A Review of Particle Swarm Optimization Methods used for Multimodal Optimization

Julio Barrera and Carlos A. Coello Coello*

CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07360, MEXICO
julio.barrera@gmail.com, ccoello@cs.cinvestav.mx

Abstract. Particle swarm optimization (PSO) is a metaheuristic inspired on the flight of a flock of birds seeking food, which has been widely used for a variety of optimization tasks [1, 2]. However, its use in multimodal optimization (i.e., single-objective optimization problems having multiple optima) has been relatively scarce.
In this chapter, we will review the most representative PSO-based approaches that have been proposed to deal with multimodal optimization problems. Such approaches include the simple introduction of powerful mutation operators, schemes to maintain diversity that were originally introduced in the genetic algorithms literature (e.g., niching [3, 4]), the exploitation of local topologies, the use of species, and clustering, among others.
Our review also includes hybrid methods in which PSO is combined with another approach to deal with multimodal optimization problems. Additionally, we also present a study in which the performance of different PSO-based approaches is assessed in several multimodal optimization problems. Finally, a case study consisting on the search of solutions for systems of nonlinear equations is also provided.

1 Introduction

Particle swarm optimization (PSO) is a bio-inspired metaheuristic that was proposed by James Kennedy and Russell Eberhart in 1995 [5]. PSO performs a population-based search, using particles to represent potential solutions within the search space. Each particle is characterized by its position, velocity, and a record of its past performance. Particles are influenced by their leaders, which are the best performers either from the entire swarm or their neighborhood. At each flight cycle, the objective function is evaluated for each particle, with respect to its current position, and that information is used to measure the quality of the particle and to determine the leader in the sub-swarms and the entire population.

* The second author is also affiliated to the UMI-LAFMIA 3175 CNRS.

Although, usually, the development of optimization algorithms considers only the search of a single optimum of a given function, this is not always the case. It is possible that the function to be optimized has multiple global optima or one global optimum with many local optima in the search space. Such functions are called *multimodal* and have been widely studied in the genetic algorithms literature [3, 4, 6].

The PSO algorithm is a relatively recent optimization algorithm, which is quite simple, since it only consists of two rules for obtaining a new solution from a previous one. In spite of its simplicity, PSO has been found to exhibit a fast convergence to the optimum (or its vicinity) in a wide variety of optimization problems, which has significantly increased its popularity in the last few years [2]. However, until now, relatively few researchers have explored the potential of PSO for multimodal optimization, although its simplicity makes PSO a good candidate for dealing with such problems. The purpose of this chapter is precisely to review the most representative research done in this regard.

The remainder of this chapter is organized as follows. Section 2 describes the main topologies commonly adopted with PSO. Some PSO variants commonly adopted in the specialized literature are briefly described in Section 3. Section 4 introduces multimodal optimization, as well as the main approaches that have been proposed to deal with this sort of problem. Section 5 contains the test problems adopted for a small comparative study that is described and discussed in Section 6. A case study consisting of finding solutions to a system of nonlinear equations is presented in Section 7. Finally, our conclusions and some possible paths for future research are presented in Sections 8 and 9, respectively.

2 PSO Topologies

In the original PSO algorithm introduced by Kennedy and Eberhart [5] the position and velocity of a particle is updated using equations (1) and (2)

$$v_{t+1} = v_t + R_1 \cdot C_1 \cdot (g - x_t) + R_2 \cdot C_2 \cdot (p - x_t) \quad (1)$$

$$x_{t+1} = x_t + v_{t+1} \quad (2)$$

where C_1, and C_2 are the "learning" constants, R_1 and R_2 are randomly generated numbers (from a uniform distribution) in the interval $[0, 1]$, g is the position of the global best particle (i.e., the particle with the best value in the entire swarm), and p is the position with the best value recorded by the particle so far. The computation of g involves an inspection of the values of all the other particles in the swarm. In other words, any particle has access to the information of any other particle in the swarm. The global best (or *gbest*) model refers to the case in which all the particles are "connected" with each other and can transfer information among them, and it is essentially the original model of the PSO algorithm. A graphical representation of the *gbest* model is shown in Figure 1.

The *gbest* model is not the only model that has been proposed for PSO [7, 8]. Another model that has been widely used is the local best (or *lbest*) model,

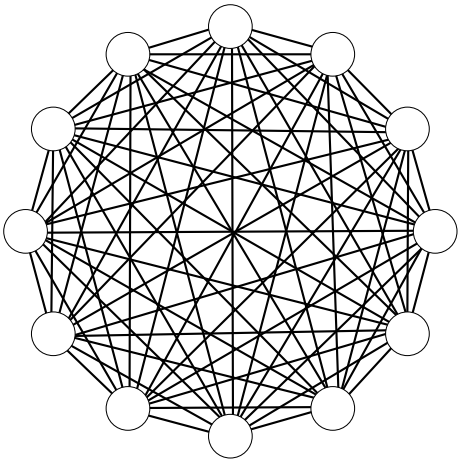

Fig. 1. Graphical representation of the *gbest* model

in which a particle is connected only with k of its neighbors and can only communicate with them. The number of neighbors of each particle is usually $k = 2$. In this case, the topology of the swarm is represented as a connected ring and its graphical representation is shown in Figure 2.

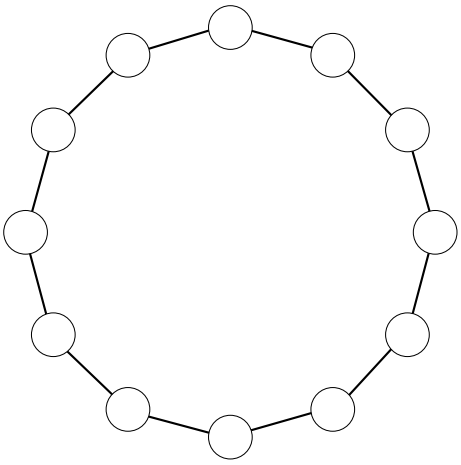

Fig. 2. Graphical representation of the *lbest* model

Another model that has been commonly adopted in the PSO literature is the von Neumann model, in which a particle can communicate with four of its neighbors using a rectangular lattice topology. A graphical representation of the von Neumann model is shown in Figure 3. The von Neumann model is

commonly adopted together with the *gbest* or *lbest* models in algorithms that use sub-swarms and a main swarm. Typically, the main swarm is arranged with the von Neumann model and the sub-swarms use either the *gbest* or the *lbest* model.

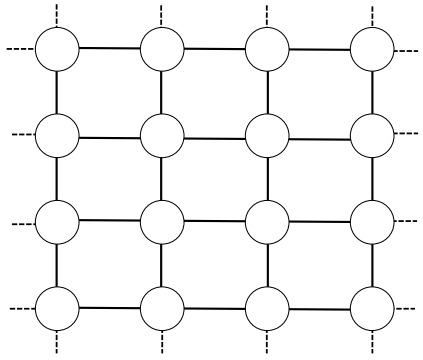

Fig. 3. Graphical representation of the von Neumann model

3 PSO variants

Other modifications have been added to the PSO algorithm, aiming to improve its convergence. The most common variations correspond to modifications on the computation of the velocity. Equation (1) shows the original method for computing the velocity of a particle. Eberhart and Shi [9] introduced the so-called *Inertia Weight model* in which the velocity of a particle at iteration t is multiplied by a constant parameter ω, called Inertia Weight, before computing the velocity for iteration $t+1$, as shown in equation (3).

$$v_{t+1} = \omega \cdot v_t + R_1 \cdot C_1 \cdot (g - x_t) + R_2 \cdot C_2 \cdot (p - x_t) \tag{3}$$

The parameter ω helps to balance between exploitation and exploration. Although the parameter ω is maintained constant in this model, Eberhart and Shi suggested in [10] that a linearly decreasing inertia weight may improve the convergence of the PSO algorithm. An initial ω_i and final ω_f values for the inertia are set, and the value of the inertia weight ω_t for the iteration t is computed using equation (4).

$$\omega_t = \omega_i - \frac{(\omega_f - \omega_i) \cdot t}{T} \tag{4}$$

where T is the total number of iterations and $t = 0, \ldots, T$. Another modification to the computation of the velocity is found in [11], where Clerc and Kennedy

presented the so-called *Constriction Factor model*. In the Constriction Factor model not only the velocity at iteration t is multiplied by a constant, but the new computed velocity is also affected, as shown in equation (5).

$$v_{t+1} = \chi \cdot [v_t + R_1 \cdot C_1 \cdot (g - x_t) + R_2 \cdot C_2 \cdot (p - x_t)] \tag{5}$$

The constriction factor constant χ is computed using equation (6).

