 2, \dots, s$) of the jth dimension ($j \in 1, 2, \dots, n$).

In case of minimization of f, the past best position $Pbest_i$ of each particle is updated as follows:

$$Pbest_i(t+1) = \begin{cases} Pbest_i(t), & \text{if } f(X_i(t+1)) \geq Pbest_i(t) \\ X_i(t+1), & \text{otherwise} \end{cases} \tag{3}$$

The global best position *Gbest*, found by the evaluations of the particles during each generation, is defined as:

$$Gbest(t+1) = \arg\min_{Pbest_i} f(Pbest_i(t+1)), \quad 1 \leq i \leq s \tag{4}$$

In the global version of PSO, the best particle *Gbest* is chosen among the whole population. The information graph is completely connected. The information links between the particles are defined only one time, we call this topology a static information topology.

The pseudo code of the original PSO is shown in Algorithm 1.

The velocity clamping is problem-dependent. In the original PSO, the particle velocities v_i can be clamped in the range $[-v_{max}, v_{max}]$ according to the problem at hand. This clamping is used to prevent the particles from moving out of the search space. Changing the values of $c_1 r_1$ and $c_2 r_2$ makes the velocity value increase [3], until it reaches its maximal bound.

2.2 Some improvements of PSO

Due to its simple implementation, minimum mathematical processing and good optimization capability, PSO has attracted a growing interest since its birth in 1995. To improve the performance of this algorithm, Shi and Eberhart [6] modified the original

Algorithm 1 The original Particle Swarm Optimization (PSO)

```
 1: Initialize randomly the position $x$ and the velocity $v$ of each particle : set $iteration = 1$.
 2: for $i = 1$ to $s$ do
 3:     $Pbest_i = X_i$
 4: end for
 5: while the stopping criterion is not satisfied do
 6:     for $i = 1$ to $s$ do
 7:         update position $X_i$ using (1) and (2)
 8:         calculate particle fitness $f(X_i)$
 9:         update $Pbest_i$, $Gbest$ using (3) and (4)
10:     end for
11:     set $iteration = iteration + 1$
12: end while
```

PSO by introducing an inertia weight w into (1), to control the influence of the direction of the particle on its future displacement. The new velocity update is calculated as follows:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1_{i,j}}(t)\left[pbest_{i,j}(t) - x_{i,j}(t)\right] + c_2 r_{2_{i,j}}(t)\left[gbest_j(t) - x_{i,j}(t)\right] \tag{5}$$

The goal of w is to balance global (wide-ranging) and local (nearby) exploration abilities of the particles. The intensity of the exploration of the search space depends on the inertia weight value. While a larger inertia weight w facilitates global exploration, a smaller one facilitates local exploration. The inertia weight attracted a lot of interest from the research community [4, 5]. In some applications, w is determined dynamically. In [2], the authors used an inertia weight starting with a value close to 0.9 and linearly decreasing to 0.4 throughout the course of the run. This strategy greatly improved the performance of PSO for several functions. Chatterjee and Siarry in [7] suggested another dynamic inertia weight.

In [8, 9], another work is proposed which is often mentioned in PSO literature. In this work, a new way of balancing global and local searches, known as constriction factor ψ, was derived from the existing constants in the velocity update equation:

$$\psi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \quad \varphi = c_1 + c_2 \text{ and } \varphi < 4 \tag{6}$$

This constriction factor is applied to the velocity update equation as follows:

$$v_{i,j}(t+1) = \psi\left(v_{i,j}(t) + c_1 r_{1_{i,j}}(t)\left[pbest_{i,j}(t) - x_{i,j}(t)\right] + c_2 r_{2_{i,j}}(t)\left[gbest_j(t) - x_{i,j}(t)\right]\right) \tag{7}$$

Clerc indicated in [9] that the use of a constriction factor may be necessary to insure convergence of the particle swarm algorithm. His research has shown that the inclusion of properly defined constriction coefficients increases the rate of convergence; further, these coefficients can prevent explosion and induce particles to converge on local optima. A comparison is performed in [5] between the influences of the inertia weight and the constriction factor.

Kennedy has proposed in [10] to use a local variable *lbest* instead of the global variable *gbest* in (1), it allows the particles to escape the local optima, but slows the convergence of the PSO. While the *lbest* swarm model uses a local topology (for example, ring topology), the *gbest* model uses a global topology (fully connected). At first, the *lbest* model using the original PSO shows inferior performance than the *gbest* model. However, Kennedy and Mendes in [11] indicate that the *lbest* model returns improved results when used in conjunction with other algorithm improvements.

Other techniques were proposed to improve the performance of PSO. These methods are hybridizations of PSO with other search techniques. Evolutionary operators, such as selection, crossover, and mutation, have been introduced to keep the best solution [12]. This technique copies the current positions and velocities of the best half of the population, to replace the worst half without changing the best values of any of the particles in the current step. Nakano et al. [13] propose a PSO approach based on the concept of tabu search. The population is divided into two swarms in this algorithm. The first swarm deals with intensification and the second one deals with diversification.

Other interesting PSO variants proposed in the literature are multi-swarm algorithms. An algorithm proposed in [14] is based on a Master-Slave model, in which a population consists of one master swarm and several slave swarms. The slave swarms execute a single PSO independently to maintain the diversity of the particles, while the master swarm evolves based on its own knowledge and on the knowledge of the slave swarms.

The algorithm, called Efficient Population Utilization Strategy for Particle Swarm Optimizer (EPUS-PSO), presented in [15], is based on a variable size of population. The authors propose three main ideas to improve the efficiency of PSO. The first one introduces a population manager. Depending on the status of the search, the population manager adds or deletes some particles. This dynamicity increases the ability of finding the global optimum. The second idea is based on a solution-sharing strategy, that allows best particles to share their information, and update their velocity. The third idea concerns the *searching range sharing* (SRS). This strategy prevents the particles from falling into local optima. Furthermore, SRS extends the exploration of the search space to unexplored zones.

A new cooperative approach to Particle Swarm Optimization (CPSO), presented in [16], employs cooperative behavior to achieve a good performance, by using multiple swarms to optimize different components of the solution vector. To improve this approach, the authors divided the initial swarm into sub-swarms to cover the n-dimensional search space. This algorithm is called CPSO-S_k. In the particular case of $n = k$, the algorithm uses one dimensional swarms to search each dimension separately, and it is called CPSO-S. A hybrid approach from the CPSO family, called CPSO-H, uses also one dimensional searches. It alternates between performing an iteration of CPSO-S and an iteration of the standard PSO algorithm.

The Comprehensive Learning Particle Swarm Optimizer (CLPSO) [17] is a variant of PSOs, which uses a novel learning strategy to update the particle's velocity according to its historical information.

Unified Particle Swarm Optimization (UPSO) [18] is a new scheme that aggregates the global and local variants of PSO, by combining their exploration and exploitation capabilities.

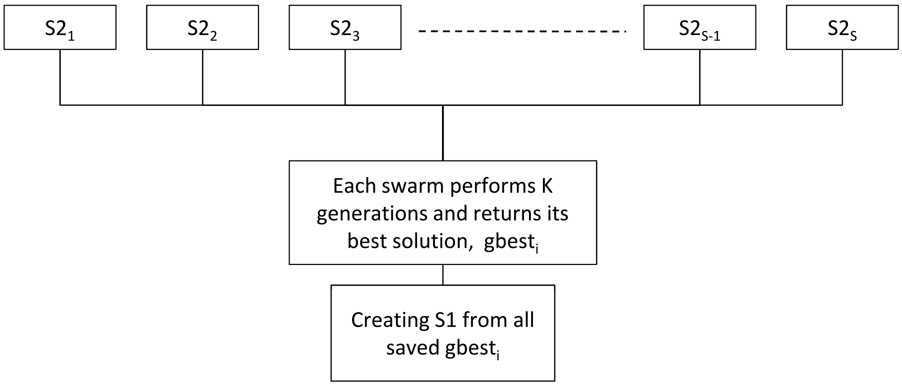

S2_1
S2_2
S2_3
S2_S-1
S2_S
Each swarm performs K generations and returns its best solution, gbest_i
Creating S1 from all saved gbest_i

