1

Metaheuristics for Controller Optimization

1.1. Introduction

Stochastic optimization using metaheuristics is well adapted for solving problems for which it is difficult to find a global optimum or good local optima using classical methods. This type of optimization has three characteristics that are often decisive in global optimization:

– optimization using metaheuristics does not require us to know the gradient of the function to be minimized, the only constraint being that we must be able to evaluate the latter, which can therefore have any form;

– it is not necessary to use a "good" initial point, the initialization being carried out at random in the search space;

– finally, this type of optimization is stochastic, which makes it possible to overcome the combinatorial explosion of possibilities and limits trapping in the local optima.

In [DRÉ 06], [YAN 10a] and [FIS 13], you will find descriptions of metaheuristic principles inspired by analogies with physics, ethology, biology, etc.

Despite the main drawback, which is the pertinence of the solution found and which can be evaluated statistically using several successive runs (which creates a generally high calculation time), this type of optimization proves useful for difficult optimization problems and is shown here, adapted for controller synthesis.

As for the calculation time, this can prove to be substantial as most of the algorithms proceed to evaluate the cost function for each individual in a population at each iteration. One way of reducing the calculation times is to parallelize them at each iteration with specialized numerical tools [MAT 10, LUS 09]; it is therefore possible to distribute the calculation charge over several processors or cores in parallel by allocating a group of individuals from the population to each processor.

In this chapter, we introduce the metaheuristics used in the rest of the book as they have given "good results" for all the automation problems we discuss. By "good result", we mean that the optimization has a good degree of reproducibility and a good rate of convergence, and provides a good optimum.

The set of these algorithms is classed into three large families:

– evolutionary approaches;

– swarm approaches;

– hybrid evolutionary/swarm approaches.

To introduce these algorithms, we consider the problem of finding X_{opt} so that $X_{opt} = \underset{X \in \Psi}{\operatorname{argmin}}\left(f\left(X\right)\right)$, where $\psi \subset \mathrm{R}^n$ is the search space for X and n is the problem's dimension. In the following, we set $\Psi = \left[X_{\min}; X_{\max}\right]$. The function *rnd*($x$, y) designs a random number uniformly generated in the interval [x, y].

1.2. Evolutionary approaches using differential evolution

1.2.1. *Standard version*

Differential evolution (DE) [STO 95] is a recent metaheuristic that belongs to the class of evolutionary algorithms (Figure 1.1), which are inspired by the theory of the evolution of species (Figure 1.2).

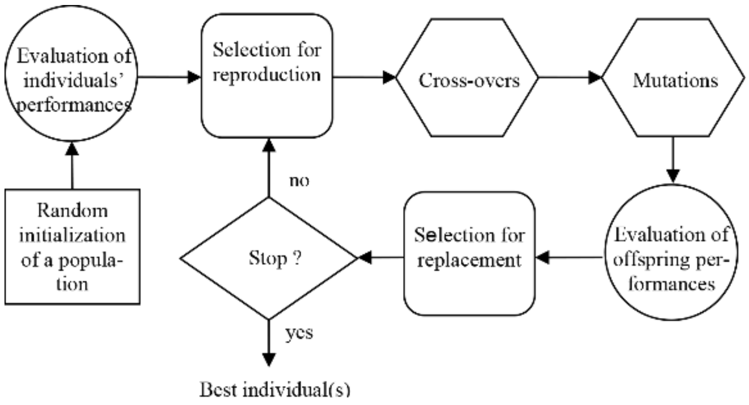
Evaluation of individuals' performances
Selection for reproduction
Cross-overs
Mutations
Random initialization of a population
no
Stop ?
Selection for replacement
Evaluation of offspring performances
yes
Best individual(s)

Figure 1.1. *Principle of evolutionary algorithms*

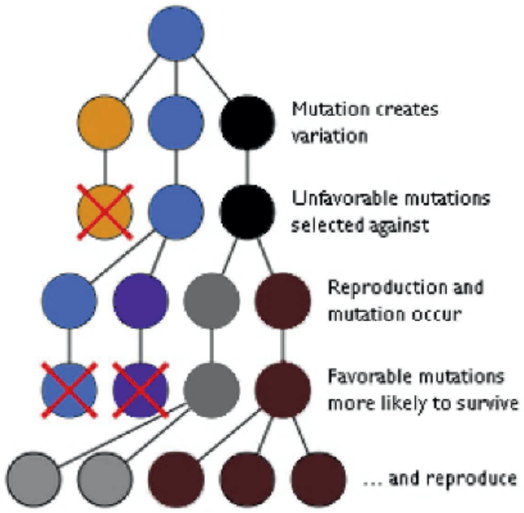
Mutation creates variation
Unfavorable mutations selected against
Reproduction and mutation occur
Favorable mutations more likely to survive
... and reproduce

Figure 1.2. *Darwin's theory. For a color version of this figure, see www.iste.co.uk/feyel/optimization.zip*

Below, we describe the classical version of the algorithm.

Initializing the algorithm

We build an initial population P from N individuals:

$$P = \{X_1, \ldots, X_i, \ldots, X_N\} \tag{1.1}$$

Each individual is defined by its n genes:

$$X_i = (x_{i1}, \ldots, x_{ij}, \ldots, x_{in}) \tag{1.2}$$

n is therefore the problem's dimension.

Assuming that $x_j \in [X_{\min j}, X_{\max j}]$, the jth gene of the ith individual is randomly chosen in its definition interval:

$$x_{ij}^{(0)} = X_{\min j} + rnd(0,1)(X_{\max j} - X_{\min j}),\ j = 1, \ldots, n,\ i = 1, \ldots, N \tag{1.3}$$

The k^{ith} iteration consists of successively carrying out the three stages described below.

