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

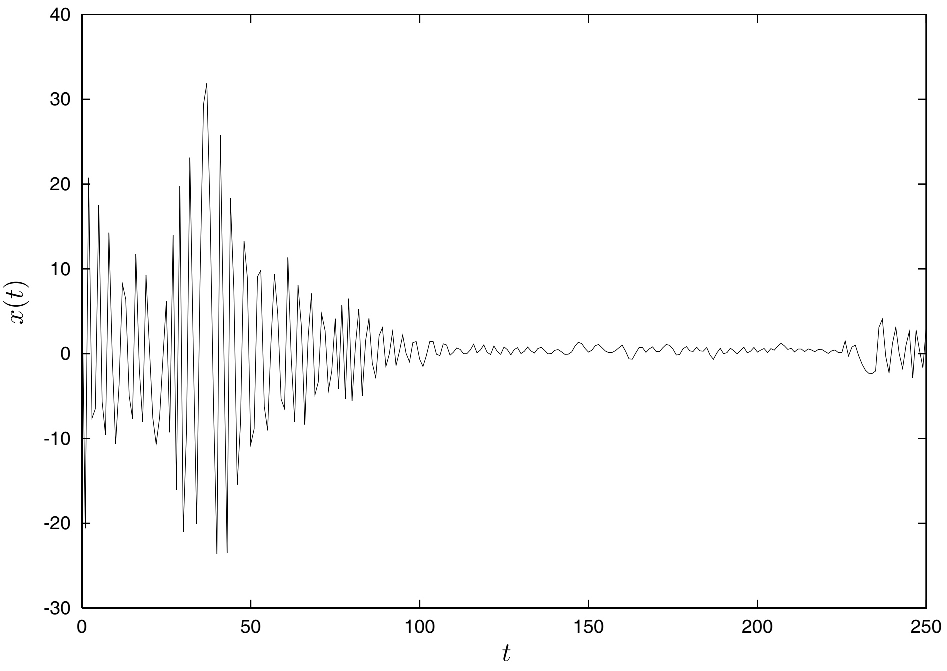

Figure 16.6 Stochastic Particle Trajectory for $w = 0.9$ and $c_1 = c_2 = 2.0$

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 $f(\hat{\mathbf{y}}(t)) \le f(\hat{\mathbf{y}}(t+1))$; $\rho(0) = 1.0$ 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,

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

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

with the corresponding velocities,

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

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

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

$$\begin{aligned} v_{ij}(t+1) &= wv_{ij}(t) + c_1r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + c_2r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t)) \\ &\quad + c_3r_{3j}(t)D(d) \end{aligned} \tag{16.84}$$

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 $C_i = 0, \forall i = 1, \ldots, n_s$;
**repeat**
    Evaluate fitness of all particles;
    Update velocities;
    Update positions;
    Calculate criticality for all particles;
    Reduce criticality for each particle;
    **while** $\exists i = 1, \ldots, n_s$ *such that* $C_i > C$ **do**
        Disperse criticality of particle $i$;
        Reinitialize $\mathbf{x}_i$;
    **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,

$$v_{ij}(t+1) = wv_{ij}(t) + c_1r_1(t)[y_{ij}(t) - x_{ij}(t)] + c_2r_2(t)[\hat{y}_j(t) - x_{ij}(t)] + a_{ij}(t) \quad (16.86)$$

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,

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

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;
```