Fig. 1 Creating swarm S1 by gathering all g_{besti} of S2_i

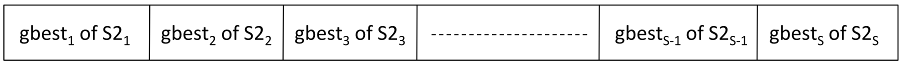


Fig. 2 The initial state of the main swarm S1

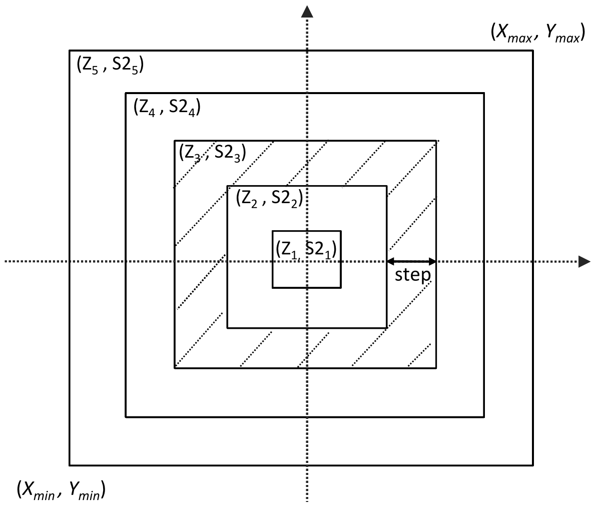
In [19], the authors propose an improved PSO based on distributed sub-swarms. These sub-swarms are gathered periodically to share their information. Then, the particles are randomly reinitialized in the search space.

Clerc propose in [20] a novel algorithm (TRIBES) that can be used as a black box [21]. In this algorithm, the swarm is divided into several sub-swarms, called tribes. Intra-tribe communications take place in each of them. For each sub-swarm, a "leader" particle can communicate with other sub-swarm leaders using inter-tribes communications.

3 The proposed method

The fast convergence of PSO algorithms can lead to a stagnation on a local optimum [2]. Thus, we need a way to increase the quality and the diversity of the particles, without decreasing the convergence speed. To do so, we propose a new PSO algorithm, called PSO-2S. This algorithm is based on Standard PSO (specifically, SPSO2006 [22] that uses a quasi-random local best topology, modified after each iteration in which the best solution has not been improved), with a new initialization strategy. This strategy is based on the use of two kinds of swarms: the main swarm denoted by S1, and s auxiliary ones denoted by S2_i, where $1 \leq i \leq s$ and s is the number of auxiliary swarms. To construct S1, we propose to perform a random initialization of each S2_i $(i = 1, \ldots, s)$. After each of these initializations, K displacements of the particles are performed, then the best solution of each S2_i is added to S1. These repeated initializations in unexplored zones of the search space lead to a more diversified initial swarm, with particles of higher quality. The particles of S2_i are moving during the K generations according to (2) and (5). The different steps of this initialization process are depicted in Fig. 1. Figure 2 presents the initial particle

Fig. 3 Partitioning of the search space



positions of the main swarm S1, which are obtained at the end of this initialization process.

When the initialization phase of the proposed algorithm is completed, $S2_i$ are no longer used. Then, S1 begins to explore the search space, in the same way that the Standard PSO algorithm, until a stopping criterion is satisfied. Two stopping criteria are used in this paper: the algorithm stops when a maximum number of evaluations is reached or when the fitness of the best solution found, denoted by *gbest*, becomes lower than a predefined threshold.

To enhance the performance of the algorithm, the search space is partitioned into several zones. Then, each initialization of $S2_i$ is performed in one of these zones. This partitioning allows a best covering of the search space, thus leading to a better exploration in each zone separately.

In order to partition the search space $[min, max]^N$ into max_{zone} zones, we define the center of the search space as a starting point of the partitioning process, according to (8). Then, we compute a *step* to determine the boundaries of each zone, as in (9). This process is illustrated in Fig. 3, where the ith swarm $S2_i$ and its attributed zone Z_i are denoted by $(Z_i, S2_i)$.

$$center = (max - min)/2 \tag{8}$$

$$step = center/max_{zone} \tag{9}$$

Each zone $[zone_{i\,min}, zone_{i\,max}]^N$ is then determined according to (10) and (11), where $num_{zone} = 1, 2, \ldots, max_{zone}$, is the current zone number.

$$zone_{i\,min} = center - num_{zone} * step \tag{10}$$

$$zone_{i\,max} = center + num_{zone} * step \tag{11}$$

The size of the zones of the partitioned search space are different ($Z_1 < Z_2 < \cdots < Z_{max_{zone}}$). Therefore, the number of particles in $S2_i$, denoted by $S2_{i\,size}$, depends on its corresponding zone size. Indeed, a small zone takes less particles and

Algorithm 2 Initialization procedure in each zone of the search space

```
1: Input: $max$, $min$, $max_{zone}$.
2: $center \leftarrow (max - min)/2$, $step \leftarrow center/max_{zone}$;
3: if ($p = 1$) then
4:     for $s = 1$ to $S2_p.size$ do
5:         $S2_p.X[s] \leftarrow alea\_X([center - step, center + step]^N)$
6:         $S2_p.V[s] \leftarrow alea\_V([center - step, center + step]^N)$
7:     end for
8: else
9:     for $s = 1$ to $S2_p.size$ do
10:        do
11:            $S2_p.X[s] \leftarrow alea\_X([center - step * p, center + step * p]^N)$
12:        while (isInside($S2_p.X[s]$, $p - 1$))
13:    end for
14:    for $s = 1$ to $S2_p.size$ do
15:        do
16:            $S2_p.V[s] \leftarrow alea\_V([center - step * p, center + step * p]^N)$
17:        while (isInside($S2_p.V[s]$, $p - 1$))
18:    end for
19:    return $S2_p$
20: end if
```

the number of particles increases when the zone becomes larger. The size of each auxiliary swarm is calculated as follows:

$$S2_{i\ size} = num_{zone} * nb_{\text{particle}} \tag{12}$$

where $num_{zone} = 1, 2, \ldots, max_{zone}$, is the current zone number and $nb_{particle}$ is a fixed value.

To avoid an overflowing number of particles, when using a high number of zones, $nb_{particle}$ must be decreased when max_{zone} increases.

The initialization procedure in each zone is presented in Algorithm 2.

To widely cover the search space, the concept of repulsion is applied [23]. This technique has been used in an agent-based optimization algorithm for dynamic environments [24], to generate an initial population of agents. At first, we initialize all the particles randomly in different zones and each particle is considered as an electron. Then, a force of $1/r^2$, where r is the distance between two particles, is applied on the particles of each zone, until the maximum displacement of a particle during an iteration, denoted by max_disp, becomes lower than a given threshold ϵ. In a normalized search space ($[0, 1]^N$, where N is the dimension of the search space), this threshold is typically equal to 10^{-4}. This value of the threshold ϵ is used several times in the literature, especially in [24]. Figure 4 presents an example of the repulsion applied to $(Z_3,\ S2_3)$.

This repulsion heuristic is presented in Algorithm 3. During each iteration of the heuristic, the repulsion force $\mathbf{D}$ applied on a particle $\mathbf{P}_i$ by the other particles is calculated. Then, the coefficient $step$ is increased or decreased in order to maximize the displacement ($step.\mathbf{D}$) of $\mathbf{P}_i$, under the constraint that the energy e of the system decreases after this displacement.