Describing the algorithm

– Step 1: mutation

Usually, two mutation schemes are used: the *rand* one and the *best* one. They are described below.

$$\begin{aligned}&\text{DE/Rand/1} \rightarrow U_i^{(k+1)} = X_{a_i}{}^{(k)} + F\left(X_{b_i}{}^{(k)} - X_{c_i}{}^{(k)}\right)\\&\text{DE/Best/1} \rightarrow U_i^{(k+1)} = X_{best}{}^{(k)} + F\left(X_{a_i}{}^{(k)} - X_{b_i}{}^{(k)}\right)\end{aligned} \qquad [1.4]$$

where $X_{a_i}, X_{b_i}, X_{c_i}$ are different and randomly chosen in the population and X_{best} is the best individual obtained since the start of the optimization, and $F \in [0,2]$ is a number called the mutation factor. Therefore, at each iteration, a number *N* of mutants U_i are generated according to one of the previous schemes (chosen once for all at the start of the optimization).

The *rand* scheme promotes diversity, whereas the *best* scheme encourages convergence. As the problems that we face are high-dimensional, we will here prioritize exploration and therefore the *rand* scheme.

– Step 2: cross-over

By crossing $U_i^{(k+1)}$ with $X_i^{(k)}$, a new individual $V_i^{(k+1)}$ is generated with genes defined in the following rule:

$$v_{ij}{}^{(k+1)} = \begin{cases} u_{ij}{}^{(k+1)} \;\; if \;\; rnd(0,1)_{ij} \le C_r \;\; or \; if \;\; j = j_i \\ \qquad\qquad x_{ij}{}^{(k)} \;\; else \end{cases} \qquad [1.5]$$

where $C_r \in [0{,}1;0{,}9]$ is the cross-over rate and *n* is an integer number randomly chosen in $\{1, 2, ..., N\}$.

– Step 3: selection

The population is finally modified with the individuals defined by:

$$X_i^{(k+1)} = \begin{cases} V_i^{(k+1)} si \;\; f\left(V_i^{(k+1)}\right) \le f\left(X_i^{(k)}\right) \\ \qquad\qquad X_i^{(k)} \;\; ifnot \end{cases} \qquad [1.6]$$

If the stop criterion is satisfied, then return the best solution found so far, if not, return to stage 1.

The algorithm's pseudo-code

The algorithm's pseudo-code is given below:

```
Generate an initial population.
While the stop criterion is not satisfied, repeat:
    For $i = 1$ to $N$ individuals, make:
        Choose randomly 3 individuals $X_{a_i}, X_{b_i}, X_{c_i}$
        Mutation:
            Create a mutant $U_i$ from individual $X_i$ and/or $X_{a_i}, X_{b_i}, X_{c_i}$
    (depending on the mutation scheme chosen at the beginning)
        Cross-over:                                                    [1.7]
            Create the trial $V_i$ from the mutant $U_i$ and $X_i$
        Selection:
            If $f(U_i) \le f(X_i)$ then we replace $X_i \leftarrow U_i$
    End For
    Calculate $X_{best}$
End While
```

Algorithm stop criterion

Several stop criteria can be used. We will use:

– the maximum number of iterations, knowing that, at each iteration, the fitness function is evaluated P times;

– a maximum number of fitness evaluations;

– premature convergence of the algorithm~~,~~ detected when all the individuals tend to be identical, that is to say, when the following relation is satisfied:

$$\exists X_{best} : \frac{\max\limits_{i=1,\dots,P} \left\| X_i^{(k)} - X_{best} \right\|}{\left\| X_{max} - X_{min} \right\|} < \varepsilon \qquad [1.8]$$

In most cases we will choose $\varepsilon = 10^{-5.}$

Parameters settings

One of the algorithm's advantages is the low number of setting parameters. Here, we will use the classical settings of [STO 96]: $F = 0.75$ and $C_r = 0.8$.

Although a population size N between $5n$ and $10n$ is generally advised, the algorithm has provided the best results using the same idea as in (Q)PSO, i.e. a number of individuals equal to:

$$N = floor\left(10 + \sqrt{n}\right) \qquad [1.9]$$

Variants of the algorithm

In the suite, $X_{a_i}, X_{b_i}, X_{c_i}, X_{d_i}, X_{e_i}$ are different individuals and are randomly chosen in the population and X_{best} is the best individual in the population obtained since the beginning of the optimization.

There are many variants of DE [MEZ 08], which most of the time consist of defining particular mutation schemes. As examples we cite:

– DE/Best/2:

$$\text{DE/Best/2} \rightarrow U_i^{(k+1)} = X_{best}{}^{(k)} + F\left(X_{a_i}{}^{(k)} - X_{b_i}{}^{(k)} + X_{c_i}{}^{(k)} - X_{d_i}{}^{(k)}\right) \qquad [1.10]$$

– DE/Rand/2:

$$\text{DE/Rand/2} \rightarrow U_i^{(k+1)} = X_{a_i}{}^{(k)} + F\left(X_{b_i}{}^{(k)} - X_{c_i}{}^{(k)} + X_{d_i}{}^{(k)} - X_{e_i}{}^{(k)}\right) \qquad [1.11]$$

– DE/Current-to-Rand/1:

$$\text{DE/Current-to-rand/1} \rightarrow U_i^{(k+1)} = X_i^{(k)} + F\left(X_{a_i}^{(k)} - X_{b_i}^{(k)}\right) + F'\left(X_{c_i}^{(k)} - X_i^{(k)}\right) \quad [1.12]$$

– DE/Current-to-Best/1:

$$\text{DE/Current-to-best/1} \rightarrow U_i^{(k+1)} = X_i^{(k)} + F\left(X_{a_i}^{(k)} - X_{b_i}^{(k)}\right) + F'\left(X_{best}^{(k)} - X_i^{(k)}\right) \quad [1.13]$$

The *rand* scheme promotes diversity (and therefore exploration of the search space), whereas the *best* scheme encourages convergence but often leads to sub-optimal solutions. This is why, with the aim of benefitting from the advantages of these two basic schemes, we suggest using the mutation scheme [1.14] which we call DE/Rand-to-best/1:

$$\text{DE/Rand/2} \rightarrow U_i^{(k+1)} = X_{a_i}^{(k)} + F\left(X_{b_i}^{(k)} - X_{c_i}^{(k)} + X_{d_i}^{(k)} - X_{e_i}^{(k)}\right) \quad [1.14]$$

F′ is another mutation factor that we choose, equal to 1 – *F* with $F' = 0.25$ here. This represents a good balance between diversity and convergence.

