Chapter 3

# Particle swarm optimization

## 3.1 Biological background

The tendency to form swarms appears in many different organisms, for instance (some species of) birds and fish. Swarming behavior offers several advantages, for example protection from predators: An animal near the centre of a swarm is unlikely to be captured by a predator. Furthermore, the members of a swarm may confuse predators through coordinated movements, such as rapid division into subgroups. Furthermore, because of the large concentration of individuals in a swarm, the risk of injury to a predator, should it attack, can be much larger than when it is attacking a single animal. On the other hand, it is not entirely evident that swarming is always beneficial: Once a predator has discovered a swarm, at least it knows where the prey is located. However, by aggregating, the prey effectively presents the predator with a needle-in-haystack problem and the benefits of doing so are apparently significant, since swarming behavior is so prevalent in nature. Swarms are formed also for other reasons than protection from predators. For example, swarming plays a role in efficient repeoduction: It is easier to find a mate in a large group. However, also in this case, there are both benefits and drawbacks, since competition obviously increases as well.

Swarming is also a prerequisite for cooperation that, in turn, plays a crucial role in, for example, the foraging (food gathering) of several species, particularly ants, bees, and termites, but also in other organisms such as birds. Here, the principle is that many eyes are more likely to find food than a single pair of eyes.

The search efficiency provided by swarming is what underlies particle swarm optimization (PSO) algorithms. Before introducing PSO in detail, however, we shall briefly consider a model for swarming in biological organisms.

### 3.1.1 A model of swarming

In many instances of swarming in animals, there is no apparent leader that the other members of the swarm follow. Thus, the swarm must be a result of *local* interactions only, and an interesting question follows: how can a coherent swarm result from such interactions? This issue has been addressed by Reynolds [3], who introduced a model for numerical simulation of the swarming of bird-like objects (or **boids** as they were called), which we will consider next.

In the description of PSO below, the $i^{\text{th}}$ member of a swarm, referred to as a *particle*, will be denoted $p_i$. Thus, in order to keep a unified notation throughout the chapter, we shall use the symbol $p_i$ also for the boids considered in this section. Let S, defined as,

$$\mathbf{S} = \{p_i,\ i = 1, \ldots, N\}. \tag{3.1}$$

denote a swarm of $N$ boids. In this model, the boids can only perceive nearby swarm mates. Thus, for each boid $i$, a **visibility sphere** $\mathbf{V}_i$ is introduced, defined as

$$\mathbf{V}_i = \{p_j \colon \|\mathbf{x}_j - \mathbf{x}_i\| < r,\ j \neq i\}, \tag{3.2}$$

where $r$ is a global constant, i.e. a property of the swarm. In each time step of the simulation, the positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$ of the boids are updated using standard Euler integration, i.e.

$$\mathbf{x}_i' = \mathbf{x}_i + \mathbf{v}_i \Delta t,\ i = 1, \ldots, N, \tag{3.3}$$

$$\mathbf{v}_i' = \mathbf{v}_i + \mathbf{a}_i \Delta t,\ i = 1, \ldots, N, \tag{3.4}$$

where $\mathbf{v}_i$ denotes the velocity of boid $i$, $\mathbf{a}_i$ its acceleration, and $\Delta t$ is the time step. Each boid is influenced by three different movement tendencies (or steers) that together determine its acceleration, namely **cohesion**, **alignment**, and **separation**. Cohesion represents the tendency of any given boid to stay near the centre of the swarm. Let $\rho_i$ denote the centre of density of the boids within the visibility sphere of boid $i$, i.e.

$$\rho_i = \sum_{p_j \in V_i} \frac{\mathbf{x}_j}{k_i}, \tag{3.5}$$

where $k_i$ is the number of boids in $V_i$. The steering vector representing cohesion is defined as

$$\mathbf{c}_i = \frac{1}{T^2}\left(\rho_i - \mathbf{x}_i\right), \tag{3.6}$$

where $T$ is a time constant, introduced in order to give $\mathbf{c}_i$ the correct dimension (acceleration). If $k_i = 0$, i.e. if no boids are within the visibility sphere of boid $i$, then $\mathbf{c}_i = \mathbf{0}$. Alignment, by contrast, is the tendency of boids to align