In Algorithm 3, the function $crop$ is used to center the repelled points in the current zone of the search space. For a given point, it returns the nearest point on the boundary

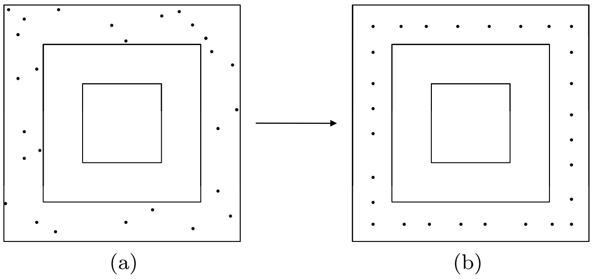


Fig. 4 Repulsion process: (**a**) $(Z_3, S2_3)$ before repulsion, (**b**) $(Z_3, S2_3)$ after repulsion

Algorithm 3 Repulsion heuristic

```
1:  $\epsilon \leftarrow 10^{-4}$
2:  $step \leftarrow 1$
3:  do
4:      $max\_disp \leftarrow 0$
5:      for each particle $\mathbf{P_i}$ do
6:          $\mathbf{D} \leftarrow \sum_{j \neq i}(\frac{\mathbf{P_i}-\mathbf{P_j}}{\|\mathbf{P_i}-\mathbf{P_j}\|^3})$
7:          $e \leftarrow \sum_{j \neq i}(\frac{1}{\|\mathbf{P_i}-\mathbf{P_j}\|^2})$
8:          do
9:              $\mathbf{p_i'} \leftarrow crop(\mathbf{p_i} + step.\mathbf{D})$
10:             if $(e < \sum_{j \neq i}(\frac{1}{\|\mathbf{P_i'}-\mathbf{P_j}\|^2}))$ then
11:                 $step \leftarrow \frac{step}{2}$
12:             else
13:                 $step \leftarrow step \times 2$
14:             end if
15:         while $step$ can still be adapted and $e < \sum_{j \neq i}(\frac{1}{\|\mathbf{P_i'}-\mathbf{P_j}\|^2})$
16:         if $\|\mathbf{P_i} - \mathbf{P_i'}\| > max\_disp$ then
17:             $max\_disp \leftarrow \|\mathbf{P_i} - \mathbf{P_i'}\|$
18:         end if
19:         $\mathbf{P_i} \leftarrow \mathbf{P_i'}$
20:     end for
21: while $(max\_disp > \epsilon)$
```

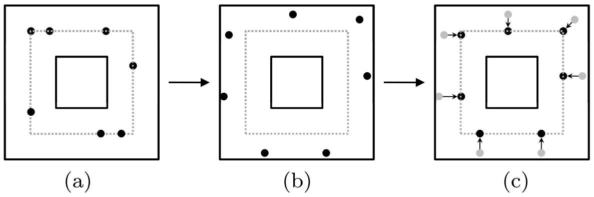


Fig. 5 The *crop* function: (**a**) the auxiliary swarm at the nth step of the repulsion heuristic after applying the *crop* function, (**b**) the auxiliary swarm at the $(n+1)$th step after repulsion, (**c**) the auxiliary swarm at the $(n+1)$th step after applying the *crop* function

of the hyper-square centered inside the current zone of the search space. This function is illustrated in Fig. 5.

The main procedure of PSO-2S algorithm is presented in Algorithm 4.

Algorithm 4 PSO-2S

```
1: Input: $max_{zone}$, $nb_{particle}$, $nb_{generation}$.
2: $S1.size \leftarrow max_{zone}$;
3: for $p = 1$ to $max_{zone}$ do
4:     $S2_p.size \leftarrow nb_{part} * p$
5:     $S2_p \leftarrow$ init_swarm($S2_p.size$, $p$)
6:     repulsion_particle($S2_p$, $p$)
7:     for $k = 1$ to $nb_{generation}$ do
8:         for $i = 1$ to $S2_p.size$ do
9:             $V_i \leftarrow \omega V_i + \rho_1(Xpbest_i - X_i) + \rho_2(Xgbest_p - X_i)$
10:            $X_i \leftarrow X_i + V_i$
11:            if $f(X_i) < Pbest_i$ then
12:                $Pbest_i \leftarrow f(X_i)$
13:                $Xpbest_i \leftarrow X_i$
14:            end if
15:            if $Pbest_i < gbest_p$ then
16:                $gbest_p \leftarrow pbest_i$
17:                $Xgbest[p] \leftarrow X_i$
18:            end if
19:        end for
20:    end for
21:    $S1.X[p] \leftarrow Xgbest[p]$
22: end for
23: while the stopping criterion is not satisfied do
24:    for $i = 1$ to $max_{zone}$ do
25:        update velocity $V_i$ and position $X_i$ using (5) and (2)
26:        update $Pbest_i$, $Gbest$ of swarm $S_1$ using (3) and (4)
27:    end for
28: end while
```

4 Experiments and discussion

4.1 Benchmark functions

In this section, we will present the benchmark functions used in our experiments [25, 26]. These functions are classified into four main groups according to their properties. These functions are used to compare the results obtained by PSO-2S with the results from other PSO algorithms available in the literature. The four groups are classified as follows:

Group A: Unimodal problem

1. Sphere function

$$f_1(x) = \sum_{i=1}^{n} x_i^2 \tag{13}$$

2. Quadric function

$$f_2(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{14}$$

Group B: Unrotated multimodal problems

3. Rosenbrock's function

$$f_3(x) = \sum_{i=1}^{n-1}\big(100\big(x_i^2 - x_{i+1}\big)^2 + (x_i - 1)^2\big) \tag{15}$$

4. Ackley's function

$$f_4(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{16}$$

5. Rastrigin's function

$$f_5(x) = \sum_{i=1}^{n}\big(x_i^2 - 10\cos(2\pi x_i) + 10\big) \tag{17}$$

6. Weierstrass's function

$$f_6(x) = \sum_{i=1}^{n}\left(\sum_{k=0}^{k\,max}\big[a^k\cos\big(2\pi b^k(x_i + 0.5)\big)\big]\right) - D\sum_{k=0}^{k\,max}\big[a^k\cos\big(2\pi b^k.0.5\big)\big]$$

$$a = 0.5,\ b = 3,\ k\,max = 20 \tag{18}$$

7. Generalized penalized function

$$f_7(x) = 0.1\left\{\begin{array}{l}\sin^2(3\pi x_1)\\ \quad + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})]\\ \quad + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\end{array}\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4) \tag{19}$$

with:

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a\\ 0, & -a \le x_i \le a\\ k(-x_i - a)^m, & x_i < -a\end{cases}$$

8. Griewank's function

$$f_8(x) = \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{20}$$

9. Tripod's function

$$f_9(x) = \left\{\begin{array}{l}\frac{1-\text{sign}(x_2)}{2}(|x_1| + |x_2 + 50|)\\ \quad + \frac{1+\text{sign}(x_2)}{2}\frac{1-\text{sign}(x_1)}{2}(1 + |x_1 + 50| + |x_2 - 50|)\\ \quad + \frac{1+\text{sign}(x_1)}{2}(2 + |x_1 - 50| + |x_2 - 50|)\end{array}\right. \tag{21}$$

with:

$$\begin{cases} \mathrm{sign}(x) = -1 & \text{if } x \le 0 \\ \mathrm{sign}(x) = 1 & \text{else.} \end{cases}$$

The last function is not so difficult, but it may be deceptive for algorithms that are easily trapped into local minima.

Group C: Rotated problems

To rotate a test function, Salomon's method [27] generates an orthogonal matrix M to rotate the benchmark functions, then we get the variable $y = M * x$ to calculate the fitness value f, where x is the original variable. When some test functions can be solved using N one-dimensional searches, where N is the dimension of the problem, the rotated functions have a higher complexity, and can no longer be solved by optimizing on each dimension independently. The rotated functions are defined as follows:

10. Rotated Quadric function

$$f_{10}(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} y_j \right)^2 \tag{22}$$

11. Rotated Rosenbrock's function

$$f_{11}(x) = \sum_{i=1}^{n-1} \left(100\left(y_i^2 - y_{i+1}\right)^2 + (y_i - 1)^2\right) \tag{23}$$

12. Rotated Ackley's function

$$f_{12}(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi y_i)\right) + 20 + e \tag{24}$$

13. Rotated Rastrigin's function

$$f_{13}(x) = \sum_{i=1}^{n} \left(y_i^2 - 10\cos(2\pi y_i) + 10\right) \tag{25}$$

14. Rotated Weierstrass's function

$$f_{14}(x) = \sum_{i=1}^{n} \left(\sum_{k=0}^{k\,max} \left[a^k \cos\left(2\pi b^k (y_i + 0.5)\right)\right] \right) - D \sum_{k=0}^{k\,max} \left[a^k \cos\left(2\pi b^k .0.5\right)\right]$$
$$a = 0.5,\ b = 3,\ k\,max = 20 \tag{26}$$