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{6}$$

where $\phi = C_1 + C_2$, and κ is an arbitrary constant in the range $[0, 1]$. The value of ϕ is constrained: $\phi > 4$.

The computation of the velocity of a particle involves the position of the global best particle, called *social component* and the position with the best value recorded is called *cognition component*. If one of these terms is omitted, the two resulting models are called *cognition only* and *social only*, respectively. In any of them, only one component is used for the velocity update equation, as shown in equation (7) for the cognition only model and in equation (8) for the social only model.

$$v_{t+1} = v_t + R \cdot C \cdot (p - x_t) \tag{7}$$

$$v_{t+1} = v_t + R \cdot C \cdot (g - x_t) \tag{8}$$

Both, the cognition and the social models have been used in combination with the Inertia Weight and the Constriction Factor models.

4 Multimodal Optimization using PSO

The different models for updating the position and velocity of a particle, and the different topologies mentioned before can be considered as the basis for PSO-based multimodal approaches. In all the PSO-based multimodal approaches analyzed here, one of the three models indicated before (i.e., Inertia Weight, Decreasing Inertia Weight or Constriction Factor) is used to update the position and velocity of a particle, regardless of the particular approach adopted to deal with a multimodal problem. In methods that implement several sub-swarms and a main swarm, it is common to use two different topologies: one for the sub-swarms and another for the main swarm.

Next, we will review the most representative approaches that have been adopted to extend PSO so that it can deal with multimodal optimization problems.

4.1 Use of Mutation

Probably the easiest approach to adapt PSO to deal with multimodal optimization problems is to add a mutation operator, such as those adopted with genetic algorithms. Esquivel and Coello Coello [12] studied the use of nonuniform mutation in PSO in the context of multimodal optimization. The nonuniform mutation operator that they adopted was originally proposed in [13] for real-coded genetic algorithms, and it operates as follows. If we have the chromosome of an individual represented as a vector of real numbers $C_t = (c_1, c_2, \ldots, c_n)$ at the iteration t and c_k is the gene to mutate, then the mutated value c'_k is computed as follows:

$$c'_k = \begin{cases} c_k + \Delta(t, UB - c_k) & \text{if } P < 0.5 \\ c_k - \Delta(t, c_k - LB) & \text{otherwise} \end{cases} \tag{9}$$

where P is a randomly generated number in the interval $[0, 1]$ with a uniform distribution, LB and UB are the lower and upper bounds for the coordinate c_k, respectively, and the function $\Delta(t, z)$ returns a value in the range $[0, z]$. The probability that $\Delta(t, z)$ returns a value close to zero must increase as t increases. The function proposed in [13] for that sake is:

$$\Delta(t, z) = z \cdot \left(1 - R^{(1 - \frac{t}{T})^b}\right) \tag{10}$$

where R is a randomly generated number in the range $[0, 1]$ using a uniform distribution, T is the total number of iterations, and b is a user-defined parameter that determines the degree of dependency in the total number of iterations (in [13], a value $b = 5$ is suggested). The algorithm proposed by Esquivel and Coello Coello uses in its underlying PSO, the Inertia Weight model and mutates a coordinate of the position of a particle based on the index value of the current iteration and a mutation probability P_m. The algorithm using the *gbest* model is outlined in Figure 4. A simple modification of the algorithm allows it to be used with the *lbest* model as well.

The mutation operator introduced in the PSO algorithm tries to prevent that the particles remain trapped in a local minimum.

4.2 Niching in PSO

Niching is a method originally developed for genetic algorithms which is designed to block convergence of the entire population towards a single solution (otherwise, the entire population of an evolutionary algorithm eventually converges to a single solution because of stochastic noise [14]). Niching is one of the earliest methods developed to deal with multimodality using evolutionary algorithms [3, 4].

A variety of niching algorithms exist (see for example [15]). However, relatively few researchers have proposed PSO-based niching approaches. The most representative are briefly discussed next.

```
swarm initialization;
for i=1 to number of particles do
    for j=1 to number of dimensions do
        Initialize x_ij with a rnd(x_max, x_min) value;
        Initialize v_ij with zero value;
        copy x_ij to p_ij;
    end
end
search the best global leader and record its position in g;
swarm flight through the search space;
repeat
    for i=1 to number of particles do
        for j=1 to number of dimensions do
            Update v_ij using p_ij and x_ij;
            Prevent explosion of v_ij;
            Update x_ij;
            if loop_number < T · P_m then
                Mutate x_ij;
            end
        end
        Evaluate fitness(x_i);
        if fitness(p_i) < fitness(x_i) then
            Update p_i;
        end
    end
until loop_number < T ;
```

Fig. 4. PSO with a nonuniform mutation operator using the *gbest* model

NichePSO The NichePSO algorithm was proposed in [16]. This approach uses the Guaranteed Convergence Particle Swarm Optimizer from van den Bergh and Engelbrecht [17] together with the *gbest* model, aiming to prevent premature convergence of the particles. Additionally, the *cognition only* model is used to encourage the local search of the particles. The method used to identify niches in the NichePSO algorithm is based on keeping track of the changes in the fitness value of the particles. If a particle does not show any change within a certain number of iterations, a niche is created, containing the particle and its closest topological neighbor. The Euclidean distance is used to determine the closeness of the particles, so that the closest topological neighbor to the particle selected is the particle within the minimum distance to it. A niche radius is computed for each sub-swarm. This radius is the maximum distance between the particles in the sub-swarm and is described in equation (11).

$$r_j = \max\{||S_{x_{j,g}} - S_{x_{j,i}}||\} \tag{11}$$

This radius is used for adding particles to a sub-swarm and to merge two sub-swarms. If a particle has a distance less than r_j to the initial particle in a sub-swarm, then the particle is absorbed in the sub-swarm. If the distance between the initial particles of two sub-swarm is less than the sum of its radius, then the sub-swarms are merged. This condition is described in equation (12).

$$||S_{x_{j1,g}} - S_{x_{j2,g}}|| < (r_{j1} + r_{j2}) \tag{12}$$

If a sub-swarm has converged to an optimum it is possible that its radius r_j has a value of zero. In this case, it is difficult to determine the merge of two sub-swarms. A threshold value μ is given and if the distance between initial particles of sub-swarms with radius close to zero is less than μ, then the sub-swarms are merged. This is expressed in equation (13). The algorithm of NichePSO is outlined in Figure 5.

$$||S_{x_{j1,g}} - S_{x_{j2,g}}|| < \mu \tag{13}$$

The NichePSO algorithm does not require knowing ahead of time the number of optima of the function. Also, it does not require setting up a fixed radius for the niches. However, it depends on a minimum threshold to determine when a particle does not show changes and a new niche can be created. It also depends on a minimum distance to determine when two sub-swarms that are close to converging to an optimum can be merged.