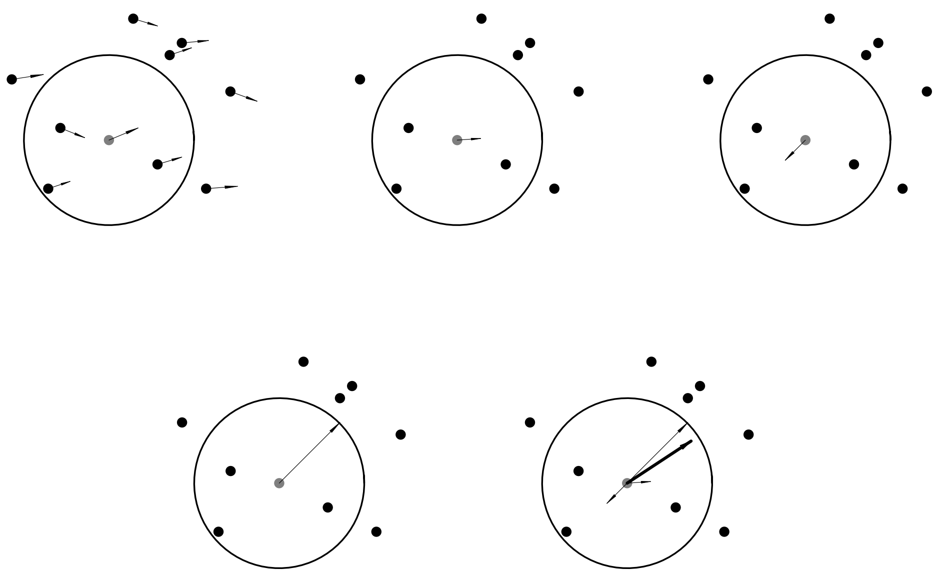

**Figure 3.1:** *A two-dimensional example of steering vectors: The upper left panel shows the positions and velocities of a swarm of boids. The circle indicates the visibility sphere of the gray particle located at its centre. The steering components* $\mathbf{c}$*,* $\mathbf{l}$*, and* $\mathbf{s}$ *are shown as arrows in the middle, right and lower left panels, respectively. The lower right panel shows the resulting acceleration vector* $\mathbf{a}$*.*

their velocities with those of their nearby swarm mates. Thus, the alignment steering vector is defined as

$$\mathbf{l}_i = \frac{1}{T} \sum_{p_j \in V_i} \frac{\mathbf{v}_j}{k_i}. \tag{3.7}$$

As in the case of cohesion, if $k_i = 0$, then $\mathbf{l}_i = \mathbf{0}$. Separation, finally, is needed in order to avoid collision with nearby boids, and the corresponding steering vector is obtained as

$$\mathbf{s}_i = \frac{1}{T^2} \sum_{p_j \in V_i} (\mathbf{x}_i - \mathbf{x}_j). \tag{3.8}$$

If $k_i = 0$, then $\mathbf{s}_i = \mathbf{0}$. Finally, combining the steering vectors, the acceleration of boid $i$ is obtained as

$$\mathbf{a}_i = C_c \mathbf{c}_i + C_l \mathbf{l}_i + C_s \mathbf{s}_i, \tag{3.9}$$

where the constants $C_c, C_l$ and $C_s$ ($\in [0,1]$) measure the relative impact of the three steering vectors. An example of steering vectors in a boids simulation is shown in Fig. 3.1.

A crucial factor in simulations based on this model is the initialization; If the initial speed of the boids is too large, the swarm will break apart. Thus, commonly, swarms are instead initialized with $\mathbf{v}_i = 0,\ i = 1,\ldots,N$. Furthermore, each boid should (at initialization) be within the visibility sphere of at least one other boid.

The simple model presented above leads, in fact, to very realistic swarm behavior, and the algorithm has (with some modifications) been used both in computer games and in movies (e.g. Jurassic Park).