15. Rotated Generalized penalized function

$$f_{15}(x) = 0.1 \left\{ \begin{array}{l} \sin^2(3\pi y_1) \\ \quad + \sum_{i=1}^{n-1} (y_i - 1)^2 [1 + \sin^2(3\pi y_{i+1})] \\ \quad + (y_n - 1)^2 [1 + \sin^2(2\pi y_n)] \end{array} \right\} + \sum_{i=1}^{n} u(y_i, 5, 100, 4) \tag{27}$$

with:

$$u(y_i, a, k, m) = \begin{cases} k(y_i - a)^m, & y_i > a \\ 0, & -a \le y_i \le a \\ k(-y_i - a)^m, & y_i < -a \end{cases}$$

Group D: Shifted multimodal problems

In this group of test functions, we shift the coordinates by an offset vector $O = (o_1, o_2, \dots, o_n)$, and add f_{bias} to the functions. Thus the global optimum is translated, and it is expected that this operation makes the problem more difficult to solve. PSO algorithms could not reduce too quickly the search space area in such shifted functions. The used offset vectors are taken from the Proceedings of the Conference CEC 2005. Examples of this group are listed below:

16. Shifted Rosenbrock's function

$$f_{16}(x) = \sum_{i=1}^{n-1} \left(100\left(z_i^2 - z_{i+1}\right)^2 + (z_i - 1)^2\right) + f_{bias_1}$$
$$z_i = x_i - o_{1i} \tag{28}$$

17. Shifted Ackley's function

$$f_{17}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi z_i)\right) + 20 + e + f_{bias_2}$$
$$z_i = x_i - o_{2i} \tag{29}$$

18. Shifted Rastrigin's function

$$f_{18}(x) = \sum_{i=1}^{n}\left(z_i^2 - 10\cos(2\pi z_i) + 10\right) + f_{bias_3}$$
$$z_i = x_i - o_{3i} \tag{30}$$

19. Shifted Griewank's function

$$f_{19}(x) = \sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f_{bias_4}$$
$$z_i = x_i - o_{4i} \tag{31}$$

Group E: Shifted unimodal problem

20. Shifted Sphere function

$$f_{20}(x) = \sum_{i=1}^{n} x_i^2 + f_{bias_5}$$
$$z_i = x_i - o_{5i} \tag{32}$$

Table 1 Parameters of the benchmark functions

f	Global optimum x^*	$f(x^*)$	Search space
f_1	$(0, 0, \ldots, 0)$	0	$[-100, 100]^N$
f_2	$(0, 0, \ldots, 0)$	0	$[-100, 100]^N$
f_3	$(1, 1, \ldots, 1)$	0	$[-2.048, 2.048]^N$
f_4	$(0, 0, \ldots, 0)$	0	$[-32.768, 32.768]^N$
f_5	$(0, 0, \ldots, 0)$	0	$[-5.12, 5.12]^N$
f_6	$(0, 0, \ldots, 0)$	0	$[-0.5, 0.5]^N$
f_7	$(1, 1, \ldots, 1)$	0	$[-50, 50]^N$
f_8	$(0, 0, \ldots, 0)$	0	$[-600, 600]^N$
f_9	$(0, 50)$	0	$[-100, 100]^2$
f_{10}	$(0, 0, \ldots, 0)$	0	$[-100, 100]^N$
f_{11}	$(1, 1, \ldots, 1)$	0	$[-2.048, 2.048]^N$
f_{12}	$(0, 0, \ldots, 0)$	0	$[-32.768, 32.768]^N$
f_{13}	$(0, 0, \ldots, 0)$	0	$[-5.12, 5.12]^N$
f_{14}	$(0, 0, \ldots, 0)$	0	$[-0.5, 0.5]^N$
f_{15}	$(1, 1, \ldots, 1)$	0	$[-50, 50]^N$
f_{16}	$(o_{11}, o_{12}, \ldots, o_{1n})$	390	$[-100, 100]^N$
f_{17}	$(o_{21}, o_{22}, \ldots, o_{2n})$	−140	$[-32.768, 32.768]^N$
f_{18}	$(o_{31}, o_{32}, \ldots, o_{3n})$	−330	$[-5.12, 5.12]^N$
f_{19}	$(o_{41}, o_{42}, \ldots, o_{4n})$	−180	$[-600, 600]^N$
f_{20}	$(o_{51}, o_{52}, \ldots, o_{5n})$	−450	$[-100, 100]^N$
V_8	$(x_1, x_2, x_3, \ldots, x_{24})$	−19.821489	$[-2, 2]^{24}$
V_9	$(y_1, y_2, y_2, \ldots, y_{27})$	−24.113360	$[-2, 2]^{27}$
V_{10}	$(z_1, z_2, z_2, \ldots, z_{30})$	−28.422532	$[-2, 2]^{30}$

The Lennard-Jones atomic cluster problem

The Lennard-Jones (LJ) problem is the simplest form of the molecular conformation problem, which consists of finding a configuration of atoms in a cluster or molecule whose potential energy is minimum [28].

We present the LJ problem as follows: we denote by K the number of clusters of atoms, $x^i = (x^i_1, x^i_2, x^i_3)$ represents the coordinates of atom i in the three-dimensional space, and let be $X = ((x^1), \ldots, (x^K))$. The LJ potential energy function $v(r_{ij})$ of a pair of atoms (i, j) is given by $v(r_{ij}) = \frac{1}{r_{ij}^{12}} - \frac{1}{r_{ij}^{6}}$, $1 \leq i, j \leq K$, where $r_{ij} = \|x^i - x^j\|$.

We can notice that, with a single pair of neutral atoms, the LJ potential function is classified as a simple unimodal function. In a cluster with a high number of atoms, the interaction between atoms increases. Then we need to calculate the LJ potential function for each pair of atoms in a cluster. Many local minima are found in the resulting rugged energy landscape. The function to be minimized in this application is as follows:

$$Min\ V_k(x) = \sum_{i<j} v\big(\|x_i - x_j\|\big) = \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} \left(\frac{1}{\|x_i - x_j\|^{12}} - \frac{1}{\|x_i - x_j\|^{6}} \right) \quad (33)$$

The parameters of the benchmark functions, the global optimum x^*, the corresponding fitness value $f(x^*)$, and the search space $[X_{min}, X_{max}]$ are given in Table 1.

The shapes of some functions used in the experiments are shown in Fig. 6.

