Chapter 16

Particle Swarm Optimization

The particle swarm optimization (PSO) algorithm is a population-based search algorithm based on the simulation of the social behavior of birds within a flock. The initial intent of the particle swarm concept was to graphically simulate the graceful and unpredictable choreography of a bird flock [449], with the aim of discovering patterns that govern the ability of birds to fly synchronously, and to suddenly change direction with a regrouping in an optimal formation. From this initial objective, the concept evolved into a simple and efficient optimization algorithm.

In PSO, individuals, referred to as particles, are "flown" through hyperdimensional search space. Changes to the position of particles within the search space are based on the social-psychological tendency of individuals to emulate the success of other individuals. The changes to a particle within the swarm are therefore influenced by the experience, or knowledge, of its neighbors. The search behavior of a particle is thus affected by that of other particles within the swarm (PSO is therefore a kind of symbiotic cooperative algorithm). The consequence of modeling this social behavior is that the search process is such that particles stochastically return toward previously successful regions in the search space.

The remainder of this chapter is organized as follows: An overview of the basic PSO, i.e. the first implementations of PSO, is given in Section 16.1. The very important concepts of social interaction and social networks are discussed in Section 16.2. Basic variations of the PSO are described in Section 16.3, while more elaborate improvements are given in Section 16.5. A discussion of PSO parameters is given in Section 16.4. Some advanced topics are discussed in Section 16.6.

16.1 Basic Particle Swarm Optimization

Individuals in a particle swarm follow a very simple behavior: to emulate the success of neighboring individuals and their own successes. The collective behavior that emerges from this simple behavior is that of discovering optimal regions of a high dimensional search space.

A PSO algorithm maintains a swarm of particles, where each particle represents a potential solution. In analogy with evolutionary computation paradigms, a *swarm* is

Computational Intelligence: An Introduction, Second Edition A.P. Engelbrecht

similar to a population, while a *particle* is similar to an individual. In simple terms, the particles are "flown" through a multidimensional search space, where the position of each particle is adjusted according to its own experience and that of its neighbors. Let $\mathbf{x}_i(t)$ denote the position of particle i in the search space at time step t; unless otherwise stated, t denotes discrete time steps. The position of the particle is changed by adding a velocity, $\mathbf{v}_i(t)$, to the current position, i.e.

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{16.1}$$

with $\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$.

It is the velocity vector that drives the optimization process, and reflects both the experiential knowledge of the particle and socially exchanged information from the particle's neighborhood. The experiential knowledge of a particle is generally referred to as the *cognitive component*, which is proportional to the distance of the particle from its own best position (referred to as the particle's *personal best* position) found since the first time step. The socially exchanged information is referred to as the *social component* of the velocity equation.

Originally, two PSO algorithms have been developed which differ in the size of their neighborhoods. These two algorithms, namely the *gbest* and *lbest* PSO, are summarized in Sections 16.1.1 and 16.1.2 respectively. A comparison between *gbest* and *lbest* PSO is given in Section 16.1.3. Velocity components are decsribed in Section 16.1.4, while an illustration of the effect of velocity updates is given in Section 16.1.5. Aspects about the implementation of a PSO algorithm are discussed in Section 16.1.6.

16.1.1 Global Best PSO

For the global best PSO, or *gbest* PSO, the neighborhood for each particle is the entire swarm. The social network employed by the *gbest* PSO reflects the star topology (refer to Section 16.2). For the star neighborhood topology, the social component of the particle velocity update reflects information obtained from all the particles in the swarm. In this case, the social information is the best position found by the swarm, referred to as $\hat{\mathbf{y}}(t)$.

For *gbest* PSO, the velocity of particle i is calculated as

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \tag{16.2}$$

where $v_{ij}(t)$ is the velocity of particle i in dimension $j = 1, \ldots, n_x$ at time step t, $x_{ij}(t)$ is the position of particle i in dimension j at time step t, c_1 and c_2 are positive acceleration constants used to scale the contribution of the cognitive and social components respectively (discussed in Section 16.4), and $r_{1j}(t), r_{2j}(t) \sim U(0,1)$ are random values in the range $[0,1]$, sampled from a uniform distribution. These random values introduce a stochastic element to the algorithm.

The personal best position, $\mathbf{y}_i$, associated with particle i is the best position the particle has visited since the first time step. Considering minimization problems, the

personal best position at the next time step, $t+1$, is calculated as

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \tag{16.3}$$

where $f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ is the fitness function. As with EAs, the fitness function measures how close the corresponding solution is to the optimum, i.e. the fitness function quantifies the performance, or quality, of a particle (or solution).

The global best position, $\hat{\mathbf{y}}(t)$, at time step t, is defined as

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \ldots, \mathbf{y}_{n_s}(t)\} | f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), \ldots, f(\mathbf{y}_{n_s}(t))\} \tag{16.4}$$

where n_s is the total number of particles in the swarm. It is important to note that the definition in equation (16.4) states that $\hat{\mathbf{y}}$ is the best position discovered by any of the particles so far – it is usually calculated as the best personal best position. The global best position can also be selected from the particles of the current swarm, in which case [359]

$$\hat{\mathbf{y}}(t) = \min\{f(\mathbf{x}_0(t)), \ldots, f(\mathbf{x}_{n_s}(t))\} \tag{16.5}$$

The *gbest* PSO is summarized in Algorithm 16.1.

Algorithm 16.1 *gbest* PSO

Create and initialize an n_x-dimensional swarm;
repeat
  for *each particle* $i = 1, \ldots, n_s$ **do**
    //set the personal best position
    if $f(\mathbf{x}_i) < f(\mathbf{y}_i)$ **then**
      $\mathbf{y}_i = \mathbf{x}_i$;
    end
    //set the global best position **if** $f(\mathbf{y}_i) < f(\hat{\mathbf{y}})$ **then**
      $\hat{\mathbf{y}} = \mathbf{y}_i$;
    end
  end
  for *each particle* $i = 1, \ldots, n_s$ **do**
    update the velocity using equation (16.2);
    update the position using equation (16.1);
  end
until *stopping condition is true*;

16.1.2 Local Best PSO

The local best PSO, or *lbest* PSO, uses a ring social network topology (refer to Section 16.2) where smaller neighborhoods are defined for each particle. The social component reflects information exchanged within the neighborhood of the particle, reflecting local knowledge of the environment. With reference to the velocity equation, the social

contribution to particle velocity is proportional to the distance between a particle and the best position found by the neighborhood of particles. The velocity is calculated as

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \tag{16.6}$$

where $\hat{y}_{ij}$ is the best position, found by the neighborhood of particle i in dimension j. The local best particle position, $\hat{\mathbf{y}}_i$, i.e. the best position found in the neighborhood $\mathcal{N}_i$, is defined as

$$\hat{\mathbf{y}}_i(t+1) \in \{\mathcal{N}_i | f(\hat{\mathbf{y}}_i(t+1)) = \min\{f(\mathbf{x})\}, \quad \forall \mathbf{x} \in \mathcal{N}_i\} \tag{16.7}$$

with the neighborhood defined as

$$\mathcal{N}_i = \{\mathbf{y}_{i-n_{\mathcal{N}_i}}(t), \mathbf{y}_{i-n_{\mathcal{N}_i}+1}(t), \ldots, \mathbf{y}_{i-1}(t), \mathbf{y}_i(t), \mathbf{y}_{i+1}(t), \ldots, \mathbf{y}_{i+n_{\mathcal{N}_i}}(t)\} \tag{16.8}$$

for neighborhoods of size $n_{\mathcal{N}_i}$. The local best position will also be referred to as the neighborhood best position.

It is important to note that for the basic PSO, particles within a neighborhood have no relationship to each other. Selection of neighborhoods is done based on particle indices. However, strategies have been developed where neighborhoods are formed based on spatial similarity (refer to Section 16.2).

There are mainly two reasons why neighborhoods based on particle indices are preferred:

1. It is computationally inexpensive, since spatial ordering of particles is not required. For approaches where the distance between particles is used to form neighborhoods, it is necessary to calculate the Euclidean distance between all pairs of particles, which is of $O(n_s^2)$ complexity.
2. It helps to promote the spread of information regarding good solutions to all particles, irrespective of their current location in the search space.

It should also be noted that neighborhoods overlap. A particle takes part as a member of a number of neighborhoods. This interconnection of neighborhoods also facilitates the sharing of information among neighborhoods, and ensures that the swarm converges on a single point, namely the global best particle. The *gbest* PSO is a special case of the *lbest* PSO with $n_{\mathcal{N}_i} = n_s$.

Algorithm 16.2 summarizes the *lbest* PSO.

16.1.3 *gbest* versus *lbest* PSO

The two versions of PSO discussed above are similar in the sense that the social component of the velocity updates causes both to move towards the global best particle. This is possible for the *lbest* PSO due to the overlapping neighborhoods.

There are two main differences between the two approaches with respect to their convergence characteristics [229, 489]:

Algorithm 16.2 *lbest* PSO

Create and initialize an n_x-dimensional swarm;
repeat
  for *each particle* $i = 1, \ldots, n_s$ **do**
    //set the personal best position
    if $f(\mathbf{x}_i) < f(\mathbf{y}_i)$ **then**
      $\mathbf{y}_i = \mathbf{x}_i$;
    end
    //set the neighborhood best position
    if $f(\mathbf{y}_i) < f(\hat{\mathbf{y}}_i)$ **then**
      $\hat{\mathbf{y}} = \mathbf{y}_i$;
    end
  end
  for *each particle* $i = 1, \ldots, n_s$ **do**
    update the velocity using equation (16.6);
    update the position using equation (16.1);
  end
until *stopping condition is true*;

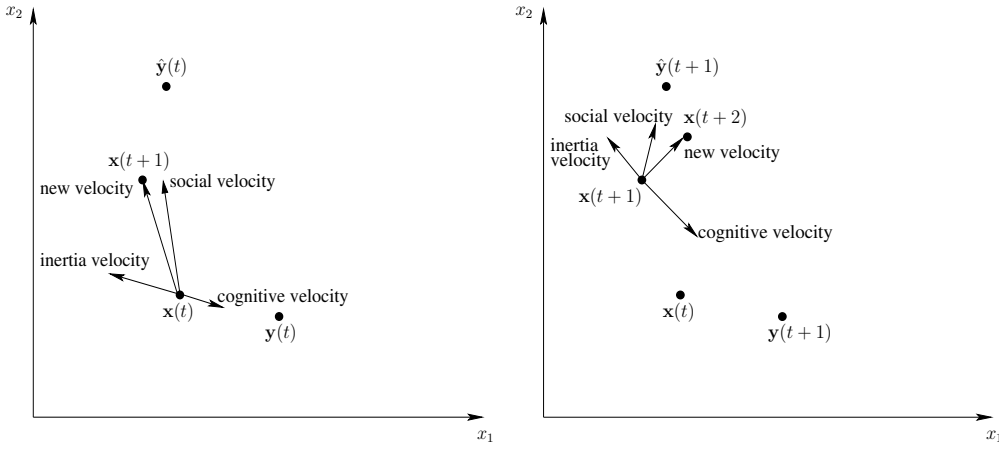
- Due to the larger particle interconnectivity of the *gbest* PSO, it converges faster than the *lbest* PSO. However, this faster convergence comes at the cost of less diversity than the *lbest* PSO.
- As a consequence of its larger diversity (which results in larger parts of the search space being covered), the *lbest* PSO is less susceptible to being trapped in local minima. In general (depending on the problem), neighborhood structures such as the ring topology used in *lbest* PSO improves performance [452, 670].

A more in-depth discussion on neighborhoods can be found in Section 16.2.

16.1.4 Velocity Components

The velocity calculation as given in equations (16.2) and (16.6) consists of three terms:

- The **previous velocity**, $\mathbf{v}_i(t)$, which serves as a memory of the previous flight direction, i.e. movement in the immediate past. This memory term can be seen as a momentum, which prevents the particle from drastically changing direction, and to bias towards the current direction. This component is also referred to as the inertia component.
- The **cognitive component**, $c_1\mathbf{r}_1(\mathbf{y}_i - \mathbf{x}_i)$, which quantifies the performance of particle i relative to past performances. In a sense, the cognitive component resembles individual memory of the position that was best for the particle. The effect of this term is that particles are drawn back to their own best positions, resembling the tendency of individuals to return to situations or places that satisfied them most in the past. Kennedy and Eberhart also referred to the cognitive component as the "nostalgia" of the particle [449].



(a) Time Step t (b) Time Step $t+1$

Figure 16.1 Geometrical Illustration of Velocity and Position Updates for a Single Two-Dimensional Particle

- The **social component**, $c_2\mathbf{r}_2(\hat{\mathbf{y}} - \mathbf{x}_i)$, in the case of the *gbest* PSO or, $c_2\mathbf{r}_2(\hat{\mathbf{y}}_i - \mathbf{x}_i)$, in the case of the *lbest* PSO, which quantifies the performance of particle i relative to a group of particles, or neighbors. Conceptually, the social component resembles a group norm or standard that individuals seek to attain. The effect of the social component is that each particle is also drawn towards the best position found by the particle's neighborhood.

The contribution of the cognitive and social components are weighed by a stochastic amount, $c_1\mathbf{r}_1$ or $c_2\mathbf{r}_2$, respectively. The effects of these weights are discussed in more detail in Section 16.4.

16.1.5 Geometric Illustration

The effect of the velocity equation can easily be illustrated in a two-dimensional vector space. For the sake of the illustration, consider a single particle in a two-dimensional search space.

An example movement of the particle is illustrated in Figure 16.1, where the particle subscript has been dropped for notational convenience. Figure 16.1(a) illustrates the state of the swarm at time step t. Note how the new position, $\mathbf{x}(t+1)$, moves closer towards the global best $\hat{\mathbf{y}}(t)$. For time step $t+1$, as illustrated in Figure 16.1(b), assume that the personal best position does not change. The figure shows how the three components contribute to still move the particle towards the global best particle.

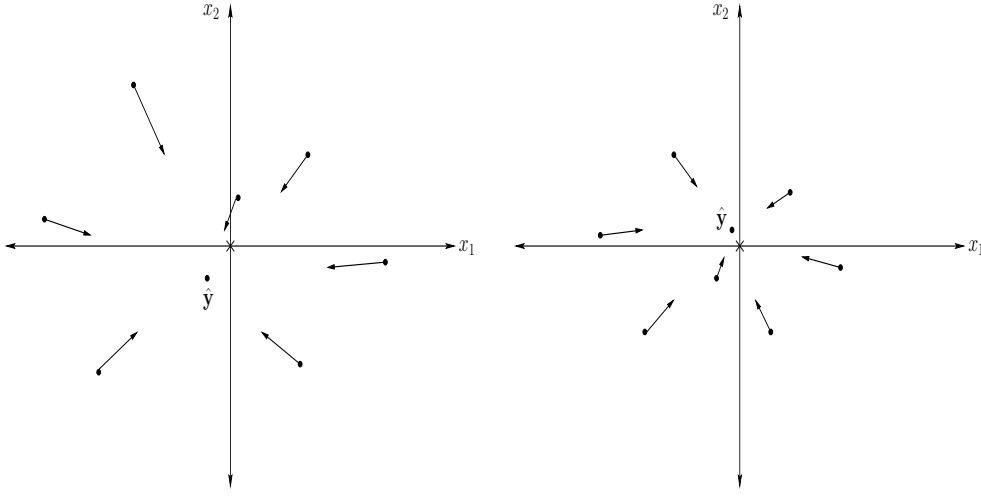
It is of course possible for a particle to overshoot the global best position, mainly due to the momentum term. This results in two scenarios:

1. The new position, as a result of overshooting the current global best, may be a

better position than the current global best. In this case the new particle position will become the new global best position, and all particles will be drawn towards it.

2. The new position is still worse than the current global best particle. In subsequent time steps the cognitive and social components will cause the particle to change direction back towards the global best.

The cumulative effect of all the position updates of a particle is that each particle converges to a point on the line that connects the global best position and the personal best position of the particle. A formal proof can be found in [863, 870].

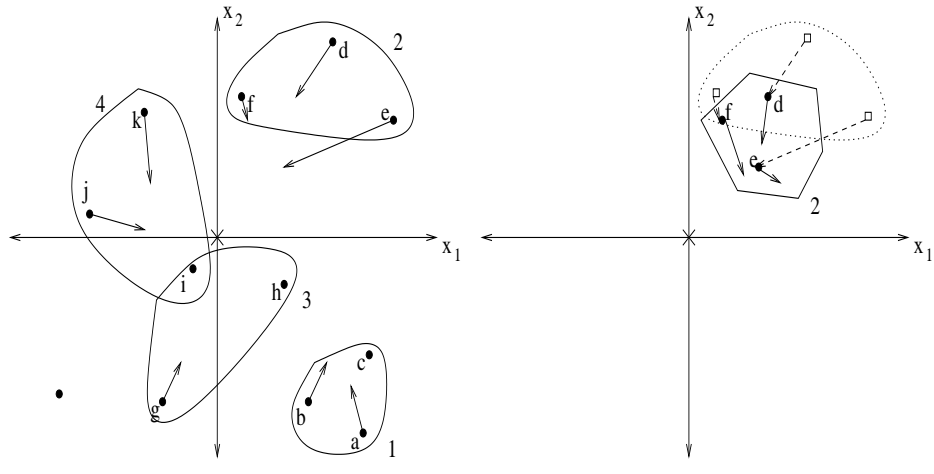


(a) At time $t = 0$ (b) At time $t = 1$

Figure 16.2 Multi-particle *gbest* PSO Illustration

Returning to more than one particle, Figure 16.2 visualizes the position updates with reference to the task of minimizing a two-dimensional function with variables x_1 and x_2 using *gbest* PSO. The optimum is at the origin, indicated by the symbol '$\times$'. Figure 16.2(a) illustrates the initial positions of eight particles, with the global best position as indicated. Since the contribution of the cognitive component is zero for each particle at time step $t = 0$, only the social component has an influence on the position adjustments. Note that the global best position does not change (it is assumed that $\mathbf{v}_i(0) = \mathbf{0}$, for all particles). Figure 16.2(b) shows the new positions of all the particles after the first iteration. A new global best position has been found. Figure 16.2(b) now indicates the influence of all the velocity components, with particles moving towards the new global best position.

Finally, the *lbest* PSO, as illustrated in Figure 16.3, shows how particles are influenced by their immediate neighbors. To keep the graph readable, only some of the movements are illustrated, and only the aggregate velocity direction is indicated. In neighborhood 1, both particles a and b move towards particle c, which is the best solution within that neighborhood. Considering neighborhood 2, particle d moves towards f, so does e. For the next iteration, e will be the best solution for neighborhood 2. Now d and



(a) Local Best Illustrated – Initial Swarm (b) Local Best – Second Swarm

Figure 16.3 Illustration of *lbest* PSO

f move towards e as illustrated in Figure 16.3(b) (only part of the solution space is illustrated). The blocks represent the previous positions. Note that e remains the best solution for neighborhood 2. Also note the general movement towards the minimum.

More in-depth analyses of particle trajectories can be found in [136, 851, 863, 870].

16.1.6 Algorithm Aspects

A few aspects of Algorithms 16.1 and 16.2 still need to be discussed. These aspects include particle initialization, stopping conditions and defining the terms iteration and function evaluation.

With reference to Algorithms 16.1 and 16.2, the optimization process is iterative. Repeated iterations of the algorithms are executed until a stopping condition is satisfied. One such iteration consists of application of all the steps within the `repeat...until` loop, i.e. determining the personal best positions and the global best position, and adjusting the velocity of each particle. Within each iteration, a number of function evaluations (FEs) are performed. A *function evaluation* (FE) refers to one calculation of the fitness function, which characterizes the optimization problem. For the basic PSO, a total of n_s function evaluations are performed per iteration, where n_s is the total number of particles in the swarm.

The first step of the PSO algorithm is to initialize the swarm and control parameters. In the context of the basic PSO, the acceleration constants, c_1 and c_2, the initial velocities, particle positions and personal best positions need to be specified. In addition, the *lbest* PSO requires the size of neighborhoods to be specified. The importance of optimal values for the acceleration constants are discussed in Section 16.4.

Usually, the positions of particles are initialized to uniformly cover the search space. It is important to note that the efficiency of the PSO is influenced by the initial diversity of the swarm, i.e. how much of the search space is covered, and how well particles are distributed over the search space. If regions of the search space are not covered by the initial swarm, the PSO will have difficulty in finding the optimum if it is located within an uncovered region. The PSO will discover such an optimum only if the momentum of a particle carries the particle into the uncovered area, provided that the particle ends up on either a new personal best for itself, or a position that becomes the new global best.

Assume that an optimum needs to be located within the domain defined by the two vectors, $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$, which respectively represents the minimum and maximum ranges in each dimension. Then, an efficient initialization method for the particle positions is:

$$x(0) = x_{min,j} + r_j(x_{max,j} - x_{min,j}), \quad \forall j = 1, \ldots, n_x, \ \forall i = 1, \ldots, n_s \tag{16.9}$$

where $r_j \sim U(0, 1)$.

The initial velocities can be initialized to zero, i.e.

$$\mathbf{v}_i(0) = \mathbf{0} \tag{16.10}$$

While it is possible to also initialize the velocities to random values, it is not necessary, and it must be done with care. In fact, considering physical objects in their initial positions, their velocities are zero – they are stationary. If particles are initialized with nonzero velocities, this physical analogy is violated. Random initialization of position vectors already ensures random positions and moving directions. If, however, velocities are also randomly initialized, such velocities should not be too large. Large initial velocities will have large initial momentum, and consequently large initial position updates. Such large initial position updates may cause particles to leave the boundaries of the search space, and may cause the swarm to take more iterations before particles settle on a single solution.