## 3.2 Algorithm

Particle swarm optimization (PSO) [1] is based on the properties of swarms. As is the case with EAs, PSO algorithms (of which there are several versions, as we shall see) attempts to capture those aspects of swarming that are important in optimization, namely the search efficiency attributable to a swarm. Essentially, in PSO, each particle [1] is associated both with a position and a velocity in the search space, as well as a method for determining the changes in velocity depending on the performance of the particle itself and that of other particles. Thus, a clear difference, compared to EAs, is the introduction of a velocity in the search space. A basic PSO algorithm is described in Algorithm 3.1. The first step is initialization of the positions $\mathbf{x}_i$ and the velocities $\mathbf{v}_i$ of each particle $p_i$, $i = 1,\ldots,N$. The appropriate number of particles will vary from problem to problem, but is typically smaller than the number of individuals used in EAs. Common values of $N$ are $20 - 40$. Positions are normally initialized randomly, using uniform sampling in a given range $[x_{\text{min}}, x_{\text{max}}]$, i.e.

$$x_{i,j} = x_{\text{min}} + r\left(x_{\text{max}} - x_{\text{min}}\right),\ i = 1,\ldots,N,\ j = 1,\ldots,n \tag{3.10}$$

where $x_{i,j}$ denotes the $j^{\text{th}}$ component of the position of particle $p_i$ and $r$ is a uniform random number in the range $[0,1]$. $N$ denotes the size of the swarm, corresponding to the population size in an EA, and $n$ is the dimensionality of the problem (the number of variables). Velocities are normally also initialized randomly, according to

$$v_{i,j} = \alpha\frac{x_{\text{min}} + r\left(x_{\text{max}} - x_{\text{min}}\right)}{\Delta t},\ i = 1,\ldots,N,\ j = 1,\ldots,n \tag{3.11}$$

where $v_{i,j}$ denotes the $j^{\text{th}}$ component of the velocity of particle $p_i$. $\alpha$ is a constant in the range $]0,1]$, and $\Delta t$ is the time step length which, for simplicity, commonly is set to 1. Restricting velocities so as to avoid uncontrollable divergence of the swarm is a crucial part of PSOs, which will be further discussed later in this chapter.

[1]As mentioned earlier in the chapter, in PSO the candidate solutions, corresponding to individuals in EAs, are referred to as *particles*.

```
1. Initialize positions and velocities of the particles $p_i$:
    1.1. $x_{i,j} = x_{\min} + r\left(x_{\max} - x_{\min}\right),\ i = 1, \ldots, N,\ j = 1, \ldots, n$
    1.2. $v_{i,j} = \alpha \frac{x_{\min}+r(x_{\max}-x_{\min})}{\Delta t},\ i = 1, \ldots, N,\ j = 1, \ldots, n$
2. Evaluate each particle in the swarm: $\mathbf{x}_i \to f(\mathbf{x}_i),\ i = 1, \ldots, N.$
3. Update the best position of each particle, and the global best position.
   Thus, for all particles $p_i$ $i = 1, \ldots, N$:
    3.1. if $f(\mathbf{x}_i) < f(\mathbf{x}_i^{\mathrm{pb}})$ then $\mathbf{x}_i^{\mathrm{pb}} \leftarrow \mathbf{x}_i$.
    3.2. if $f(\mathbf{x}_i) < f(\mathbf{x}^{\mathrm{sb}})$ then $\mathbf{x}^{\mathrm{sb}} \leftarrow \mathbf{x}_i$.
4. Update particle velocities and positions:
    4.1. $v_{i,j} \leftarrow v_{i,j} + c_1 q \left(\frac{x_{i,j}^{\mathrm{pb}} - x_{i,j}}{\Delta t}\right) + c_2 r \left(\frac{x_j^{sb} - x_{i,j}}{\Delta t}\right),\ i = 1, \ldots, N,\ j = 1, \ldots, n$
    4.2. $x_{i,j} \leftarrow x_{i,j} + v_{i,j}\Delta t,\ i = 1, \ldots, N,\ j = 1, \ldots, n.$
5. Return to step 2, unless the termination criterion has been reached.
```

**Algorithm 3.1:** *Basic particle swarm optimization. $N$ denotes the number of particles in the swarm, and $n$ denotes the dimensionality (number of variables) of the problem under study. It has been assumed that the goal is to minimize the objective function $f(\mathbf{x})$. See the main text for a complete description of the algorithm.*

Once initialization has been completed, the next step is to evaluate the performance of each particle. As in an EA, the detailed nature of the evaluation depends, of course, on the problem at hand. Also, the sign of the inequalities in Algorithm 3.1 depends on whether the goal is to maximize or minimize the value of the objective function (here, minimization has been assumed). In the simple case of function minimization, the function value $f(\mathbf{x})$ can serve as the performance measure.