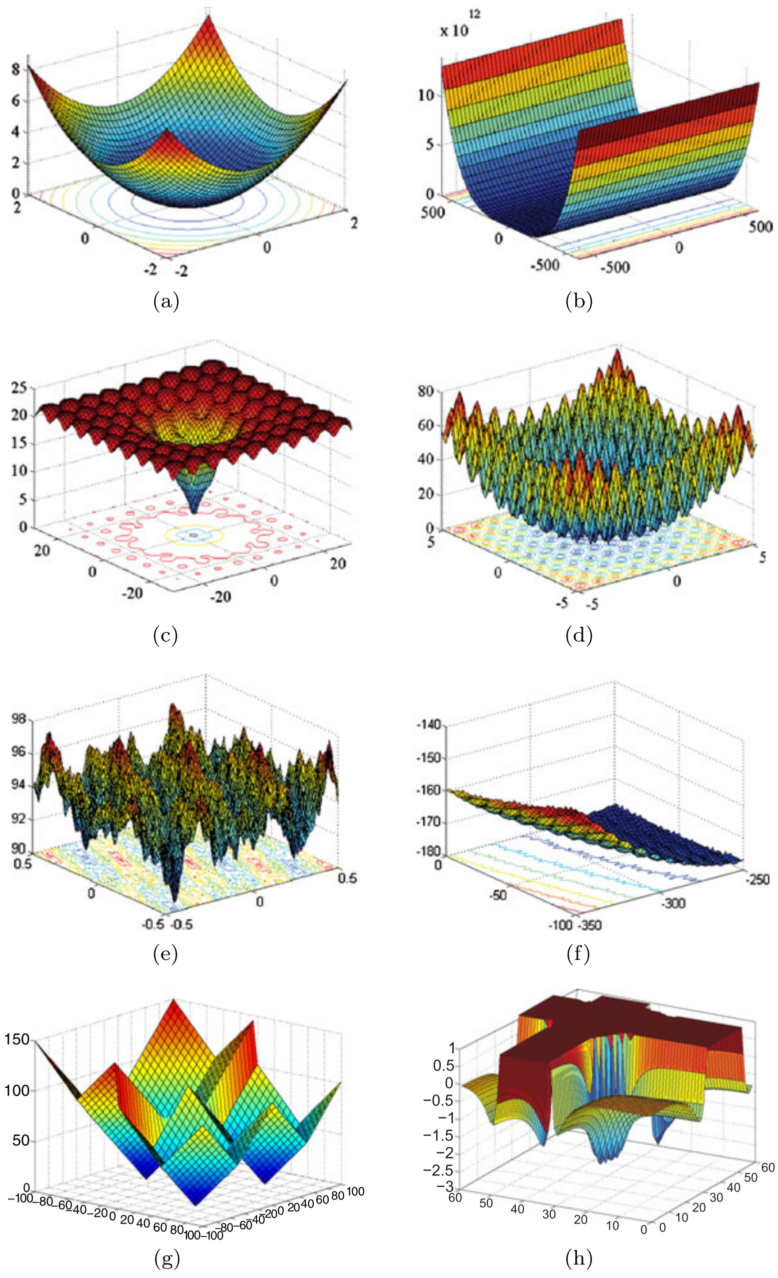


Fig. 6 Shapes of some functions used in the experiments (**a**) Sphere, (**b**) Rosenbrock, (**c**) Ackley, (**d**) Rastrigin, (**e**) Weierstrass, (**f**) Griewank, (**g**) Tripod, (**h**) Lennard-Jones (8 atoms)

Table 2 Parameter settings of the compared PSO algorithms

PSO variants	Inertia weight	Acceleration coefficients
PSO-2S	$w = \frac{1}{2*\ln(2)}$	$c_1 = c_2 = 0.5 + \ln(2)$
EPUS-PSO [15]	$w = \frac{1}{2*\ln(2)}$	$c_1 = c_2 = 0.5 + \ln(2)$
CLPSO [17]	$w(g) = 0.9 - \frac{0.5*g}{max_{gen}}$	$c_1 = c_2 = 1.49$
CPSO-H [16]	$w(g) = 0.9 - \frac{0.5*g}{max_{gen}}$	$c_1 = c_2 = 1.49$
UPSO [18]	$w = 0.72$	$c_1 = c_2 = 1.49$
SPSO2007 [22]	$w = \frac{1}{2*\ln(2)}$	$c_1 = c_2 = 0.5 + \ln(2)$

4.2 Parameter settings and initialization

Experiments were done to compare our proposed algorithm with five PSO algorithms. PSO-2S and SPSO2007 were implemented in C. All other PSO variants were implemented using Matlab 7.1. Their results and their parameter settings are taken from [15]. The PSO-2S and the SPSO2007 use the random *lbest* topology to exchange information between the particles, UPSO [18] uses a combination of *lbest* and *gbest* topologies, and CLPSO [17] uses the comprehensive learning strategy. The learning strategy of CLPSO abandons the global best information, and for all other particles past best information is used to update particles velocity instead.

At first, we compare PSO-2S, EUPSO-PSO [15], CLPSO [17], CPSO-H [16], UPSO [18] and SPSO2007 [22] using the first three groups A, B and C of test functions, without f_8 and f_9 functions. The second comparison was performed to show the significance of using the repulsion heuristic in PSO-2S. Hence, PSO-2S is compared to SPSO2007 [22], one time using the repulsion heuristic, and a second time without using it. The comparison is performed using some functions from the groups A and B, and all shifted functions in the group D.

These PSO algorithms and their parameter settings are listed in Table 2.

The first comparison is done in 10 and 30 dimensions. The maximum number of function evaluations (FEs) is set to 40000 and 150000 for 10-D and 30-D respectively. The parameter of our proposed algorithm, denoted by max_{zone}, is set to 20. Hence, the size of the main swarm S1 is equal to max_{zone}. In 10-D and 30-D, the parameter $nb_{particle}$ is set to 2, and the number K of generations of swarm S2 is set to 5. The population size of SPSO2007 was set to 16 and 20, in 10-D and 30-D, respectively. For all other approaches, the population size was set to 20 in both 10-D and 30-D. In these experiments, each algorithm was run 30 times and its mean value and standard deviation were calculated. The execution time of the 30 runs was saved for PSO-2S and SPSO2007.

For the second comparison, we compare the results in terms of *Success rate*, *Mean best value* and *Minimal error*. The *Acceptable error* and the maximum FEs depend on the problem used. Table 3 gives the settings of the *Search space*, the *Acceptable error* and the *Max FEs* parameters used in this comparison for each test function. Unlike the first comparison, each algorithm is run 100 times.

Table 3 Parameters of the test functions used in the second comparison

f	Function	Search space	Acceptable error	Max. FEs
f_3	*Rosenbrock*	$[-10, 10]^{30}$	0.0001	40000
f_4	*Ackley*	$[-32, 32]^{30}$	0.0001	40000
f_5	*Rastrigin*	$[-10, 10]^{30}$	0.0001	40000
f_8	*Griewank*	$[-100, 100]^{30}$	0.0001	40000
f_9	*Tripod*	$[-100, 100]^{2}$	0.0001	40000
f_{16}	*Shifted Rosenbrock*	$[-100, 100]^{10}$	0.01	100000
f_{17}	*Shifted Ackley*	$[-32, 32]^{30}$	0.0001	100000
f_{18}	*Shifted Rastrigin*	$[-5, 5]^{30}$	0.0001	100000
f_{19}	*Shifted Griewank*	$[-600, 600]^{30}$	0.0001	100000
f_{20}	*Shifted Sphere*	$[-100, 100]^{30}$	0.0001	100000

Table 4 Computational complexity analysis of PSO-2S

Process	Complexity
Total number of evaluations at the initialization	$O(K\, max_{zone}^2\, nb_{particle})$
Number of generations at the initialization	$O(K\, max_{zone}\, nb_{particle})$
Number of generations left after initialization	$O(\frac{max_{eval}}{max_{zone}} - K\, max_{zone}\, nb_{particle})$
Random generation of the particles	$O(N\, max_{zone})$
Repulsion heuristic of the particles	$O((\log(\frac{1}{\epsilon}))^2 N\, max_{zone}^2)$
One generation of PSO	$O(max_{zone}^2 + N\, max_{zone})$
PSO-2S at the initialization step	$O\begin{pmatrix}(K\, max_{zone}\, nb_{particle}) \\ \times (max_{zone}^2 + N\, max_{zone}) \\ + (\log(\frac{1}{\epsilon}))^2 N\, max_{zone}^2\end{pmatrix}$
PSO-2S at the main step	$O\begin{pmatrix}(\frac{max_{eval}}{max_{zone}} - K\, max_{zone}\, nb_{particle}) \\ \times (max_{zone}^2 + N\, max_{zone})\end{pmatrix}$
Total PSO-2S complexity	$O(max_{eval}(max_{zone} + N) + (\log(\frac{1}{\epsilon}))^2 N\, max_{zone}^2)$

4.3 Computational complexity analysis of PSO-2S

The processes performed in PSO-2S are summarized in Table 4, with their computational complexity. The complexity of PSO-2S, calculated by summing the complexities of these processes, is also given. Finally, for a fixed set of parameters of PSO-2S, and for a fixed maximum number of evaluations max_{eval}, we get a worst case computational complexity of $O(N)$ for PSO-2S, where N is the dimension of the problem.

4.4 Sensitivity analysis of PSO-2S

The sensitivity of PSO-2S was studied by varying the value of two of its parameters separately, while the others were left fixed to their default values in Table 2. Each parameter was studied this way, by applying PSO-2S on f_{17} and f_{18} in 30-D. The algorithm was stopped when 75000 evaluations were performed. For both test problems, the *Mean best* and the *Std. dev* were computed from 30 runs of the algorithm.

At first, we studied the sensitivity of the parameter K. $nb_{particle}$ was set to 2 and max_{zone} was set to 20. The parameter K varied from 0 to 25 by steps of 5. Figure 7a and b show the value of the best solution found, in y-axis, for each value of the parameter K, in x-axis. As it can be seen, it is better to perform several generations of the swarm $S2$ to construct the swarm $S1$, in order to obtain a good performance. In f_{17}, we can note that *Mean best* oscillates when K increases, and becomes stable when $K = 10$. While for f_{18} we notice an improvement of *Mean best* when K increases. Hence, K has to be fitted to the problem used. Besides, the standard deviation does not vary significantly with the value of K.