Use of sub-swarms Another niching method is the creation of a fixed number of sub-swarms in the same search space and prevent the exploration of a same area by two or more sub-swarms. An example of this sort of method is found in the work of Zhang et al. [18] in which the Hill-Valley function [19] is used

```
1  Initialize main particle swarm;
2  Train main swarm particles using the cognition only model;
3  Update fitness of each main swarm particle;
4  foreach sub-swarm do
5      Train sub-swarm particles using one iteration of the GCPSO algorithm;
6      Update each particle's fitness;
7      Update swarm radius;
8  end
9  If possible, merge sub-swarms;
10 Allow sub-swarms to absorb any particles from the main swarm that moved into
   it;
11 Search main swarm for any particle that meets the partition criteria;
12 If any is found, then create a new sub-swarm with this particle and its closest
   neighbor;
```

Fig. 5. The nichePSO algorithm

to determine if a particle belongs to a niche. If a particle does not belong to a niche, it is penalized. This prevents the exploration of more than one sub-swarm in the same area. This simple penalty rule is expressed in equation (14).

$$eval(x_i) = \begin{cases} f(x_i) & \text{if } \textit{hill-valley}(x_i, x_{best}) = 1 \\ f(x_i) - p(x_i) & \text{otherwise} \end{cases} \tag{14}$$

where $p(x)$ is a penalty function [20]. The penalty function can be static (i.e., the same penalty value is always used) with a large value for the penalty factor.

The Hill-Valley algorithm tries to determine if two points are in the same valley of a function. A set of interior points are computed following a line between the points that are being tested. The best fitness value is set as the optimal fitness for comparison purposes, with respect to the fitness values of the interior points. In case of searching for a minimum, the lowest fitness value of the points being tested is set as the minimal for comparison purposes. If all the interior points have a fitness value lower than the minimal value, then the points being tested are in the same niche. The Hill-Valley algorithm is shown in Figure 6.

The niching algorithm of Zhang et al. [18] consists of sequentially creating a given number N of sub-swarms. Each sub-swarm explores the search space until a certain criterion is fulfilled. In this case, the sub-swarm explores for a certain number of iterations. The algorithm of sequential niching is shown in Figure 7.

In the sequential PSO niching algorithm of Zhang et al. [18] is necessary to set the number of sub-swarms to be used. If there is no prior knowledge of the number of optima of the function, it is necessary to experiment with the number of niches in the algorithm. It is also required to set the number of interior points for the Hill-Valley algorithm, but unlike NichePSO, this value does not depend on threshold parameters.

```
1 minfit = min(fitness(i_p), fitness(i_q));
2 for j=0 to samples length do
3     Calculate point i_interior on the line between the points i_q and i_p;
4     if minfit > fitness(i_interior) then
5         return 1;
6     end
7 end
8 return 0;
```

Fig. 6. The Hill-Valley algorithm

```
1  repeat
2      Run a new sub-swarm;
3      N = N + 1;
4      for k = 1 to N − 1 do
5          if hill_valley(x_i, x_k) == 0 then
6              Change the fitness of x_i;
7          else
8              Keep the original fitness;
9          end
10     end
11     Train the sub-swarm until a convergence criterion is met;
12 until N is greater than a given value or we reached a maximum number of
   iterations ;
```

Fig. 7. The ASNPSO algorithm

The problem of choosing the number of niches and how to determine if a particle belongs to a niche is addressed by Bird and Li [21]. In that work the number of niches and their radii is adaptively computed.

To initialize the radius value, Bird and Li [21] first compute the distance between each possible pair of particles and set the initial radius to the average of the minimum distances between particles as shown in equation (15).

$$r = \frac{\sum_{i=1}^{n} d_i}{n} \tag{15}$$

where

$$d_i = min\{||p_i - p_j|| \ |j \neq i\} \tag{16}$$

Then, to create a niche, each particle is tested versus all the rest of the particles in the swarm. This information is stored in an array s that keeps track of how many steps the particles are close to each other. If this distance is less than the initial radius value for at least two particles' steps, then a niche is created.

The algorithm for the creation of niches is divided in two parts: first, a graph is built, representing the particles that are close to each other within a distance less than the radius, for more than two particles' steps. The algorithm for creating the graph is shown in Figure 8.

Upon creating the graph, each particle is verified and it is grouped in a niche with the particles having a distance less than the initial radius. All particles that already belong to a niche are marked as visited in order to optimize the algorithm. The algorithm to create the niches is shown in Figure 9.

The particles that belong to a niche are updated using the Constriction Factor equations, adopting the *gbest* model in each niche. The particles that do not belong to a niche form a main swarm that uses a von Neumann topology and are updated using the Constriction Factor model. At each iteration of the algorithm, the niches are recalculated, which affects the performance of the algorithm.

4.3 Clustering techniques in PSO

Clustering into a PSO algorithm was first introduced by Kennedy [22]. The method of "stereotyping" adopted by Kennedy, consists of using the k-means algorithm to partition the main swarm into k sub-swarms. The k-means algorithm is shown in Figure 10. In each sub-swarm, the position of the global best is replaced by the position of the centroid of the sub-swarm.

In his work, Kennedy uses the Constriction Factor model for the PSO algorithm and explores the effects of exchanging the global best g, and the best recorded position p of a particle by the centroid of the sub-swarm in both the *gbest* and the *lbest* models.

```
1  Determine r using equation (15);
2  Create an undirected graph G containing a node for each particle, but no edges;
3  for i = 1 to n − 1 do
4      for j = i + 1 to n do
5          if ||p_i − p_j|| < r then
6              Increment s_ij;
7              if s_ij < 4 then
8                  s_ij ← 4;
9              end
10             if s_ij >= 2 then
11                 Create an edge in G from p_i to p_j;
12             end
13         else
14             Decrement s_ij;
15             if s_ij < 0 then
16                 s_ij ← 0;
17             end
18         end
19     end
20 end
```

Fig. 8. Procedure to create the niches graph

```
1  Create a set variable visited ← ∅;
2  Create a set variable reachable;
3  for i=1 to n do
4      if p_i ∉ visited and d_i < r then
5          reachable ← {p_i, p_j ∈ P|p_j ∀j reachable from p_i ∈ G};
6          Create a new niche s ← {p_j};
7          for p ∈ reachable and p ∉ visited do
8              visited ← visited ∪ {p};
9              s ← s ∪ {p};
10         end
11     end
12 end
```

Fig. 9. Creating the niches from the graph G

```
1  Initialize k centroids;
2  repeat
3      Compute the distance of each point to the k clusters;
4      Assign each point to the nearest cluster;
5      Recompute the centroid of each cluster;
6  until stop criteria is met ;
```

Fig. 10. k-means algorithm

A more recent work by Passaro and Starita [23] tries to improve the previous clustering approach by introducing several modifications. First, the number k of sub-swarms is estimated by testing different values of k and by using a Bayesian information criterion to decide which value of k is optimal [24]. The authors also limit the number of particles in each sub-swarm to correspond to the mean of the number of particles in each sub-swarm after the initial partition. If a cluster exceeds the mean of the number of particles, the particles with the worst fitness values are removed and added to the main swarm. The particles in the main swarm are updated separately using the *lbest* model with a von Neumann topology. The algorithm for identifying a niche is shown in Figure 11.

```
1  Cluster particle's pbest with the k-means algorithm;
2  Compute the average number of particles per cluster, N_avg;
3  Set N_u = 0;
4  foreach Cluster C_j do
5      if N_j > N_avg then
6          remove the N_j − N_avg particles from C_j;
7          add N_j − N_avg to N_u;
8      end
9      Adapt the neighborhood structure for the particles in C_j;
10 end
11 Reinitialize the N_u un-niched particles;
```

Fig. 11. Algorithm for the identification of niches

The algorithm of Passaro and Starita also recalculates the niches at intervals of c iterations. The procedure for indentifying niches is used to form again the sub-swarms and the main swarm with the *lbest* model. The algorithm is shown in Figure 12.

Although the optimal number of niches is computed, it is necessary to setup a maximum value of niches to carry out this calculation. It is also needed to determine the number of iterations c before recalculating the niches.