Next, the velocities and positions of all particles should be updated. As the aim is to reach optimal values of the objective function, the procedure for determining velocities should, of course, keep track of the performance of the particles thus far. In fact, two such measures are stored and used in PSO, namely (1) the best position $\mathbf{x}_i^{\mathrm{pb}}$ so far, of particle $i$, and (2) the best performance $\mathbf{x}^{\mathrm{sb}}$ so far, of *any* particle in the swarm. Thus, after the evaluation of a particle $p_i$, the two performance tests described in step 3 in Algorithm 3.1 are carried out. The first test is straightforward and simply consists of comparing the performance of particle $p_i$ with its previous best performance. The second test, however, can be carried out in different ways, depending on whether the *best performance of any particle in the swarm* is taken to refer to the *current* swarm or *all* particles considered thus far, and also depending on whether the comparison includes *all* particles of the swarm or only particles in a neighbourhood (a concept that will be further discussed below) of particle $p_i$. In Algorithm 3.1 it is assumed that the comparison in Step 3.2 involves the whole swarm, and that the best-ever position is used as the benchmark. Thus, in this case, after the first evaluation of all particles, $\mathbf{x}^{\mathrm{sb}}$ is set to the best position thus found. $\mathbf{x}^{\mathrm{sb}}$ is then stored, and is updated only when the condition in Step 3.2 of the algorithm is fulfilled.

Given the current values of $\mathbf{x}_i^{\mathrm{pb}}$ and $\mathbf{x}^{\mathrm{sb}}$, the velocity of particle $p_i$ is then updated according to

$$v_{i,j} \leftarrow v_{i,j} + c_1 q \left( \frac{x_{i,j}^{\mathrm{pb}} - x_{i,j}}{\Delta t} \right) + c_2 r \left( \frac{x_j^{\mathrm{sb}} - x_{i,j}}{\Delta t} \right), \; j = 1, \ldots, n, \tag{3.12}$$

where $q$ and $r$ are uniform random numbers in $[0, 1]$, and $c_1$ and $c_2$ are constants, typically both set to 2, so that the mean of the two factors $c_1 q$ and $c_2 r$ is equal to 1. The factor proportional to $c_1$ is sometimes referred to as the **cognitive component** and the factor proportional to $c_2$ the **social component**. The cognitive component measures the degree of self-confidence of a particle, i.e. the degree to which it trusts its own previous performance as a guide towards obtaining better results. Similarly, the social component measures a particle's trust in the ability of the other swarm members to find better solutions. Next, the position of particle $p_i$ is updated as

$$x_{i,j} \leftarrow x_{i,j} + v_{i,j} \Delta t, \; j = 1, \ldots, n. \tag{3.13}$$

This completes the first **iteration**. Steps 2, 3, and 4 of Algorithm 3.1 are then repeated until a satisfactory solution has been found.

## 3.3 Properties of PSO

As in the case of EAs, there exists many variations on the theme provided by the basic PSO described in Algorithm 3.1, some of which will now be described.

### 3.3.1 Best-in-swarm vs. best-ever

A crucial component in PSO is the concept of the best-of-swarm performance, i.e. $\mathbf{x}^{\mathrm{sb}}$ as introduced in Algorithm 3.1 above. The first modification we will consider concerns the scope of the comparison with respect to the iterations carried out during optimization. In Algorithm 3.1, we determined $\mathbf{x}^{\mathrm{sb}}$ as the

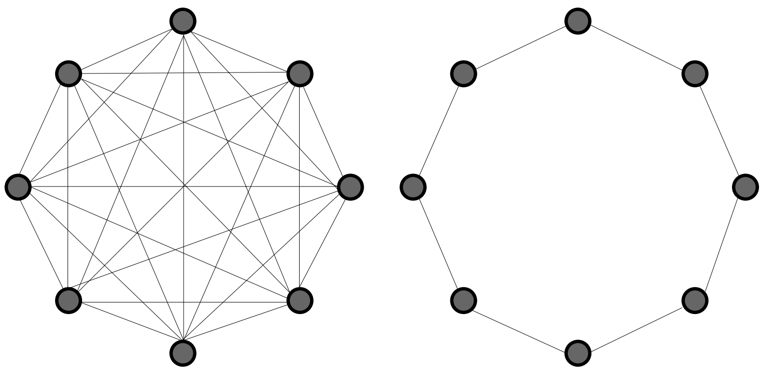

**Figure 3.2:** *Particle neighbourhoods in PSO, shown for the case $N = 8$. Left panel: a fully connected neighbourhood. Right panel: a neighbourhood with restricted connectivity.*

best-ever position $\mathbf{x}^{\mathrm{sb,e}}$ of any particle of the swarm. An alternative approach is to consider only the best position $\mathbf{x}^{\mathrm{sb,c}}$ in the *current* swarm, i.e. among the $N$ particles forming the current iteration. Note that, apart from the determination of the best-in-swarm position, all other steps are identical to those of Algorithm 3.1. In terms of computer programming, the only difference between the two methods is the single line of code needed to reset the best-of-swarm in each iteration.