As we will see further on, this mutation scheme proves to be very effective for solving control problems.

The versions retained

We will keep the two versions of DE which have proven to be the most effective in our study: *Rnd_DE* and *Rnd2Bst_DE*, and which correspond, respectively, to the mutation schemes DE/Rand/1 and DE/Rand-to-best/1.

1.2.2. *Perturbed version*

DE has proven very effective for solving numerous complex problems [PRI 05]. Compared with many other metaheuristics, DE is very simple to implement. Moreover, its low number of setting parameters makes the algorithm relatively simple to set to achieve good results. However, like other metaheuristics, DE can converge prematurely, particularly when the population is small. One simple idea therefore consists of increasing the size of the population but at the cost of greater calculation times. Thus, some studies have been designed to maintain the algorithm's diversity. For example, the adaptive variants of DE [BRE 06] are useful, but they often introduce several additional setting parameters. Other ideas involve hybridizing the DE with other metaheuristics such as cuckoo search (CS) [WAN 12], which we will describe further on: generally, algorithms that are fairly effective but much more complex to implement are obtained. More recently in [FAJ 12], a random perturbation mechanism applied to V_i, after the cross-over step has been considered, is envisaged. Linked to different selection strategies, this mechanism seems to be effective, but at the cost of a much more complex selection phase than in the original version of DE.

To limit the premature convergence of DE, we suggest completing the mutation phase of DE with a simple mechanism consisting of perturbing the mutants U_i according to the following rule:

$$U_i^{(k+1)} = U_i^{(k+1)} + \text{rnd}(-1,1) X_{f_i} \quad \textit{if} \;\; \text{rnd}(0,1) < p_m \qquad [1.15]$$

where p_m is the probability of perturbing the mutants' and X_{f_i} is randomly chosen in the population, different to X_i and distinct from all the elements used in the mutation scheme used to generate U_i. A typically suitable value for p_m is 0.025.

We will call this extension *Perturbed Differential Evolution* (PDE), which can be linked to all mutation schemas. The only difference between DE and PDE lies in a simple additional rule. In Appendix F, you will find a comparison of performances between DE and PDE.

In this study, we will, in particular, keep variant ***Rnd2Bst_PDE***, which consists of the combination of mutation scheme DE/Rand-to-best/1 with PDE and which has given especially interesting results.

1.3. Swarm approaches

1.3.1. *Particle swarm optimization algorithm*

Principle

The particle swarm optimization algorithm (Particle Swarm Optimization – PSO) [CLE 06, BRA 07, KEN 95] is a metaheuristic inspired by the social behavior of flocks of birds (Figure 1.3) or schools of fish (Figure 1.4).

Figure 1.3. *A flock of birds*

Figure 1.4. *A school of fish*

N_P particles move through the search space with their own speed. They are capable of recalling where they had their best performance. In the *LocalBest* version of PSO presented here, for a given particle, we define its neighbors (or friend particles) as a subset of particles with which it can communicate. Therefore, at each instant, each particle knows the characteristics of the best of its neighbors. The following expressions are used:

– $X_p^{(k)}$ (resp. $V_p^{(k)}$): position (resp. velocity) of the particle p at iteration k;

– $B_p^{(k)} = \arg\min\left(f\left(X_p^{(k-1)}\right), f\left(X_p^{(k)}\right)\right)$: best position found by the particle p until iteration k;

– $\Omega\left(X_p^{(k)}\right) \subset \{1, 2, ..., N_P\}$: set of particles defining the neighborhood of particle p;

– $G_p^{(k)} = \underset{X \in \left\{B_i^{(k)}, i \in \Omega\left(X_p^{(k)}\right)\right\}}{\arg\min} f(X)$: best position found by the neighborhood of particle p until iteration k.

Each particle moves in the search space according to the following transition rule:

$$\begin{aligned} V_p^{(k+1)} &= w.V_p^{(k)} + c_1 \otimes \left(B_p^{(k)} - X_p^{(k)}\right) + c_2 \otimes \left(G_p^{(k)} - X_p^{(k)}\right) \\ X_p^{(k+1)} &= X_p^{(k)} + V_p^{(k+1)} \end{aligned} \qquad [1.16]$$

where

– $\otimes$ is the element-wise product;

– w is the inertia factor;

– $c_1 = rnd(0, \bar{c}_1)$, $c_2 = rnd(0, \bar{c}_2)$ are the acceleration factors.

Therefore, each particle of the swarm moves depending on its own velocity and its best position yet obtained, as well as the position of the best of its neighbors (Figure 1.5).

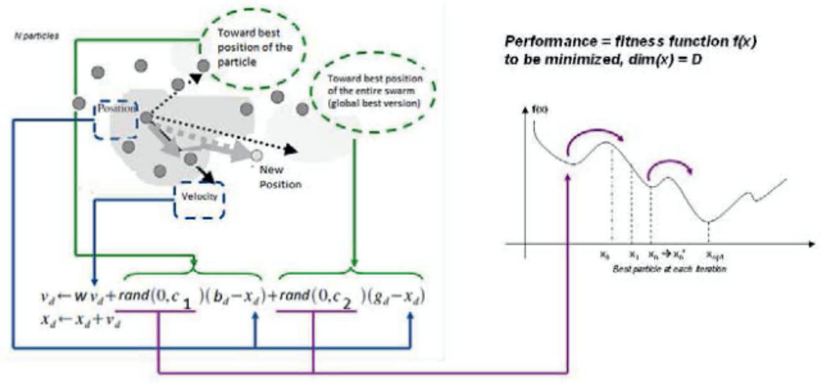
N particles
Toward best position of the particle
Toward best position of the entire swarm (global best version)
Position
Velocity
New Position
Performance = fitness function f(x) to be minimized, dim(x) = D
f(x)
x_0
x_1
$x_n \rightarrow x_n'$
x_{opt}
Best particle at each iteration
$v_d \leftarrow w\, v_d + rand(0, c_1)(b_d - x_d) + rand(0, c_2)(g_d - x_d)$
$x_d \leftarrow x_d + v_d$

Figure 1.5. *PSO Principle. For a color version of this figure, see www.iste.co.uk/feyel/optimization.zip*

In the case of the *GlobalBest* version of the algorithm, $G_p^{(k)}$ is replaced by $G^{(k)} = \underset{X \in \left\{B_i^{(k)}, i=\{1,2,\ldots,N_P\}\right\}}{\arg\min} f(X)$ which is the best position that the swarm has found at iteration k.