4.4 PSO with Species

The use of species for dealing with multimodal optimization problems, within a genetic algorithms context, is introduced in Li et al. [25]. In this method, individuals with high fitness values are selected as "seeds" to form clusters of individuals called "species", around the seeds. The procedure to select the seeds and form species consists of the following steps:

1. The best individual of a population is selected as a seed.
2. All the individuals with a distance less than a parameter value r from the seed are considered to belong to the same species.

```
1  Initialize particles with random positions and velocities;
2  Set particles' pbest to their current positions;
3  Calculate particles' fitness;
4  for t = 0 to T − 1 do
5      if t mod c = 0 then
6          Execute the procedure Identify Niches;
7      end
8      Update particles' velocities;
9      Update particles' positions;
10     Recalculate particles' fitness;
11     Update particles' and neighborhood best positions;
12 end
```

Fig. 12. The kPSO algorithm

3. The above procedure is repeated by selecting the next seed from the particles that do not belong to a species until all the particles are part of one.

The seeds selected in a generation are reintroduced in the next generation by comparing the fitness values of the new individuals within the species. If the individual with the worst fitness value in the species has a worse fitness value than the seed, then the seed replaces the worst individual of the species. Otherwise, the seed replaces the worst individual in the population even if the seed has a worse fitness value than the individual being replaced.

The concept of species was introduced into PSO by Li [26], with some changes. First, the species' seeds are not conserved nor reintroduced into the swarm. Also, the position of the seed of a species replaces the position of the global best for each particle in the species. The algorithm for the creation of the species is shown in Figure 13.

After the creation of the species, all particles are updated using the Constriction Factor equations. The algorithm for the species-based PSO approach is shown in Figure 14.

In the Species Particle Swarm Optimization algorithm, it is not necessary to set the number of sub-swarms that will be created before starting to iterate, since such number depends on the radius parameter, which is a value that needs to be set by the user.

4.5 Other Methods for Multimodal Optimization

The niching and species methods are not the only approaches that have been used with the PSO algorithm, since several other techniques have been proposed in the specialized literature as well. For example, a cooperative PSO was presented by van den Bergh and Engelbrecht [27]. This approach follows the idea from Potter and de Jong [28] who proposed that the population of an evolutionary algorithm is divided into sub-populations that cooperate unlike the niche methods in which

```
1  S = ∅;
2  while the end of L_sort has not been reached do
3      found ← FALSE;
4      forall p ∈ S do
5          if d(s, p) ≤ R_s then
6              found ← TRUE;
7              break;
8          end
9      end
10     if Not found then
11         S ← S ∪ {s}
12     end
13 end
```

Fig. 13. The algorithm for determining the species' seeds

```
1  Generate the initial population;
2  repeat
3      Evaluate all particles in the population;
4      Sort particles in descending order of their fitness value;
5      Determine the species' seeds for the current population;
6      Assign each species' seed identified as the g particle to all individuals
       identified in the same species;
7      Update the velocity and position of all the particles;
8  until Termination condition ;
```

Fig. 14. The species-based PSO

the sub-populations compete. Initially, in the Cooperative PSO algorithm, the main swarm is divided into several sub-swarms, each of which searches in only one dimension of decision variable space. Each sub-swarm cooperates by searching one coordinate of the solution. To evaluate the fitness function of a particle in a sub-swarm, an auxiliary vector is used. This vector has as its coordinates to the position of the particle with the best fitness value of each sub-swarm. In order to evaluate a particle in the jth sub-swarm the jth coordinate of the auxiliary vector is replaced with the position of the particle, and the fitness value of the particle is set to the fitness of the auxiliary vector. If some of the variables of the fitness function are correlated, it is preferred that the search space of a sub-swarm includes the correlated variables. This can be done easily with the partition model of the Cooperative PSO, but the information of correlated variables is not always available. An approach to try to group correlated variables is to partition the search space arbitrarily into k subspaces, with $k < D$ and D the dimension of the search space, as to create k sub-swarms that search into each sub-space, separately. The Cooperative PSO algorithm has a drawback: it can be trapped in pseudo-minima [27]. In order to overcome this problem, a phase is added to the Cooperative PSO algorithm, in which all the sub-swarms are considered as a single main swarm and the Constriction Factor model is used to update the position and velocity of the particles. This is called the Hybrid Cooperative PSO and its algorithm is shown in Figure 15.

The number of sub-swarms and the dimension of the subspace in which they search are randomly set and may not be the best possible for the fitness function being optimized.

Other method that implements in part a strategy similar to the Cooperative PSO is the Comprehensive Learning PSO developed by Liang et al. [29]. This algorithm uses a *social only* model in which only the social component of the velocity update equation is used together with the Inertia Weight model. Thus, to compute the velocity of a particle, equation (17) is used.

$$v_{t+1,d} = \omega \cdot v_{t,d} + C \cdot R_d \cdot (p_{g,f(d)} - x_d) \tag{17}$$

where the subindex d corresponds to the dimension, and $p_{g,f(d)}$ is an exemplar computed for the particle. Exemplars are computed using an idea similar to the one introduced by van den Bergh and Engelbrecht [27] and replace the position of the global best particle. To compute an exemplar, a coordinate of the $p_{g,f(d)}$ position is selected according to the following method:

1. A random number $rand$ is computed for each coordinate d. If $rand >= Pc_i$ (Pc_i is a probability for the particle i), then the value of the dth coordinate of the best recorded position of the particle i is copied to the dth coordinate of the exemplar.
2. Otherwise, if $rand < Pc_i$, then two particles are randomly selected from the swarm, so that they are different from particle i. Additionally, a tournament-like selection is done with the two particles selected. Also the dth coordinate

```
1  b(j, z) = (g_1, ..., g_{j-1}, z, g_{j+1}, ..., g_k);
2  K_1 = n mod K;
3  K_2 = K − (n mod K);
4  Initialize K_1⌈n/K⌉-dimensional PSOs: P_j, j ∈ [1..K_1];
5  Initialize K_2⌊n/K⌋-dimensional PSOs: P_j, j ∈ [(K_1 + 1)..K];
6  Initialize an n-dimensional PSO: Q;
7  repeat
8      foreach swarm i ∈ [1..K] do
9          foreach particle j ∈ [1..s] do
10             if f(b(i, x_ij)) < f(b(i, p_ij)) then
11                 p_ij = x_ij;
12             end
13             if f(b(i, p_ij)) < f(b(i, g_ij)) then
14                 g_ij = p_ij;
15             end
16         end
17         Perform updates on P_j;
18     end
19     Select random k ∼ U(1, s/2) such that p_k ≠ g;
20     x_k = b(1, g_1);
21     foreach particle j ∈ [1..s] do
22         if f(x_j) < f(p_j) then
23             p_j = x_j;
24         end
25         if f(p_j) < f(g) then
26             g = p_j;
27         end
28     end
29     Perform updates in Q;
30     foreach swarm j ∈ [1..K] do
31         Select random k ∼ U(1, s/2) such that p_jk ≠ g_j;
32         x_jk = g_j;
33     end
34 until stopping condition is true ;
```

Fig. 15. Pseudocode for the CPSO-H$_k$ algorithm

of the best recorded position of the particle with a better fitness value is copied to the dth coordinate of the exemplar.

3. If the condition $rand < Pc_i$ is never true for any coordinate d, then a random number k is computed with $k \in 1, \ldots, D$ (D is the number of decision variables), and the procedure of step 2 is repeated for the coordinate k.