The personal best position for each particle is initialized to the particle's position at time step $t = 0$, i.e.

$$\mathbf{y}_i(0) = \mathbf{x}_i(0) \tag{16.11}$$

Different initialization schemes have been used by researchers to initialize the particle positions to ensure that the search space is uniformly covered: Sobol sequences [661], Faure sequences [88, 89, 90], and nonlinear simplex method [662].

While it is important for application of the PSO to solve real-world problems, in that particles are uniformly distributed over the entire search space, uniformly distributed particles are not necessarily good for empirical studies of different algorithms. This typical initialization method can give false impressions of the relative performance of algorithms as shown in [312]. For many of the benchmark functions used to evaluate the performance of optimization algorithms (refer, for example, to Section A.5.3), uniform initialization will result in particles being symmetrically distributed around the optimum of the function to be optimized. In most cases it is then trivial for an

optimization algorithm to find the optimum. Gehlhaar and Fogel suggest initializing in areas that do not contain the optima, in order to validate the ability of the algorithm to locate solutions outside the initialized space [312].

The last aspect of the PSO algorithms concerns the stopping conditions, i.e. criteria used to terminate the iterative search process. A number of termination criteria have been used and proposed in the literature. When selecting a termination criterion, two important aspects have to be considered:

1. The stopping condition should not cause the PSO to prematurely converge, since suboptimal solutions will be obtained.
2. The stopping condition should protect against oversampling of the fitness. If a stopping condition requires frequent calculation of the fitness function, computational complexity of the search process can be significantly increased.

The following stopping conditions have been used:

- **Terminate when a maximum number of iterations, or FEs, has been exceeded.** It is obvious to realize that if this maximum number of iterations (or FEs) is too small, termination may occur before a good solution has been found. This criterion is usually used in conjunction with convergence criteria to force termination if the algorithm fails to converge. Used on its own, this criterion is useful in studies where the objective is to evaluate the best solution found in a restricted time period.
- **Terminate when an acceptable solution has been found.** Assume that $\mathbf{x}^*$ represents the optimum of the objective function f. Then, this criterion will terminate the search process as soon as a particle, $\mathbf{x}_i$, is found such that $f(\mathbf{x}_i) \leq |f(\mathbf{x}^*) - \epsilon|$; that is, when an acceptable error has been reached. The value of the threshold, ϵ, has to be selected with care. If ϵ is too large, the search process terminates on a bad, suboptimal solution. On the other hand, if ϵ is too small, the search may not terminate at all. This is especially true for the basic PSO, since it has difficulties in refining solutions [81, 361, 765, 782]. Furthermore, this stopping condition assumes prior knowledge of what the optimum is – which is fine for problems such as training neural networks, where the optimum is usually zero. It is, however, the case that knowledge of the optimum is usually not available.
- **Terminate when no improvement is observed over a number of iterations.** There are different ways in which improvement can be measured. For example, if the average change in particle positions is small, the swarm can be considered to have converged. Alternatively, if the average particle velocity over a number of iterations is approximately zero, only small position updates are made, and the search can be terminated. The search can also be terminated if there is no significant improvement over a number of iterations. Unfortunately, these stopping conditions introduce two parameters for which sensible values need to be found: (1) the window of iterations (or function evaluations) for which the performance is monitored, and (2) a threshold to indicate what constitutes unacceptable performance.
- **Terminate when the normalized swarm radius is close to zero.** When

the normalized swarm radius, calculated as [863]

$$R_{norm} = \frac{R_{max}}{\text{diameter}(S)} \tag{16.12}$$

where diameter(S) is the diameter of the initial swarm and the maximum radius, R_{max}, is

$$R_{max} = ||\mathbf{x}_m - \hat{\mathbf{y}}||, \quad m = 1, \ldots, n_s \tag{16.13}$$

with

$$||\mathbf{x}_m - \hat{\mathbf{y}}|| \geq ||\mathbf{x}_i - \hat{\mathbf{y}}||, \quad \forall i = 1, \ldots, n_s \tag{16.14}$$

is close to zero, the swarm has little potential for improvement, unless the global best is still moving. In the equations above, $||\bullet||$ is a suitable distance norm, e.g. Euclidean distance.

The algorithm is terminated when $R_{norm} < \epsilon$. If ϵ is too large, the search process may stop prematurely before a good solution is found. Alternatively, if ϵ is too small, the search may take excessively more iterations for the particles to form a compact swarm, tightly centered around the global best position.

Algorithm 16.3 Particle Clustering Algorithm

Initialize cluster $C = \{\hat{\mathbf{y}}\}$;
for *about 5 times* **do**
Calculate the centroid of cluster C:

$$\overline{\mathbf{x}} = \frac{\sum_{i=1, \mathbf{x}_i \in C}^{|C|} \mathbf{x}_i}{|C|} \tag{16.15}$$

for $\forall \mathbf{x}_i \in S$ **do**
if $||\mathbf{x}_i - \overline{\mathbf{x}}|| < \epsilon$ **then**
$C \leftarrow C \cup \{\mathbf{x}_i\}$;
end
endFor
endFor

A more aggressive version of the radius method above can be used, where particles are clustered in the search space. Algorithm 16.3 provides a simple particle clustering algorithm from [863]. The result of this algorithm is a single cluster, C. If $|C|/n_s > \delta$, the swarm is considered to have converged. If, for example, $\delta = 0.7$, the search will terminate if at least 70% of the particles are centered around the global best position. The threshold ϵ in Algorithm 16.3 has the same importance as for the radius method above. Similarly, if δ is too small, the search may terminate prematurely.

This clustering approach is similar to the radius approach except that the clustering approach will more readily decide that the swarm has converged.

- **Terminate when the objective function slope is approximately zero**. The stopping conditions above consider the relative positions of particles in the search space, and do not take into consideration information about the slope of

the objective function. To base termination on the rate of change in the objective function, consider the ratio [863],

$$f^{'}(t) = \frac{f(\hat{\mathbf{y}}(t)) - f(\hat{\mathbf{y}}(t-1))}{f(\hat{\mathbf{y}}(t))} \tag{16.16}$$

If $f^{'}(t) < \epsilon$ for a number of consecutive iterations, the swarm is assumed to have converged. This approximation to the slope of the objective function is superior to the methods above, since it actually determines if the swarm is still making progress using information about the search space.

The objective function slope approach has, however, the problem that the search will be terminated if some of the particles are attracted to a local minimum, irrespective of whether other particles may still be busy exploring other parts of the search space. It may be the case that these exploring particles could have found a better solution had the search not terminated. To solve this problem, the objective function slope method can be used in conjunction with the radius or cluster methods to test if all particles have converged to the same point before terminating the search process.

In the above, convergence does not imply that the swarm has settled on an optimum (local or global). With the term *convergence* is meant that the swarm has reached an equilibrium, i.e. just that the particles converged to a point, which is not necessarily an optimum [863].

16.2 Social Network Structures

The feature that drives PSO is social interaction. Particles within the swarm learn from each other and, on the basis of the knowledge obtained, move to become more similar to their "better" neighbors. The social structure for PSO is determined by the formation of overlapping neighborhoods, where particles within a neighborhood influence one another. This is in analogy with observations of animal behavior, where an organism is most likely to be influenced by others in its neighborhood, and where organisms that are more successful will have a greater influence on members of the neighborhood than the less successful.

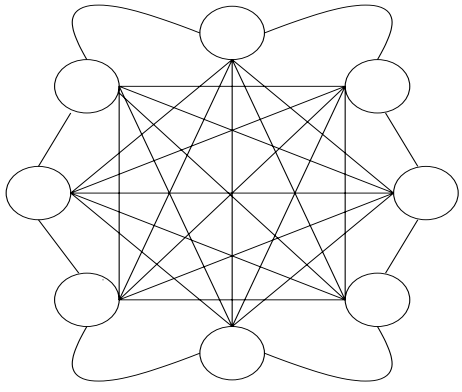
Within the PSO, particles in the same neighborhood communicate with one another by exchanging information about the success of each particle in that neighborhood. All particles then move towards some quantification of what is believed to be a better position. The performance of the PSO depends strongly on the structure of the social network. The flow of information through a social network, depends on (1) the degree of connectivity among nodes (members) of the network, (2) the amount of clustering (clustering occurs when a node's neighbors are also neighbors to one another), and (3) the average shortest distance from one node to another [892].

With a highly connected social network, most of the individuals can communicate with one another, with the consequence that information about the perceived best member quickly filters through the social network. In terms of optimization, this means faster

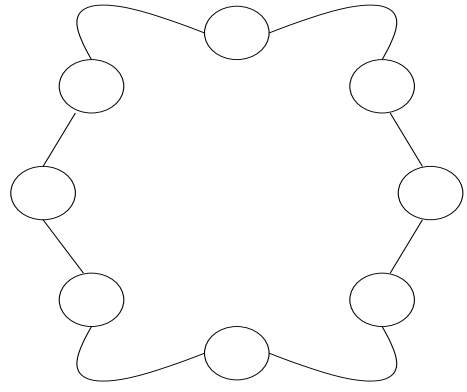
convergence to a solution than for less connected networks. However, for the highly connected networks, the faster convergence comes at the price of susceptibility to local minima, mainly due to the fact that the extent of coverage in the search space is less than for less connected social networks. For sparsely connected networks with a large amount of clustering in neighborhoods, it can also happen that the search space is not covered sufficiently to obtain the best possible solutions. Each cluster contains individuals in a tight neighborhood covering only a part of the search space. Within these network structures there usually exist a few clusters, with a low connectivity between clusters. Consequently information on only a limited part of the search space is shared with a slow flow of information between clusters.

Different social network structures have been developed for PSO and empirically studied. This section overviews only the original structures investigated [229, 447, 452, 575]:

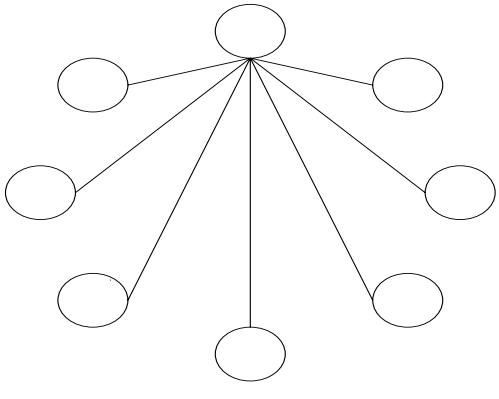
- The **star** social structure, where all particles are interconnected as illustrated in Figure 16.4(a). Each particle can therefore communicate with every other particle. In this case each particle is attracted towards the best solution found by the entire swarm. Each particle therefore imitates the overall best solution. The first implementation of the PSO used a star network structure, with the resulting algorithm generally being referred to as the *gbest* PSO. The *gbest* PSO has been shown to converge faster than other network structures, but with a susceptibility to be trapped in local minima. The *gbest* PSO performs best for unimodal problems.
- The **ring** social structure, where each particle communicates with its $n_{\mathcal{N}}$ immediate neighbors. In the case of $n_{\mathcal{N}} = 2$, a particle communicates with its immediately adjacent neighbors as illustrated in Figure 16.4(b). Each particle attempts to imitate its best neighbor by moving closer to the best solution found within the neighborhood. It is important to note from Figure 16.4(b) that neighborhoods overlap, which facilitates the exchange of information between neighborhoods and, in the end, convergence to a single solution. Since information flows at a slower rate through the social network, convergence is slower, but larger parts of the search space are covered compared to the star structure. This behavior allows the ring structure to provide better performance in terms of the quality of solutions found for multi-modal problems than the star structure. The resulting PSO algorithm is generally referred to as the *lbest* PSO.
- The **wheel** social structure, where individuals in a neighborhood are isolated from one another. One particle serves as the focal point, and all information is communicated through the focal particle (refer to Figure 16.4(c)). The focal particle compares the performances of all particles in the neighborhood, and adjusts its position towards the best neighbor. If the new position of the focal particle results in better performance, then the improvement is communicated to all the members of the neighborhood. The wheel social network slows down the propagation of good solutions through the swarm.
- The **pyramid** social structure, which forms a three-dimensional wire-frame as illustrated in Figure 16.4(d).
- The **four clusters** social structure, as illustrated in Figure 16.4(e). In this



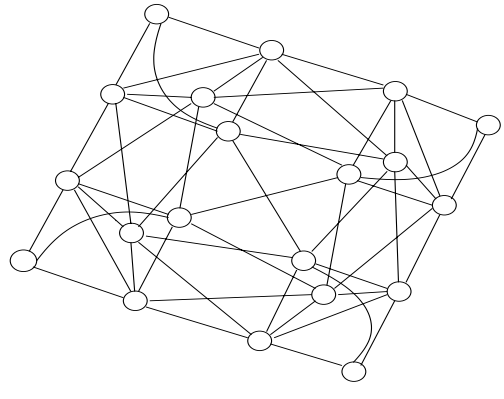
(a) Star



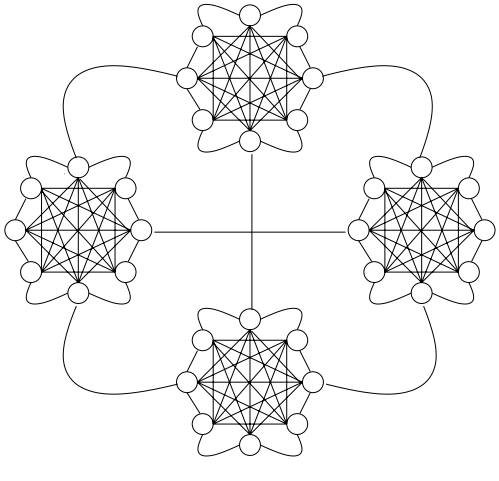
(b) Ring



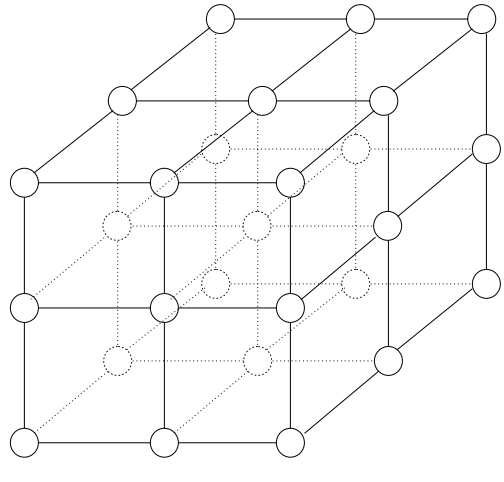
(c) Wheel



(d) Pyramid



(e) Four Clusters



(f) Von Neumann

Figure 16.4 Example Social Network Structures

network structure, four clusters (or cliques) are formed with two connections between clusters. Particles within a cluster are connected with five neighbors.

- The **Von Neumann** social structure, where particles are connected in a grid structure as illustrated in Figure 16.4(f). The Von Neumann social network has been shown in a number of empirical studies to outperform other social networks in a large number of problems [452, 670].

While many studies have been done using the different topologies, there is no outright best topology for all problems. In general, the fully connected structures perform best for unimodal problems, while the less connected structures perform better on multi-modal problems, depending on the degree of particle interconnection [447, 452, 575, 670].

Neighborhoods are usually determined on the basis of particle indices. For example, for the *lbest* PSO with $n_{\mathcal{N}} = 2$, the neighborhood of a particle with index i includes particles $i-1, i$ and $i+1$. While indices are usually used, Suganthan based neighborhoods on the Euclidean distance between particles [820].

16.3 Basic Variations

The basic PSO has been applied successfully to a number of problems, including standard function optimization problems [25, 26, 229, 450, 454], solving permutation problems [753] and training multi-layer neural networks [224, 225, 229, 446, 449, 854]. While the empirical results presented in these papers illustrated the ability of the PSO to solve optimization problems, these results also showed that the basic PSO has problems with consistently converging to good solutions. A number of basic modifications to the basic PSO have been developed to improve speed of convergence and the quality of solutions found by the PSO. These modifications include the introduction of an inertia weight, velocity clamping, velocity constriction, different ways of determining the personal best and global best (or local best) positions, and different velocity models. This section discusses these basic modifications.

16.3.1 Velocity Clamping

An important aspect that determines the efficiency and accuracy of an optimization algorithm is the exploration–exploitation trade-off. *Exploration* is the ability of a search algorithm to explore different regions of the search space in order to locate a good optimum. *Exploitation*, on the other hand, is the ability to concentrate the search around a promising area in order to refine a candidate solution. A good optimization algorithm optimally balances these contradictory objectives. Within the PSO, these objectives are addressed by the velocity update equation.

The velocity updates in equations (16.2) and (16.6) consist of three terms that contribute to the step size of particles. In the early applications of the basic PSO, it was found that the velocity quickly explodes to large values, especially for particles far

from the neighborhood best and personal best positions. Consequently, particles have large position updates, which result in particles leaving the boundaries of the search space – the particles diverge. To control the global exploration of particles, velocities are clamped to stay within boundary constraints [229]. If a particle's velocity exceeds a specified maximum velocity, the particle's velocity is set to the maximum velocity. Let $V_{max,j}$ denote the maximum allowed velocity in dimension j. Particle velocity is then adjusted before the position update using,

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1) & \text{if } v'_{ij}(t+1) < V_{max,j} \\ V_{max,j} & \text{if } v'_{ij}(t+1) \geq V_{max,j} \end{cases} \tag{16.17}$$

where v'_{ij} is calculated using equation (16.2) or (16.6).

The value of $V_{max,j}$ is very important, since it controls the granularity of the search by clamping escalating velocities. Large values of $V_{max,j}$ facilitate global exploration, while smaller values encourage local exploitation. If $V_{max,j}$ is too small, the swarm may not explore sufficiently beyond locally good regions. Also, too small values for $V_{max,j}$ increase the number of time steps to reach an optimum. Furthermore, the swarm may become trapped in a local optimum, with no means of escape. On the other hand, too large values of $V_{max,j}$ risk the possibility of missing a good region. The particles may jump over good solutions, and continue to search in fruitless regions of the search space. While large values do have the disadvantage that particles may jump over optima, particles are moving faster.

This leaves the problem of finding a good value for each $V_{max,j}$ in order to balance between (1) moving too fast or too slow, and (2) exploration and exploitation. Usually, the $V_{max,j}$ values are selected to be a fraction of the domain of each dimension of the search space. That is,

$$V_{max,j} = \delta(x_{max,j} - x_{min,j}) \tag{16.18}$$

where $x_{max,j}$ and $x_{min,j}$ are respectively the maximum and minimum values of the domain of $\mathbf{x}$ in dimension j, and $\delta \in (0, 1]$. The value of δ is problem-dependent, as was found in a number of empirical studies [638, 781]. The best value should be found for each different problem using empirical techniques such as cross-validation.

There are two important aspects of the velocity clamping approach above that the reader should be aware of:

1. Velocity clamping does not confine the positions of particles, only the step sizes as determined from the particle velocity.
2. In the above equations, explicit reference is made to the dimension, j. A maximum velocity is associated with each dimension, proportional to the domain of that dimension. For the sake of the argument, assume that all dimensions are clamped with the same constant V_{max}. Therefore if a dimension, j, exists such that $x_{max,j} - x_{min,j} << V_{max}$, particles may still overshoot an optimum in dimension j.

While velocity clamping has the advantage that explosion of velocity is controlled, it also has disadvantages that the user should be aware of (that is in addition to

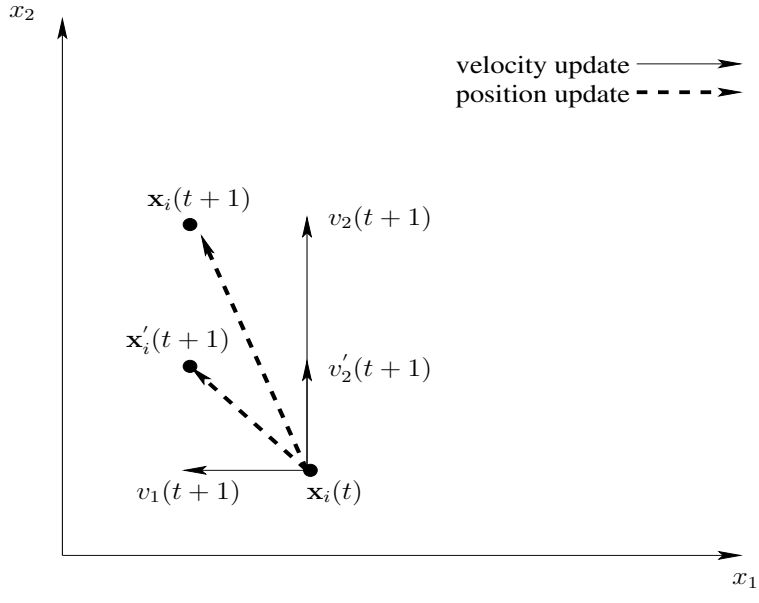


Figure 16.5 Effects of Velocity Clamping

the problem-dependent nature of $V_{max,j}$ values). Firstly, velocity clamping changes not only the step size, but also the direction in which a particle moves. This effect is illustrated in Figure 16.5 (assuming two-dimensional particles). In this figure, $\mathbf{x}_i(t+1)$ denotes the position of particle i without using velocity clamping. The position $\mathbf{x}_i'(t+1)$ is the result of velocity clamping on the second dimension. Note how the search direction and the step size have changed. It may be said that these changes in search direction allow for better exploration. However, it may also cause the optimum not to be found at all.