While the *GlobalBest* version favors convergence, the *LocalBest* version favors exploration and proves to be better adapted to high-dimensional problems: we will therefore work on this version here.

Particles confinement

To ensure the algorithm's convergence, the particles' position and speed should be confined:

$$\begin{aligned} V_p^{(k+1)} &= \min\left(\max\left(V_p^{(k+1)}, V_{\min}\right), V_{\max}\right) \\ X_p^{(k+1)} &= \min\left(\max\left(X_p^{(k+1)}, X_{\min}\right), X_{\max}\right) \end{aligned} \qquad [1.17]$$

where

– $[X_{\min}, X_{\max}]$ is the search space interval;

– $[V_{\min}, V_{\max}]$ limits a particles' maximum speed of movement and:

$$\begin{aligned} V_{\max} &= \delta\left(X_{\max} - X_{\min}\right),\ \delta \in \left]0,1\right] \\ V_{\min} &= -V_{\max} \end{aligned} \qquad [1.18]$$

Initialization of the algorithm

Initializing the algorithm means randomly choosing particles in the search space interval. It is the same for their velocity:

$$\begin{aligned} X_p^{(0)} &= X_{\min} + rnd(0,1)\left(X_{\max} - X_{\min}\right) \\ V_p^{(0)} &= V_{\min} + rnd(0,1)\left(V_{\max} - V_{\min}\right) \end{aligned} \quad [1.19]$$

The algorithm’s pseudo-code

The algorithm’s pseudo-code is given below:

```
Randomly generating particle positions and velocities in the search space using [1.19].
While the stop criterion is not satisfied, repeat:
    For $p = 1$ to $N_P$ particles:
        Compute the velocity $V_p$ of the particle,
        $V_p = \min(\max(V_p, V_{min}), V_{max})$,
        Compute the position $X_p$ of the particle,
        $X_p = \min(\max(X_p, X_{min}), X_{max})$                [1.20]
        If $f(X_p) < f(B_p)$ then $B_p = X_p$, end if
    End For
    Compute $G_p = \underset{X \in \{B_i, i \in \Omega(X_p)\}}{\arg\min} f(X),\ p = 1$ à $N_P$
    Compute $G = \underset{X \in \{B_i, i = \{1,2,\ldots,N_P\}\}}{\arg\min} f(X)$
End while
```

Canonical setting of the algorithm

The standard version of PSO (*Std_PSO*) is defined hereafter with the following settings (called canonical) for which, generally, we obtain good results:

– Swarm size $N_P = floor(10+\sqrt{n})$, where n is the number of optimization variables, in other words the problem’s dimension;

– $w = 0.729$; $\bar{c}_1 = \bar{c}_2 = 1.494 = 2.05 \times 0.729$;

– $\dim\left(\Omega\left(X_p^{(k)}\right)\right) = 3$;

– $\delta = 0.9$.

Several topologies exist for the choice of the neighborhood $\Omega\left(X_p^{(k)}\right)$, but, for the sake of simplicity, we often use a cyclic neighborhood (*social ring*) defined once at initialization. Therefore, the neighborhood of particle *i* is defined by the following set of particles:

$$\Omega(X_i) = \left(X_{(i-1)\ \text{modulo}(N_P)}\ ;\ X_i\ ;\ X_{(i+1)\ \text{modulo}(N_P)}\right) \qquad [1.21]$$

Explanations for these settings can be found in [CLE 02], [CLE 12] and [CLE 04].

The algorithm's stop criterion

Several stop criteria can be used. We will retain:

– the maximum number of iterations, knowing that, at each iteration, the fitness function is evaluated N_P times;

– a maximum number of fitness evaluations;

– premature convergence of the algorithm, which is detected when all the particles tend to be identical, that is to say, when the following relation is satisfied:

$$\exists G : \frac{\max\limits_{i=1,\dots,N_P} \left\| X_i^{(k)} - G \right\|}{\left\| X_{max} - X_{min} \right\|} < \varepsilon \qquad [1.22]$$

In most cases, $\varepsilon = 10^{-5}$ will be chosen.

Variants of the algorithm

The previous metaheuristic forms a basis from which many versions have been developed [SED 09]. As the idea is not to substitute a controller setting problem with another parameter setting problem, it is a good policy to not make the algorithm more complicated.

In order to limit the algorithm's premature convergence, we will keep the *TVRndIW_PSO*[1] version in which the inertia factor has been varied at each iteration according to the law [EBE 01]:

$$w_k = rnd\left(\frac{1}{2},1\right) \qquad [1.23]$$

The inertia factor's mean value therefore remains around 0.75. By slightly modifying the inertia factor at each iteration, we thus create diversity, which makes the algorithm more effective for high-dimensional problems. The other setting parameters remain unchanged from the canonical version.

The versions retained

We will keep the two versions of PSO (*LocalBest* version) that have proven to be most effective in our study: *Std_PSO* and *TVRndIW_PSO*.