To compute the value of the probability Pc_i for the particle i, the authors use an empirical relation. For example, to compute the probability values Pc_i in the range $[0.05, 0.5]$, equation (18) is used.

$$Pc_i = 0.05 + 0.45 \cdot \frac{e^{\left(\frac{10(i-1)}{ps-1}\right)} - 1}{e^{10} - 1} \tag{18}$$

where ps is the number of particles in the swarm. The procedure for computing an exemplar is shown in Figure 16.

```
1  for d=1 to D do
2      if rand < Pc_i then
3          f1_d = ⌈rand1_d · ps⌉;
4          f2_d = ⌈rand2_d · ps⌉;
5          if fitness(p_f1_d) > fitness(p_f2_d) then
6              f_d = f_1d;
7          else
8              f_d = f_2d;
9          end
10     else
11         f_d = i;
12     end
13 end
```

Fig. 16. Selection of the exemplar dimension for particle i

The Comprehensive Learning PSO algorithm also keeps track of the changes in the fitness of the particles. If the particle does not show changes in its best recorded position after m iterations, then a new exemplar is computed for the particle. The value for m is arbitrary and the authors use $m = 7$, which was empirically obtained. The algorithm also uses a policy of not updating the fitness value nor the best recorded position if the particle is out of the allowable search space. It also updates the global best of the swarm each time that a particle is updated. This is similar to the evaluation in the case of a *lbest* topology. The algorithm of the Comprehensive Learning PSO is shown in Figure 17.

The Comprehensive Learning PSO algorithm requires several parameters: the probability Pc_i of computing an exemplar, the refreshing gap m to re-compute an exemplar if the best position of a particle does not present changes (this

```
1  Initialize swarm;
2  for k = 1 to max_gen do
3      ω = ω_0 − (ω_0 − ω_1) · k/max_gen;
4      for i = 1 to ps do
5          if flag_i >= m then
6              Select an exemplar for particle i;
7              flag_i = 0;
8          end
9          Update velocity and position of the particles;
10         if x_i ∈ [X_min, X_max] then
11             Update fitness x_i;
12             if fitness(x_i) > fitness(p_i) then
13                 p_i = x_i;
14                 flag_i = 0;
15                 if fitness(x_i) > fitness(g) then
16                     g = x_i;
17                 end
18             else
19                 flag_i = flag_i + 1;
20             end
21         end
22     end
23 end
```

Fig. 17. The CLPSO algorithm

value is based on the Decreasing Inertia PSO). Additionally, an initial ω_i and a final ω_f must also be set by the user. In spite of the number of parameters to set and its relative complexity, Comprehensive Learning PSO has obtained better results than many other PSO-based methods in a variety of multimodal optimization problems.

4.6 Hybrid Methods

The PSO algorithm has also been hybridized with other algorithms in an attempt to improve its performance in multimodal optimization problems. That is the case of the hybrid of PSO and Differential Evolution (DE) introduced by Pant et al. [30]. This algorithm works in two phases. First, at each iteration, a mutant of each particle is computed by adopting the same procedure from the DE algorithm. Then, the crossover operator from DE is applied. If the new particle created with the DE procedure has a better fitness value than the particle being compared, then the position of the particle is replaced by the position of the newly generated particle. If the fitness of the new particle is worse, then the particle is removed and a new particle is generated. The fitness value of the particle with the updated position is compared with the fitness value of the particle in the previous position. If the new particle has a better fitness value, then the position of the particle is updated to the new computed position using the equations of the PSO algorithm. If the fitness value is worse, then the position of the particle remains unchanged. The algorithm of the hybrid of DE and PSO is shown in Figure 18.

A similar hybrid is presented in the work of Shelokar et al. [31], in which a hybrid of PSO and ant colony optimization (ACO) is presented. In this hybrid, a colony of ants is created within the swarm of particles. Each ant is related to a particle, and after a particle is updated, its related ant is also updated, by following a simple pheromone scheme. If the fitness value of the ant is better than the fitness value of the particle, then the position of the particle is replaced with the position of the ant. The algorithm of this hybrid is shown in Figure 19 where Z_t^i is computed according to equation (19).

$$z_t^i = N(g, \sigma) \tag{19}$$

where $N(g, \sigma)$ is a normal distribution with mean g and standard deviation σ. The value of σ is decreased at each iteration by multiplying it by a constant d with values in the range $[0.25, 997]$. If the value of σ is less than a lower bound σ_m, then the value of σ is set to $\sigma = \sigma_m$ and remains constant for the rest of the iterations.

5 Test Functions

In order to have a better idea of the differences in performance of some of the approaches previously discussed, we decided to perform a small comparative study.

```
Initialize swarm;
for i = 1 to N do
    Select r1, r2, r3 ∈ N;
    for j = 1 to D do
        Select j_rand ∈ D;
        if rand() < Cr or j = j_rand then
            U_{ji,g+1} = x_{r1,g} + F · (x_{r2,g} − x_{r3,g});
        end
        if f(U_{ji,g+1}) < f(x_{ji,g}) then
            x_{ji,g+1} = U_{ji,g+1};
        else
            Compute tx_{ji} using PSO equations;
            if f(tx_{ji}) < f(x_{ji,g}) then
                x_{ji,g+1} = tx_{ji};
            else
                x_{ji,g+1} = x_{ji,g};
            end
        end
    end
end
```

Fig. 18. Hybrid of PSO and DE

```
Initialize swarm;
Initialize ants;
repeat
    Update particles' position and velocity;
    Evaluate the fitness function of each particle;
    Generate P solutions z_t^i using equation (19);
    for i = 1 to ps do
        if f(z_t^i) < f(x_i) then
            x_i = z_t^i;
        end
        Update p_i of particles;
    end
    Update g best;
until t = T ;
```

Fig. 19. Hybrid of PSO and ACO

For that sake, we used a set of five test functions commonly adopted in the multimodal optimization literature. The selected problems are: Rastrigin's function [32, 33], Griewank's function [34], Ackley's function [35, 36], Schwefel's function [37], and Shubert's function. Equations (20) to (24) show the test functions in the same order as indicated above.

$$f_1(x) = nA + \sum_{i=1}^{n} x_i^2 - A\cos(\omega x_i) \tag{20}$$

$$f_2(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{21}$$

$$f_3(x) = -20e^{-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{30}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e \tag{22}$$

$$f_4(x) = 418.9829n + \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right) \tag{23}$$

$$f_5(x) = \prod_{i=1}^{n}\sum_{j=1}^{5} j\cos[(j+1)x_i + j] \tag{24}$$

Except for Shubert's test problem, which has multiple global optima, the rest of the test functions adopted have only one global optimum, but contain several local optima. Shubert's test problem has several global optima distributed uniformly throughout the search space. In Table 1, we show a summary of the features of the test functions adopted, including their search range, the number of optima in the search range and the position of their global optima.

Table 1. Summary of the features of the test functions adopted

Test Function	Optimum	Optimum fitness value	Search range
Rastrigin	$[0, \ldots, 0]$	0	$[-5.12, -5.12]^D$
Griewank	$[0, \ldots, 0]$	0	$[-600.0, 600.0]^D$
Ackley	$[0, \ldots, 0]$	0	$[-15.0, 30.0]^D$
Schwefel	$[1, \ldots, 1]$	0	$[-500.0, 500.0]^D$
Shubert	18 (in two dimensions)	-186.7309	$[-10.0, 10.0]^2$

6 Experiments and Results

The algorithms applied to the test functions are the following: PSO with mutation [12], species PSO [25], comprehensive learning PSO [29], and the hybrid of PSO and ACO [31].

Next, we describe the setup for each algorithm. It is important to note that the values adopted for the parameters of each approach follow those proposed by their authors.

For the PSO with mutation we used: swarm size = 40 particles, learning constant values $C_1 = C_2 = 1.3$, inertia weight $\omega = 0.3$, mutation probability $P_m = 0.9$.

The PSO with species used: swarm size = 40 particles, learning constants values $C_1 = C_2 = 2.05$, radius value between 1/10 and 1/20 of the allowable variables range.

For the Comprehensive Learning PSO we used: swarm size = 40 particles, initial and final inertia values $\omega_i = 0.9$, and $\omega_f = 0.4$, learning constant values $C_1 = C_2 = 1.49455$, refreshing gap $m = 7$.