Another problem with velocity clamping occurs when all velocities are equal to the maximum velocity. If no measures are implemented to prevent this situation, particles remain to search on the boundaries of a hypercube defined by $[\mathbf{x}_i(t) - \mathbf{V}_{max}, \mathbf{x}_i(t) + \mathbf{V}_{max}]$. It is possible that a particle may stumble upon the optimum, but in general the swarm will have difficulty in exploiting this local area. This problem can be solved in different ways, with the introduction of an inertia weight (refer to Section 16.3.2) being one of the first solutions. The problem can also be solved by reducing $V_{max,j}$ over time. The idea is to start with large values, allowing the particles to explore the search space, and then to decrease the maximum velocity over time. The decreasing maximum velocity constrains the exploration ability in favor of local exploitation at mature stages of the optimization process. The following dynamic velocity approaches have been used:

- Change the maximum velocity when no improvement in the global best position

has been seen over τ consecutive iterations [766]:

$$V_{max,j}(t+1) = \begin{cases} \gamma V_{max,j}(t) & \text{if } f(\hat{\mathbf{y}}(t)) \geq f(\hat{\mathbf{y}}(t-t^{'})) \quad \forall\, t^{'} = 1, \ldots, n_{t'} \\ V_{max,j}(t) & \text{otherwise} \end{cases} \tag{16.19}$$

where γ decreases from 1 to 0.01 (the decrease can be linear or exponential using an annealing schedule similar to that given in Section 16.3.2 for the inertia weight).

- Exponentially decay the maximum velocity, using [254]

$$V_{max,j}(t+1) = (1 - (t/n_t)^{\alpha}) V_{max,j}(t) \tag{16.20}$$

where α is a positive constant, found by trial and error, or cross-validation methods; n_t is the maximum number of time steps (or iterations).

Finally, the sensitivity of PSO to the value of δ (refer to equation (16.18)) can be reduced by constraining velocities using the hyperbolic tangent function, i.e.

$$v_{ij}(t+1) = V_{max,j} \tanh\left(\frac{v_{ij}^{'}(t+1)}{V_{max,j}}\right) \tag{16.21}$$

where $v_{ij}^{'}(t+1)$ is calculated from equation (16.2) or (16.6).

16.3.2 Inertia Weight

The inertia weight was introduced by Shi and Eberhart [780] as a mechanism to control the exploration and exploitation abilities of the swarm, and as a mechanism to eliminate the need for velocity clamping [227]. The inertia weight was successful in addressing the first objective, but could not completely eliminate the need for velocity clamping. The inertia weight, w, controls the momentum of the particle by weighing the contribution of the previous velocity – basically controlling how much memory of the previous flight direction will influence the new velocity. For the *gbest* PSO, the velocity equation changes from equation (16.2) to

$$v_{ij}(t+1) = w v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \tag{16.22}$$

A similar change is made for the *lbest* PSO.

The value of w is extremely important to ensure convergent behavior, and to optimally tradeoff exploration and exploitation. For $w \geq 1$, velocities increase over time, accelerating towards the maximum velocity (assuming velocity clamping is used), and the swarm diverges. Particles fail to change direction in order to move back towards promising areas. For $w < 1$, particles decelerate until their velocities reach zero (depending on the values of the acceleration coefficients). Large values for w facilitate exploration, with increased diversity. A small w promotes local exploitation. However, too small values eliminate the exploration ability of the swarm. Little momentum is then preserved from the previous time step, which enables quick changes in direction. The smaller w, the more do the cognitive and social components control position updates.

As with the maximum velocity, the optimal value for the inertia weight is problem-dependent [781]. Initial implementations of the inertia weight used a static value for the entire search duration, for all particles for each dimension. Later implementations made use of dynamically changing inertia values. These approaches usually start with large inertia values, which decreases over time to smaller values. In doing so, particles are allowed to explore in the initial search steps, while favoring exploitation as time increases. At this time it is crucial to mention the important relationship between the values of w, and the acceleration constants. The choice of value for w has to be made in conjunction with the selection of the values for c_1 and c_2. Van den Bergh and Engelbrecht [863, 870] showed that

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{16.23}$$

guarantees convergent particle trajectories. If this condition is not satisfied, divergent or cyclic behavior may occur. A similar condition was derived by Trelea [851].

Approaches to dynamically varying the inertia weight can be grouped into the following categories:

- **Random adjustments**, where a different inertia weight is randomly selected at each iteration. One approach is to sample from a Gaussian distribution, e.g.

$$w \sim N(0.72, \sigma) \tag{16.24}$$

  where σ is small enough to ensure that w is not predominantly greater than one. Alternatively, Peng *et al.* used [673]

$$w = (c_1 r_1 + c_2 r_2) \tag{16.25}$$

  with no random scaling of the cognitive and social components.

- **Linear decreasing**, where an initially large inertia weight (usually 0.9) is linearly decreased to a small value (usually 0.4). From Naka *et al.* [619], Ratnaweera *et al.* [706], Suganthan [820], Yoshida *et al.* [941]

$$w(t) = (w(0) - w(n_t))\frac{(n_t - t)}{n_t} + w(n_t) \tag{16.26}$$

  where n_t is the maximum number of time steps for which the algorithm is executed, $w(0)$ is the initial inertia weight, $w(n_t)$ is the final inertia weight, and $w(t)$ is the inertia at time step t. Note that $w(0) > w(n_t)$.

- **Nonlinear decreasing**, where an initially large value decreases nonlinearly to a small value. Nonlinear decreasing methods allow a shorter exploration time than the linear decreasing methods, with more time spent on refining solutions (exploiting). Nonlinear decreasing methods will be more appropriate for smoother search spaces. The following nonlinear methods have been defined:
  - From Peram *et al.* [675],

$$w(t+1) = \frac{(w(t) - 0.4)(n_t - t)}{n_t + 0.4} \tag{16.27}$$

    with $w(0) = 0.9$.

– From Venter and Sobieszczanski-Sobieski [874, 875],

$$w(t+1) = \alpha w(t^{'}) \tag{16.28}$$

where $\alpha = 0.975$, and $t^{'}$ is the time step when the inertia last changed. The inertia is only changed when there is no significant difference in the fitness of the swarm. Venter and Sobieszczanski-Sobieski measure the variation in particle fitness of a 20% subset of randomly selected particles. If this variation is too small, the inertia is changed. An initial inertia weight of $w(0) = 1.4$ is used with a lower bound of $w(n_t) = 0.35$. The initial $w(0) = 1.4$ ensures that a large area of the search space is covered before the swarm focuses on refining solutions.

– Clerc proposes an adaptive inertia weight approach where the amount of change in the inertia value is proportional to the relative improvement of the swarm [134]. The inertia weight is adjusted according to

$$w_i(t+1) = w(0) + (w(n_t) - w(0))\frac{e^{m_i(t)} - 1}{e^{m_i(t)} + 1} \tag{16.29}$$

where the relative improvement, m_i, is estimated as

$$m_i(t) = \frac{f(\hat{\mathbf{y}}_i(t)) - f(\mathbf{x}_i(t))}{f(\hat{\mathbf{y}}_i(t)) + f(\mathbf{x}_i(t))} \tag{16.30}$$

with $w(n_t) \approx 0.5$ and $w(0) < 1$.

Using this approach, which was developed for velocity updates without the cognitive component, each particle has its own inertia weight based on its distance from the local best (or neighborhood best) position. The local best position, $\hat{\mathbf{y}}_i(t)$ can just as well be replaced with the global best position $\hat{\mathbf{y}}(t)$.

Clerc motivates his approach by considering that the more an individual improves upon his/her neighbors, the more he/she follows his/her own way, and vice versa. Clerc reported that this approach results in fewer iterations [134].

- **Fuzzy adaptive inertia**, where the inertia weight is dynamically adjusted on the basis of fuzzy sets and rules. Shi and Eberhart [783] defined a fuzzy system for the inertia adaptation to consist of the following components:
  – Two inputs, one to represent the fitness of the global best position, and the other the current value of the inertia weight.
  – One output to represent the change in inertia weight.
  – Three fuzzy sets, namely LOW, MEDIUM and HIGH, respectively implemented as a left triangle, triangle and right triangle membership function [783].
  – Nine fuzzy rules from which the change in inertia is calculated. An example rule in the fuzzy system is [229, 783]:

```
if normalized best fitness is LOW, and
  current inertia weight value is LOW
then the change in weight is MEDIUM
```

- **Increasing inertia**, where the inertia weight is linearly increased from 0.4 to 0.9 [958].

The linear and nonlinear adaptive inertia methods above are very similar to the temperature schedule of simulated annealing [467, 649] (also refer to Section A.5.2).

16.3.3 Constriction Coefficient

Clerc developed an approach very similar to the inertia weight to balance the exploration–exploitation trade-off, where the velocities are constricted by a constant χ, referred to as the constriction coefficient [133, 136]. The velocity update equation changes to:

$$v_{ij}(t+1) = \chi[v_{ij}(t) + \phi_1(y_{ij}(t) - x_{ij}(t)) + \phi_2(\hat{y}_j(t) - x_{ij}(t))] \tag{16.31}$$

where

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi(\phi-4)}|} \tag{16.32}$$

with $\phi = \phi_1 + \phi_2, \phi_1 = c_1 r_1$ and $\phi_2 = c_2 r_2$. Equation (16.32) is used under the constraints that $\phi \geq 4$ and $\kappa \in [0, 1]$. The above equations were derived from a formal eigenvalue analysis of swarm dynamics [136].

The constriction approach was developed as a natural, dynamic way to ensure convergence to a stable point, without the need for velocity clamping. Under the conditions that $\phi \geq 4$ and $\kappa \in [0, 1]$, the swarm is guaranteed to converge. The constriction coefficient, χ, evaluates to a value in the range $[0, 1]$ which implies that the velocity is reduced at each time step.

The parameter, κ, in equation (16.32) controls the exploration and exploitation abilities of the swarm. For $\kappa \approx 0$, fast convergence is obtained with local exploitation. The swarm exhibits an almost hill-climbing behavior. On the other hand, $\kappa \approx 1$ results in slow convergence with a high degree of exploration. Usually, κ is set to a constant value. However, an initial high degree of exploration with local exploitation in the later search phases can be achieved using an initial value close to one, decreasing it to zero.

The constriction approach is effectively equivalent to the inertia weight approach. Both approaches have the objective of balancing exploration and exploitation, and in doing so of improving convergence time and the quality of solutions found. Low values of w and χ result in exploitation with little exploration, while large values result in exploration with difficulties in refining solutions. For a specific χ, the equivalent inertia model can be obtained by simply setting $w = \chi, \phi_1 = \chi c_1 r_1$ and $\phi_2 = \chi c_2 r_2$. The differences in the two approaches are that

- velocity clamping is not necessary for the constriction model,
- the constriction model guarantees convergence under the given constraints, and
- any ability to regulate the change in direction of particles must be done via the constants ϕ_1 and ϕ_2 for the constriction model.

While it is not necessary to use velocity clamping with the constriction model, Eberhart and Shi showed empirically that if velocity clamping and constriction are used together, faster convergence rates can be obtained [226].

16.3.4 Synchronous versus Asynchronous Updates

The *gbest* and *lbest* PSO algorithms presented in Algorithms 16.1 and 16.2 perform synchronous updates of the personal best and global (or local) best positions. Synchronous updates are done separately from the particle position updates. Alternatively, asynchronous updates calculate the new best positions after each particle position update (very similar to a steady state GA, where offspring are immediately introduced into the population). Asynchronous updates have the advantage that immediate feedback is given about the best regions of the search space, while feedback with synchronous updates is only given once per iteration. Carlisle and Dozier reason that asynchronous updates are more important for *lbest* PSO where immediate feedback will be more beneficial in loosely connected swarms, while synchronous updates are more appropriate for *gbest* PSO [108].

Selection of the global (or local) best positions is usually done by selecting the absolute best position found by the swarm (or neighborhood). Kennedy proposed to select the best positions randomly from the neighborhood [448]. This is done to break the effect that one, potentially bad, solution drives the swarm. The random selection was specifically used to address the difficulties that the *gbest* PSO experience on highly multi-modal problems. The performance of the basic PSO is also strongly influenced by whether the best positions (*gbest* or *lbest*) are selected from the particle positions of the current iterations, or from the personal best positions of all particles. The difference between the two approaches is that the latter includes a memory component in the sense that the best positions are the best positions found over all iterations. The former approach neglects the temporal experience of the swarm. Selection from the personal best positions is similar to the "hall of fame" concept (refer to Sections 8.5.9 and 15.2.1) used within evolutionary computation.

16.3.5 Velocity Models

Kennedy [446] investigated a number of variations to the full PSO models presented in Sections 16.1.1 and 16.1.2. These models differ in the components included in the velocity equation, and how best positions are determined. This section summarizes these models.

Cognition-Only Model

The cognition-only model excludes the social component from the original velocity equation as given in equation (16.2). For the cognition-only model, the velocity update

changes to

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) \tag{16.33}$$

The above formulation excludes the inertia weight, mainly because the velocity models in this section were investigated before the introduction of the inertia weight. However, nothing prevents the inclusion of w in equation (16.33) and the velocity equations that follow in this section.

The behavior of particles within the cognition-only model can be likened to nostalgia, and illustrates a stochastic tendency for particles to return toward their previous best position.

From empirical work, Kennedy reported that the cognition-only model is slightly more vulnerable to failure than the full model [446]. It tends to locally search in areas where particles are initialized. The cognition-only model is slower in the number of iterations it requires to reach a good solution, and fails when velocity clamping and the acceleration coefficient are small. The poor performance of the cognitive model is confirmed by Carlisle and Dozier [107], but with respect to dynamic changing environments (refer to Section 16.6.3). The cognition-only model was, however, successfully used within niching algorithms [89] (also refer to Section 16.6.4).

Social-Only Model

The social-only model excludes the cognitive component from the velocity equation:

$$v_{ij}(t+1) = v_{ij}(t) + c_2 r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t)) \tag{16.34}$$

for the *gbest* PSO. For the *lbest* PSO, $\hat{y}_j$ is simply replaced with $\hat{y}_{ij}$.

For the social-only model, particles have no tendency to return to previous best positions. All particles are attracted towards the best position of their neighborhood.

Kennedy empirically illustrated that the social-only model is faster and more efficient than the full and cognitive models [446], which is also confirmed by the results from Carlisle and Dozier [107] for dynamic environments.

Selfless Model

The selfless model is basically the social model, but with the neighborhood best solution only chosen from a particle's neighbors. In other words, the particle itself is not allowed to become the neighborhood best. Kennedy showed the selfless model to be faster than the social-only model for a few problems [446]. Carlisle and Dozier's results show that the selfless model performs poorly for dynamically changing environments [107].

16.4 Basic PSO Parameters

The basic PSO is influenced by a number of control parameters, namely the dimension of the problem, number of particles, acceleration coefficients, inertia weight, neighborhood size, number of iterations, and the random values that scale the contribution of the cognitive and social components. Additionally, if velocity clamping or constriction is used, the maximum velocity and constriction coefficient also influence the performance of the PSO. This section discusses these parameters.

The influence of the inertia weight, velocity clamping threshold and constriction coefficient has been discussed in Section 16.3. The rest of the parameters are discussed below:

- **Swarm size**, n_s, i.e. the number of particles in the swarm: the more particles in the swarm, the larger the initial diversity of the swarm – provided that a good uniform initialization scheme is used to initialize the particles. A large swarm allows larger parts of the search space to be covered per iteration. However, more particles increase the per iteration computational complexity, and the search degrades to a parallel random search. It is also the case that more particles may lead to fewer iterations to reach a good solution, compared to smaller swarms. It has been shown in a number of empirical studies that the PSO has the ability to find optimal solutions with small swarm sizes of 10 to 30 particles [89, 865]. Success has even been obtained for fewer than 10 particles [863]. While empirical studies give a general heuristic of $n_s \in [10, 30]$, the optimal swarm size is problem-dependent. A smooth search space will need fewer particles than a rough surface to locate optimal solutions. Rather than using the heuristics found in publications, it is best that the value of n_s be optimized for each problem using cross-validation methods.
- **Neighborhood size**: The neighborhood size defines the extent of social interaction within the swarm. The smaller the neighborhoods, the less interaction occurs. While smaller neighborhoods are slower in convergence, they have more reliable convergence to optimal solutions. Smaller neighborhood sizes are less susceptible to local minima. To capitalize on the advantages of small and large neighborhood sizes, start the search with small neighborhoods and increase the neighborhood size proportionally to the increase in number of iterations [820]. This approach ensures an initial high diversity with faster convergence as the particles move towards a promising search area.
- **Number of iterations**: The number of iterations to reach a good solution is also problem-dependent. Too few iterations may terminate the search prematurely. A too large number of iterations has the consequence of unnecessary added computational complexity (provided that the number of iterations is the only stopping condition).
- **Acceleration coefficients**: The acceleration coefficients, c_1 and c_2, together with the random vectors $\mathbf{r}_1$ and $\mathbf{r}_2$, control the stochastic influence of the cognitive and social components on the overall velocity of a particle. The constants c_1 and c_2 are also referred to as trust parameters, where c_1 expresses how much

confidence a particle has in itself, while c_2 expresses how much confidence a particle has in its neighbors. With $c_1 = c_2 = 0$, particles keep flying at their current speed until they hit a boundary of the search space (assuming no inertia). If $c_1 > 0$ and $c_2 = 0$, all particles are independent hill-climbers. Each particle finds the best position in its neighborhood by replacing the current best position if the new position is better. Particles perform a local search. On the other hand, if $c_2 > 0$ and $c_1 = 0$, the entire swarm is attracted to a single point, $\hat{\mathbf{y}}$. The swarm turns into one stochastic hill-climber.

Particles draw their strength from their cooperative nature, and are most effective when nostalgia (c_1) and envy (c_2) coexist in a good balance, i.e. $c_1 \approx c_2$. If $c_1 = c_2$, particles are attracted towards the average of $\mathbf{y}_i$ and $\hat{\mathbf{y}}$ [863, 870]. While most applications use $c_1 = c_2$, the ratio between these constants is problem-dependent. If $c_1 >> c_2$, each particle is much more attracted to its own personal best position, resulting in excessive wandering. On the other hand, if $c_2 >> c_1$, particles are more strongly attracted to the global best position, causing particles to rush prematurely towards optima. For unimodal problems with a smooth search space, a larger social component will be efficient, while rough multi-modal search spaces may find a larger cognitive component more advantageous.

Low values for c_1 and c_2 result in smooth particle trajectories, allowing particles to roam far from good regions to explore before being pulled back towards good regions. High values cause more acceleration, with abrupt movement towards or past good regions.

Usually, c_1 and c_2 are static, with their optimized values being found empirically. Wrong initialization of c_1 and c_2 may result in divergent or cyclic behavior [863, 870].

Clerc [134] proposed a scheme for adaptive acceleration coefficients, assuming the social velocity model (refer to Section 16.3.5):

$$c_2(t) = \frac{c_{2,min} + c_{2,max}}{2} + \frac{c_{2,max} - c_{2,min}}{2} + \frac{e^{-m_i(t)} - 1}{e^{-m_i(t)} + 1} \tag{16.35}$$

where m_i is as defined in equation (16.30). The formulation of equation (16.30) implies that each particle has its own adaptive acceleration as a function of the slope of the search space at the current position of the particle.

Ratnaweera *et al.* [706] builds further on a suggestion by Suganthan [820] to linearly adapt the values of c_1 and c_2. Suganthan suggested that both acceleration coefficients be linearly decreased, but reported no improvement in performance using this scheme [820]. Ratnaweera *et al.* proposed that c_1 decreases linearly over time, while c_2 increases linearly [706]. This strategy focuses on exploration in the early stages of optimization, while encouraging convergence to a good optimum near the end of the optimization process by attracting particles more towards the neighborhood best (or global best) positions. The values of $c_1(t)$ and $c_2(t)$ at time step t is calculated as

$$c_1(t) = (c_{1,min} - c_{1,max})\frac{t}{n_t} + c_{1,max} \tag{16.36}$$

$$c_2(t) = (c_{2,max} - c_{2,min})\frac{t}{n_t} + c_{2,min} \tag{16.37}$$

where $c_{1,max} = c_{2,max} = 2.5$ and $c_{1,min} = c_{2,min} = 0.5$.

A number of theoretical studies have shown that the convergence behavior of PSO is sensitive to the values of the inertia weight and the acceleration coefficients [136, 851, 863, 870]. These studies also provide guidelines to choose values for PSO parameters that will ensure convergence to an equilibrium point. The first set of guidelines are obtained from the different constriction models suggested by Clerc and Kennedy [136]. For a specific constriction model and selected ϕ value, the value of the constriction coefficient is calculated to ensure convergence.

For an unconstricted simplified PSO system that includes inertia, the trajectory of a particle converges if the following conditions hold [851, 863, 870, 937]:

$$1 > w > \frac{1}{2}(\phi_1 + \phi_2) - 1 \geq 0 \tag{16.38}$$

and $0 \leq w < 1$. Since $\phi_1 = c_1 U(0,1)$ and $\phi_2 = c_2 U(0,1)$, the acceleration coefficients, c_1 and c_2 serve as upper bounds of ϕ_1 and ϕ_2. Equation (16.38) can then be rewritten as

$$1 > w > \frac{1}{2}(c_1 + c_2) - 1 \geq 0 \tag{16.39}$$

Therefore, if w, c_1 and c_2 are selected such that the condition in equation (16.39) holds, the system has guaranteed convergence to an equilibrium state.

The heuristics above have been derived for the simplified PSO system with no stochastic component. It can happen that, for stochastic ϕ_1 and ϕ_2 and a w that violates the condition stated in equation (16.38), the swarm may still converge. The stochastic trajectory illustrated in Figure 16.6 is an example of such behavior. The particle follows a convergent trajectory for most of the time steps, with an occasional divergent step.

Van den Bergh and Engelbrecht show in [863, 870] that convergent behavior will be observed under stochastic ϕ_1 and ϕ_2 if the ratio,

$$\phi_{ratio} = \frac{\phi_{crit}}{c_1 + c_2} \tag{16.40}$$

is close to 1.0, where

$$\phi_{crit} = \sup \phi \,|\, 0.5\,\phi - 1 < w, \quad \phi \in (0, c_1 + c_2] \tag{16.41}$$

It is even possible that parameter choices for which $\phi_{ratio} = 0.5$, may lead to convergent behavior, since particles spend 50% of their time taking a step along a convergent trajectory.