1.3.2. *Quantum particle swarm algorithm*

Principle

An important property of PSO [CLE 02] is that the algorithm's convergence is reached if the *i*th particle converges on its local attractor, whose coordinates are defined in an *n*-dimensional search space (*n* being the problem's dimension) by $P_i^{(k)} = \left(p_{i1}{}^{(k)}, p_{i2}{}^{(k)}, \ldots, p_{ij}{}^{(k)}, \ldots, p_{in}{}^{(k)}\right)$ with:

$$p_{ij}{}^{(k)} = \frac{c_1 b_{ij}{}^{(k)} + c_2 g_j{}^{(k)}}{c_1 + c_2} = \phi b_{ij}{}^{(k)} + (1-\phi) g_j{}^{(k)} \qquad [1.24]$$

φ is a random number distributed uniformly in [0.1] and:

$$\begin{aligned} B_i^{(k)} &= \left(b_{i1}{}^{(k)}, b_{i2}{}^{(k)}, \ldots, b_{ij}{}^{(k)}, \ldots, b_{in}{}^{(k)}\right) \\ G^{(k)} &= \left(g_1{}^{(k)}, g_2{}^{(k)}, \ldots, g_j{}^{(k)}, \ldots, g_n{}^{(k)}\right) \end{aligned} \qquad [1.25]$$

1 TVRndIW: time varying randomly inertia weight.

In quantum particle swarm optimization (QPSO) [SUN 12a], the swarm is a quantum system, and each particle has a quantum behavior with its quantum state modeled by a wave function $\psi\left(\vec{X},t\right)$ that should meet the Schrödinger equation:

$$i\hbar\frac{\partial\psi\left(\vec{X},t\right)}{\partial t}=\hat{H}\left(\vec{X}\right)\psi\left(\vec{X},t\right) \qquad [1.26]$$

where $\hbar$ is the Planck constant and $i=\sqrt{-1}$. The Hamiltonian operator is defined for a particle with a mass m in a potential field $V\left(\vec{X}\right)$ by:

$$\hat{H}\left(\vec{X}\right)=-\frac{\hbar^2}{2m}\nabla^2+V\left(\vec{X}\right) \qquad [1.27]$$

Moreover, $Q=|\psi|^2$ represents the particle's probability density function to be in $\vec{X}$ and is such that:

$$\int Q(\vec{X})d\vec{X}=1 \qquad [1.28]$$

In the case of movement in a one-dimensional space and in a case where the abscissa point x_0 is the center of the potential well:

$$\begin{aligned}&V\left(X\right)=-\gamma\delta\left(X-x_0\right)=-\gamma\delta\left(Y\right)\\&\left.\begin{aligned}&\delta\left(Y\right)=1 \text{ si } Y=0\\&\delta\left(Y\right)=0 \text{ si } Y\neq 0\end{aligned}\right\}\end{aligned} \qquad [1.29]$$

where ε is the potential well's intensity and the function δ(x) is equal to 1 when x is equal to 0, and 0 if not. By denoting E the particle's energy ($E < 0$), the Hamiltonian is written as:

$$\hat{H}\left(Y\right)=-\frac{\hbar^2}{2m}\frac{d^2}{dY^2}-\left(\gamma\delta(Y)+E\right) \qquad [1.30]$$

Therefore, at equilibrium, the particle's state satisfies the following stationary Schrödinger equation (we retain the time dependency of [1.26]):

$$\hat{H}\left(Y\right)\psi\left(Y\right)=0 \qquad [1.31]$$

In other words:

$$\frac{d^2\psi}{dY^2}+\frac{2m}{\hbar^2}\left(\gamma\delta(Y)+E\right)\psi\left(Y\right)=0 \qquad [1.32]$$

A solution such as $\psi \to 0$ when $|Y| \to \infty$ is:

$$\psi(Y) = \frac{1}{\sqrt{L}} e^{-|Y|/L},\ L = \frac{2E}{\gamma},\ E = \frac{\hbar^2}{2m} \qquad [1.33]$$

and:

$$Q(Y) = \frac{1}{L} e^{-2|Y|/L} \qquad [1.34]$$

It is verified that:

$$\begin{aligned}\int_{-\infty}^{+\infty} \frac{1}{L} e^{-2|Y|/L} dY &= \int_{-\infty}^{0} \frac{1}{L} e^{2Y/L} dY + \int_{0}^{+\infty} \frac{1}{L} e^{-2Y/L} dY \\ &= \left[\frac{1}{2} e^{2Y/L}\right]_{-\infty}^{0} + \left[-\frac{1}{2} e^{-2Y/L}\right]_{0}^{+\infty} \\ &= 1\end{aligned} \qquad [1.35]$$

The corresponding probability distribution is written as:

$$F(Y) = 1 - e^{-2|Y|/L} \qquad [1.36]$$

The *i*th particle has the coordinates $X_i = (x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{in})$.

Inspired by analysis of the traditional PSO's convergence, the *i*th particle moves at iteration *k* around a potential well centered on $p_{ij}^{(k)}$ for the *j*th dimension (Figure 1.6).

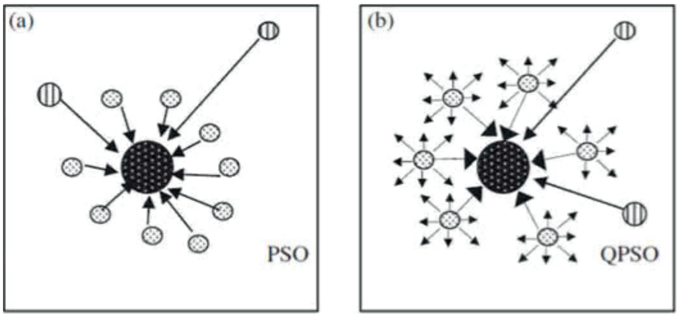

Figure 1.6. *The particles' movement in PSO (a) and QPSO (b)*

The corresponding wave function at iteration $k + 1$ is written as:

$$\psi\left(x_{ij}^{(k+1)}\right) = \frac{e^{\frac{-\left|x_{ij}^{(k)} - p_{ij}^{(k)}\right|}{L_{ij}^{(k)}}}}{\sqrt{L_{ij}^{(k)}}} \qquad [1.37]$$

The corresponding probability distribution is written as:

$$F\left(x_{ij}^{(k+1)}\right) = 1 - e^{-2\frac{\left|x_{ij}^{(k)} - p_{ij}^{(k)}\right|}{L_{ij}^{k}}} \qquad [1.38]$$

from which we infer that:

$$x_{ij}^{(k)} = p_{ij}^{(k)} \pm \frac{L_{ij}^{(k)}}{2} \ln\left(\frac{1}{1 - F\left(x_{ij}^{(k+1)}\right)}\right) \qquad [1.39]$$