Finally, for the hybrid of PSO and ACO we used: swarm size = 40 particles, learning constant values $C_1 = C_2 = 2.0$, initial and final inertia values $\omega_i = 0.7$, $\omega_f = 0.4$, $d = 0.5$, $\sigma_m = 10^{-4}$.

In order to allow a fair comparison, all the algorithms performed 5,000 iterations. The convergence plots for all the test functions are shown in Figures 20, 21, 22, 23 and 24. Table 2 shows the mean of the best fitness values obtained by the algorithms after 5000 iterations for the test functions adopted.

Table 2. Mean of the best fitness value after 5000 iterations.

	CLPSO	APSO	SPSO	MPSO
Rastrigin	1.5	188.56	175.44	436.33
Griewank	1.68	838.41	15.75	0.97
Ackley	1.96e-07	19.36	10.71	3.59
Schwefel	13.29	1.28e+13	6.81e+9	6.21e+4
Shubert	-186.56	-185.87	-79.94	136.88

From the plots shown in Figures 20 to 24, and from the summary shown in Table 2, we can observe that the Comprehensive Learning PSO algorithm has the best convergence and it is consistent in all the test functions adopted. The Species PSO algorithm depends strongly on the selection of the radius, but it tends to stabilize faster than the others. The PSO with mutation has a lower rate of convergence but it shows good results in almost all the test functions tested, which seems to indicate that the simple introduction of a good mutation operator may be enough if the cost of evaluating the objective function is not significant. The hybrid of PSO and ACO had the worst results both in terms of convergence rate and accuracy, but it shows good results and better stability than any other algorithm in Shubert's test function.

In Figure 24 we show the convergence graphs for Shubert's test function. Although it is analyzed only in two dimensions, this test function has 18 global optima and, as we can observe from Figure 24, for almost all the algorithms, the value of the reported optimum shows oscillations. In the case of the PSO with mutation the reported optimum value tends to diverge.

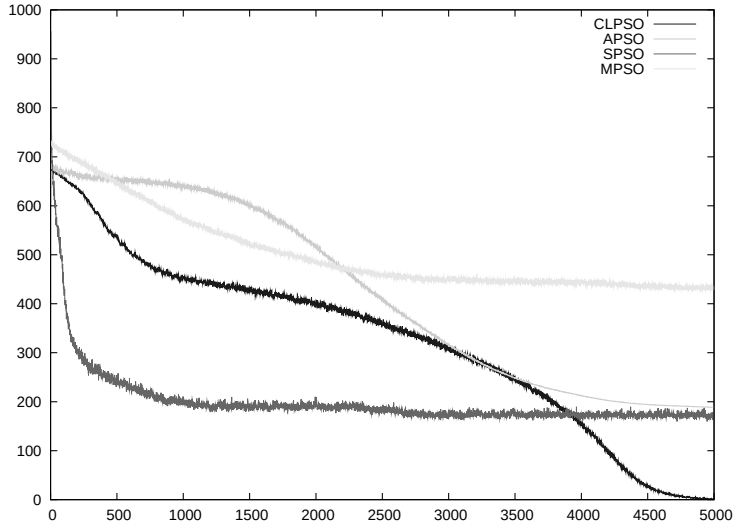

Fig. 20. Diagram of convergence for the four algorithms applied to Rastrigin's function.

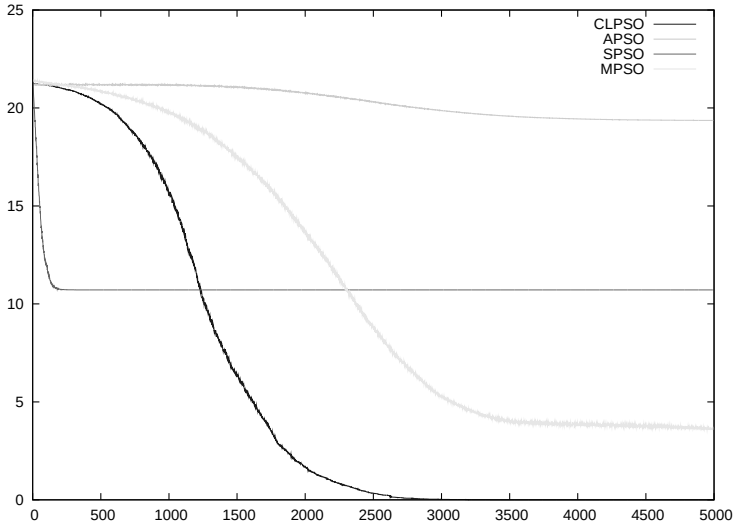

Fig. 21. Diagram of convergence for the four algorithms applied to Ackley's function.

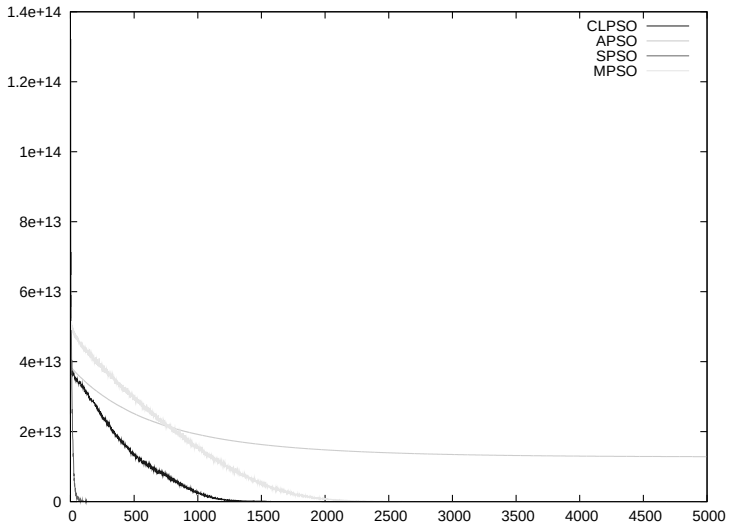

Fig. 22. Diagram of convergence for the four algorithms applied to Schwefel's function.

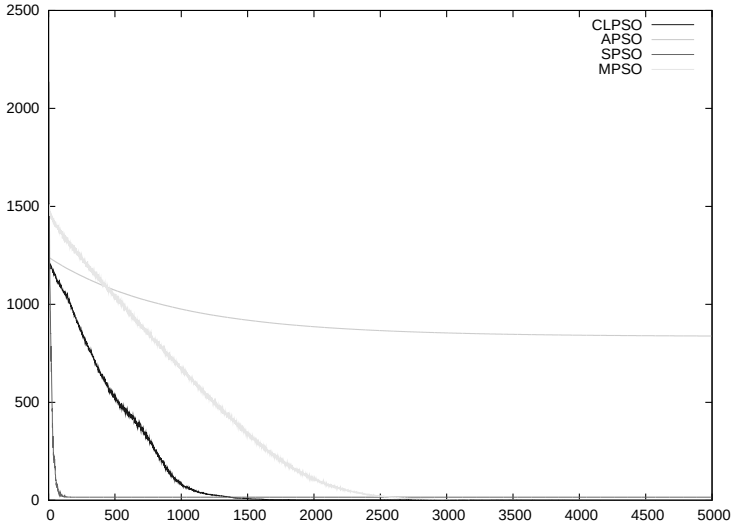

Fig. 23. Diagram of convergence for the four algorithms applied to Griewank's function.