16.5 Single-Solution Particle Swarm Optimization

Initial empirical studies of the basic PSO and basic variations as discussed in this chapter have shown that the PSO is an efficient optimization approach – for the benchmark

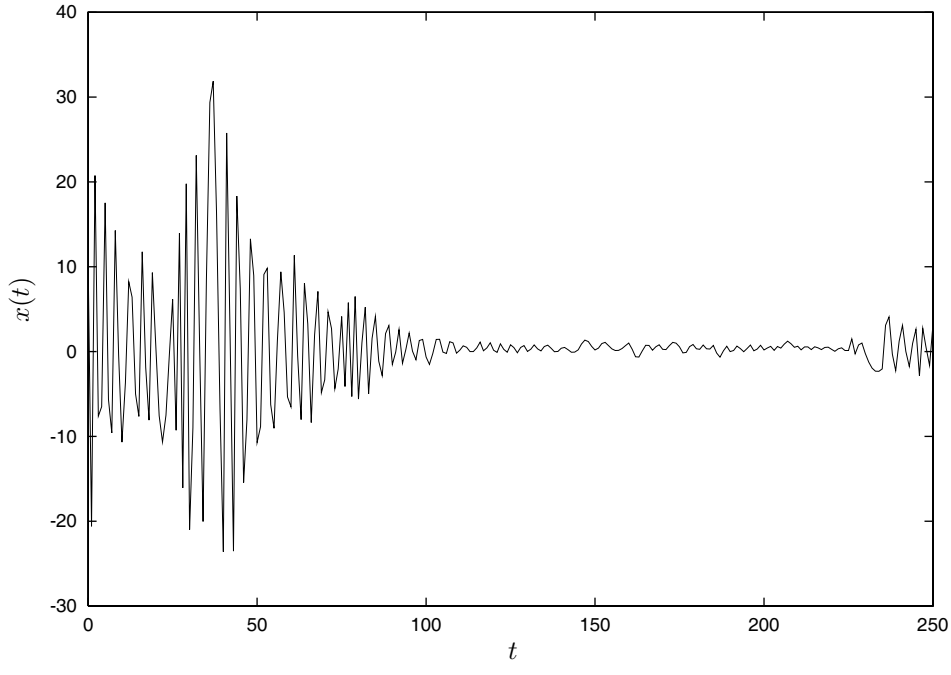


Figure 16.6 Stochastic Particle Trajectory for $w = 0.9$ and $c_1 = c_2 = 2.0$

problems considered in these studies. Some studies have shown that the basic PSO improves on the performance of other stochastic population-based optimization algorithms such as genetic algorithms [88, 89, 106, 369, 408, 863]. While the basic PSO has shown some success, formal analysis [136, 851, 863, 870] has shown that the performance of the PSO is sensitive to the values of control parameters. It was also shown that the basic PSO has a serious defect that may cause stagnation [868].

A variety of PSO variations have been developed, mainly to improve the accuracy of solutions, diversity and convergence behavior. This section reviews some of these variations for locating a single solution to unconstrained, single-objective, static optimization problems. Section 16.5.2 considers approaches that differ in the social interaction of particles. Some hybrids with concepts from EC are discussed in Section 16.5.3. Algorithms with multiple swarms are discussed in Section 16.5.4. Multi-start methods are given in Section 16.5.5, while methods that use some form of repelling mechanism are discussed in Section 16.5.6. Section 16.5.7 shows how PSO can be changed to solve binary-valued problems.

Before these PSO variations are discussed, Section 16.5.1 outlines a problem with the basic PSO.

16.5.1 Guaranteed Convergence PSO

The basic PSO has a potentially dangerous property: when $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}$, the velocity update depends only on the value of $w\mathbf{v}_i$. If this condition is true for all particles and it persists for a number of iterations, then $w\mathbf{v}_i \to 0$, which leads to stagnation of the search process. This point of stagnation may not necessarily coincide with a local minimum. All that can be said is that the particles converged to the best position found by the swarm. The PSO can, however, be pulled from this point of stagnation by forcing the global best position to change when $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}$.

The guaranteed convergence PSO (GCPSO) forces the global best particle to search in a confined region for a better position, thereby solving the stagnation problem [863, 868]. Let τ be the index of the global best particle, so that

$$\mathbf{y}_\tau = \hat{\mathbf{y}} \tag{16.42}$$

GCPSO changes the position update to

$$x_{\tau j}(t+1) = \hat{y}_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_2(t)) \tag{16.43}$$

which is obtained using equation (16.1) if the velocity update of the global best particle changes to

$$v_{\tau j}(t+1) = -x_{\tau j}(t) + \hat{y}_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_{2j}(t)) \tag{16.44}$$

where $\rho(t)$ is a scaling factor defined in equation (16.45) below. Note that only the global best particle is adjusted according to equations (16.43) and (16.44); all other particles use the equations as given in equations (16.1) and (16.2).

The term $-x_{\tau j}(t)$ in equation (16.44) resets the global best particle's position to the position $\hat{y}_j(t)$. The current search direction, $wv_{\tau j}(t)$, is added to the velocity, and the term $\rho(t)(1 - 2r_{2j}(t))$ generates a random sample from a sample space with side lengths $2\rho(t)$. The scaling term forces the PSO to perform a random search in an area surrounding the global best position, $\hat{\mathbf{y}}(t)$. The parameter, $\rho(t)$ controls the diameter of this search area, and is adapted using

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if } \#successes(t) > \epsilon_s \\ 0.5\rho(t) & \text{if } \#failures(t) > \epsilon_f \\ \rho(t) & \text{otherwise} \end{cases} \tag{16.45}$$

where $\#successes$ and $\#failures$ respectively denote the number of consecutive successes and failures. A failure is defined as $f(\hat{\mathbf{y}}(t)) \leq f(\hat{\mathbf{y}}(t+1))$; $\rho(0) = 1.0$ was found empirically to provide good results [863, 868]. The threshold parameters, ϵ_s and ϵ_f adhere to the following conditions:

$$\#successes(t+1) > \#successes(t) \Rightarrow \#failures(t+1) = 0 \tag{16.46}$$

$$\#failures(t+1) > \#failures(t) \Rightarrow \#successes(t+1) = 0 \tag{16.47}$$

The optimal choice of values for ϵ_s and ϵ_f is problem-dependent. Van den Bergh *et al.* [863, 868] recommends that $\epsilon_s = 15$ and $\epsilon_f = 5$ be used for high-dimensional search

spaces. The algorithm is then quicker to punish poor ρ settings than it is to reward successful ρ values.

Instead of using static ϵ_s and ϵ_f values, these values can be learnt dynamically. For example, increase s_c each time that $\#failures > \epsilon_f$, which makes it more difficult to reach the success state if failures occur frequently. Such a conservative mechanism will prevent the value of ρ from oscillating rapidly. A similar strategy can be used for ϵ_s.

The value of ρ determines the size of the local area around $\hat{\mathbf{y}}$ where a better position for $\hat{\mathbf{y}}$ is searched. GCPSO uses an adaptive ρ to find the best size of the sampling volume, given the current state of the algorithm. When the global best position is repeatedly improved for a specific value of ρ, the sampling volume is increased to allow step sizes in the global best position to increase. On the other hand, when ϵ_f consecutive failures are produced, the sampling volume is too large and is consequently reduced. Stagnation is prevented by ensuring that $\rho(t) > 0$ for all time steps.

16.5.2 Social-Based Particle Swarm Optimization

Social-based PSO implementations introduce a new social topology, or change the way in which personal best and neighborhood best positions are calculated.

Spatial Social Networks

Neighborhoods are usually formed on the basis of particle indices. That is, assuming a ring social network, the immediate neighbors of a particle with index i are particles with indices $(i - 1 \mod n_s)$ and $(i - 1 \mod n_s)$, where n_s is the total number of particles in the swarm. Suganthan proposed that neighborhoods be formed on the basis of the Euclidean distance between particles [820]. For neighborhoods of size $n_{\mathcal{N}}$, the neighborhood of particle i is defined to consist of the $n_{\mathcal{N}}$ particles closest to particle i. Algorithm 16.4 summarizes the spatial neighborhood selection process.

Calculation of spatial neighborhoods require that the Euclidean distance between all particles be calculated at each iteration, which significantly increases the computational complexity of the search algorithm. If n_t iterations of the algorithm is executed, the spatial neighborhood calculation adds a $O(n_t n_s^2)$ computational cost. Determining neighborhoods based on distances has the advantage that neighborhoods change dynamically with each iteration.

Fitness-Based Spatial Neighborhoods

Braendler and Hendtlass [81] proposed a variation on the spatial neighborhoods implemented by Suganthan, where particles move towards neighboring particles that have found a good solution. Assuming a minimization problem, the neighborhood of

Algorithm 16.4 Calculation of Spatial Neighborhoods; $\mathcal{N}_i$ is the neighborhood of particle i

Calculate the Euclidean distance $\mathcal{E}(\mathbf{x}_{i_1}, \mathbf{x}_{i_2}), \forall i_1, i_2 = 1, \ldots, n_s$;
$S = \{i : i = 1, \ldots, n_s\}$;
for $i = 1, \ldots, n_s$ **do**
  $S^{'} = S$;
  for $i^{'} = 1, \ldots, n_{\mathcal{N}_{i'}}$ **do**
    $\mathcal{N}_i = \mathcal{N}_i \cup \{\mathbf{x}_{i''} : \mathcal{E}(\mathbf{x}_i, \mathbf{x}_{i''}) = \min\{\mathcal{E}(\mathbf{x}_i, \mathbf{x}_{i'''}), \forall \mathbf{x}_{i'''} \in S^{'}\}$;
    $S^{'} = S^{'} \setminus \{\mathbf{x}_{i''}\}$;
  end
end

particle i is defined as the $n_{\mathcal{N}}$ particles with the smallest value of

$$\mathcal{E}(\mathbf{x}_i, \mathbf{x}_{i'}) \times f(\mathbf{x}_{i'}) \tag{16.48}$$

where $\mathcal{E}(\mathbf{x}_i, \mathbf{x}_{i'})$ is the Euclidean distance between particles i and $i^{'}$, and $f(\mathbf{x}_{i'})$ is the fitness of particle $i^{'}$. Note that this neighborhood calculation mechanism also allows for overlapping neighborhoods.

Based on this scheme of determining neighborhoods, the standard *lbest* PSO velocity equation (refer to equation (16.6)) is used, but with $\hat{\mathbf{y}}_i$ the neighborhood-best determined using equation (16.48).

Growing Neighborhoods

As discussed in Section 16.2, social networks with a low interconnection converge slower, which allows larger parts of the search space to be explored. Convergence of the fully interconnected star topology is faster, but at the cost of neglecting parts of the search space. To combine the advantages of better exploration by neighborhood structures and the faster convergence of highly connected networks, Suganthan combined the two approaches [820]. The search is initialized with an *lbest* PSO with $n_{\mathcal{N}} = 2$ (i.e. with the smallest neighborhoods). The neighborhood sizes are then increased with increase in iteration until each neighborhood contains the entire swarm (i.e. $n_{\mathcal{N}} = n_s$).

Growing neighborhoods are obtained by adding particle position $\mathbf{x}_{i_2}(t)$ to the neighborhood of particle position $\mathbf{x}_{i_1}(t)$ if

$$\frac{||\mathbf{x}_{i_1}(t) - \mathbf{x}_{i_2}(t)||_2}{d_{max}} < \epsilon \tag{16.49}$$

where d_{max} is the largest distance between any two particles, and

$$\epsilon = \frac{3t + 0.6n_t}{n_t} \tag{16.50}$$

with n_t the maximum number of iterations.

This allows the search to explore more in the first iterations of the search, with faster convergence in the later stages of the search.

Hypercube Structure

For binary-valued problems, Abdelbar and Abdelshahid [5] used a hypercube neighborhood structure. Particles are defined as neighbors if the Hamming distance between the bit representation of their indices is one. To make use of the hypercube topology, the total number of particles must be a power of two, where particles have indices from 0 to $2^{n_{\mathcal{N}}} - 1$. Based on this, the hypercube has the properties [5]:

- Each neighborhood has exactly $n_{\mathcal{N}}$ particles.
- The maximum distance between any two particles is exactly $n_{\mathcal{N}}$.
- If particles i_1 and i_2 are neighbors, then i_1 and i_2 will have no other neighbors in common.

Abdelbar and Abdelshahid found that the hypercube network structure provides better results than the *gbest* PSO for the binary problems studied.

Fully Informed PSO

Based on the standard velocity updates as given in equations (16.2) and (16.6), each particle's new position is influenced by the particle itself (via its personal best position) and the best position in its neighborhood. Kennedy and Mendes observed that human individuals are not influenced by a single individual, but rather by a statistical summary of the state of their neighborhood [453]. Based on this principle, the velocity equation is changed such that each particle is influenced by the successes of all its neighbors, and not on the performance of only one individual. The resulting PSO is referred to as the *fully informed* PSO (FIPS).

Two models are suggested [453, 576]:

- Each particle in the neighborhood, $\mathcal{N}_i$, of particle i is regarded equally. The cognitive and social components are replaced with the term

$$\sum_{m=1}^{n_{\mathcal{N}_i}} \frac{\mathbf{r}(t)(\mathbf{y}_m(t) - \mathbf{x}_i(t))}{n_{\mathcal{N}_i}} \tag{16.51}$$

  where $n_{\mathcal{N}_i} = |\mathcal{N}_i|$, $\mathcal{N}_i$ is the set of particles in the neighborhood of particle i as defined in equation (16.8), and $\mathbf{r}(t) \sim U(0, c_1 + c_2)^{n_x}$. The velocity is then calculated as

$$\mathbf{v}_i(t+1) = \chi \left(\mathbf{v}_i(t) + \sum_{m=1}^{n_{\mathcal{N}_i}} \frac{\mathbf{r}(t)(\mathbf{y}_m(t) - \mathbf{x}_i(t))}{n_{\mathcal{N}_i}} \right) \tag{16.52}$$

Although Kennedy and Mendes used constriction models of type $1'$ (refer to Section 16.3.3), inertia or any of the other models may be used.

Using equation (16.52), each particle is attracted towards the average behavior of its neighborhood.

Note that if $\mathcal{N}_i$ includes only particle i and its best neighbor, the velocity equation becomes equivalent to that of *lbest* PSO.

- A weight is assigned to the contribution of each particle based on the performance of that particle. The cognitive and social components are replaced by

$$\frac{\sum_{m=1}^{n_{\mathcal{N}_i}} \frac{\phi_m \mathbf{p}_m(t)}{f(\mathbf{x}_m(t))}}{\sum_{m=1}^{n_{\mathcal{N}_i}} \frac{\phi_m}{f(\mathbf{x}_m(t))}} \tag{16.53}$$

where $\phi_m \sim U(0, \frac{c_1+c_2}{n_{\mathcal{N}_i}})$, and

$$\mathbf{p}_m(t) = \frac{\phi_1 \mathbf{y}_m(t) + \phi_2 \hat{\mathbf{y}}_m(t)}{\phi_1 + \phi_2} \tag{16.54}$$

The velocity equation then becomes

$$\mathbf{v}_i(t+1) = \chi \left(\mathbf{v}_i(t) + \frac{\sum_{m=1}^{n_{\mathcal{N}_i}} \left(\frac{\phi_m \mathbf{p}_m(t)}{f(\mathbf{x}_m(t))} \right)}{\sum_{m=1}^{n_{\mathcal{N}_i}} \left(\frac{\phi_m}{f(\mathbf{x}_m(t))} \right)} \right) \tag{16.55}$$

In this case a particle is attracted more to its better neighbors.

A disadvantage of the FIPS is that it does not consider the fact that the influences of multiple particles may cancel each other. For example, if two neighbors respectively contribute the amount of a and $-a$ to the velocity update of a specific particle, then the sum of their influences is zero. Consider the effect when all the neighbors of a particle are organized approximately symmetrically around the particle. The change in weight due to the FIPS velocity term will then be approximately zero, causing the change in position to be determined only by $\chi \mathbf{v}_i(t)$.

Barebones PSO

Formal proofs [851, 863, 870] have shown that each particle converges to a point that is a weighted average between the personal best and neighborhood best positions. If it is assumed that $c_1 = c_2$, then a particle converges, in each dimension to

$$\frac{y_{ij}(t) + \hat{y}_{ij}(t)}{2} \tag{16.56}$$

This behavior supports Kennedy's proposal to replace the entire velocity by random numbers sampled from a Gaussian distribution with the mean as defined in equation (16.56) and deviation,

$$\sigma = |y_{ij}(t) - \hat{y}_{ij}(t)| \tag{16.57}$$

The velocity therefore changes to

$$v_{ij}(t+1) \sim N\left(\frac{y_{ij}(t)+\hat{y}_{ij}(t)}{2}, \sigma\right) \tag{16.58}$$

In the above, $\hat{y}_{ij}$ can be the global best position (in the case of *gbest* PSO), the local best position (in the case of a neighborhood-based algorithm), a randomly selected neighbor, or the center of a FIPS neighborhood. The position update changes to

$$x_{ij}(t+1) = v_{ij}(t+1) \tag{16.59}$$

Kennedy also proposed an alternative to the barebones PSO velocity in equation (16.58), where

$$v_{ij}(t+1) = \begin{cases} y_{ij}(t) & \text{if } U(0,1) < 0.5 \\ N(\frac{y_{ij}(t)+\hat{y}_{ij}(t)}{2}, \sigma) & \text{otherwise} \end{cases} \tag{16.60}$$

Based on equation (16.60), there is a 50% chance that the j-th dimension of the particle dimension changes to the corresponding personal best position. This version can be viewed as a mutation of the personal best position.

16.5.3 Hybrid Algorithms

This section describes just some of the PSO algorithms that use one or more concepts from EC.

Selection-Based PSO

Angeline provided the first approach to combine GA concepts with PSO [26], showing that PSO performance can be improved for certain classes of problems by adding a selection process similar to that which occurs in evolutionary algorithms. The selection procedure as summarized in Algorithm 16.5 is executed before the velocity updates are calculated.

Algorithm 16.5 Selection-Based PSO

Calculate the fitness of all particles;
for *each particle* $i = 1, \ldots, n_s$ **do**
    Randomly select n_{ts} particles;
    Score the performance of particle i against the n_{ts} randomly selected particles;
end
Sort the swarm based on performance scores;
Replace the worst half of the swarm with the top half, without changing the personal best positions;

Although the bad performers are penalized by being removed from the swarm, memory of their best found positions is not lost, and the search process continues to build on previously acquired experience. Angeline showed empirically that the selection based PSO improves on the local search capabilities of PSO [26]. However, contrary to one of the objectives of natural selection, this approach significantly reduces the diversity of the swarm [363]. Since half of the swarm is replaced by the other half, diversity is decreased by 50% at each iteration. In other words, the selection pressure is too high.

Diversity can be improved by replacing the worst individuals with mutated copies of the best individuals. Performance can also be improved by considering replacement only if the new particle improves the fitness of the particle to be deleted. This is the approach followed by Koay and Srinivasan [470], where each particle generates offspring through mutation.

Reproduction

Reproduction refers to the process of producing offspring from individuals of the current population. Different reproduction schemes have been used within PSO. One of the first approaches can be found in the Cheap-PSO developed by Clerc [134], where a particle is allowed to generate a new particle, kill itself, or modify the inertia and acceleration coefficient, on the basis of environment conditions. If there is no sufficient improvement in a particle's neighborhood, the particle spawns a new particle within its neighborhood. On the other hand, if a sufficient improvement is observed in the neighborhood, the worst particle of that neighborhood is culled.

Using this approach to reproduction and culling, the probability of adding a particle decreases with increasing swarm size. On the other hand, a decreasing swarm size increases the probability of spawning new particles.

The Cheap-PSO includes only the social component, where the social acceleration coefficient is adapted using equation (16.35) and the inertia is adapted using equation (16.29).

Koay and Srinivasan [470] implemented a similar approach to dynamically changing swarm sizes. The approach was developed by analogy with the natural adaptation of the amoeba to its environment: when the amoeba receives positive feedback from its environment (i.e. that sufficient food sources exist), it reproduces by releasing more spores. On the other hand, when food sources are scarce, reproduction is reduced. Taking this analogy to optimization, when a particle finds itself in a potential optimum, the number of particles is increased in that area. Koay and Srinivasan spawn only the global best particle (assuming *gbest* PSO) in order to reduce the computational complexity of the spawning process. The choice of spawning only the global best particle can be motivated by the fact that the global best particle will be the first particle to find itself in a potential optimum. Stopping conditions as given in Section 16.1.6 can be used to determine if a potential optimum has been found. The spawning process is summarized in Algorithm 16.6. It is also possible to apply the same process to the neighborhood best positions if other neighborhood networks are used, such as the ring structure.