### 3.3.2 Neighbourhood topologies

In the boids model presented at the beginning of this chapter, a visibility sphere was associated with each boid, and only those boids that happened to be inside this sphere influenced the acceleration of the boid under consideration. A similar idea (albeit with an important difference, see below) has been introduced in connection with PSO, namely the concept of **neighbourhoods**..

Let $\mathbf{x}_i^{\mathrm{sb,\,n}}$ denote the best particle (i.e. the one associated with the lowest value of the objective function), among the neighbours of particle $i$. In Algorithm 3.1, the neighbourhood included all particles in the swarm, so that $\mathbf{x}_i^{\mathrm{sb,\,n}} = \mathbf{x}^{\mathrm{sb}}$ (independent of $i$). Such a neighbourhood is shown in the left panel of Fig. 3.2, where the discs represent the particles and the lines emanating from any disc determine the neighbours of the particle in question.

However, there are many alternatives to the fully connected neighbourhood. Another example is shown in the right panel of the figure, in which each particle is only connected to its nearest neighbours on either side. Obviously, intermediate cases can be defined as well, in which each particle is connected to the $K$ nearest neighbours on each side.

Note that, unlike the visibility spheres introduced in connection with boids (see Subsect. 3.1.1), the topological constructs shown in Fig. 3.2 are defined in an abstract space different from the $n$-dimensional search space; two neigh-

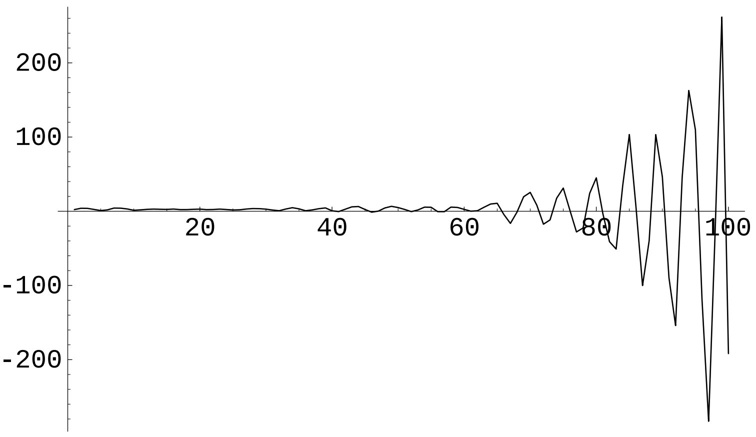

**Figure 3.3:** *Typical trajectory of a single particle, integrated (in one dimension) over 100 time steps, using Eq. (3.12), with $c_1 = c_2 = 1.5$, $x^{\text{pb}} = 2$, $x^{\text{sb}} = 2.5$, $\Delta t = 1$. The integration was started at $x = 1, v = 0$.*

bours linked together as neighbours may be located at very different places in the search space. Furthermore, the neighbourhood structure normally remains fixed throughout optimization whereas, of course, the positions $\mathbf{x}_i$ in the *search* space vary with every iteration, according to Eqs. (3.12) and (3.13).

The definition of neighbourhood structures with restricted connectivity in PSO serves the same purpose as the various procedures introduced for the prevention of premature convergence in EAs.

### 3.3.3 Maintaining coherence

The choice of the parameters $c_1$ and $c_2$ has a strong influence on the trajectories of particles in the swarm. As noted above, a common choice is to take $c_1 = c_2 = 2$, in which case equal weight is given to the cognitive and social parts of the velocity equation (Eq. (3.12)). Other choices are possible as well. However, the sum of $c_1$ and $c_2$ should be upper bounded by 4, i.e.

$$c_1 + c_2 \leq 4. \tag{3.14}$$

One can prove (after removing the stochastic components, i.e the random numbers $q$ and $r$ in Eq. (3.12)) that the trajectories will remain bounded only if $c_1 + c_2 \leq 4$.