Similarly, we studied the sensitivity of max_{zone} for the same functions. $nb_{particle}$ was still set to 2 and the value of K was fixed to 5. The obtained results show that *Mean best* improves as the value of max_{zone} increases, for the two test functions f_{17} and f_{18}. *Std. dev* decreases as the value of max_{zone} increases, for both test problems. Figure 7c and d show this improvement of *Mean best* and *Std. dev*.

To conclude, one can notice that the influence of max_{zone} on the performance of PSO-2S is more important than the one of the parameter K. Hence, we can set K to a low value (around 5) in order to use a low number of evaluations per zone of the search space. This way, we can use a high number of zones, without increasing the number of evaluations used during the whole initialization process.

4.5 Experimental results

1. *Results on the* 10-*D problems*: Table 5 presents the mean best value and the standard deviation (*mean* $\pm$ *std. dev.*) of 30 runs of six PSO algorithms on the thirteen test functions from the first three groups (A, B and C), using 10 dimensions. The best results among those obtained by the five algorithms are shown in bold.

From these experiments, we can notice that the proposed algorithm obtains the best results on most of the test functions used. The two algorithms that obtain the best performances on these test functions are PSO-2S and EPUS-PSO. PSO-2S outperforms EPUS-PSO on 7 functions among 13, it obtains similar results on 3 functions, and it is outperformed by EPUS-PSO on 3 functions. Besides, we can notice that PSO-2S and EPUS-PSO perform significantly better than the other tested algorithms on f_5, f_7 and f_{15}. On the other hand, the proposed algorithm is better than SPSO2007 for all test functions, except for f_1.

The execution times of PSO-2S and SPSO2007 for 30 runs in 10-D are presented in Table 6. One can notice that the execution time of PSO-2S, using the repulsion heuristic, is still competitive compared to the one of SPSO2007. It can be observed that our algorithm outperforms SPSO2007 for all test functions, except f_6. As it

Table 5 Results on the 10-D problems

f	PSO-2S	EPUS-PSO	CLPSO	CPSO-H	UPSO	SPSO2007
f_1	1.05e–086 ± 3.10e–086	**5.55e–153 ± 2.44e–134**	4.02e–021 ± 6.49e–021	9.01e–010 ± 1.38e–009	5.52e–090 ± 1.18e–089	4.00e–101 ± 1.80e–100
f_2	**6.18e–028 ± 2.13e–027**	9.32e–012 ± 4.16e–011	5.04e+002 ± 1.99e+002	1.62e+003 ± 9.35e+002	7.18e+001 ± 7.63e+001	1.83e–027 ± 4.03e–027
f_3	**2.35e–002 ± 1.56e–002**	9.13e–002 ± 1.05e–001	2.84e+000 ± 1.05e+000	1.08e+000 ± 1.18e+000	2.89e–001 ± 7.36e–002	3.10e–001 ± 9.95e–001
f_4	4.12e–015 ± 1.21e–015	**2.72e–015 ± 1.50e–015**	1.78e–011 ± 1.63e–011	7.02e–006 ± 5.77e–006	3.32e–015 ± 8.86e–016	3.85e–001 ± 2.07e–001
f_5	**0.00e+000 ± 0.00e+000**	**0.00e+000 ± 0.00e+000**	2.35e–009 ± 2.72e–009	3.83e–010 ± 7.36e–010	9.28e+000 ± 2.97e+000	5.11e+000 ± 2.29e+000
f_6	**5.85e–003 ± 2.35e–002**	3.21e–002 ± 8.11e–002	2.06e+000 ± 4.80e–001	4.12e+000 ± 1.72e+000	1.94e+000 ± 1.13e+000	4.04e–002 ± 1.78e–001
f_7	**1.35e–032 ± 5.47e–048**	**1.35e–032 ± 5.47e–048**	4.32e–001 ± 4.10e–001	1.09e+006 ± 2.14e+006	**1.35e–032 ± 5.47e–048**	7.32e–004 ± 2.74e–003
f_{10}	**3.38e–009 ± 9.02e–009**	3.99e–007 ± 1.63e–006	8.27e+002 ± 3.35e+002	2.15e+003 ± 1.19e+003	1.01e+002 ± 9.50e+001	1.02e–007 ± 4.12e–007
f_{11}	**4.21e–001 ± 7.77e–002**	8.96e–001 ± 2.15e+000	6.44e+000 ± 4.56e+000	3.01e+000 ± 2.52e+000	1.11e+000 ± 1.18e+000	5.73e–001 ± 1.05e+000
f_{12}	1.56e+001 ± 2.28e+000	**2.61e–015 ± 1.57e–015**	8.45e–008 ± 2.97e–007	1.31e+000 ± 9.68e–001	7.70e–002 ± 2.88e–001	2.03e+001 ± 8.54e–002
f_{13}	**2.65e–001 ± 9.93e–001**	7.26e+000 ± 4.31e+000	8.85e+000 ± 3.07e+000	2.81e+001 ± 1.39e+001	1.18e+001 ± 5.66e+000	1.26e+001 ± 7.00e+000
f_{14}	**6.43e–001 ± 3.45e–001**	1.12e+000 ± 1.36e+000	1.98e+000 ± 5.38e–001	3.48e+000 ± 1.41e+000	1.84e+000 ± 1.02e+000	4.20e+000 ± 1.50e+000
f_{15}	**1.35e–32 ± 5.47e–048**	**1.35e–032 ± 5.47e–048**	4.36e–001 ± 5.55e–001	1.09e+006 ± 2.37e+006	2.77e+000 ± 6.22e+000	1.10e–003 ± 3.30e–003
Performance of PSO-2S		7/10	12/13	12/13	9/12	12/13

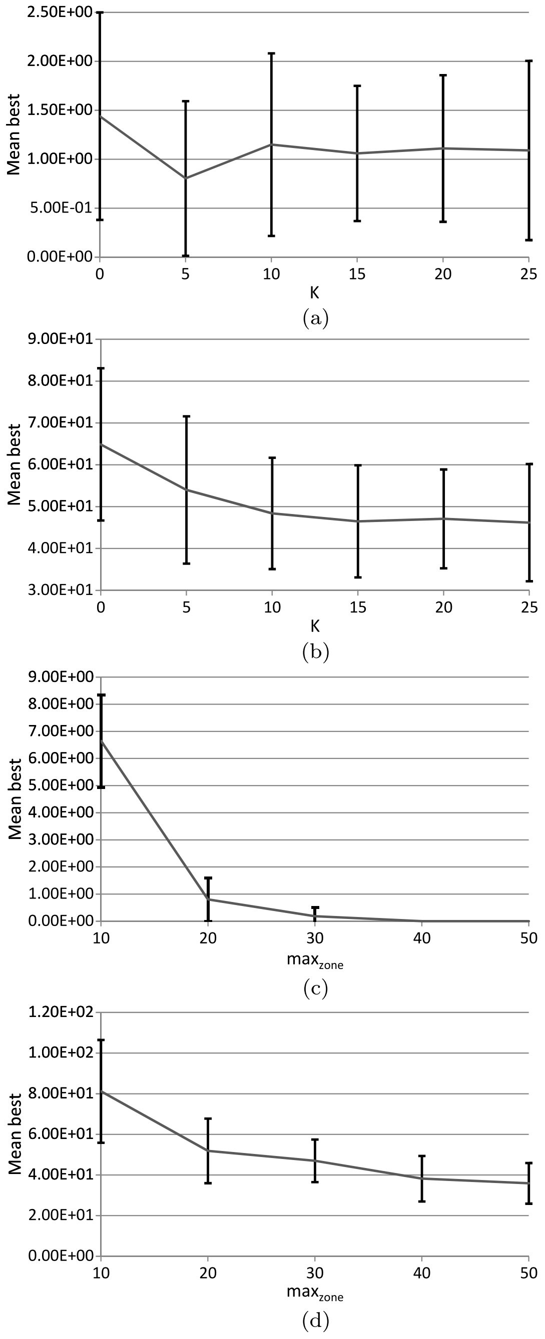


Fig. 7 Sensitivity analysis of PSO-2S: (**a**) the evolution of the parameter K for f_{17}, (**b**) the evolution of the parameter K for f_{18}, (**c**) the evolution of the parameter max_{zone} for f_{17}, (**d**) the evolution of the parameter max_{zone} for f_{18}

was mentioned above, the other competing algorithms were implemented in Matlab, therefore they do not take part in the comparison of execution times.