Algorithm 16.6 Global Best Spawning Algorithm

if $\hat{\mathbf{y}}(t)$ *is in a potential minimum* **then**
  repeat
    $\hat{\mathbf{y}} = \hat{\mathbf{y}}(t)$;
    for *NumberOfSpawns=1 to 10* **do**
      for $a = 1$ *to NumberOfSpawns* **do**
        $\hat{\mathbf{y}}_a = \hat{\mathbf{y}}(t) + N(0, \sigma)$;
        if $f(\hat{\mathbf{y}}_a) < f(\hat{\mathbf{y}})$ **then**
          $\hat{\mathbf{y}} = \hat{\mathbf{y}}_a$;
        end
      end
    end
  until $f(\hat{\mathbf{y}}) \geq f(\hat{\mathbf{y}}(t))$;
  $\hat{\mathbf{y}}(t) = \hat{\mathbf{y}}$;
end

Løvberg *et al.* [534, 536] used an arithmetic crossover operator to produce offspring from two randomly selected particles. Assume that particles a and b are selected for crossover. The corresponding positions, $\mathbf{x}_a(t)$ and $\mathbf{x}_b(t)$ are then replaced by the offspring,

$$\mathbf{x}_{i_1}(t+1) = \mathbf{r}(t)\mathbf{x}_{i_1}(t) + (\mathbf{1} - \mathbf{r}(t))\mathbf{x}_{i_2}(t) \quad (16.61)$$

$$\mathbf{x}_{i_2}(t+1) = \mathbf{r}(t)\mathbf{x}_{i_2}(t) + (\mathbf{1} - \mathbf{r}(t))\mathbf{x}_{i_1}(t) \quad (16.62)$$

with the corresponding velocities,

$$\mathbf{v}_{i_1}(t+1) = \frac{\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)}{||\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)||}||\mathbf{v}_{i_1}(t)|| \quad (16.63)$$

$$\mathbf{v}_{i_2}(t+1) = \frac{\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)}{||\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)||}||\mathbf{v}_{i_2}(t)|| \quad (16.64)$$

where $\mathbf{r}_1(t) \sim U(0,1)^{n_x}$. The personal best position of an offspring is initialized to its current position. That is, if particle i_1 was involved in the crossover, then $\mathbf{y}_{i_1}(t+1) = \mathbf{x}_{i_1}(t+1)$.

Particles are selected for breeding at a user-specified breeding probability. Given that this probability is less than one, not all of the particles will be replaced by offspring. It is also possible that the same particle will be involved in the crossover operation more than once per iteration. Particles are randomly selected as parents, not on the basis of their fitness. This prevents the best particles from dominating the breeding process. If the best particles were allowed to dominate, the diversity of the swarm would decrease significantly, causing premature convergence. It was found empirically that a low breeding probability of 0.2 provides good results [536].

The breeding process is done for each iteration after the velocity and position updates have been done.

The breeding mechanism proposed by Løvberg *et al.* has the disadvantage that parent particles are replaced even if the offspring is worse off in fitness. Also, if $f(\mathbf{x}_{i_1}(t+1)) >$

$f(\mathbf{x}_{i_1}(t))$ (assuming a minimization problem), replacement of the personal best with $\mathbf{x}_{i_1}(t+1)$ loses important information about previous personal best positions. Instead of being attracted towards the previous personal best position, which in this case will have a better fitness, the offspring is attracted to a worse solution. This problem can be addressed by replacing $\mathbf{x}_{i_1}(t)$ with its offspring only if the offspring provides a solution that improves on particle i_1's personal best position, $\mathbf{y}_{i_1}(t)$.

Gaussian Mutation

Gaussian mutation has been applied mainly to adjust position vectors after the velocity and update equations have been applied, by adding random values sampled from a Gaussian distribution to the components of particle position vectors, $\mathbf{x}_i(t+1)$.

Miranda and Fonseca [596] mutate only the global best position as follows,

$$\hat{\mathbf{y}}(t+1) = \hat{\mathbf{y}}^{'}(t+1) + \eta^{'}\mathbf{N}(0,1) \tag{16.65}$$

where $\hat{\mathbf{y}}^{'}(t+1)$ represents the unmutated global best position as calculated from equation (16.4), $\eta^{'}$ is referred to as a learning parameter, which can be a fixed value, or adapted as a strategy parameter as in evolutionary strategies.

Higashi and Iba [363] mutate the components of particle position vectors at a specified probability. Let $\mathbf{x}_i^{'}(t+1)$ denote the new position of particle i after application of the velocity and position update equations, and let P_m be the probability that a component will be mutated. Then, for each component, $j = 1, \ldots, n_x$, if $U(0,1) < P_m$, then component $x_{ij}^{'}(t+1)$ is mutated using [363]

$$x_{ij}(t+1) = x_{ij}^{'}(t+1) + N(0,\sigma)x_{ij}^{'}(t+1) \tag{16.66}$$

where the standard deviation, σ, is defined as

$$\sigma = 0.1(x_{max,j} - x_{min,j}) \tag{16.67}$$

Wei *et al.* [893] directly apply the original EP Gaussian mutation operator:

$$x_{ij}(t+1) = x_{ij}^{'}(t+1) + \eta_{ij}(t)N_j(0,1) \tag{16.68}$$

where η_{ij} controls the mutation step sizes, calculated for each dimension as

$$\eta_{ij}(t) = \eta_{ij}(t-1)\, \mathrm{e}^{\tau^{'}N(0,1)+\tau N_j(0,1)} \tag{16.69}$$

with τ and $\tau^{'}$ as defined in equations (11.54) and (11.53).

The following comments can be made about this mutation operator:

- With $\eta_{ij}(0) \in (0,1]$, the mutation step sizes decrease with increase in time step, t. This allows for initial exploration, with the solutions being refined in later time steps. It should be noted that convergence cannot be ensured if mutation step sizes do not decrease over time.

- All the components of the same particle are mutated using the same value, $N(0,1)$. However, for each component, an additional random number, $N_j(0,1)$, is also used. Each component will therefore be mutated by a different amount.
- The amount of mutation depends on the dimension of the problem, by the calculation of τ and $\tau^{'}$. The larger n_x, the smaller the mutation step sizes.

Secrest and Lamont [772] adjust particle positions as follows,

$$\mathbf{x}_i(t+1) = \begin{cases} \mathbf{y}_i(t) + \mathbf{v}_i(t+1) & \text{if } U(0,1) > c_1 \\ \hat{\mathbf{y}}(t) + \mathbf{v}_i(t+1) & \text{otherwise} \end{cases} \tag{16.70}$$

where

$$\mathbf{v}_i(t+1) = |\mathbf{v}_i(t+1)|\mathbf{r}_\theta \tag{16.71}$$

In the above, $\mathbf{r}_\theta$ is a random vector with magnitude of one and angle uniformly distributed from 0 to 2π. $|\mathbf{v}_i(t+1)|$ is the magnitude of the new velocity, calculated as

$$|\mathbf{v}_i(t+1)| = \begin{cases} N(0,(1-c_2)||\mathbf{y}_i(t) - \hat{\mathbf{y}}(t)||_2) & \text{if } U(0,1) > c_1 \\ N(0,c_2||\mathbf{y}_i(t) - \hat{\mathbf{y}}(t)||_2) & \text{otherwise} \end{cases} \tag{16.72}$$

Coefficient $c_1 \in (0,1)$ specifies the trust between the global and personal best positions. The larger c_1, the more particles are placed around the global best position; $c_2 \in (0,1)$ establishes the point between the global best and the personal best to which the corresponding particle will converge.

Cauchy Mutation

In the fast EP, Yao *et al.* [934, 936] showed that mutation step sizes sampled from a Cauchy distribution result in better performance than sampling from a Gaussian distribution. This is mainly due to the fact that the Cauchy distribution has more of its probability in the tails of the distribution, resulting in an increased probability of large mutation step sizes (therefore, better exploration). Stacey *et al.* [808] applied the EP Cauchy mutation operator to PSO. Given that n_x is the dimension of particles, each dimension is mutated with a probability of $1/n_x$. If a component $x_{ij}(t)$ is selected for mutation, the mutation step size, $\Delta x_{ij}(t)$, is sampled from a Cauchy distribution with probability density function given by equation (11.10).

Differential Evolution Based PSO

Differential evolution (DE) makes use of an arithmetic crossover operator that involves three randomly selected parents (refer to Chapter 13). Let $\mathbf{x}_1(t) \neq \mathbf{x}_2(t) \neq \mathbf{x}_3(t)$ be three particle positions randomly selected from the swarm. Then each dimension of particle i is calculated as follows:

$$x_{ij}^{'}(t+1) = \begin{cases} x_{1j}(t) + \beta(x_{2j}(t) - x_{3j}(t)) & \text{if } U(0,1) \leq P_c \text{ or } j = U(1,n_x) \\ x_{ij}(t) & \text{otherwise} \end{cases} \tag{16.73}$$

where $P_c \in (0,1)$ is the probability of crossover, and $\beta > 0$ is a scaling factor.

The position of the particle is only replaced if the offspring is better. That is, $\mathbf{x}_i(t+1) = \mathbf{x}_i^{'}(t+1)$ only if $f(\mathbf{x}_i^{'}(t+1)) < f(\mathbf{x}_i(t))$, otherwise $\mathbf{x}_i(t+1) = \mathbf{x}_i(t)$ (assuming a minimization problem).

Hendtlass applied the above DE process to the PSO by executing the DE crossover operator from time to time [360]. That is, at specified intervals, the swarm serves as population for the DE algorithm, and the DE algorithm is executed for a number of iterations. Hendtlass reported that this hybrid produces better results than the basic PSO. Kannan *et al.* [437] applies DE to each particle for a number of iterations, and replaces the particle with the best individual obtained from the DE process.

Zhang and Xie [954] followed a somewhat different approach where only the personal best positions are changed using the following operator:

$$y_{ij}^{'}(t+1) = \begin{cases} \hat{y}_{ij}(t) + \delta_j & \text{if } U(0,1) < P_c \text{ and } j = U(1,n_x) \\ y_{ij}(t) & \text{otherwise} \end{cases} \tag{16.74}$$

where δ is the general difference vector defined as,

$$\delta_j = \frac{y_{1j}(t) - y_{2j}(t)}{2} \tag{16.75}$$

with $y_{1j}(t)$ and $y_{2j}(t)$ randomly selected personal best positions. Then, $y_{ij}(t+1)$ is set to $y_{ij}^{'}(t+1)$ only if the new personal best has a better fitness evaluation.

16.5.4 Sub-Swarm Based PSO

A number of cooperative and competitive PSO implementations that make use of multiple swarms have been developed. Some of these are described below.

Multi-phase PSO approaches divide the main swarm of particles into subgroups, where each subgroup performs a different task, or exhibits a different behavior. The behavior of a group or task performed by a group usually changes over time in response to the group's interaction with the environment. It can also happen that individuals may migrate between groups.

The breeding between sub-swarms developed by Løvberg *et al.* [534, 536] (refer to Section 16.5.3) is one form of cooperative PSO, where cooperation is implicit in the exchange of genetic material between parents of different sub-swarms.

Al-Kazemi and Mohan [16, 17, 18] explicitly divide the main swarm into two sub-swarms, of equal size. Particles are randomly assigned to one of the sub-swarms. Each sub-swarm can be in one of two phases:

- **Attraction phase**, where the particles of the corresponding sub-swarm are allowed to move towards the global best position.
- **Repulsion phase**, where the particles of the corresponding sub-swarm move away from the global best position.

For their multi-phase PSO (MPPSO), the velocity update is defined as [16, 17]:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 x_{ij}(t) + c_2 \hat{y}_j(t) \tag{16.76}$$

The personal best position is excluded from the velocity equation, since a hill-climbing procedure is followed where a particle's position is only updated if the new position results in improved performance.

Let the tuple (w, c_1, c_2) represent the values of the inertia weight, w, and acceleration coefficients c_1 and c_2. Particles that find themselves in phase 1 exhibit an attraction towards the global best position, which is achieved by setting $(w, c_1, c_2) = (1, -1, 1)$. Particles in phase 2 have $(w, c_1, c_2) = (1, 1, -1)$, forcing them to move away from the global best position.

Sub-swarms switch phases either

- when the number of iterations in the current phase exceeds a user specified threshold, or
- when particles in any phase show no improvement in fitness during a user-specified number of consecutive iterations.

In addition to the velocity update as given in equation (16.76), velocity vectors are periodically initialized to new random vectors. Care should be taken with this process, not to reinitialize velocities when a good solution is found, since it may pull particles out of this good optimum. To make sure that this does not happen, particle velocities can be reinitialized based on a reinitialization probability. This probability starts with a large value that decreases over time. This approach ensures large diversity in the initial steps of the algorithm, emphasizing exploration, while exploitation is favored in the later steps of the search.

Cooperation between the subgroups is achieved through the selection of the global best particle, which is the best position found by all the particles in both sub-swarms.

Particle positions are not updated using the standard position update equation. Instead, a hill-climbing process is followed to ensure that the fitness of a particle is monotonically decreasing (increasing) in the case of a minimization (maximization) problem. The position vector is updated by randomly selecting ς consecutive components from the velocity vector and adding these velocity components to the corresponding position components. If no improvement is obtained for any subset of ς consecutive components, the position vector does not change. If an improvement is obtained, the corresponding position vector is accepted. The value of ς changes for each particle, since it is randomly selected, with $\varsigma \sim U(1, \varsigma_{max})$, with ς_{max} initially small, increasing to a maximum of n_x (the dimension of particles).

The attractive and repulsive PSO (ARPSO) developed by Riget and Vesterstrøm [729, 730, 877] follows a similar process where the entire swarm alternates between an attraction and repulsion phase. The difference between the MPPSO and ARPSO lies in the velocity equation, in that there are no explicit sub-swarms in ARPSO, and ARPSO uses information from the environment to switch between phases. While

ARPSO was originally applied to one swarm, nothing prevents its application to sub-swarms.

The ARPSO is based on the diversity guided evolutionary algorithm developed by Ursem [861], where the standard Gaussian mutation operator is changed to a directed mutation in order to increase diversity. In ARPSO, if the diversity of the swarm, measured using

$$\text{diversity}(S(t)) = \frac{1}{n_s}\sum_{i=1}^{n_s}\sqrt{\sum_{j=1}^{n_x}(x_{ij}(t)-\overline{x}_j(t))^2} \tag{16.77}$$

where $\overline{x}_j(t)$ is the average of the j-th dimension over all particles, i.e.

$$\overline{x}_j(t) = \frac{\sum_{i=1}^{n_s} x_{ij}(t)}{n_s} \tag{16.78}$$

is greater than a threshold, φ_{min}, then the swarm switches to the attraction phase; otherwise the swarm switches to the repulsion phase until a threshold diversity, φ_{max} is reached. The attraction phase uses the basic PSO velocity and position update equations. For the repulsion phase, particles repel each other using,

$$v_{ij}(t+1) = wv_{ij}(t) - c_1r_{1j}(t)(y_{ij}(t)-x_{ij}(t)) - c_2r_{2j}(t)(\hat{y}_{ij}(t)-x_{ij}(t)) \tag{16.79}$$

Riget and Vesterstrøm used $\varphi_{min} = 5\times10^{-6}$ and $\varphi_{max} = 0.25$ as proposed by Ursem.

In the division of labor PSO (DLPSO), each particle has a response threshold, θ_{ik}, for each task, k. Each particle receives task-related stimuli, s_{ik}. If the received stimuli are much lower than the response threshold, the particle engages in the corresponding task with a low probability. If $s_{ik} > \theta_{ik}$, the probability of engaging in the task is higher. Let ϑ_{ik} denote the state of particle i with respect to task k. If $\vartheta_{ik} = 0$, then particle i is not performing task k. On the other hand, if $\vartheta_{ik} = 1$, then task k is performed by particle i. At each time step, each particle determines if it should become active or inactive on the basis of a given probability. An attractive particle performs task k with probability

$$P(\vartheta_{ik}=0\rightarrow\vartheta_{ik}=1) = P_{\theta_{ik}}(\theta_{ik},s_{ik}) = \frac{s_{ik}^{\alpha}}{\theta_{ik}^{\alpha}+s_{ik}^{\alpha}} \tag{16.80}$$

where α controls the steepness of the response function, P_θ. For high α, high probabilities are assigned to values of s_{ik} just above θ_{ik}.

The probability of an active particle to become inactive is taken as a constant, user-specified value, P_ϑ.

The DLPSO uses only one task (the task subscript is therefore omitted), namely local search. In this case, the stimuli is the number of time steps since the last improvement in the fitness of the corresponding particle. When a particle becomes active, the local search is performed as follows: if $\vartheta_i = 1$, then $\mathbf{x}_i(t) = \hat{\mathbf{y}}_i(t)$ and $\mathbf{v}_i(t)$ is randomly assigned with length equal to the length of the velocity of the neighborhood best particle. The probability of changing to an inactive particle is $P_\vartheta = 1$, meaning that a

particle becomes inactive immediately after its local search task is completed. Inactive particles follow the velocity and position updates of the basic PSO. The probability, $P(\vartheta_i = 1 \rightarrow \vartheta_i = 0)$ is kept high to ensure that the PSO algorithm does not degrade to a pure local search algorithm.

The local search, or exploitation, is only necessary when the swarm starts to converge to a solution. The probability of task execution should therefore be small initially, increasing over time. To achieve this, the absolute ageing process introduced by Théraulaz *et al.* [842] for ACO is used in the DLPSO to dynamically adjust the values of the response thresholds

$$\theta_i(t) = \beta \, \mathrm{e}^{-\alpha t} \tag{16.81}$$

with $\alpha, \beta > 0$.

Empirical results showed that the DLPSO obtained significantly better results as a GA and the basic PSO [877, 878].

Krink and Løvberg [490, 534] used the life-cycle model to change the behavior of individuals. Using the life-cycle model, an individual can be in any of three phases: a PSO particle, a GA individual, or a stochastic hill-climber. The life-cycle model used here is in analogy with the biological process where an individual progresses through various phases from birth to maturity and reproduction. The transition between phases of the life-cycle is usually triggered by environmental factors.

For the life-cycle PSO (LCPSO), the decision to change from one phase to another depends on an individual's success in searching the fitness landscape. In the original model, all individuals start as particles and exhibit behavior as dictated by the PSO velocity and position update equations. The second phase in the life-cycle changes a particle to an individual in a GA, where its behavior is governed by the process of natural selection and survival of the fittest. In the last phase, an individual changes into a solitary stochastic hill-climber. An individual switches from one phase to the next if its fitness is not improved over a number of consecutive iterations. The LCPSO is summarized in Algorithm 16.7.

Application of the LCPSO results in the formation of three subgroups, one for each behavior (i.e. PSO, GA or hill-climber). Therefore, the main population may at the same time consist of individuals of different behavior.

While the original implementation initialized all individuals as PSO particles, the initial population can also be initialized to contain individuals of different behavior from the first time step. The rationale for starting with PSO particles can be motivated by the observation that the PSO has been shown to converge faster than GAs. Using hill-climbing as the third phase also makes sense, since exploitation should be emphasized in the later steps of the search process, with the initial PSO and GA phases focusing on exploration.

Algorithm 16.7 Life-Cycle PSO

```
Initialize a population of individuals;
repeat
    for all individuals do
        Evaluate fitness;
        if fitness did not improve then
            Switch to next phase;
        end
    end
    for all PSO particles do
        Calculate new velocity vectors;
        Update positions;
    end
    for all GA individuals do
        Perform reproduction;
        Mutate;
        Select new population;
    end
    for all Hill-climbers do
        Find possible new neighboring solution;
        Evaluate fitness of new solution;
        Move to new solution with specified probability;
    end
until stopping condition is true;
```

Cooperative Split PSO

The cooperative split PSO, first introduced in [864] by Van den Bergh and Engelbrecht, is based on the cooperative coevolutionary genetic algorithm (CCGA) developed by Potter [686] (also refer to Section 15.3). In the cooperative split PSO, denoted by CPSO-S_K, each particle is split into K separate parts of smaller dimension [863, 864, 869]. Each part is then optimized using a separate sub-swarm. If $K = n_x$, each dimension is optimized by a separate sub-swarm, using any PSO algorithm. The number of parts, K, is referred to as the *split factor*.

The difficulty with the CPSO-S_K algorithm is how to evaluate the fitness of the particles in the sub-swarms. The fitness of each particle in sub-swarm S_k cannot be computed in isolation from other sub-swarms, since a particle in a specific sub-swarm represents only part of the complete n_x-dimensional solution. To solve this problem, a context vector is maintained to represent the n_x-dimensional solution. The simplest way to construct the context vector is to concatenate the global best positions from the K sub-swarms. To evaluate the fitness of particles in sub-swarm S_k, all the components of the context vector are kept constant except those that correspond to the components of sub-swarm S_k. Particles in sub-swarm S_k are then swapped into the corresponding positions of the context vector, and the original fitness function is used to evaluate the fitness of the context vector. The fitness value obtained is then assigned as the fitness

of the corresponding particle of the sub-swarm.

This process has the advantage that the fitness function, f, is evaluated after each subpart of the context vector is updated, resulting in a much finer-grained search. One of the problems with optimizing the complete n_x-dimensional problem is that, even if an improvement in fitness is obtained, some of the components of the n_x-dimensional vector may move away from an optimum. The improved fitness could have been obtained by a sufficient move towards the optimum in the other vector components. The evaluation process of the CPSO-S_K addresses this problem by tuning subparts of the solution vector separately.

Algorithm 16.8 Cooperative Split PSO Algorithm

$K_1 = n_x \bmod K$;
$K_2 = K - (n_x \bmod K)$;
Initialize K_1 $\lceil n_x/K \rceil$-dimensional swarms;
Initialize K_2 $\lfloor n_x/K \rfloor$-dimensional swarms;
repeat
  for *each sub-swarm* S_k,$k = 1, \ldots, K$ **do**
    for *each particle* $i = 1, \ldots, S_k.n_s$ **do**
      if $f(\mathbf{b}(k, S_k.\mathbf{x}_i)) < f(\mathbf{b}(k, S_k.\mathbf{y}_i))$ **then**
        $S_k.\mathbf{y}_i = S_k.\mathbf{x}_i$;
      end
      if $f(\mathbf{b}(k, S_k.\mathbf{y}_i)) < f(\mathbf{b}(k, S_k.\hat{\mathbf{y}}))$ **then**
        $S_k.\hat{\mathbf{y}} = S_k.\mathbf{y}_i$;
      end
    end
    Apply velocity and position updates;
  end
until *stopping condition is true*;

The CPSO-S_K algorithm is summarized in Algorithm 16.8. In this algorithm, $\mathbf{b}(k, \mathbf{z})$ returns an n_x-dimensional vector formed by concatenating the global best positions from all the sub-swarms, except for the k-th component which is replaced with $\mathbf{z}$, where $\mathbf{z}$ represents the position vector of any particle from sub-swarm S_k. The context vector is therefore defined as

$$\mathbf{b}(k, \mathbf{z}) = (S_1.\hat{\mathbf{y}}, \ldots, S_{k-1}.\hat{\mathbf{y}}, \mathbf{z}, S_{k+1}.\hat{\mathbf{y}}, \ldots, S_K.\hat{\mathbf{y}}) \tag{16.82}$$

While the CPSO-S_K algorithm has shown significantly better results than the basic PSO, it has to be noted that performance degrades when correlated components are split into different sub-swarms. If it is possible to identify which parameters correlate, then these parameters can be grouped into the same swarm — which will solve the problem. However, such prior knowledge is usually not available. The problem can also be addressed to a certain extent by allowing a particle to become the global best or personal best of its sub-swarm only if it improves the fitness of the context vector.