However, even if $c_1 + c_2 \leq 4$, the trajectories of particles moving under the influence of Algorithm 3.1 will, in fact, diverge eventually, due to the influence of stochastic variables $q$ and $r$. An example is shown in Fig. 3.3, in which the position of a single particle is integrated with $c_1 = c_2 = 1.5$. As is evident from the figure, even though $c_1 + c_2 < 4$, the trajectory eventually diverges.

Thus, in practice, the divergence of particle trajectories must somehow be controlled. The simplest way of doing so is to introduce a limit on particle velocities. Typically, velocities are restricted such that

$$|v_{i,j}| < v_{\max} = \frac{(x_{\max} - x_{\min})}{\Delta t}, \; j = 1, \ldots, n. \tag{3.15}$$

Thus if, after an update using Eq. (3.12), $v_{i,j} > v_{\max}$, then $v_{i,j}$ is simply set equal to $v_{\max}$. Similarly if $v_{i,j} < -v_{\max}$, $v_{i,j}$ is set equal to $-v_{\max}$.

This, however, is not the only way to maintain swarm coherence. In fact, in the literature on PSO, there exists several studies concerning the use of **constriction coefficients** that modify the velocity equation (Eq. (3.12)), the position equation (Eq. (3.13)), or both equations. As an example, consider the modified rule

$$v_{i,j} \leftarrow \chi \left( v_{i,j} + c_1 q \left( \frac{x_{i,j}^{\text{pb}} - x_{i,j}}{\Delta t} \right) + c_2 r \left( \frac{x_j^{\text{sb}} - x_{i,j}}{\Delta t} \right) \right), \; j = 1, \ldots, n. \tag{3.16}$$

It can be shown that the trajectories obtained using Eq. (3.16) do not diverge if $\chi$ is taken as

$$\chi = \frac{2\xi}{|2 - \xi + \sqrt{\xi^2 - 4\xi}|}, \tag{3.17}$$

where (note!) $\xi \equiv c_1 + c_2 > 4$.

## 3.3.4 Inertia weight

As a further modification of Algorithm 3.1 one may introduce a parameter that determines the relative influence of previous velocities on the current velocity of a particle. Consider the modified velocity equation

$$v_{i,j} \leftarrow w v_{i,j} + c_1 q \left( \frac{x_{i,j}^{\text{pb}} - x_{i,j}}{\Delta t} \right) + c_2 r \left( \frac{x_j^{\text{sb}} - x_{i,j}}{\Delta t} \right), \; j = 1, \ldots, n. \tag{3.18}$$

Here, $w$ is referred to as the **inertia weight**. If $w > 1$, the particle favors exploration over exploitation, i.e. it assigns relatively less significance to the cognitive and social components than if $w < 1$, in which case the particle is more attracted towards the current best positions. As in the case of EAs, exploration plays a more important role than exploitation in the early stages of optimization, and vice versa towards the end. Thus, a common strategy is to start with a value larger than 1 ($w = 1.4$, say), and then reduce $w$ by a constant factor $\beta \in ]0, 1[$ (typically very close to 1) in each iteration, until $w$ reaches a lower bound (typically around 0.3-0.4). In fact, the use of an inertia weight is so common to warrant its inclusion in a *standard PSO algorithm*. Thus, here, we define the standard PSO algorithm as Algorithm 3.1, but with step 4.1 defined by Eq. (3.18). Note that the standard algorithm, shown in Algorithm 3.2, uses the best-so-far position in the entire swarm (i.e. with full connectivity) in the determination of the social component of the velocity change, as well as a maximum velocity (rather than a constriction coefficient).

```
1. Initialize positions and velocities of the particles $p_i$:
    1.1. $x_{i,j} = x_{\min} + r\left(x_{\max} - x_{\min}\right)$, (random), $i = 1, \ldots, N,\ j = 1, \ldots, n$
    1.2. $v_{i,j} = \alpha \frac{x_{\min} + r(x_{\max} - x_{\min})}{\Delta t},\ i = 1, \ldots, N,\ j = 1, \ldots, n$
2. Evaluate each particle in the swarm: $\mathbf{x}_i \to f(\mathbf{x}_i),\ i = 1, \ldots, N$.
3. Update the best position of each particle, and the global best position.
   Thus, for all particles $p_i\ i = 1, \ldots, N$:
    3.1. if $f(\mathbf{x}_i) < f(\mathbf{x}_i^{\mathrm{pb}})$ then $\mathbf{x}_i^{\mathrm{pb}} \leftarrow \mathbf{x}_i$.
    3.2. if $f(\mathbf{x}_i) < f(\mathbf{x}^{\mathrm{sb}})$ then $\mathbf{x}^{\mathrm{sb}} \leftarrow \mathbf{x}_i$.
4. Update particle velocities and positions:
    4.1. $v_{i,j} \leftarrow w v_{i,j} + c_1 q \left(\frac{x_{i,j}^{\mathrm{pb}} - x_{i,j}}{\Delta t}\right) + c_2 r \left(\frac{x_j^{\mathrm{sb}} - x_{i,j}}{\Delta t}\right),\ i = 1, \ldots, N,\ j = 1, \ldots, n$
    4.2. Restrict velocities, such that $|v_{i,j}| < v_{\max}$.
    4.3. $x_{i,j} \leftarrow x_{i,j} + v_{i,j}\Delta t,\ i = 1, \ldots, N,\ j = 1, \ldots, n$.
5. Return to step 2, unless the termination criterion has been reached.
```