Table 6 Execution times of 30 runs on the 10-D and 30-D problems for SPSO2007 and PSO-2S (best results are in bold)

Function	PSO-2S	SPSO2007	PSO-2S	SPSO2007
	Run time (10-D)		Run time (30-D)	
f_1	**10.016**	11.259	**38.178**	50.625
f_2	**4.731**	12.559	**36.585**	61.446
f_3	**6.864**	11.042	**39.852**	51.445
f_4	**10.657**	13.356	135.147	**69.378**
f_5	**5.855**	12.801	118.646	**67.110**
f_6	143.526	**106.760**	1472.139	**1038.837**
f_7	**9.250**	16.110	**119.396**	122.355
f_{10}	**5.550**	12.328	**56.818**	80.664
f_{11}	**8.276**	12.057	**64.566**	71.624
f_{12}	**11.947**	14.117	107.005	**88.712**
f_{13}	7.737	13.451	**83**.418	83.819
f_{14}	**103.474**	107.067	1603.414	**1078.324**
f_{15}	**10.224**	16.585	**98.412**	124.757

2. *Results on the* 30*-D problems*: Our experimental results for the 30-D problems are shown in Table 7. We compare the mean best value and the standard deviation of 30 runs of the six PSO algorithms on the same thirteen test functions, as in Table 5.

In the 30-D problems, that are more complicated than the 10-D ones, PSO-2S still obtains good performances. It shows that PSO-S2 scales up well with the problem dimension. As one can see, the results obtained in 30-D are indeed not significantly different than those obtained in 10-D. Moreover, our proposed algorithm outperforms the other PSO algorithms on eight test functions, and EPUS-PSO is still ranked at the second place in the comparison. Especially, PSO-2S significantly outperforms the other algorithms for f_2, f_7 and f_{10}. The proposed algorithm is better than SPSO2007 for all functions, except for f_{12}. One can notice also that CLPSO performs significantly better than the other algorithms for f_{12}.

As for the 10-D problems, we computed the execution times of PSO-2S and SPSO2007 algorithms in 30-D. As shown in Table 6, PSO-2S is still competitive with the standard version of PSO. Even more, our algorithm achieves a similar performance in terms of execution time, compared to SPSO2007.

3. *Comparison of PSO-2S with SPSO*2007: In this section, we will compare our results to the results of SPSO2007, with and without using the repulsion heuristic. These results are performed in terms of *Mean best value*, *Success rate*, *Minimal error* and *Run time* of 100 runs for each algorithm. The parameters of the test functions are presented in Table 3.

The obtained results are given in Table 8. For most of the test functions, our proposed algorithm outperforms SPSO2007, except for f_{19}. The success rates reported in this table show the efficiency of the proposed algorithm.

To ensure the performance of PSO-2S, using the repulsion heuristic, we performed a Wilcoxon-Mann-Whitney test on the results obtained by SPSO2007 and PSO-2S,

Table 7 Results on the 30-D problems

f	PSO-2S	EPUS-PSO	CLPSO	CPSO-H	UPSO	SPSO2007
f_1	3.02e–119 ± 1.56e–118	**8.50e–263 ± 1.02e–262**	1.34e–025 ± 1.71e–025	1.57e–010 ± 2.99e–10	8.79e–123 ± 3.56e–122	3.36e–107 ± 1.79e–106
f_2	**6.18e–011 ± 2.63e–011**	197.e+0000 ± 4.69e+000	2.41e+004 ± 6.78e+003	8.36e+004 ± 6.58e+004	1.07e+004 ± 4.23e+003	8.25e–008 ± 9.00e–008
f_3	**9.75e+000 ± 9.59e–001**	2.55e+001 ± 3.53e–001	2.01e+001 ± 3.37e+000	1.03e+001 ± 1.37e+001	1.31e+001 ± 2.98e+000	1.23e+001 ± 1.47e+000
f_4	1.63e–001 ± 3.85e–001	**3.91e–015 ± 1.07e–015**	9.90e–014 ± 3.80e–014	3.90e–006 ± 5.98e–006	1.33e+000 ± 8.58e–001	1.36e+000 ± 1.13e+000
f_5	1.00e+000 ± 1.18e+001	**0.00e+000 ± 0.00e+000**	1.87e–009 ± 5.34e–009	3.15e–010 ± 1.05e–009	7.63e+001 ± 1.45e+001	4.88e+001 ± 1.44e+001
f_6	**3.63e–001 ± 2.79e–001**	4.08e–001 ± 1.22e+000	1.49e+001 ± 1.69e+000	1.36e+000 ± 3.38e+000	2.06e+000 ± 3.37e+000	1.55e+001 ± 1.09e+000
f_7	**1.35e–032 ± 5.47e–048**	**1.35e–032 ± 5.47e–048**	1.62e+003 ± 5.92e+003	2.58e+008 ± 4.41e+008	1.34e+001 ± 2.34e+001	4.21e–001 ± 1.10e+000
f_{10}	**7.80e–005 ± 9.25e–005**	1.20e+001 ± 2.46e+001	2.38e+004 ± 6.49e+003	3.87e+004 ± 1.71e+004	1.27e+004 ± 4.77e+003	1.09e–001 ± 1.80e–001
f_{11}	**1.39e+001 ± 1.48e–001**	1.96e+001 ± 2.91e+001	3.04e+001 ± 1.32e+001	3.03e+001 ± 2.59e+001	2.15e+001 ± 1.89e+001	2.77e+001 ± 1.31e+001
f_{12}	1.69e+001 ± 1.66e+000	6.81e–001 ± 1.02e+000	**2.22e–006 ± 8.29e–006**	1.52e+000 ± 8.30e–001	1.63e+000 ± 9.49e–001	2.09e+000 ± 1.31e+000
f_{13}	**1.40e+000 ± 2.73e–001**	3.76e+001 ± 1.48e+001	4.70e+001 ± 6.85e+000	9.52e+001 ± 2.93e+001	7.70e+001 ± 1.70e+001	9.04e+001 ± 4.03e+001
f_{14}	**3.34e+000 ± 1.22e+000**	4.26e+000 ± 2.97e+000	1.51e+001 ± 1.93e+000	1.36e+001 ± 3.56e+000	2.02e+001 ± 4.19e+000	3.27e+001 ± 3.19e+000
f_{15}	1.89e–002 ± 2.38e–002	**1.35e–032 ± 5.47e–048**	3.43e+004 ± 1.57e+005	2.44e+008 ± 4.66e+008	2.47e+001 ± 2.77e+001	5.37e+001 ± 2.92e+001
Performance of PSO-2S		7/12	10/13	10/13	11/13	12/13

Table 8 Results of SPSO2007 and PSO-2S

	f	SPSO2007	PSO-2S with repulsion	PSO-2S without repulsion		
		Mean best	Mean best	Mean best	W	P-value
		Min. error	Min. error	Min. error		
		Suc. rate	Suc. rate	Suc. rate		
		Run time	Run time	Run time		
B	f_3	3.15e+001	**2.23e+001**	2.24e+001	6957	**1.714e–06**
		1.82e–001	2.13e+001	2.14e+001		
		0.0%	0.0%	0.0%		
		46.750 s	24.938 s	21.021 s		
	f_4	9.94e–001	**2.03e–001**	7.34e–001	6909	**3.024e–06**
		7.11e–001	8.08e–005	8.69e–005		
		33%	**81%**	47%		
		46.313 s	38.234 s	27.389 s		
	f_5	5.71e+001	**2.00e+000**	7.50e+000	10000	< **2.2e–16**
		2.49e+001	6.55e–005	8.92e–005		
		0.0%	**24%**	5%		
		62.835 s	64.805 s	33.160 s		
	f_8	1.31e–002	**2.93e–003**	4.24e–003	6341.5	**0.001049**
		4.90e–005	5.30e–005	6.19e–005		
		43%	**77%**	61%		
		46.542 s	47.028 s	24.935 s		
	f_9	6.02e–001	**2.45e–003**	2.05e–001	5946	**0.02050**
		3.43e–006	1.79e–005	7.31e–006		
		51%	**98%**	75%		
		19.047 s	9.153 s	4.572 s		
D	f_{16}	2.81e+000	**2.06e+000**	2.89e+000	4070	**0.02293**
		7.26e–003	9.53e–003	9.57e–003		
		76%	73%	**77%**		
		59.282 s	31.202 s	17.615 s		
	f_{17}	1.05e+000	**2.34e–001**	1.27e+000	6135.5	**0.000343**
		7.69e–005	7.96e–005	8.87e–005		
		36%	**63%**	24%		
		108.104 s	197.423 s	150.201 s		
	f_{18}	5.44e+001	**4.32e+001**	5.43e+001	7824	**0.00668**
		2.92e+001	1.39e+001	3.08e+001		
		0.0%	0.0%	0.0%		
		154.163 s	148.035 s	98.578 s		
	f_{19}	**2.36e–002**	3.35e–002	3.78e–002	4618	0.3512
		2.72e–005	7.52e–005	8.03e–005		
		36%	38%	**42%**		
		117.191 s	211.496 s	120.180 s		
E	f_{20}	8.39e–005	**6.23e–005**	9.31e–005	3444	**0.0001442**
		1.88e–005	3.74e–005	7.50e–005		
		100%	**100%**	**100%**		
		12.185 s	28.686 s	10.201 s		