Algorithm 16.9 summarizes a hybrid search where the CPSO and GCPSO algorithms are interweaved. Additionally, a rudimentary form of cooperation is implemented

between the CPSO-S_K and GCPSO algorithms. Information about the best positions discovered by each algorithm is exchanged by copying the best discovered solution of the one algorithm to the swarm of the other algorithm. After completion of one iteration of the CPSO-S_K algorithm, the context vector is used to replace a randomly selected particle from the GCPSO swarm (excluding the global best, $Q.\hat{\mathbf{y}}$). After completion of a GCPSO iteration, the new global best particle, $Q.\hat{\mathbf{y}}$, is split into the required subcomponents to replace a randomly selected individual of the corresponding CPSO-S_K algorithm (excluding the global best positions).

Predator-Prey PSO

The predator-prey relationship that can be found in nature is an example of a competitive environment. This behavior has been used in the PSO to balance exploration and exploitation [790]. By introducing a second swarm of predator particles, scattering of prey particles can be obtained by having prey particles being repelled by the presence of predator particles. Repelling facilitates better exploration of the search space, while the consequent regrouping promotes exploitation.

In their implementation of the predator–prey PSO, Silva *et al.* [790] use only one predator to pursue the global best prey particle. The velocity update for the predator particle is defined as

$$\mathbf{v}_p(t+1) = \mathbf{r}(\hat{\mathbf{y}}(t) - \mathbf{x}_p(t)) \tag{16.83}$$

where $\mathbf{v}_p$ and $\mathbf{x}_p$ are respectively the velocity and position vectors of the predator particle, p; $\mathbf{r} \sim U(0, V_{max,p})^{n_x}$. The speed at which the predator catches the best prey is controlled by $V_{max,p}$. The larger $V_{max,p}$, the larger the step sizes the predator makes towards the global best.

The prey particles update their velocity using

$$\begin{aligned} v_{ij}(t+1) &= wv_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + c_2 r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t)) \\ &\quad + c_3 r_{3j}(t) D(d) \end{aligned} \tag{16.84}$$

where d is the Euclidean distance between prey particle i and the predator, $r_{3j}(t) \sim U(0,1)$, and

$$D(d) = \alpha \, \mathrm{e}^{-\beta d} \tag{16.85}$$

$D(d)$ quantifies the influence that the predator has on the prey. The influence grows exponentially with proximity, and is further controlled by the positive constants α and β.

Components of the position vector of a particle is updated using equation (16.84) based on a "fear" probability, P_f. For each dimension, if $U(0,1) < P_f$, then position $x_{ij}(t)$ is updated using equation (16.84); otherwise the standard velocity update is used.

Algorithm 16.9 Hybrid of Cooperative Split PSO and GCPSO

$K_1 = n_x \bmod K$;
$K_2 = K - (n_x \bmod K)$;
Initialize K_1 $\lceil n_x/K \rceil$-dimensional swarms:$S_k, k = 1, \ldots, K$;
Initialize K_2 $\lfloor n_x/K \rfloor$-dimensional swarms:$S_k, k = K+1, \ldots, K$;
Initialize an n-dimensional swarm, Q;
repeat
  for *each sub-swarm* S_k, $k = 1, \ldots, K$ **do**
    for *each particle* $i = 1, \ldots, S_k.n_s$ **do**
      if $f(\mathbf{b}(k, S_k.\mathbf{x}_i)) < f(\mathbf{b}(k, S_k.\mathbf{y}_i))$ **then**
        $S_k.\mathbf{y}_i = S_k.\mathbf{x}_i$;
      end
      if $f(\mathbf{b}(k, S_k.\mathbf{y}_i)) < f(\mathbf{b}(k, S_k.\hat{\mathbf{y}}))$ **then**
        $S_k.\hat{\mathbf{y}} = S_k.\mathbf{y}_i$;
      end
    end
    Apply velocity and position updates;
  end
  Select a random $l \sim U(1, n_s/2) | Q.\mathbf{y}_l \neq Q.\hat{\mathbf{y}}$;
  Replace $Q.\mathbf{x}_l$ with the context vector $\mathbf{b}$;
  for *each particle* $i = 1, \ldots, n_s$ **do**
    if $f(Q.\mathbf{x}_i) < f(Q.\mathbf{y}_i)$ **then**
      $Q.\mathbf{y}_i = Q.\mathbf{x}_i$;
    end
    if $f(Q.\mathbf{y}_i) < f(Q.\hat{\mathbf{y}})$ **then**
      $Q.\hat{\mathbf{y}} = Q.\mathbf{y}_i$;
    end
  end
  Apply GCPSO velocity and position updates;
  for *each swarm* $S_k, k = 1, \ldots, K$ **do**
    Select a random $m \sim U(1, S_k.n_s/2) | S_k.\mathbf{y}_m \neq S_k.\hat{\mathbf{y}}$;
    Replace $S_k.\mathbf{x}_m$ with the corresponding components of $Q.\hat{\mathbf{y}}$;
  end
until *stopping condition is true*;

16.5.5 Multi-Start PSO Algorithms

One of the major problems with the basic PSO is lack of diversity when particles start to converge to the same point. As an approach to prevent the premature stagnation of the basic PSO, several methods have been developed to continually inject randomness, or chaos, into the swarm. This section discusses these approaches, collectively referred to as multi-start methods.

Multi-start methods have as their main objective to increase diversity, whereby larger parts of the search space are explored. Injection of chaos into the swarm introduces a negative entropy. It is important to remember that continual injection of random

positions will cause the swarm never to reach an equilibrium state. While not all the methods discussed below consider this fact, all of the methods can address the problem by reducing the amount of chaos over time.

Kennedy and Eberhart [449] were the first to mention the advantages of randomly reinitializing particles, a process referred to as *craziness*. Although Kennedy mentioned the potential advantages of a craziness operator, no evaluation of such operators was given. Since then, a number of researchers have proposed different approaches to implement a craziness operator for PSO.

When considering any method to add randomness to the swarm, a number of aspects need to be considered, including what should be randomized, when should randomization occur, how should it be done, and which members of the swarm will be affected? Additionally, thought should be given to what should be done with personal best positions of affected particles. These aspects are discussed next.

The diversity of the swarm can be increased by randomly initializing position vectors [534, 535, 863, 874, 875, 922, 923] and/or velocity vectors [765, 766, 922, 923, 924].

By initializing positions, particles are physically relocated to a different, random position in the search space. If velocity vectors are randomized and positions kept constant, particles retain their memory of their current and previous best solutions, but are forced to search in different random directions. If a better solution is not found due to random initialization of the velocity vector of a particle, the particle will again be attracted towards its personal best position.

If position vectors are initialized, thought should be given to what should be done with personal best positions and velocity vectors. Total reinitialization will have a particle's personal best also initialized to the new random position [534, 535, 923]. This effectively removes the particle's memory and prevents the particle from moving back towards its previously found best position (depending on when the global best position is updated). At the first iteration after reinitialization the "new" particle is attracted only towards the previous global best position of the swarm. Alternatively, reinitialized particles may retain their memory of previous best positions. It should be noted that the latter may have less diversity than removing particle memories, since particles are immediately moving back towards their previous personal best positions. It may, of course, happen that a new personal best position is found *en route*. When positions are reinitialized, velocities are usually initialized to zero, to have a zero momentum at the first iteration after reinitialization. Alternatively, velocities can be initialized to small random values [923]. Venter and Sobieszczanski-Sobieski [874, 875] initialize velocities to the cognitive component before reinitialization. This ensures a momentum back towards the personal best position.

The next important question to consider is when to reinitialize. If reinitialization happens too soon, the affected particles may not have had sufficient time to explore their current regions before being relocated. If the time to reinitialization is too long, it may happen that all particles have already converged. This is not really a problem, other than wasted computational time since no improvements are seen in this state. Several approaches have been identified to decide when to reinitialize:

- At fixed intervals, as is done in the mass extinction PSO developed by Xie *et al.* [923, 924]. As discussed above, fixed intervals may prematurely reinitialize a particle.
- Probabilistic approaches, where the decision to reinitialize is based on a probability. In the dissipative PSO, Xie *et al.* [922] reinitialize velocities and positions based on chaos factors that serve as probabilities of introducing chaos in the system. Let c_v and c_l, with $c_v, c_l \in [0, 1]$, be respectively the chaos factors for velocity and location. Then, for each particle, i, and each dimension, j, if $r_{ij} \sim U(0,1) < c_v$, then the velocity component is reinitialized to $v_{ij}(t+1) = U(0,1)V_{max,j}$. Also, if $r_{ij} \sim U(0,1) < c_l$, then the position component is initialized to $x_{ij}(t+1) \sim U(x_{min,j}, x_{max,j})$. A problem with this approach is that it will keep the swarm from reaching an equilibrium state. To ensure that an equilibrium can be reached, while still taking advantage of chaos injection, start with large chaos factors that reduce over time. The initial large chaos factors increase diversity in the first phases of the search, allowing particles to converge in the final stages. A similar probabilistic approach to reinitializing velocities is followed in [765, 766].
- Approaches based on some "convergence" condition, where certain events trigger reinitialization. Using convergence criteria, particles are allowed to first exploit their local regions before being reinitialized.

  Venter and Sobieszczanski-Sobieski [874, 875] and Xie *et al.* [923] initiate reinitialization when particles do not improve over time. Venter and Sobieszczanski-Sobieski evaluate the variation in particle fitness of the current swarm. If the variation is small, then particles are centered in close proximity to the global best position. Particles that are two standard deviations away from the swarm center are reinitialized. Xie *et al.* count for each $\mathbf{x}_i \neq \hat{\mathbf{y}}$ the number of times that $f(\mathbf{x}_i) - f(\hat{\mathbf{y}}) < \epsilon$. When this count exceeds a given threshold, the corresponding particle is reinitialized. Care should be taken in setting values for ϵ and the count threshold. If ϵ is too large, particles will be reinitialized before having any chance of exploiting their current regions.

  Clerc [133] defines a hope and re-hope criterion. If there is still hope that the objective can be reached, particles are allowed to continue in their current search directions. If not, particles are reinitialized around the global best position, taking into consideration the local shape of the objective function.

  Van den Bergh [863] defined a number of convergence tests for a multi-start PSO, namely the normalized swarm radius condition, the particle cluster condition and the objective function slope condition (also refer to Section 16.1.6 for a discussion on these criteria).

  Løvberg and Krink [534, 535] use self-organized criticality (SOC) to determine when to reinitialize particles. Each particle maintains an additional variable, C_i, referred to as the *criticality* of the particle. If two particles are closer than a threshold distance, ϵ, from one another, then both have their criticality increased by one. The larger the criticality of all particles, the more uniform the swarm becomes. To prevent criticality from building up, each C_i is reduced by a fraction in each iteration. As soon as $C_i > C$, where C is the global criticality limit, particle i is reinitialized. Its criticality is distributed to its immediate neighbors

and $C_i = 0$. Løvberg and Krink also set the inertia weight value of each particle to $w_i = 0.2 + 0.1C_i$. This forces the particle to explore more when it is too similar to other particles.

The next issue to consider is which particles to reinitialize. Obviously, it will not be a good idea to reinitialize the global best particle! From the discussions above, a number of selection methods have already been identified. Probabilistic methods decide which particles to reinitialize based on a user-defined probability. The convergence methods use specific convergence criteria to identify particles for reinitialization. For example, SOC PSO uses criticality measures (refer to Algorithm 16.11), while others keep track of the improvement in particle fitness. Van den Bergh [863] proposed a random selection scheme, where a particle is reinitialized at each t_r iteration (also refer to Algorithm 16.10). This approach allows each particle to explore its current region before being reinitialized. To ensure that the swarm will reach an equilibrium state, start with a large $t_r < n_s$, which decreases over time.

Algorithm 16.10 Selection of Particles to Reinitialize; τ indicates the index of the global best particle

Create and initialize an n_x-dimensional PSO: S;
$s_{idx} = 0$;
repeat
  if $s_{idx} \neq \tau$ **then**
    $S.\mathbf{x}_{idx} \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$;
  end
  $s_{idx} = (s_{idx} + 1) \bmod t_r$;
  for *each particle* $i = 1, \ldots, n_s$ **do**
    if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ **then**
      $S.\mathbf{y}_i = S.\mathbf{x}$;
    end
    if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ **then**
      $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
    end
  end
  Update velocities;
  Update position;
until *stopping condition is true*;

Finally, how are particles reinitialized? As mentioned earlier, velocities and/or positions can be reinitialized. Most approaches that reinitialize velocities set each velocity component to a random value constrained by the maximum allowed velocity. Venter and Sobieszczanski-Sobieski [874, 875] set the velocity vector to the cognitive component after reinitialization of position vectors.

Position vectors are usually initialized to a new position subject to boundary constraints; that is, $x_{ij}(t+1) \sim U(x_{min,j}, x_{max,j})$. Clerc [133] reinitializes particles on the basis of estimates of the local shape of the objective function. Clerc [135] also proposes alternatives, where a particle returns to its previous best position, and from

Algorithm 16.11 Self-Organized Criticality PSO

```
Create and initialize an n_x-dimensional PSO: S;
Set C_i = 0, ∀i = 1, ..., n_s;
repeat
    Evaluate fitness of all particles;
    Update velocities;
    Update positions;
    Calculate criticality for all particles;
    Reduce criticality for each particle;
    while ∃i = 1, ..., n_s such that C_i > C do
        Disperse criticality of particle i;
        Reinitialize x_i;
    end
until stopping condition is true;
```

there moves randomly for a fixed number of iterations.

A different approach to multi-start PSOs is followed in [863], as summarized in Algorithm 16.12. Particles are randomly initialized, and a PSO algorithm is executed until the swarm converges. When convergence is detected, the best position is recorded and all particles randomly initialized. The process is repeated until a stopping condition is satisfied, at which point the best recorded solution is returned. The best recorded solution can be refined using local search before returning the solution. The convergence criteria listed in Section 16.1.6 are used to detect convergence of the swarm.

16.5.6 Repelling Methods

Repelling methods have been used to improve the exploration abilities of PSO. Two of these approaches are described in this section.

Charged PSO

Blackwell and Bentley developed the charged PSO based on an analogy of electrostatic energy with charged particles [73, 74, 75]. The idea is to introduce two opposing forces within the dynamics of the PSO: an attraction to the center of mass of the swarm and inter-particle repulsion. The attraction force facilitates convergence to a single solution, while the repulsion force preserves diversity.

The charged PSO changes the velocity equation by adding a particle acceleration, $\mathbf{a}_i$, to the standard equation. That is,

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t)[\hat{y}_j(t) - x_{ij}(t)] + a_{ij}(t) \quad (16.86)$$

Algorithm 16.12 Multi-start Particle Swarm Optimization; $\hat{\mathbf{y}}$ is the best solution over all the restarts of the algorithm

Create and initialize an n_x-dimensional swarm, S;
repeat
  if $f(S.\hat{\mathbf{y}}) < f(\hat{\mathbf{y}})$ **then**
    $\hat{\mathbf{y}} = S.\hat{\mathbf{y}}$;
  end
  if *the swarm S has converged* **then**
    Reinitialize all particles;
  end
  for *each particle* $i = 1, \cdots, S.n_s$ **do**
    if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ **then**
      $S.\mathbf{y}_i = S.\mathbf{x}$;
    end
    if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ **then**
      $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
    end
  end
  Update velocities;
  Update position;
until *stopping condition is true*;
Refine $\hat{\mathbf{y}}$ using local search;
Return $\hat{\mathbf{y}}$ as the solution;

The acceleration determines the magnitude of inter-particle repulsion, defined as [75]

$$\mathbf{a}_i(t) = \sum_{l=1, i\neq l}^{n_s} \mathbf{a}_{il}(t) \tag{16.87}$$

with the repulsion force between particles i and l defined as

$$\mathbf{a}_{il}(t) = \begin{cases} \left(\frac{Q_i Q_l}{||\mathbf{x}_i(t)-\mathbf{x}_l(t)||^3}\right)(\mathbf{x}_i(t)-\mathbf{x}_l(t)) & \text{if } R_c \leq ||\mathbf{x}_i(t)-\mathbf{x}_l(t)|| \leq R_p \\ \left(\frac{Q_i Q_l(\mathbf{x}_i(t)-\mathbf{x}_l(t))}{R_c^2||\mathbf{x}_i(t)-\mathbf{x}_l(t)||}\right) & \text{if } ||\mathbf{x}_i(t)-\mathbf{x}_l(t)|| < R_c \\ 0 & \text{if } ||\mathbf{x}_i(t)-\mathbf{x}_l(t)|| > R_p \end{cases} \tag{16.88}$$

where Q_i is the charged magnitude of particle i, R_c is referred to as the core radius, and R_p is the perception limit of each particle.

Neutral particles have a zero charged magnitude, i.e. $Q_i = 0$. Only when $Q_i \neq 0$ are particles charged, and do particles repel from each other. Therefore, the standard PSO is a special case of the charged PSO with $Q_i = 0$ for all particles. Particle avoidance (inter-particle repulsion) occurs only when the separation between two particles is within the range $[R_c, R_p]$. In this case the smaller the separation, the larger the repulsion between the corresponding particles. If the separation between two particles becomes very small, the acceleration will explode to large values. The consequence will be that the particles never converge due to extremely large repulsion forces. To

prevent this, acceleration is fixed at the core radius for particle separations less than R_c. Particles that are far from one another, i.e. further than the particle perception limit, R_p, do not repel one another. In this case $\mathbf{a}_{il}(t) = 0$, which allows particles to move towards the global best position. The value of R_p will have to be optimized for each application.

The acceleration, $\mathbf{a}_i(t)$, is determined for each particle before the velocity update.

Blackwell and Bentley suggested as electrostatic parameters, $R_c = 1, R_p = \sqrt{3}x_{max}$ and $Q_i = 16$ [75].

Electrostatic repulsion maintains diversity, enabling the swarm to automatically detect and respond to changes in the environment. Empirical evaluations of the charged PSO in [72, 73, 75] have shown it to be very efficient in dynamic environments. Three types of swarms were defined and studied:

- **Neutral swarm**, where $Q_i = 0$ for all particles $i = 1, \ldots, n_s$.
- **Charged swarm**, where $Q_i > 0$ for all particles $i = 1, \ldots, n_s$. All particles therefore experience repulsive forces from the other particles (when the separation is less than r_p).
- **Atomic swarm**, where half of the swarm is charged ($Q_i > 0$) and the other half is neutral ($Q_i = 0$).

It was found that atomic swarms perform better than charged and neutral swarms [75]. As a possible explanation of why atomic swarms perform better than charged swarms, consider as worst case what will happen when the separation between particles never gets below R_c. If the separation between particles is always greater than or equal to R_c, particles repel one another, which never allows particles to converge to a single position. Inclusion of neutral particles ensures that these particles converge to an optimum, while the charged particles roam around to automatically detect and adjust to environment changes.

Particles with Spatial Extention

Particles with spatial extension were developed to prevent the swarm from prematurely converging [489, 877]. If one particle locates an optimum, then all particles will be attracted to the optimum – causing all particles to cluster closely. The spatial extension of particles allows some particles to explore other areas of the search space, while others converge to the optimum to further refine it. The exploring particles may locate a different, more optimal solution.

The objective of spatial extension is to dynamically increase diversity when particles start to cluster. This is achieved by adding a radius to each particle. If two particles collide, i.e. their radii intersect, then the two particles bounce off. Krink *et al.* [489] and Vesterstrøm and Riget [877] investigated three strategies for spatial extension:

- **random bouncing**, where colliding particles move in a random new direction at the same speed as before the collision;

- **realistic physical bouncing**; and
- simple **velocity-line bouncing**, where particles continue to move in the same direction but at a scaled speed. With scale factor in $[0, 1]$ particles slow down, while a scale factor greater than one causes acceleration to avoid a collision. A negative scale factor causes particles to move in the opposite direction to their previous movement.

Krink *et al.* showed that random bouncing is not as efficient as the consistent bouncing methods.

To ensure convergence of the swarm, particles should bounce off on the basis of a probability. An initial large bouncing probability will ensure that most collisions result in further exploration, while a small bouncing probability for the final steps will allow the swarm to converge. At all times, some particles will be allowed to cluster together to refine the current best solution.

16.5.7 Binary PSO

PSO was originally developed for continuous-valued search spaces. Kennedy and Eberhart developed the first discrete PSO to operate on binary search spaces [450, 451]. Since real-valued domains can easily be transformed into binary-valued domains (using standard binary coding or Gray coding), this binary PSO can also be applied to real-valued optimization problems after such transformation (see [450, 451] for applications of the binary PSO to real-valued problems).

For the binary PSO, particles represent positions in binary space. Each element of a particle's position vector can take on the binary value 0 or 1. Formally, $\mathbf{x}_i \in \mathbb{B}^{n_x}$, or $x_{ij} \in \{0, 1\}$. Changes in a particle's position then basically implies a mutation of bits, by flipping a bit from one value to the other. A particle may then be seen to move to near and far corners of a hypercube by flipping bits.