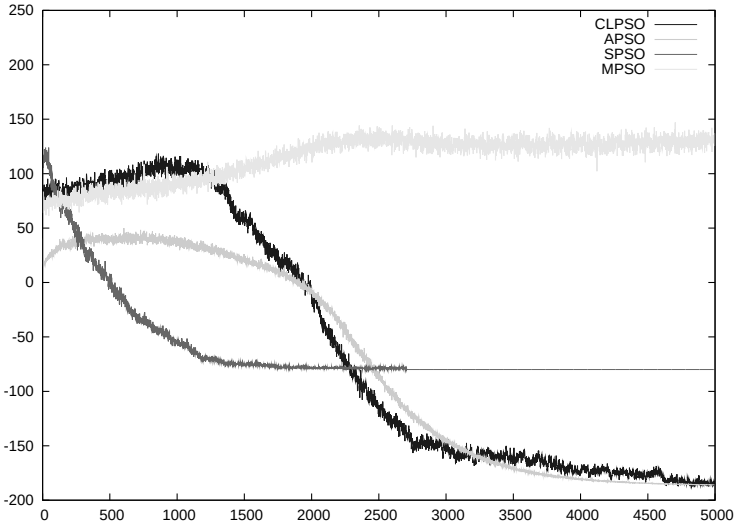

Fig. 24. Diagram of convergence for the four algorithms applied to Shubert's function.

7 Search of Solutions of a System of Nonlinear Equations

The study of dynamical systems is of great importance in almost all fields of knowledge. A dynamical system is commonly modeled using a system of ordinary differential equations such as the one represented in equation (25).

$$\dot{x} = f(x) \tag{25}$$

with $x \in R^n$. The first step in the analysis of a dynamical system is to find a fixed point of the system. A fixed point is a point in which all the derivatives are equal to zero, that is, a point x' such that

$$0 = f(x') \tag{26}$$

To find a fixed point we need to find a solution of the system of equations $f(x)$ that most of the time is nonlinear.

An example of the physical interpretation of this sort of system is the electrical power system of three nodes [38, 39] shown in Figure 25.

The corresponding dynamic system is modeled with equations (27) to (30).

$$\dot{\delta}_m = \omega \tag{27}$$

$$\dot{\omega} = \frac{1}{M}(P_m - D_m\omega + VE_mY_m \sin(\delta - \delta_m - \theta_m) +$$

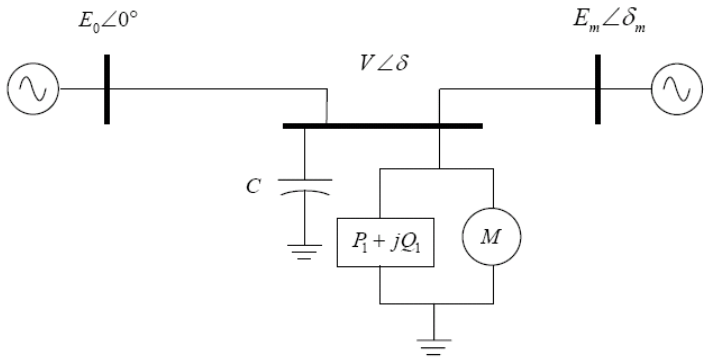

Fig. 25. Diagram of a power system of three nodes

$$E_m^2 Y_m \sin\theta_m) \tag{28}$$

$$\dot{\delta} = \frac{1}{K_{qw}}(-K_{qv2}V^2 - K_{qv}V + Q - Q_0 - Q_1) \tag{29}$$

$$\dot{V} = \frac{1}{TK_{qw}K_{pv}}[-K_{qw}(P_0 + P_1 - P) + (K_{pw}K_{qv} - K_{qw}K_{pv})V + K_{pw}(Q_0 + Q_1 + Q) + K_{pw}K_{qv2}V^2] \tag{30}$$

where

$$P = -VE_0'Y_0'\sin(\delta + \theta_0') - VE_mY_m\sin(\delta - \delta_m + \theta_m) + V^2(Y_0'\sin\theta_0' + Y_m\sin\theta_m) \tag{31}$$

$$Q = VE_0'Y_0'\cos(\delta + \theta_0') + VE_mY_m\cos(\delta - \delta_m + \theta_m) - V^2(Y_0'\cos\theta_0' + Y_m\cos\theta_m) \tag{32}$$

Only four variables are considered: δ_m, ω, δ, and V. All the other symbols are set to constant values as shown in Table 3. We need to find a fixed point in the search space defined by the range values for the variables shown in Table 4.

For this example, we used the Comprehensive Learning PSO with a swarm size of 10 particles, initial and final inertia values $\omega_i = 0.9$, $\omega_f = 0.4$, learning constant values $C_1 = C_2 = 1.49455$, refreshing gap $m = 7$ and a total of $1,000$ iterations. The fitness function is $|f(x)|$ with f being the system of equations (27) to (30) and $x = (\delta_m, \omega, \delta, V)$. After applying Comprehensive Learning PSO, a solution is found in the point $(0.047, 0.0, 0.310, 1.35)$ with an error of 10^{-4} that is an acceptable error value for this application.

The PSO algorithm has several advantages when adopted for searching solutions for systems of nonlinear equations: PSO does not require of a "good" initial point to perform the search, and the search space can be bounded by lower and upper values for each decision variable. Additionally, no continuity or

Table 3. Constant values for the symbols in the system of nonlinear equations

symbol	value	symbol	value	symbol	value
K_{pw}	0.4	K_{pv}	0.3	K_{qw}	-0.03
K_{qv2}	2.1	T	8.5	K_{qv}	-2.8
P_0	0.6	P_1	0.0	Q_0	1.3
E'_0	2.5	P_m	1.0	E_m	1.0
M	0.3	Y_m	5.0	Y'_0	8.0
θ'_0	12.0	Q_1	10.0	θ_m	-5.0
D_m	0.05				

Table 4. Values of the search ranges for the decision variables

variable	range
δ_m	$[0, 1]$
ω	$[-1, 1]$
δ	$[0, 1]$
V	$[0, 2]$

differentiability of the objective function is required. What can be considered as the main disadvantage of PSO in this sort of application is its relatively poor accuracy, which is caused by the coarse granularity of the search performed by the algorithm. This can, of course, be improved either by running the PSO algorithm for a larger number of iterations (although at a higher computational cost) or by post-processing the solution produced by the PSO algorithm with a traditional numerical optimization technique.

8 Conclusions

In this chapter, we have shown a review of the most representative PSO-based methods available for multimodal optimization. As we saw, most of these methods were either adapted or inspired by research reported in the genetic algorithms literature. There are, however, other methods that are not directly derived from such literature, because they rely on hybridizations between PSO and another metaheuristic (e.g., differential evolution or ant colony optimization), with the clear aim of benefitting from the advantages of both types of approaches.

A particular case is the clustering method originally developed by Kennedy [22] and further modified by Bird and Li [21], since this sort of approach was specifically designed for a PSO algorithm.

It is worth mentioning that all the methods analyzed here have been developed using as a basis the known models for updating the position and velocity of the PSO algorithm, namely the Inertia Weight and the Constriction Factor. To the authors' best knowledge, there are no models currently available for updating the position and velocity of a particle, that had been specifically developed

for dealing with multimodal optimization problems. This is also true for the topologies of the swarm, since the authors are not aware of any new topologies that had been developed specifically for multimodal problems. It is worth noting, however, that in the method for computing the niche parameters for Bird and Li's algorithm [21], a graph based on the closeness of the particles is used to determine if a particle belongs to a niche. Nevertheless, once the sub-swarms are built they use a *gbest* topology in each sub-swarm and a von Neumann topology with the particles that do not belong to a sub-swarm, but are part of the main swarm. A particular case is the Comprehensive Learning PSO algorithm. Although in this algorithm the Inertia Weight is used together with the *social only* model, the position of the best particle in the swarm is replaced with a computed exemplar for each particle, which is an *ad-hoc* method specifically developed for this approach.

In general, all the methods studied show advantages over the use of a basic PSO algorithm, but they also introduce new parameters that need to be set by the user. In most cases, the parameters were empirically tuned after performing a series of experiments [29]. The current research in the area includes the development of methods for adaptively computing the parameters required for dealing with multimodal optimization problems (e.g., [21]).

We have also conducted a small comparative study in which we analyzed the performance of several of the algorithms discussed in this chapter. This study showed that Comprehensive Learning PSO had the best overall results both in terms of quality of the final solutions and of consistency in reaching such results.