**Algorithm 3.2:** *A standard particle swarm optimization algorithm. Note the introduction of the inertia weight, and the explicit restriction of particle velocities. As in Algorithm 3.1, it has been assumed that the best-ever position is used in step 3.2, and that the neighbourhood of any particle includes all other particles.*

### 3.3.5 Elite particle

Another slight modification is the introduction of an **elite particle**, that is, a particle that is placed exactly at the best position. Thus, if $p_1$ is chosen as the elite particle, one would have

$$\mathbf{x}_1 = \mathbf{x}^{\mathrm{sb}}. \tag{3.19}$$

Thus, the position of the elite particle is only updated if a new best-so-far position is encountered (by one of the other particles).

### 3.3.6 Craziness operator

The final PSO component that will be considered here is the so called **craziness operator**. This operator, which is typically applied with a given probability $p_{\mathrm{cr}}$, sets the velocity of the particle $p_i$ in question to a uniform random value within the allowed range. Thus, if craziness is applied, the velocity of the particle changes according to

$$v_{i,j} = -v_{\max} + 2rv_{\max}, \tag{3.20}$$

where $r$ is a random number in $[0, 1]$. The craziness operator, which in some way serves the same function as mutations in an EA, can be said to have a biological motivation: In flocks of birds one can observe that, from time to time, one bird suddently shoots off in a seemingly random direction (only to re-enter the swarm shortly thereafter).

## 3.4 Discrete versions

In the PSO algorithms presented above, it is assumed that the variables $x_j$ take values in a subset of $\mathbf{R}^n$. However, with only slight modifications, PSO can also be used in connection with integer programming problems, where the variables take integer values. Here, two discrete PSO algorithms will be considered. The first method considered, which is based on variable truncation, can be applied to any integer programming problem, whereas the second method (binary PSO) is applicable to problems in which the variables take binary values, i.e. $x_j \in \{0, 1\}$.

### 3.4.1 Variable truncation

The variable truncation PSO algorithm is very straightforward: At initialization, random positions are generated as usual, but are then truncated (component by component) to their nearest integer values. The determination of new velocities and positions is then carried out exactly as in the continuous version. Once new positions have been obtained, each component of the position vector is truncated to the nearest integer value, and the objective function is computed. Moreover, since the internal workings of the algorithm are identical to the standard (continuous) PSO, the discussion above concerning specialized operators and parameter settings is valid in the discrete case as well.