One of the first problems to address in the development of the binary PSO, is how to interpret the velocity of a binary vector. Simply seen, velocity may be described by the number of bits that change per iteration, which is the Hamming distance between $\mathbf{x}_i(t)$ and $\mathbf{x}_i(t+1)$, denoted by $\mathcal{H}(\mathbf{x}_i(t), \mathbf{x}_i(t+1))$. If $\mathcal{H}(\mathbf{x}_i(t), \mathbf{x}_i(t+1)) = 0$, zero bits are flipped and the particle does not move; $||\mathbf{v}_i(t)|| = 0$. On the other hand, $||\mathbf{v}_i(t)|| = n_x$ is the maximum velocity, meaning that all bits are flipped. That is, $\mathbf{x}_i(t+1)$ is the complement of $\mathbf{x}_i(t)$. Now that a simple interpretation of the velocity of a bit-vector is possible, how is the velocity of a single bit (single dimension of the particle) interpreted?

In the binary PSO, velocities and particle trajectories are rather defined in terms of probabilities that a bit will be in one state or the other. Based on this probabilistic view, a velocity $v_{ij}(t) = 0.3$ implies a 30% chance to be bit 1, and a 70% chance to be bit 0. This means that velocities are restricted to be in the range $[0, 1]$ to be interpreted as a probability. Different methods can be employed to normalize velocities such that $v_{ij} \in [0, 1]$. One approach is to simply divide each v_{ij} by the maximum velocity, $V_{max,j}$. While this approach will ensure velocities are in the range [0,1], consider

what will happen when the maximum velocities are large, with $v_{ij}(t) << V_{max,j}$ for all time steps, $t = 1, \ldots, n_t$. This will limit the maximum range of velocities, and thus the chances of a position to change to bit 1. For example, if $V_{max,j} = 10$, and $v_{ij}(t) = 5$, then the normalized velocity is $v'_{ij}(t) = 0.5$, with only a 50% chance that $x_{ij}(t+1) = 1$. This normalization approach may therefore cause premature convergence to bad solutions due to limited exploration abilities.

A more natural normalization of velocities is obtained by using the sigmoid function. That is,

$$v'_{ij}(t) = \text{sig}(v_{ij}(t)) = \frac{1}{1 + \text{e}^{-v_{ij}(t)}} \tag{16.89}$$

Using equation (16.89), the position update changes to

$$x_{ij}(t+1) = \begin{cases} 1 & \text{if } r_{3j}(t) < \text{sig}(v_{ij}(t+1)) \\ 0 & \text{otherwise} \end{cases} \tag{16.90}$$

with $r_{3j}(t) \sim U(0,1)$. The velocity, $v_{ij}(t)$, is now a probability for $x_{ij}(t)$ to be 0 or 1. For example, if $v_{ij}(t) = 0$, then $\text{prob}(x_{ij}(t+1) = 1) = 0.5$ (or 50%). If $v_{ij}(t) < 0$, then $\text{prob}(x_{ij}(t+1) = 1) < 0.5$, and if $v_{ij}(t) > 0$, then $\text{prob}(x_{ij}(t+1) = 1) > 0.5$. Also note that $\text{prob}(x_{ij}(t) = 0) = 1 - \text{prob}(x_{ij}(t) = 1)$. Note that x_{ij} can change even if the value of v_{ij} does not change, due to the random number r_{3j} in the equation above.

It is only the calculation of position vectors that changes from the real-valued PSO. The velocity vectors are still real-valued, with the same velocity calculation as given in equation (16.2), but including the inertia weight. That is, $\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}} \in \mathbb{B}^{n_x}$ while $\mathbf{v}_i \in \mathbb{R}^{n_x}$.

The binary PSO is summarized in Algorithm 16.13.

Algorithm 16.13 *binary* PSO

Create and initialize an n_x-dimensional swarm;
repeat
  for *each particle* $i = 1, \ldots, n_s$ **do**
    if $f(\mathbf{x}_i) < f(\mathbf{y}_i)$ **then**
      $\mathbf{y}_i = \mathbf{x}_i$;
    end
    if $f(\mathbf{y}_i) < f(\hat{\mathbf{y}})$ **then**
      $\hat{\mathbf{y}} = \mathbf{y}_i$;
    end
  end
  for *each particle* $i = 1, \ldots, n_s$ **do**
    update the velocity using equation (16.2);
    update the position using equation (16.90);
  end
until *stopping condition is true*;

16.6 Advanced Topics

This section describes a few PSO variations for solving constrained problems, multi-objective optimization problems, problems with dynamically changing objective functions and to locate multiple solutions.

16.6.1 Constraint Handling Approaches

A very simple approach to cope with constraints is to reject infeasible particles, as follows:

- Do not allow infeasible particles to be selected as personal best or neighborhood global best solutions. In doing so, infeasible particles will never influence other particles in the swarm. Infeasible particles are, however, pulled back to feasible space due to the fact that personal best and neighborhood best positions are in feasible space. This approach can only be efficient if the ratio of number of infeasible particles to feasible particles is small. If this ratio is too large, the swarm may not have enough diversity to effectively cover the (feasible) space.
- Reject infeasible particles by replacing them with new randomly generated positions from the feasible space. Reinitialization within feasible space gives these particles an opportunity to become best solutions. However, it is also possible that these particles (or any other particle for that matter), may roam outside the boundaries of feasible space. This approach may be beneficial in cases where the feasible space is made up of a number of disjointed regions. The strategy will allow particles to cross boundaries of one feasible region to explore another feasible region.

Most applications of PSO to constrained problems make use of penalty methods to penalize those particles that violate constraints [663, 838, 951].

Similar to the approach discussed in Section 12.6.1, Shi and Krohling [784] and Laskari *et al.* [504] converted the constrained problem to an unconstrained Lagrangian.

Repair methods, which allow particles to roam into infeasible space, have also been developed. These are simple methods that apply repairing operators to change infeasible particles to represent feasible solutions. Hu and Eberhart *et al.* [388] developed an approach where particles are not allowed to be attracted by infeasible particles.

The personal best of a particle changes only if the fitness of the current position is better and if the current position violates no constraints. This ensures that the personal best positions are always feasible. Assuming that neighborhood best positions are selected from personal best positions, it is also guaranteed that neighborhood best positions are feasible. In the case where the neighborhood best positions are selected from the current particle positions, neighborhood best positions should be updated only if no constraint is violated. Starting with a feasible set of particles, if a particle moves out of feasible space, that particle will be pulled back towards feasible space. Allowing particles to roam into infeasible space, and repairing them over time by

moving back to feasible best positions facilitates better exploration. If the feasible space consists of disjointed feasible regions, the chances are increased for particles to explore different feasible regions.

El-Gallad *et al.* [234] replaced infeasible particles with their feasible personal best positions. This approach assumes feasible initial particles and that personal best positions are replaced only with feasible solutions. The approach is very similar to that of Hu and Eberhart [388], but with less diversity. Replacement of an infeasible particle with its feasible personal best, forces an immediate repair. The particle is immediately brought back into feasible space. The approach of Hu and Eberhart allow an infeasible particle to explore more by pulling it back into feasible space over time. But, keep in mind that during this exploration, the infeasible particle will have no influence on the rest of the swarm while it moves within infeasible space.

Venter and Sobieszczanski-Sobieski [874, 875] proposed repair of infeasible solutions by setting

$$\mathbf{v}_i(t) = \mathbf{0} \tag{16.91}$$

$$\mathbf{v}_i(t+1) = c_1\mathbf{r}_1(t)(\mathbf{y}_i(t) - \mathbf{x}_i(t)) + c_2\mathbf{r}_2(t)(\hat{\mathbf{y}}(t) - \mathbf{x}_i(t)) \tag{16.92}$$

for all infeasible particles, i. In other words, the memory of previous velocity (direction of movement) is deleted for infeasible particles, and the new velocity depends only on the cognitive and social components. Removal of the momentum has the effect that infeasible particles are pulled back towards feasible space (assuming that the personal best positions are only updated if no constraints are violated).

16.6.2 Multi-Objective Optimization

A great number of PSO variations can be found for solving MOPs. This section describes only a few of these but provides references to other approaches.

The *dynamic neighborhood* MOPSO, developed by Hu and Eberhart [387], dynamically determines a new neighborhood for each particle in each iteration, based on distance in objective space. Neighborhoods are determined on the basis of the simplest objective. Let $f_1(\mathbf{x})$ be the simplest objective function, and let $f_2(\mathbf{x})$ be the second objective. The neighbors of a particle are determined as those particles closest to the particle with respect to the fitness values for objective $f_1(\mathbf{x})$. The neighborhood best particle is selected as the particle in the neighborhood with the best fitness according to the second objective, $f_1(\mathbf{x})$. Personal best positions are replaced only if a particle's new position dominates its current personal best solution.

The dynamic neighborhood MOPSO has a few disadvantages:

- It is not easily scalable to more than two objectives, and its usability is therefore restricted to MOPs with two objectives.
- It assumes prior knowledge about the objectives to decide which is the most simple for determination of neighborhoods.

- It is sensitive to the ordering of objectives, since optimization is biased towards improving the second objective.

Parsopoulos and Vrahatis [659, 660, 664, 665] developed the *vector evaluated* PSO (VEPSO), on the basis of the vector-evaluated genetic algorithm (VEGA) developed by Schaffer [761] (also refer to Section 9.6.3). VEPSO uses two sub-swarms, where each sub-swarm optimizes a single objective. This algorithm is therefore applicable to MOPs with only two objectives. VEPSO follows a kind of coevolutionary approach. The global best particle of the first swarm is used in the velocity equation of the second swarm, while the second swarm's global best particle is used in the velocity update of the first swarm. That is,

$$\begin{aligned} S_1.v_{ij}(t+1) &= wS_1.v_{ij}(t) + c_1 r_{1j}(t)(S_1.y_{ij}(t) - S_1.x_{ij}(t)) \\ &+ c_2 r_{2j}(t)(S_2.\hat{y}_i(t) - S_1.x_{ij}(t)) \qquad (16.93) \\ S_2.v_{ij}(t+1) &= wS_2.v_{ij}(t) + c_1 r_{1j}(t)(S_2.y_{ij}(t) - S_2.x_{ij}(t)) \\ &+ c_2 r_{ij}(t)(S_1.\hat{y}_j(t) - S.x_{2j}(t)) \qquad (16.94) \end{aligned}$$

where sub-swarm S_1 evaluates individuals on the basis of objective $f_1(\mathbf{x})$, and sub-swarm S_2 uses objective $f_2(\mathbf{x})$.

The MOPSO algorithm developed by Coello Coello and Lechuga is one of the first PSO-based MOO algorithms that extensively uses an archive [147, 148]. This algorithm is based on the Pareto archive ES (refer to Section 12.6.2), where the objective function space is separated into a number of hypercubes.

A truncated archive is used to store non-dominated solutions. During each iteration, if the archive is not yet full, a new particle position is added to the archive if the particle represents a non-dominated solution. However, because of the size limit of the archive, priority is given to new non-dominated solutions located in less populated areas, thereby ensuring that diversity is maintained. In the case that members of the archive have to be deleted, those members in densely populated areas have the highest probability of deletion. Deletion of particles is done during the process of separating the objective function space into hypercubes. Densely populated hypercubes are truncated if the archive exceeds its size limit. After each iteration, the number of members of the archive can be reduced further by eliminating from the archive all those solutions that are now dominated by another archive member.

For each particle, a global guide is selected to guide the particle toward less dense areas of the Pareto front. To select a guide, a hypercube is first selected. Each hypercube is assigned a selective fitness value,

$$f_{sel}(H_h) = \frac{\alpha}{f_{del}(H_h)} \qquad (16.95)$$

where $f_{del}(H_h) = H_h.n_s$ is the deletion fitness value of hypercube H_h; $\alpha = 10$ and $H_h.n_s$ represents the number of nondominated solutions in hypercube H_h. More densely populated hypercubes will have a lower score. Roulette wheel selection is then used to select a hypercube, H_h, based on the selection fitness values. The global guide for particle i is selected randomly from among the members of hypercube H_h.

Hence, particles will have different global guides. This ensures that particles are attracted to different solutions. The local guide of each particle is simply the personal best position of the particle. Personal best positions are only updated if the new position, $\mathbf{x}_i(t+1) \prec \mathbf{y}_i(t)$. The global guide replaces the global best, and the local guide replaces the personal best in the velocity update equation.

In addition to the normal position update, a mutation operator (also referred to as a craziness operator in the context of PSO) is applied to the particle positions. The degree of mutation decreases over time, and the probability of mutation also decreases over time. That is,

$$x_{ij}(t+1) = N(0, \sigma(t))x_{ij}(t) + v_{ij}(t+1) \tag{16.96}$$

where, for example

$$\sigma(t) = \sigma(0)\, \mathrm{e}^{-t} \tag{16.97}$$

with $\sigma(0)$ an initial large variance.

The MOPSO developed by Coello Coello and Lechuga is summarized in Algorithm 16.14.

Algorithm 16.14 Coello Coello and Lechuga MOPSO

Create and initialize an n_x-dimensional swarm S;
Let $A = \emptyset$ and $A.n_s = 0$;
Evaluate all particles in the swarm;
for *all non-dominated* $\mathbf{x}_i$ **do**
    $A = A \cup \{\mathbf{x}_i\}$;
end
Generate hypercubes;
Let $\mathbf{y}_i = \mathbf{x}_i$ for all particles;
repeat
    Select global guide, $\hat{\mathbf{y}}$;
    Select local guide, $\mathbf{y}_i$;
    Update velocities using equation (16.2);
    Update positions using equation (16.96);
    Check boundary constraints;
    Evaluate all particles in the swarm;
    Update the repository, A;
until *stopping condition is true*;

Li [517] applied a nondominated sorting approach similar to that described in Section 12.6.2 to PSO. Other MOO algorithms can be found in [259, 389, 609, 610, 940, 956].

16.6.3 Dynamic Environments

Early results of the application of the PSO to type I environments (refer to Section A.9) with small spatial severity showed that the PSO has an implicit ability to track changing optima [107, 228, 385]. Each particle progressively converges on a point on the line that connects its personal best position with the global best position [863, 870]. The trajectory of a particle can be described by a sinusoidal wave with diminishing amplitude around the global best position [651, 652]. If there is a small change in the location of an optimum, it is likely that one of these oscillating particles will discover the new, nearby optimum, and will pull the other particles to swarm around the new optimum.

However, if the spatial severity is large, causing the optimum to be displaced outside the radius of the contracting swarm, the PSO will fail to locate the new optimum due to loss of diversity. In such cases mechanisms need to be employed to increase the swarm diversity.

Consider spatial changes where the value of the optimum remains the same after the change, i.e. $f(\mathbf{x}^*(t)) = f(\mathbf{x}^*(t+1))$, with $\mathbf{x}^*(t) \neq \mathbf{x}^*(t+1)$. Since the fitness remains the same, the global best position does not change, and remains at the old optimum. Similarly, if $f(\mathbf{x}^*(t)) > f(\mathbf{x}^*(t+1))$, assuming minimization, the global best position will also not change. Consequently, the PSO will fail to track such a changing minimum. This problem can be solved by re-evaluating the fitness of particles at time $t+1$ and updating global best and personal best positions. However, keep in mind that the same problem as discussed above may still occur if the optimum is displaced outside the radius of the swarm.

One of the goals of optimization algorithms for dynamic environments is to locate the optimum and then to track it. The self-adaptation ability of the PSO to track optima (as discussed above) assumes that the PSO did not converge to an equilibrium state in its first goal to locate the optimum. When the swarm reaches an equilibrium (i.e. converged to a solution), $\mathbf{v}_i = \mathbf{0}$. The particles have no momentum and the contributions from the cognitive and social components are zero. The particles will remain in this stable state even if the optimum does change. In the case of dynamic environments, it is possible that the swarm reaches an equilibrium if the temporal severity is low (in other words, the time between consecutive changes is large). It is therefore important to read the literature on PSO for dynamic environments with this aspect always kept in mind.

The next aspects to consider are the influence of particle memory, velocity clamping and the inertia weight. The question to answer is: to what extent do these parameters and characteristics of the PSO limit or promote tracking of changing optima? These aspects are addressed next:

- Each particle has a memory of its best position found thus far, and velocity is adjusted to also move towards this position. Similarly, each particle retains information about the global best (or local best) position. When the environment changes this information becomes stale. If, after an environment change, particles are still allowed to make use of this, now stale, information, they are

drawn to the old optimum – an optimum that may no longer exist in the changed environment. It can thus be seen that particle memory (from which the global best is selected) is detrimental to the ability of PSO to track changing optima. A solution to this problem is to reinitialize or to re-evaluate the personal best positions (particle memory).

- The inertia weight, together with the acceleration coefficients balance the exploration and exploitation abilities of the swarm. The smaller w, the less the swarm explores. Usually, w is initialized with a large value that decreases towards a small value (refer to Section 16.3.2). At the time that a change occurs, the value of w may have reduced too much, thereby limiting the swarm's exploration ability and its chances of locating the new optimum. To alleviate this problem, the value of w should be restored to its initial large value when a change in the environment is detected. This also needs to be done for the acceleration coefficients if adaptive accelerations are used.
- Velocity clamping also has a large influence on the exploration ability of PSO (refer to 16.3.1). Large velocity clamping (i.e. small $V_{max,j}$ values) limits the step sizes, and more rapidly results in smaller momentum contributions than lesser clamping. For large velocity clamping, the swarm will have great difficulty in escaping the old optimum in which it may find itself trapped. To facilitate tracking of changing objectives, the values of $V_{max,j}$ therefore need to be chosen carefully.

When a change in environment is detected, the optimization algorithm needs to react appropriately to adapt to these changes. To allow timeous and efficient tracking of optima, it is important to correctly and timeously detect if the environment did change. The task is easy if it is known that changes occur periodically (with the change frequency known), or at predetermined intervals. It is, however, rarely the case that prior information about when changes occur is available. An automated approach is needed to detect changes based on information received from the environment.

Carlisle and Dozier [106, 109] proposed the use of a sentry particle, or more than one sentry particle. The task of the sentry is to detect any change in its local environment. If only one sentry is used, the same particle can be used for all iterations. However, feedback from the environment will then only be from one small part of the environment. Around the sentry the environment may be static, but it may have changed elsewhere. A better strategy is to select the particle randomly from the swarm for each iteration. Feedback is then received from different areas of the environment.

A sentry particle stores a copy of its most recent fitness value. At the start of the next iteration, the fitness of the sentry particle is re-evaluated and compared with its previously stored value. If there is a difference in the fitness, a change has occurred.

More sentries allow simultaneous feedback from more parts of the environment. If multiple sentries are used detection of changes is faster and more reliable. However, the fitness of a sentry particle is evaluated twice per iteration. If fitness evaluation is costly, multiple sentries will significantly increase the computational complexity of the search algorithm.

As an alternative to using randomly selected sentry particles, Hu and Eberhart [385]

proposed to monitor changes in the fitness of the global best position. By monitoring the fitness of the global best particle, change detection is based on globally provided information. To increase the accuracy of change detection, Hu and Eberhart [386] later also monitored the global second-best position. Detection of the second-best and global best positions limits the occurrence of false alarms. Monitoring of these global best positions is based on the assumption that if the optimum location changes, then the optimum value of the current location also changes.

One of the first studies in the application of PSO to dynamic environments came from Carlisle and Dozier [107], where the efficiency of different velocity models (refer to Section 16.3.5) has been evaluated. Carlisle and Dozier observed that the social-only model is faster in tracking changing objectives than the full model. However, the reliability of the social-only model deteriorates faster than the full model for larger update frequencies. The selfless and cognition-only models do not perform well on changing environments. Keep in mind that these observations were without changing the original velocity models, and should be viewed under the assumption that the swarm had not yet reached an equilibrium state. Since this study, a number of other studies have been done to investigate how the PSO should be changed to track dynamic optima. These studies are summarized in this section.

From these studies in dynamic environments, it became clear that diversity loss is the major reason for the failure of PSO to achieve more efficient tracking.

Eberhart and Shi [228] proposed using the standard PSO, but with a dynamic, randomly selected inertia coefficient. For this purpose, equation (16.24) is used to select a new $w(t)$ for each time step. In their implementation, $c_1 = c_2 = 1.494$, and velocity clamping was not done. As motivation for this change, recall from Section 16.3.1 that velocity clamping restricts the exploration abilities of the swarm. Therefore, removal of velocity clamping facilitates larger exploration, which is highly beneficial. Furthermore, from Section 16.3.2, the inertia coefficient controls the exploration–exploitation trade-off. Since it cannot be predicted in dynamically changing environments if exploration or exploitation is preferred, the randomly changing $w(t)$ ensures a good mix of focusing on both exploration and exploitation.

While this PSO implementation presented promising results, the efficiency is limited to type I environments with a low severity and type II environments where the value of the optimum is better after the change in the environment. This restriction is mainly due to the memory of particles (i.e. the personal best positions and the global best selected from the personal best positions), and that changing the inertia will not be able to kick the swarm out of the current optimum when $\mathbf{v}_i(t) \approx 0, \forall i = 1, \ldots, n_s$.

A very simple approach to increase diversity is to reinitialize the swarm, which means that all particle positions are set to new random positions, and the personal best and neighborhood best positions are recalculated. Eberhart and Shi [228] suggested the following approaches:

- Do not reinitialize the swarm, and just continue to search from the current position. This approach only works for small changes, and when the swarm has not yet reached an equilibrium state.