Let us imagine that uniformly distributed draws are made for *F*, thus $F \approx U(0,1)$ and so $1 - F \approx U(0,1)$. By denoting $u = 1 - F$, it becomes:

$$x_{ij}^{(k)} = p_{ij}^{(k)} \pm \frac{L_{ij}^{(k)}}{2} \ln\left(\frac{1}{u}\right), \ u = U(0,1) \qquad [1.40]$$

Therefore, in [SUN 12a] and [SUN 12b], we suggest replacing the traditional PSO's evolution law [1.16] with:

$$x_{ij}^{(k+1)} = p_{ij}^{(k)} \pm \beta \left|C_j^{(k)} - x_{ij}^{(k+1)}\right| \ln\left(\frac{1}{u_{ij}^{(k+1)}}\right) \qquad [1.41]$$

where $C^{(k)} = \left(C_1^{(k)}, C_2^{(k)}, \ldots, C_n^{(k)}\right)$ is the mean of the best positions found by the particles at iteration *k*, that is to say:

$$C^{(k)} = \left(\frac{1}{P}\sum_{i=1}^{P} b_{i1}^{(k)}, \frac{1}{P}\sum_{i=1}^{P} b_{i2}^{(k)} \ldots, \frac{1}{P}\sum_{i=1}^{P} b_{in}^{(k)}\right) \qquad [1.42]$$

$u_{ij}^{(k+1)}$ is a random number distributed uniformly in [0.1] and β is the contraction–expansion coefficient (CEC) and is used to control the algorithm's rate of convergence.

Particles confinement

To ensure the algorithm's convergence, the particles' position should be confined in the search space:

$$X_p^{(k+1)} = \min\left(\max\left(X_p^{(k+1)}, X_{\min}\right), X_{\max}\right) \quad [1.43]$$

Initializing the algorithm

Initializing the algorithm consists of randomly choosing particles in the search space interval:

$$X_p^{(0)} = X_{\min} + rnd(0,1)\left(X_{\max} - X_{\min}\right) \quad [1.44]$$

The algorithm's pseudo-code

The algorithm's pseudo-code is given below:

```
Randomly generate particle positions in the search space
While the stop criterion is not satisfied, repeat:
    Compute the best mean position $C$ with [1.42],
    For $i$ = 1 to $N_P$ particles, do:
        if $f(X_i) < f(B_i)$ then $B_i = X_i$ end if
        For $j$ =1 to $n$ dimensions, do:
            $\varphi = rnd(0.1)$; $u = rnd(0.1)$;
            $p_{ij} = \varphi .b_{ij} + (1-\varphi).g_j$
            if rnd(0,1) > 0,5                                  [1.45]
                $x_{ij} = p_{ij} + \beta.\text{abs}(C_j - x_{ij}).\log(1/u)$
            else
                $x_{ij} = p_{ij} - \beta.\text{abs}(C_j - x_{ij}).\log(1/u)$
            end if
            $x_{ij} = \min(\max(x_{ij}, X_{min\_j}), X_{max\_j})$
        End for
    End for
    Find $G = \underset{X \in \{B_i, i=\{1,2,\ldots,N_P\}\}}{\arg\min} f(X)$
End while
```

Setting the algorithm

One advantage of this algorithm is its low number of setting parameters. The only parameter to set is β. The drawback is that the algorithm’s behavior is very sensitive to β. However, in general, a constant value of β around 0.75 leads to good results [SUN 12a]. This is the *StaticCEC_QPSO* version.

Following the same idea as the traditional PSO, the size of the swarm is fixed at $N_P = floor(10 + \sqrt{n}\,)$, where n is the number of optimization variables, that is to say the problem’s dimension.

The algorithm’s stop criterion

Several stop criteria can be used. We will retain:

– the maximum number of iterations, knowing that, at each iteration, the cost function is evaluated N_P times;

– a maximum number of fitness evaluations;

– the algorithm’s premature convergence, which is detected when all the particles tend to be identical, that is to say, when the following relationship is satisfied:

$$\exists G : \frac{\max\limits_{i=1,\dots,N_P} \left\| X_i^{(k)} - G \right\|}{\left\| X_{max} - X_{min} \right\|} < \varepsilon \tag{1.46}$$

In most cases, we will choose $\varepsilon = 10^{-5}$.

Variants of the algorithm

Many versions of QPSO exist [SUN 12a]. Most consist of varying β according to particular laws. For example, we can create diversity at each iteration by randomly varying β as follows:

$$\beta(k) = rnd\left(\beta_{\min}, \beta_{\max}\right) \; ; \; \beta_{\max} = 1{,}0 \; ; \; \beta_{\min} = 0{,}5 \tag{1.47}$$

We will retain version *TVDecCEC_QPSO* [SUN 12a, SUN 12b], in which β decreases during an N-iterations optimization process according to the following law:

$$\beta(k) = \beta_{\max} - \frac{\beta_{\max} - \beta_{\min}}{N} \times k \; ; \; \beta_{\max} = 1{,}0 ; \beta_{\min} = 0{,}5 \tag{1.48}$$

The versions retained

Finally, we will retain the two versions of QPSO that have proven to be most effective in our study: *StaticCEC_QPSO* and *TVDecCEC_QPSO*.

1.3.3. *Artificial bee colony optimization algorithm*

Principle

Artificial bee colony (ABC) optimization [KAR 07, KAR 08] is a recent metaheuristic inspired by the natural model of the behavior of honey bees when they search for food.

The bees' process of searching for food is based on a very effective movement mechanism. It enables them to draw the attention of the other bees in the colony to food sources found, with the aim of collecting diverse resources. In fact, the bees use a set of wriggling dances as a means of communication with each other. These dances enable the bees to share information on the direction, distance and quantity of nectar with their peers. The collective collaboration and knowledge of bees in the same colony is based on the exchange of information on the quantity of nectar in the food source found by the various members.