### 3.4.2 Binary PSO

In certain cases of integer programming, it is necessary to restrict the particle positions to a given subset of $\mathbf{Z}^n$. One such case is binary programming, where the components $x_j$ of the particle positions are restricted to $\{0, 1\}$. Note that a restriction on velocities does *not* imply a similar restriction on particle positions. Thus, a dedicated restriction mechanism for particle positions is needed. One such mechanism, introduced by Kennedy and Eberhart [2] is shown in Algorithm 3.3. Here, the variables $v_{i,j}$ are updated as usual, but

```
1. Initialize positions and velocities of the particles $p_i$:
    1.1. $x_{i,j} \in \{0,1\}$, (random), $i = 1,\ldots,N,\ j = 1,\ldots,n$
    1.2. $v_{i,j} = \alpha \frac{x_{\min} + r(x_{\max} - x_{\min})}{\Delta t},\ i = 1,\ldots,N,\ j = 1,\ldots,n$
2. Evaluate each particle in the swarm: $\mathbf{x}_i \to f(\mathbf{x}_i),\ i = 1,\ldots,N$.
3. Update the best position of each particle, and the global best position.
   Thus, for all particles $p_i$ $i = 1,\ldots,N$:
    3.1. if $f(\mathbf{x}_i) < f(\mathbf{x}_i^{\mathrm{pb}})$ then $\mathbf{x}_i^{\mathrm{pb}} \leftarrow \mathbf{x}_i$.
    3.2. if $f(\mathbf{x}_i) < f(\mathbf{x}^{\mathrm{sb}})$ then $\mathbf{x}^{\mathrm{sb}} \leftarrow \mathbf{x}_i$.
4. Update particle velocities and positions:
    4.1. $v_{i,j} \leftarrow wv_{i,j} + c_1 q \left(\frac{x_{i,j}^{\mathrm{pb}} - x_{i,j}}{\Delta t}\right) + c_2 r \left(\frac{x_j^{\mathrm{sb}} - x_{i,j}}{\Delta t}\right),\ i = 1,\ldots,N,\ j = 1,\ldots,n$
    4.2. Restrict velocities, such that $|v_{i,j}| < v_{\max} (\approx 4)$.
    4.3. Compute $\sigma(v_{i,j}) = \frac{1}{1+\mathrm{e}^{-v_{i,j}}},\ i = 1,\ldots,N,\ j = 1,\ldots,n$
    4.4. Generate a uniform random number $r \in [0,1]$ and update $x_{i,j}$ as
         $x_{i,j} \leftarrow \begin{cases} 0 & \text{if } r > \sigma(v_{i,j}) \\ 1 & \text{otherwise} \end{cases}$
5. Return to step 2, unless the termination criterion has been reached.
```

**Algorithm 3.3:** *Binary particle swarm optimization. The main difference compared to Algorithm 3.1 is the interpretation of the velocities. As in Algorithm 3.1, it has been assumed that the best-ever position is used in step 3.2, and that the neighbourhood of any particle includes all other particles.*

their interpretation is different: instead of directly modifying positions as in Eq. (3.13), the $v_{i,j}$ are passed through a squashing function $\sigma$, given by

$$\sigma(v_{i,j}) = \frac{1}{1 + \mathrm{e}^{-v_{i,j}}}, \tag{3.21}$$

to generate the probability of setting $x_{i,j}$ equal to 1. Thus, with this modification, the positions are restricted to the desired set $\{0,1\}$. As for the variable truncation PSO algorithm, the discussion concerning special operators and parameter settings in Sect. 3.3 is still mostly valid. One exception, however, is the truncation of velocities. In binary PSO, explicit truncation of velocities is used (rather than a constriction coefficient), such that the $v_{i,j}$ are typically restricted to the range $|v_{i,j}| < v_{\max}$, where $v_{\max} \approx 4$, in order to avoid situations in which the probability of setting $x_{i,j}$ to any particular value (0 or 1) becomes too high. With $|v_{i,j}|$ restricted to 4 or less, there is always a probability of at least $\sigma(v_{\max}) = 0.018$ of modifying the value of any position component $x_{i,j}$ for a particle $p_i$.

Some problems, such as e.g. decision-making problems involving a sequence of yes-or-no decisions fall naturally into the category of binary programming, where binary PSO is directly applicable. Moreover, continuous optimization problems can also be solved using binary PSO, simply by employing a binary encoding scheme for the continuous variables, much as in a standard GA (see Chapter 2, p. 48).

# Bibliography

[1] Kennedy, J. and Eberhart, R.C. *Particle swarm optimization*, In: Proc. of the IEEE International Conference on Neural Networks, Man, and Cybernetics, pp. 1942-1948, 1995

[2] Kennedy, J. and Eberhart, R.C. *A discrete binary version of the particle swarm algorithm*, In: Proc. of the 1997 Conference on Systems, Man, and Cybernetics, pp. 4104-4109, 1997

[3] Reynolds, C.W. *Flocks, herds, and schools: A distributed behavioral model*, Computer Graphics, **21**, pp. 25-34, 1987