- Reinitialize the entire swarm. While this approach does neglect the swarm's memory and breaks the swarm out of a current optimum, it has the disadvantage that the swarm needs to search all over again. If changes are not severe, the consequence is an unnecessary increase in computational complexity due to an increase in the number of iterations to converge to the new optimum. The danger is also that the swarm may converge on a worse local optimum. Reinitialization of the entire swarm is more effective under severe environment changes.
- Reinitialize only parts of the swarm, but make sure to retain the global best position. The reinitialization of a percentage of the particle positions injects more diversity, and also preserves memory of the search space by keeping potentially good particles (existing particles may be close to the changed optimum). Hu and Eberhart experimented with a number of reinitialization percentages [386], and observed that lower reinitialization is preferred for smaller changes. Larger changes require more particles to be reinitialized for efficient tracking. It was also found empirically that total reinitialization is not efficient.

Particles' memory of previous best positions is a major cause of the inefficiency of the PSO in tracking changing optima. This statement is confirmed by the results of Carlisle and Dozier [107], where it was found that the cognition-only model showed poor performance. The cognitive component promotes the nostalgic tendency to return to previously found best positions. However, after a change in the environment, these positions become stale since they do not necessarily reflect the search space for the changed environment. The consequence is that particles are attracted to outdated positions.

Carlisle and Dozier [107], and Hu and Eberhart [385] proposed that the personal best positions of all particles be reset to the current position of the particle when a change is detected. This action effectively clears the memory of all particles. Resetting of the personal best positions is only effective when the swarm has not yet converged to a solution [386]. If this is the case, $w(t)\mathbf{v}(t) \approx 0$ and the contribution of the cognitive component will be zero for all particles, since $(\mathbf{y}_i(t) - x_i(t)) = 0, \forall i = 1, \ldots, n_s$. Furthermore, the contribution of the social component will also be approximately zero, since, at convergence, all particles orbit around the same global best position (with an approximately zero swarm radius). Consequently, velocity updates are very small, as is the case for position updates.

To address the above problem, resetting of the personal best positions can be combined with partial reinitialization of the swarm [386]. In this case, reinitialization increases diversity while resetting of the personal best positions prevents return to out-of-date positions.

Looking more carefully at the resetting of personal best positions, it may not always be a good strategy to simply reset all personal best positions. Remember that by doing so, all particles forget any experience that they have gained about the search space during the search process. It might just be that after an environment change, the personal best position of a particle is closer to the new goal than the current position. Under less severe changes, it is possible that the personal best position remains the best position for that particle. Carlisle and Dozier proposed that the personal best

positions first be evaluated after the environment change and if the personal best position is worse than the current position for the new environment, only then reset the personal best position [109].

The global best position, if selected from the personal best positions, also adds to the memory of the swarm. If the environment changes, then the global best position is based on stale information. To address this problem, any of the following strategies can be followed:

- After resetting of the personal best positions, recalculate the global best position from the more up-to-date personal best positions.
- Recalculate the global best position only if it is worse under the new environment.
- Find the global best position only from the current particle positions and not from the personal best positions [142].

The same reasoning applies to neighborhood best positions.

The charged PSO discussed in Section 16.5.6 has been developed to track dynamically changing optima, facilitated by the repelling mechanism. Other PSO implementations for dynamic environments can be found in [142, 518, 955].

16.6.4 Niching PSO

Although the basic PSO was developed to find single solutions to optimization problems, it has been observed for PSO implementations with special parameter choices that the basic PSO has an inherent ability to find multiple solutions. Agrafiotis and Cedeño, for example, observed for a specific problem that particles coalesce into a small number of local minima [11]. However, any general conclusions about the PSO's niching abilities have to be reached with extreme care, as explained below.

The main driving forces of the PSO are the cognitive and social components. It is the social component that prevents speciation. Consider, for example, the *gbest* PSO. The social component causes all particles to be attracted towards the best position obtained by the swarm, while the cognitive component exerts an opposing force (if the global best position is not the same as a particle's personal best position) towards a particle's own best solution. The resulting effect is that particles converge to a point between the global best and personal best positions, as was formally proven in [863, 870]. If the PSO is executed until the swarm reaches an equilibrium point, then each particle will have converged to such a point between the global best position and the personal best position of that particle, with a final zero velocity.

Brits *et al.* [91] showed empirically that the *lbest* PSO succeeds in locating a small percentage of optima for a number of functions. However, keep in mind that these studies terminated the algorithms when a maximum number of function evaluations was exceeded, and not when an equilibrium state was reached. Particles may therefore still have some momentum, further exploring the search space. It is shown in [91] that the basic PSO fails miserably compared to specially designed *lbest* PSO niching algorithms. Of course, the prospect of locating more multiple solutions will improve with

an increase in the number of particles, but at the expense of increased computational complexity. The basic PSO also fails another important objective of niching algorithms, namely to maintain niches. If the search process is allowed to continue, more particles converge to the global best position in the process to reach an equilibrium, mainly due to the social component (as explained above).

Emanating from the discussion in this section, it is desirable to rather adapt the basic PSO with true speciation abilities. One such algorithm, the NichePSO is described next.

The NichePSO was developed to find multiple solutions to general multi-modal problems [89, 88, 91]. The basic operating principle of NichePSO is the self-organization of particles into independent sub-swarms. Each sub-swarm locates and maintains a niche. Information exchange is only within the boundaries of a sub-swarm. No information is exchanged between sub-swarms. This independency among sub-swarms allows sub-swarms to maintain niches. To emphasize: each sub-swarm functions as a stable, individual swarm, evolving on its own, independent of individuals in other swarms.

The NichePSO starts with one swarm, referred to as the *main* swarm, containing all particles. As soon as a particle converges on a potential solution, a sub-swarm is created by grouping together particles that are in close proximity to the potential solution. These particles are then removed from the main swarm, and continue within their sub-swarm to refine (and to maintain) the solution. Over time, the main swarm shrinks as sub-swarms are spawned from it. NichePSO is considered to have converged when sub-swarms no longer improve the solutions that they represent. The global best position from each sub-swarm is then taken as an optimum (solution).

The NichePSO is summarized in Algorithm 16.15. The different steps of the algorithm are explained in more detail in the following sections.

Algorithm 16.15 NichePSO Algorithm

Create and initialize a n_x-dimensional *main* swarm, S;
repeat
  Train the main swarm, S, for one iteration using the *cognition-only* model;
  Update the fitness of each main swarm particle, $S.\mathbf{x}_i$;
  for *each sub-swarm* S_k **do**
    Train sub-swarm particles, $S_k.\mathbf{x}_i$, using a full model PSO;
    Update each particle's fitness;
    Update the swarm radius $S_k.R$;
  endFor
  If possible, merge sub-swarms;
  Allow sub-swarms to absorb any particles from the main swarm that moved into the sub-swarm;
  If possible, create new sub-swarms;
until *stopping condition is true*;
Return $S_k.\hat{\mathbf{y}}$ for each sub-swarm S_k as a solution;

Main Swarm Training

The main swarm uses a cognition-only model (refer to Section 16.3.5) to promote exploration of the search space. Because the social component serves as an attractor for all particles, it is removed from the velocity update equation. This allows particles to converge on different parts of the search space.

It is important to note that velocities must be initialized to zero.

Sub-swarm Training

The sub-swarms are independent swarms, trained using a full model PSO (refer to Section 16.1). Using a full model to train sub-swarms allows particle positions to be adjusted on the basis both of particle's own experience (the cognitive component) and of socially obtained information (the social component). While any full model PSO can be used to train the sub-swarms, the NichePSO as presented in [88, 89] uses the GCPSO (discussed in Section 16.5.1). The GCPSO is used since it has guaranteed convergence to a local minimum [863], and because the GCPSO has been shown to perform well on extremely small swarms [863]. The latter property of GCPSO is necessary because sub-swarms initially consist of only two particles. The *gbest* PSO has a tendency to stagnate with such small swarms.

Identification of Niches

A sub-swarm is formed when a particle seems to have converged on a solution. If a particle's fitness shows little change over a number of iterations, a sub-swarm is created with that particle and its closest topological neighbor. Formally, the standard deviation, σ_i, in the fitness $f(\mathbf{x}_i)$ of each particle is tracked over a number of iterations. If $\sigma_i < \epsilon$, a sub-swarm is created. To avoid problem-dependence, σ_i is normalized according to the domain. The closest neighbor, l, to the position $\mathbf{x}_i$ of particle i is computed using Euclidean distance, i.e.

$$l = \arg\min_a \{||\mathbf{x}_i - \mathbf{x}_a||\} \tag{16.98}$$

with $1 \leq i, a \leq S.n_s, i \neq a$ and $S.n_s$ is the size of the main swarm.

The sub-swarm creation, or niche identification process is summarized in Algorithm 16.16. In this algorithm, $\mathcal{Q}$ represents the set of sub-swarms, $\mathcal{Q} = \{S_1, \cdots, S_K\}$, with $|\mathcal{Q}| = K$. Each sub-swarm has $S_k.n_s$ particles. During initialization of the NichePSO, K is initialized to zero, and $\mathcal{Q}$ is initialized to the empty set.

Absorption of Particles into a Sub-swarm

It is likely that particles of the main swarm move into the area of the search space covered by a sub-swarm S_k. Such particles are merged with the sub-swarm, for the following reasons:

Algorithm 16.16 NichePSO Sub-swarm Creation Algorithm

if $\sigma_i < \epsilon$ **then**
  $k = k + 1$;
  Create sub-swarm $S_k = \{\mathbf{x}_i, \mathbf{x}_l\}$;
  Let $\mathcal{Q} \leftarrow \mathcal{Q} \cup S_k$;
  Let $S \leftarrow S \setminus S_k$;
end

- Inclusion of particles that traverse the search space of an existing sub-swarm may improve the diversity of the sub-swarm.
- Inclusion of such particles into a sub-swarm will speed up their progression towards an optimum through the addition of social information within the sub-swarm.

More formally, if for particle i,

$$||\mathbf{x}_i - S_k.\hat{\mathbf{y}}|| \leq S_k.R \tag{16.99}$$

then absorb particle i into sub-swarm S_k:

$$S_k \leftarrow S_k \cup \{\mathbf{x}_i\} \tag{16.100}$$
$$S \leftarrow S \setminus \{\mathbf{x}_i\} \tag{16.101}$$

In equation (16.99), $S_k.R$ refers to the radius of sub-swarm S_k, defined as

$$S_k.R = \max\{||S_k.\hat{\mathbf{y}} - S_k.\mathbf{x}_i||\}, \quad \forall i = 1, \ldots, S_k.n_s \tag{16.102}$$

where $S_k.\hat{\mathbf{y}}$ is the global best position of sub-swarm S_k.

Merging Sub-swarms

It is possible that more than one sub-swarm form to represent the same optimum. This is due to the fact that sub-swarm radii are generally small, approximating zero as the solution represented is refined over time. It may then happen that a particle that moves toward a potential solution is not absorbed into a sub-swarm that is busy refining that solution. Consequently, a new sub-swarm is created. This leads to the redundant refinement of the same solution by multiple swarms. To solve this problem, similar sub-swarms are merged. Swarms are considered similar if the hyperspace defined by their particle positions and radii intersect. The new, larger sub-swarm then benefits from the social information and experience of both swarms. The resulting sub-swarm usually exhibits larger diversity than the original, smaller sub-swarms.

Formally stated, two sub-swarms, S_{k_1} and S_{k_2}, intersect when

$$||S_{k_1}.\hat{\mathbf{y}} - S_{k_2}.\hat{\mathbf{y}}|| < (S_{k_1}.R + S_{k_2}.R) \tag{16.103}$$

If $S_{k_1}.R = S_{k_2}.R = 0$, the condition in equation (16.103) fails, and the following merging test is considered:

$$||S_{k_1}.\hat{\mathbf{y}} - S_{k_2}.\hat{\mathbf{y}}|| < \mu \tag{16.104}$$

where μ is a small value close to zero, e.g. $\mu = 10^{-3}$. If μ is too large, the result may be that dissimilar swarms are merged with the consequence that a candidate solution may be lost.

To avoid tuning of μ over the domain of the search space, $||S_{k_1}.\hat{\mathbf{y}} - S_{k_2}.\hat{\mathbf{y}}||$ are normalized to the interval $[0, 1]$.

Stopping Conditions

Any of a number of stopping conditions can be used to terminate the search for multiple solutions. It is important that the stopping conditions ensure that each sub-swarm has converged onto a unique solution.

The reader is referred to [88] for a more detailed analysis of the NichePSO.

16.7 Applications

PSO has been used mostly to optimize functions with continuous-valued parameters. One of the first applications of PSO was in training neural networks, as summarized in Section 16.7.1. A game learning application of PSO is summarized in Section 16.7.3. Other applications are listed in Table 16.1.

16.7.1 Neural Networks

The first applications of PSO was to train feedforward neural networks (FFNN) [224, 446]. These first studies in training FFNNs using PSO have shown that the PSO is an efficient alternative to NN training. Since then, numerous studies have further explored the power of PSO as a training algorithm for a number of different NN architectures. Studies have also shown for specific applications that NNs trained using PSO provide more accurate results. Section 16.7.1 and Section 16.7.1 respectively address supervised and unsupervised training. NN architecture selection approaches are discussed in Section 16.7.2.

Supervised Learning

The main objective in supervised NN training is to adjust a set of weights such that an objective (error) function is minimized. Usually, the SSE error is used as the objective function (refer to equation (2.17)).

In order to use PSO to train an NN, a suitable representation and fitness function needs to be found. Since the objective is to minimize the error function, the fitness function is simply the given error function (e.g. the SSE given in equation (2.17)). Each particle represents a candidate solution to the optimization problem, and since the weights of a trained NN are a solution, a single particle represents one complete network. Each component of a particle's position vector represents one NN weight or bias. Using this representation, any of the PSO algorithms can be used to find the best weights for an NN to minimize the error function.

Eberhart and Kennedy [224, 446] provided the first results of applying the basic PSO to the training of FFNNs. Mendes *et al.* [575] evaluated the performance of different neighborhood topologies (refer to Section 16.2) on training FFNNs. The topologies tested included the star, ring, pyramid, square and four clusters topologies. Hirata *et al.* [367], and Gudise and Venayagamoorthy [338] respectively evaluated the ability of *lbest* and *gbest* PSO to train FFNNs. Al-Kazemi and Mohan applied the multi-phase PSO (refer to Section 16.5.4) to NN training. Van den Bergh and Engelbrecht showed that the cooperative PSO (refer to Section 16.5.4) and GCPSO (refer to Section 16.5.1) perform very well as NN training algorithms for FFNNs [863, 864]. Settles and Rylander also applied the cooperative PSO to NN training [776].

He *et al.* [359] used the basic PSO to train a fuzzy NN, after which accurate rules have been extracted from the trained network.

The real power of PSO as an NN training algorithm was illustrated by Engelbrecht and Ismail [247, 406, 407, 408] and Van den Bergh and Engelbrecht [867] in training NNs with product units. The basic PSO and cooperative PSO have been shown to outperform optimizers such as gradient-descent, LeapFrog (refer to Section 3.2.4), scaled conjugate gradient (refer to Section 3.2.3) and genetic algorithms (refer to Chapter 9). Paquet and Engelbrecht [655, 656] have further illustrated the power of the linear PSO in training support vector machines.

Salerno [753] used the basic PSO to train Elman recurrent neural networks (RNN). The PSO was successful for simple problems, but failed to train an RNN for parsing natural language phrases. Tsou and MacNish [854] also showed that the basic PSO fails to train certain RNNs, and developed a Newton-based PSO that successfully trained a RNN to learn regular language rules. Juang [430] combined the PSO as an operator in a GA to evolve RNNs. The PSO was used to enhance elitist individuals.

Unsupervised Learning

While plenty of work has been done in using PSO algorithms to train supervised networks, not much has been done to show how PSO performs as an unsupervised training algorithm. Xiao *et al.* [921] used the *gbest* PSO to evolve weights for a self-organizing map (SOM) [476] (also refer to Section 4.5) to perform gene clustering. The training process consists of two phases. The first phase uses PSO to find an initial weight set for the SOM. The second phase initializes a PSO with the weight set obtained from the first phase. The PSO is then used to refine this weight set.

Messerschmidt and Engelbrecht [580], and Franken and Engelbrecht [283, 284, 285] used the *gbest*, *lbest* and Von Neumann PSO algorithms as well as the GCPSO to coevolve neural networks to approximate the evaluation function of leaf nodes in game trees. No target values were available; therefore NNs compete in game tournaments against groups of opponents in order to determine a score or fitness for each NN. During the coevolutionary training process, weights are adjusted using PSO algorithms to have NNs (particles) move towards the best game player. The coevolutionary training process has been applied successfully to the games of tick-tack-toe, checkers, bao, the iterated prisoner's dilemma, and a probabilistic version of tick-tack-toe. For more information, refer to Section 15.2.3.

16.7.2 Architecture Selection

Zhang and Shao [949, 950] proposed a PSO model to simultaneously optimize NN weights and architecture. Two swarms are maintained: one swarm optimizes the architecture, and the other optimizes weights. Particles in the architecture swarm are two-dimensional, with each particle representing the number of hidden units used and the connection density. The first step of the algorithm randomly initializes these architecture particles within predefined ranges.

The second swarm's particles represent actual weight vectors. For each architecture particle, a swarm of particles is created by randomly initializing weights to correspond with the number of hidden units and the connection density specified by the architecture particle. Each of these swarms is evolved using a PSO, where the fitness function is the MSE computed from the training set. After convergence of each NN weights swarm, the best weight vector is identified from each swarm (note that the selected weight vectors are of different architectures). The fitness of these NNs are then evaluated using a validation set containing patterns not used for training. The obtained fitness values are used to quantify the performance of the different architecture specifications given by the corresponding particles of the architecture swarm. Using these fitness values, the architecture swarm is further optimized using PSO.

This process continues until a termination criterion is satisfied, at which point the global best particle is one with an optimized architecture and weight values.

16.7.3 Game Learning

Messerschmidt and Engelbrecht [580] developed a PSO approach to train NNs in a coevolutionary mechanism to approximate the evaluation function of leaf nodes in a game tree as described in Section 15.2.3. The initial model was applied to the simple game of tick-tack-toe.

As mentioned in Section 15.2.3 the training process is not supervised. No target evaluation of board states is provided. The lack of desired outputs for the NN necessitates a coevolutionary training mechanism, where NN agents compete against other agents, and all inferior NNs strive to beat superior NNs. For the PSO coevolutionary training

algorithm, summarized in Algorithm 16.17, a swarm of particles is randomly created, where each particle represents a single NN. Each NN plays in a tournament against a group of randomly selected opponents, selected from a competition pool (usually consisting of all the current particles of the swarm and all personal best positions). After each NN has played against a group of opponents, it is assigned a score based on the number of wins, losses and draws achieved. These scores are then used to determine personal best and neighborhood best solutions. Weights are adjusted using the position and velocity updates of any PSO algorithm.

Algorithm 16.17 PSO Coevolutionary Game Training Algorithm

```
Create and randomly initialize a swarm of NNs;
repeat
    Add each personal best position to the competition pool;
    Add each particle to the competition pool;
    for each particle (or NN) do
        Randomly select a group of opponents from the competition pool;
        for each opponent do
            Play a game (using game trees to determine next moves) against the
            opponents, playing as first player;
            Record if game was won, lost or drawn;
            Play a game against same opponent, but as the second player;
            Record if game was won, lost or drawn;
        end
        Determine a score for each particle;
        Compute new personal best positions based on scores;
    end
    Compute neighbor best positions;
    Update particle velocities;
    Update particle positions;
until stopping condition is true;
Return global best particle as game-playing agent;
```

The basic algorithm as given in Algorithm 16.17 has been applied successfully to the zero-sum games of tick-tack-toe [283, 580], checkers [284], and bao [156]. Franken and Engelbrecht also applied the approach to the non-zero-sum game, the iterated prisoner's dilemma [285]. A variant of the approach, using two competing swarms has recently been used to train agents for a probabilistic version of tick-tac-toe [654].

16.8 Assignments

1. Discuss in detail the differences and similarities between PSO and EAs.
2. Discuss how PSO can be used to cluster data.
3. Why is it better to base the calculation of neighborhoods on the index assigned to particles and not on geometrical information such as Euclidean distance?

Table 16.1 Applications of Particle Swarm Optimization

Application	References
Clustering	[638, 639]
Design	[7, 316, 700, 874, 875]
Scheduling	[7, 470, 707, 754]
Planning	[771, 775, 856, 886]
Controllers	[143, 155, 297, 959]
Power systems	[6, 296, 297, 299, 437, 444]
Bioinformatics	[627, 705, 921]
Data mining	[805]

4. Explain how PSO can be used to approximate functions using an n-th order polynomial.
5. Show how PSO can be used to solve a system of equations.
6. If the basic PSO is used to solve a system of equations, what problem(s) do you foresee? How can these be addressed?
7. How can PSO be used to solve problems with discrete-valued parameters?
8. For the predator-prey PSO, what will be the effect if more than one predator is used?
9. Critically discuss the following strategy applied to a dynamic inertia weight: Start with an inertia weight of 2.0, and linearly decrease it to 0.5 as a function of the iteration number.
10. The GCPSO was developed to address a specific problem with the standard PSO. What is this problem? If mutation is combined with the PSO, will this problem be addressed?
11. Consider the following adaptation of the standard *gbest* PSO algorithm: all particles, except for the *gbest* particle, use the standard PSO velocity update and position update equations. The new position of the *gbest* particle is, however, determined by using the LeapFrog algorithm. Comment on this strategy. What advantages do you see, any disadvantages? Will it solve the problem of the standard PSO in the question above?
12. Explain why the basic *gbest* PSO cannot be used to find niches (multiple solutions), neither in parallel nor sequentially (assuming that the fitness function is not allowed to be augmented).
13. Explain why velocities should be initialized to zero for the NichePSO.
14. Can it be said that PSO implements a form of
    (a) competitive coevolution?
    (b) cooperative coevolution?

    Justify your answers.
15. Discuss the validity of the following statement: "*PSO is an EA.*"