In a bee colony optimization algorithm, a source of nectar corresponds to a possible solution of the problem to be solved. The colony of artificial bees consists of three types of bees: the "employed" bees, the onlooker bees and the scouts:

– the employed bee exploits the food source found. It relies on its memory and tries to make changes to its current position (solution) to find a new position (i.e. food source);

– the onlooker waits for the employed bees to return to the dance field to observe their dances and collect information on the sources of nectar that they have found;

– the scout bee exploits the search space by launching a random search for a new food source.

Initializing the algorithm

The initial population is formed of a number of N_{FS} food sources randomly generated in the search space. Assuming that the population's *i*th food source is represented by $X_i = \left(x_{i1},\ldots,x_{ij},\ldots,x_{in}\right)$, where n is the problem's dimension, each food source is generated by:

$$x_{ij} = X_{\min j} + rnd(0,1)\left(X_{\max j} - X_{\min j}\right),\ j = 1,\ldots,n,\ i = 1,\ldots,N_{FS} \qquad [1.49]$$

These food sources are randomly assigned to N_O employed bees in the hive and the values of the corresponding cost functions are evaluated. Here, we will assume that $N_O = N_{FS}$.

Similarly, we will suppose that the number of onlooker bees N_S is equal to N_O.

Finally, each food source has an abandonment counter initialized at 0.

The k^{ith} iteration consists of successively carrying out the three phases described below.

Algorithm description

– Employed bee phase:

In this phase, each employed bee generates a new food source $x_{new,}$ chosen in the neighborhood of the current position found:

$$x_{newij} = x_{ij} + rnd(-1,1)\left(x_{ij} - x_{kj}\right), \quad i = 1,\ldots,N_O \qquad [1.50]$$

where $k \in \left\{1,2,3,\ldots,N_{FS}\right\}$, so that $k \neq i$ and $j \in \left\{1,2,3,\ldots,n\right\}$ are chosen at random.

We then proceed to a selection after evaluating the cost function for this new solution. X_{new} replaces X_i in the population if $f\left(X_{new}\right) \leq f\left(X_i\right)$ (and its abandonment counter is re-initialized), otherwise X_i is retained and its abandonment counter is incremented.

– Onlooker phase:

In this stage, the onlookers recover information on the quantity of nectar in the source x_i from the employed bees. The probability p_i that the onlookers choose the source x_i is determined as follows:

$$p_i = \frac{fit_i}{\sum_{k=1}^{N_{FS}} fit_k} \qquad [1.51]$$

where fit_i is the quantity of nectar in the ith food source X_i.

In the literature, we also find the following expression [OZT 12]:

$$p_i = 0{,}1 + \frac{0{,}9\, fit_i}{\max\limits_k \left(fit_k\right)} \qquad [1.52]$$

The quantity of nectar linked to the source X_i is determined by:

$$fit_i = \begin{cases} \dfrac{1}{1+f(X_i)}, \ if \ f(X_i) \geq 0 \\ \\ 1+|f(X_i)|, \ if \ f(X_i) < 0 \end{cases} \qquad [1.53]$$

According to [1.51] and [1.52], it is clear that the higher the fit_i is, the greater the probability of selecting the corresponding source X_i is.

Once the ith source is selected by the onlookers, they modify it using [1.50].

If the source thus modified is better than X_i, then the modified source replaces X_i in the population (its abandonment counter is then reinitialized), otherwise X_i is retained and its abandonment counter is incremented.

– Scout phase:

The ith food source is abandoned if it cannot be improved after a set number of tests, T_{limit} decided beforehand, (in other words, if its abandonment counter exceeds T_{limit}), and the corresponding employed bee then becomes a scout bee. The scout bee then generates its food source randomly:

$$x_{ij} = X_{\min j} + rnd(0,1)\left(X_{\max j} - X_{\min j}\right), \ j = 1,\ldots,n \qquad [1.54]$$

Pseudo-code of the algorithm

The pseudo-code of the algorithm is given below:

```
For each employed bee
    Generate an initial solution $X_i$ for each worker bee with [1.49],
    Evaluate the cost function $f(X_\mathrm{i})$
    Evaluate the associated quantity of nectar ($fit_i$) using [1.53],
    Initialize an abandonment counter $C_i = 0$.

The stop criterion is not satisfied, repeat:

Employed Phase
        For each employed bee
        Generate a new position $X_{new}$ with [1.50],                    [1.55]
        Evaluate the corresponding cost function $f(X_{new})$,
        Evaluate its quantity of nectar ($fit_{new}$) using [1.53],
        If $f(X_{new}) \leq f(X_i)$, then
                    $X_i \leftarrow X_{new}$
            $C_i = 0$
        Else
            $C_i = C_i + 1$
        End If
    End For

Evaluate the probability of transmitting information associated with each source $p_i$ using [1.51],

Onlooker phase
        For each onlooker
            if rnd(0,1) < $p_i$
              Generate a new position $X_{new}$ with [1.50],
              Evaluate the corresponding cost function $f(X_{new})$,
              Evaluate its quantity of nectar ($fit_{new}$) using [1.53]
            If $f(X_{new}) \leq f(X_i)$, then
                  $X_i \leftarrow X_{new}$
                  $C_i = 0$
              Else
                  $C_i = C_i + 1$
              End If

  Scout phase
        For each scout
              If $C_i \geq T_{limit}$
```

```
                Generate a new position with [1.54],
                Evaluate the corresponding cost
                Evaluate its quantity of nectar ($fit_i$) using [1.53],
                $C_i = 0$
        Retain the best solution $X_{best}$
        End while
```

Stop criterion

Several stop criteria can be used. We will retain:

– the maximum number of iterations, knowing that, at each iteration, the fitness function is evaluated P times;

– a maximum number of evaluations of the cost function;

– the algorithm's premature convergence, which is detected when all the bees tend to be identical, in other words when the following relation is met:

$$\exists X_{best} : \frac{\max\limits_{i=1,\ldots,P} \left\| X_i^{(k)} - X_{best} \right\|}{\left\| X_{max} - X_{min} \right\|} < \varepsilon \qquad [1.56]$$

In most cases, we will choose $\varepsilon = 10^{-5}$.

Variants of the algorithm

The algorithm's parameters are the number of food sources N_{FS}, the number of employed bees N_O, the number of onlookers N_S and, finally, the limit T_{limit} at which the food source should be abandoned.