Finally, we presented an application of PSO in a case study in which the problem is multimodal: the search of solutions to a set of nonlinear equations. The results in this case were very encouraging and showed some of the advantages of adopting PSO with respect to traditional numerical optimization techniques (e.g., the use of bounded decision variables, and the use of randomly generated solutions to initiate the search).

9 Future work

There are several areas that remain open for those interested in working in this area. First, the development of new PSO-based methods for multimodal optimization still has a lot to offer. For example, the development of new approaches that have less (or no) parameters that have to be fine-tuned by the user and that remain effective in a variety of test problems is an interesting venue for future research.

There are approaches, such as those based on clustering methods, that do not use any information from the fitness values of the particles. Such information could be used, for example, to improve the computation of the centroids of the clusters.

Finally, the development of additional hybrid methods is also an interesting path for future research, that can give us more insights regarding the behavior and potential advantages and disadvantages of different metaheuristics.

References

1. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm intelligence. Morgan Kaufmann, San Francisco (2001)
2. Engelbrecht, A.P.: Fundamentals of Computational Swarm Intelligence. John Wiley & Sons (2006)
3. Goldberg, D.E., Richardson, J.: Genetic algorithm with sharing for multimodal function optimization. In Grefenstette, J.J., ed.: Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Hillsdale, New Jersey, Lawrence Erlbaum (1987) 41–49
4. Deb, K., Goldberg, D.E.: An Investigation of Niche and Species Formation in Genetic Function Optimization. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, George Mason University, Morgan Kaufmann Publishers (June 1989) 42–50
5. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, Piscataway, New Jersey, USA, IEEE Press (1995) 1942–1948
6. Deb, K., Kumar, A.: Real-coded Genetic Algorithms with Simulated Binary Crossover: Studies on Multimodal and Multiobjective Problems. Complex Systems **9** (1995) 431–454
7. Kennedy, J.: Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of the 1999 IEEE Congress on Evolutionary Computation (CEC'99). Volume 3., IEEE Press (July 1999) 1931–1938
8. Kennedy, J., Mendes, R.: Population structure and particle swarm performance. In: Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC'2002), Washington, DC, USA, IEEE Press (2002) 1671–1676
9. Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation Proceedings, IEEE Press (1998) 69–73
10. Shi, Y., Eberhart, R.C.: Empirical study of particle swarm optimization. In: Proceedings of the 1999 IEEE Congress on Evolutionary Computation (CEC'1999). Volume 3., IEEE Press (July 1999) 1945–1950
11. Clerc, M., Kennedy, J.: The particle swarm: explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation **6**(1) (2002) 58–73
12. Esquivel, S.C., Coello Coello, C.A.: On the use of particle swarm optimization with multimodal functions. In: Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC'2003). Volume 2., IEEE Press (December 2003) 1130–1136
13. Michalewicz, Z.: Genetic algorithms + data structures = evolution programs (3rd ed.). Springer-Verlag, London, UK (1996)
14. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co., Reading, Massachusetts (1989)
15. Mahfoud, S.W.: Niching Methods for Genetic Algorithms. PhD thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, Illinois (May 1995)
16. Brits, R., Engelbrecht, A., van den Bergh, F.: A Niching Particle Swarm Optimizer. In Wang, L., Tan, K.C., Furuhashi, T., Kim, J.H., Yao, X., eds.: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02). Volume 2., Orchid Country Club, Singapore, Nanyang Technical University (November 2002) 692–696

17. van den Bergh, F., Engelbrecht, A.: A new locally convergent particle swarm optimiser. In: 2002 IEEE International Conference on Systems, Man and Cybernetics. Volume 3., IEEE Press (October 2002)
18. Zhang, J., Huang, D.S., Liu, K.H.: Multi-Sub-Swarm Particle Swarm Optimization Algorithm for Multimodal Function Optimization. In: 2007 IEEE Congress on Evolutionary Computation (CEC'2007), Singapore, IEEE Press (September 2007) 3215–3220
19. Ursem, R.K.: Multinational evolutionary algorithms. In: Proceedings of the Congress on Evolutionary Computation, IEEE Press (1999) 1633–1640
20. Yeniay, Özgür.: Penalty Function Methods for Constrained Optimization with Genetic Algorithms. Mathematical and Computational Applications **10**(1) (2005) 45–56
21. Bird, S., Li, X.: Adaptively Choosing Niching Parameters in a PSO. In et al., M.K., ed.: 2006 Genetic and Evolutionary Computation Conference (GECCO'2006). Volume 1., Seattle, Washington, USA, ACM Press. ISBN 1-59593-186-4 (July 2006) 3–9
22. Kennedy, J.: Stereotyping: improving particle swarm performance with cluster analysis. In: Proceedings of the 2000 Congress on Evolutionary Computation. Volume 2., IEEE Press (2000) 1507–1512
23. Passaro, A., Starita, A.: Particle swarm optimization for multimodal functions: a clustering approach. Journal of Artificial Evolution and Applications **8**(2) (2008) 1–15
24. Pelleg, D., Moore, A.: X-means: Extending k-means with efficient estimation of the number of clusters. In: Proceedings of the Seventeenth International Conference on Machine Learning, San Francisco, California, USA, Morgan Kaufmann (2000) 727–734
25. Li, J.P., Balazs, M.E., Parks, G.T., Clarkson, P.J.: A species conserving genetic algorithm for multimodal function optimization. Evolutionary Computation **10**(3) (2002) 207–234
26. Li, X.: Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In et al., K.D., ed.: Genetic and Evolutionary Computation–GECCO 2004. Proceedings of the Genetic and Evolutionary Computation Conference. Part I, Seattle, Washington, USA, Springer-Verlag, Lecture Notes in Computer Science Vol. 3102 (June 2004) 105–116
27. van den Bergh, F., Engelbrecht, A.: A cooperative approach to particle swarm optimization. IEEE Transactions on Evolutionary Computation **8**(3) (June 2004) 225–239
28. Potter, M.A., de Jong, K.: A Cooperative Coevolutionary Approach to Function Optimization. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature—PPSN III, Jerusalem, Israel, Springer-Verlag. Lecture Notes in Computer Science Vol. 866 (October 1994) 249–257
29. Liang, J., Qin, A., Suganthan, P., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Transactions on Evolutionary Computation **10**(3) (June 2006) 281–295
30. Pant, M., Thangaraj, R., Grosan, C., Abraham, A.: Hybrid differential evolution - particle swarm optimization algorithm for solving global optimization problems. In: Third International Conference on Digital Information Management (ICDIM' 2008). (November 2008) 18–24
31. Shelokar, P., Siarry, P., Jayaraman, V., Kulkarni, B.: Particle swarm and ant colony algorithms hybridized for improved continuous optimization. Applied Mathematics and Computation **188**(1) (2007) 129–142

32. Törn, A., Zilinskas, A.: Global Optimization. Lecture Notes in Computer Science **350** (1989)
33. Mühlenbein, H., Schomisch, D., Born, J.: The Parallel Genetic Algorithm as Function Optimizer. Parallel Computing **17**(6-7) (1991) 619–632
34. Bäck, T., Fogel, D., Michalewicz, Z.: Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York (1997)
35. Ackley, D.H.: A connectionist machine for genetic hillclimbing. Kluwer, Boston (1987)
36. Bäck, T.: Evolutionary algorithms in theory and practice. Oxford University Press (1996)
37. Schwefel, H.P.: Numerical Optimization of Computer Models. John Wiley & Sons (1981)
38. Dobson, I., Chiang, H.D., Thorp., J.S.: A model of voltage collapse in electric power systems. In: IEEE proceedings of 27th Conference on Decision and Control, Austin Texas (December 1988) 2104–2109
39. Walve, K.: Modeling of power system components at severe disturbances. In: CIGRÉ paper 38-18, International conference on large high voltage electric systems. (August 1986)