Table 9 Experimental results for LJ problem

$V_k(x)$	SPSO2007			PSO-2S		
	Mean best $\pm$ *std. dev*	*Min. error*	*Suc. rate*	*Mean best* $\pm$ *std. dev*	*Min. error*	*Suc. rate*
V_8	1.17e+000 ± 1.80e+000	0.000048	34%	1.47e+000 ± **9.14e–002**	0.000087	28%
V_9	2.57e+000 ± 2.99e+000	0.000080	14%	2.31e+000 ± **6.06e–002**	0.000095	10%
V_{10}	4.66e+000 ± 4.22e+000	0.000092	2%	1.52e+000 ± **2.28e–001**	0.000000	6%

to determine if their performances are significantly different or not. PSO-2S and SPSO2007 algorithms have been performed 100 times, and the values of the best solutions found at the end of each run were used to calculate the values of W and P*-value*. These values are reported in Table 8 for each function.

The functions for which the value of P*-value* is displayed in bold are those for which PSO-2S is significantly better than SPSO2007. We can conclude that PSO-2S outperforms SPSO2007, especially for f_{19}, with a confidence level of 95% (P*-value* ≤ 0.05).

We have tested the Lennard-Jones problem with 8, 9 and 10 atoms. Known global minimum energy values are -19.821489, -24.113360 and -28.422532 respectively [29]. The *maximum FEs* and the *Acceptable error* were set to 65000 and 0.0001 respectively, for all LJ test problems. The *Mean best* ± *std. dev*, *Minimal error* and *Success rates* are shown in Table 9, for 100 runs of PSO-2S.

According to the *Mean best* and the *Success rate* in Table 9, we cannot determine if our algorithm is better than the other. However, we can see that PSO-2S obtains lower *Std. dev* on the three tests of LJ problem. Hence, we can conclude that PSO-2S is more robust than SPSO2007 in these tests.

5 Conclusion

This paper proposes a novel algorithm called PSO-2S. It is based on multi-swarm, and it uses charged particles in a partitioned search space for solving both unimodal and multimodal problems. Three ideas have been added to the original PSO. The first idea consists in using two swarms, a main one and an auxiliary one. The second idea is the partitioning of the search space into several zones. The last one is to apply a repulsion heuristic on particles to increase the diversity. These ideas have the attractive property of not introducing so many complex operations to the original PSO.

We can observe, from the analysis and experiments, that PSO-2S outperforms the other variants on most of the tested problems. We can conclude that the strategies used in PSO-2S make PSO more robust, and improve significantly its performance.

In conclusion, these new ideas presented above opened the gate to new methods that could be applied to other optimization algorithms, such as evolutionary algorithms.

References

1. Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, pp. 39–43 (1995)
2. Shi, Y., Eberhart, R.C.: Empirical study of particle swarm optimization. In: Proceedings of the 1999 Congress on Evolutionary Computation, pp. 1945–1950 (1999)
3. Eberhart, R.C., Simpson, P.K., Dobbins, R.W.: Evolutionary computation implementations. In: Computational Intelligence PC Tools, pp. 212–226. Academic Press Professional, San Diego (1996)
4. Shi, Y., Eberhart, R.C.: Fuzzy adaptive particle swarm optimization. In: Proceedings of the Congress on Evolutionary Computation, pp. 101–106 (2001)
5. Eberhart, R.C., Shi, Y.: Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the IEEE International Conference on Evolutionary Computation, vol. 1, pp. 84–88 (2000)
6. Shi, Y., Eberhart, R.C.: A modified particle swarm optimizer. In: Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 69–73 (1998)
7. Chatterjee, A., Siarry, P.: Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. Comput. Oper. Res. **33**, 859–871 (2006)
8. Clerc, M., Kennedy, J.: The particle swarm: explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. Evol. Comput. **6**, 58–73 (2002)
9. Clerc, M.: The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1951–1957 (1999)
10. Kennedy, J.: Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of the Congress on Evolutionary Computation, pp. 1931–1938 (1999)
11. Kennedy, J., Mendes, R.: Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. IEEE Trans. Syst. Man Cybern., Part C, Appl. Rev. **36**(4), 515–519 (2006)
12. Angeline, P.J.: Using selection to improve particle swarm optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 84–89 (1998)
13. Nakano, S., Ishigame, A., Yasuda, K.: Particle swarm optimization based on the concept of tabu search. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 3258–3263 (2007)
14. Niu, B., Zhu, Y., He, X., Wu, H.: MCPSO: a multi-swarm cooperative particle swarm optimizer. Appl. Math. Comput. **185**, 1050–1062 (2007)
15. Hsieh, S.T., Sun, T.Y., Liu, C.C., Tsai, S.J.: Efficient population utilization strategy for particle swarm optimizer. IEEE Trans. Syst. Man Cybern., Part B, Cybern. **39**(2), 444–456 (2009)
16. Van den Bergh, F., Engelbrecht, A.P.: A cooperative approach to particle swarm optimization. IEEE Trans. Evol. Comput. **8**, 225–239 (2004)
17. Liang, J.J., Qin, A.K.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans. Evol. Comput. **10**(3), 281–295 (2006)
18. Parsopoulos, K.E., Vrahatis, M.N.: UPSO: a unified particle swarm optimization scheme. In: Proceedings of the International Conference of Computational Methods in Sciences and Engineering, vol. 1, pp. 868–873 (2004)
19. Jiang, Y., Hu, T., Huang, C.C., Wu, X.: An improved particle swarm optimization algorithm. Appl. Math. Comput. **193**, 231–239 (2007)
20. Clerc, M.: Particle Swarm Optimization. ISTE (International Scientific and Technical Encyclopaedia) (2006)
21. Cooren, Y.: Perfectionnement d'un algorithme adaptatif d'Optimisation par Essaim Particulaire. Applications en génie médical et en électronique. PhD thesis, Université Paris-Est Créteil, France (2008)
22. Particle Swarm Central. http://particleswarm.info
23. Conway, J., Sloane, N.: Sphere Packings, Lattices and Groups. Springer, Berlin (1988), 2nd edn. (1993), 3rd edn. (1999)
24. Lepagnot, J., Nakib, A., Oulhadj, H., Siarry, P.: A new multiagent algorithm for dynamic continuous optimization. Int. J. Appl. Metaheuristic Comput. **1**(1), 16–38 (2010)
25. Tu, Z.G., Yong, L.: A robust stochastic genetic algorithm (StGA) for global numerical optimization. IEEE Trans. Evol. Comput. **8**(5), 456–470 (2004)
26. Yao, X., Liu, Y., Lin, G.M.: Evolutionary programming made faster. IEEE Trans. Evol. Comput. **3**(2), 82–102 (1999)
27. Salomon, R.: Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. Biosystems **39**, 263–278 (1996)
28. Ellen, F.: Global optimization of Lennard-Jones clusters. PhD thesis, McMaster University, Hamilton, Ontario, 26 February 2002
29. Hoare, M.R.: Structure and dynamics of simple microclusters. Adv. Chem. Phys. **40**, 49–135 (1979)