The standard version of the algorithm consists of taking:

$$N_{FS} = N_O = N_S \qquad [1.57]$$

Therefore, there remain only two parameters to set.

Although a population size N_{FS} about $10n$ is generally advised, the algorithm has given its best results using the same idea as in (Q)PSO, in other words:

$$N_{FS} = floor\left(10 + \sqrt{n}\right) \qquad [1.58]$$

Finally, the literature recommends choosing [KAR 08]:

$$T_{limit} = n \times N_{FS} \tag{1.59}$$

1.3.4. *Cuckoo search algorithm*

Principle

The CS is a recent metaheuristic [YAN 09a] inspired by the reproduction method of some species of cuckoo. In fact, their reproduction strategy has the peculiarity that the females lay their eggs in other species' nests (where the eggs have a similar appearance (Figure 1.7)). The eggs can therefore be hatched by substitute parents. Moreover, when the cuckoo eggs succeed in hatching in the host nest (they hatch more rapidly), the cuckoo chicks have a reflex that causes them to throw the eggs of the host species out of the nest and even imitate the cry of the host chicks with the aim of being fed by the host species.

Figure 1.7. *A cuckoo egg (gray) in a dunnock nest. For a color version of this figure, see www.iste.co.uk/feyel/optimization.zip*

However, it is possible for the cuckoo eggs to be discovered; in this case, the substitute parents remove them from the nest, or abandon the nest and start their brood elsewhere. This metaheuristic is therefore based on the cuckoo species' parasitic behavior, linked to a "Levy flight" displacement specific to certain birds and certain species of flies.

The CS algorithm is based on the following rules:

– each cuckoo lays only one egg at a time and places it in a nest chosen randomly;

– the best nests with high-quality eggs (solutions) are kept for subsequent generations;

– the number of host nests is fixed and the egg laid by the cuckoo can be discovered by the host species with a probability $p_a \in [0,1]$. In this case, the host bird either removes the egg from the nest, or leaves the nest and builds a new one. To simplify, this last hypothesis can be approached by replacing a fraction p_a of n nests with new ones.

In CS, each egg in a nest represents a solution and each cuckoo can lay a single egg (which represents a solution), the aim is to use the new and potentially best solution to replace a poorer solution in a nest. Although the algorithm can be extended to a more complex case where each nest contains several eggs representing a set of solutions, here we use the simplest version where each nest only contains a single egg. In this case, there is no longer a distinction between egg, nest and cuckoo, and each nest corresponds to an egg that also represents a cuckoo.

Levy flight

In the framework of CS, the cuckoos' displacement step is determined by Levy flight (Figure 1.8). The Levy flight is a random walk in which the steps have a length with a particular probability distribution (Levy distribution), the steps' direction being isotropic and random. The Levy flight is a class of random walk in which the jumps are distributed according to the Levy distribution, which consists of a power law with infinite variance and infinite mean of type:

$$Levy(\beta) \sim y = x^{-\beta},\ 1 < \beta \leq 3 \qquad [1.60]$$

In the case of CS, the use of the Levy flight improves and optimizes the search: new solutions are generated by a Levy random walk around the best solution yet obtained, which accelerates the global search.

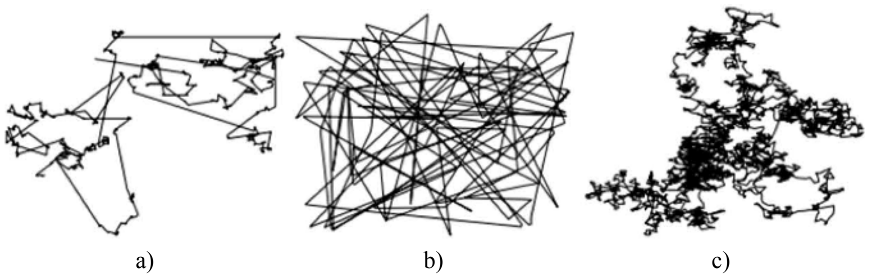

Figure 1.8. *Levy flight a), simple random walk b) and Brownian movement c)*

From the point of view of implementation, generating a random number with the Levy flight follows two stages: the choice of a random direction and the generation of a step that ought to obey the Levy distribution. A direction can be generated from a uniform distribution, whereas generating steps is more delicate. There are several methods for arriving at this, but one of the simplest and effective methods consists of using the Mantegna formulae to determine step s:

$$s = \frac{u}{|v|^{1/\beta}} \qquad [1.61]$$

where u and v are normal distributions centered such that:

$$u = N\left(0, \sigma_u^2\right), \quad v = N\left(0, \sigma_v^2\right) \qquad [1.62]$$

with:

$$\sigma_u^2 = \frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma\left((1+\beta)/2\right)\beta 2^{(\beta-1)/2}}, \quad \sigma_v^2 = 1 \qquad [1.63]$$

where $\Gamma(z)$ is the gamma function:

$$\Gamma(z) = \int_0^{+\infty} t^{z-1} e^{-t} dt \qquad [1.64]$$

Initializing the algorithm

The initial population consists of a number N_H of host nests randomly generated in the search space. We recall that the numbers of host nests, cuckoos and eggs are equal.

Assuming that the *i*th host nest is represented by $X_i = \left(x_{i1},...,x_{ij},...,x_{in}\right)$, where *n* is the problem's dimension, each nest is generated by:

$$x_{ij} = X_{\min j} + rnd(0,1)\left(X_{\max j} - X_{\min j}\right),\ j = 1,\ldots,n,\ i = 1,\ldots,N_H \qquad [1.65]$$

The corresponding cost functions are evaluated.

Describing the algorithm

– Step 1: global search

The first step consists of carrying out a global search. To do this, new cuckoo (solution) tests are generated from existing cuckoos (solution) by performing the Levy flight. This consists of using the following evolution law for each cuckoo:

$$\left.\begin{aligned} X_{new\,i} &= X_i + \alpha \otimes Levy(\beta) \\ F_{new\,i} &= f\left(X_{new\,i}\right) \end{aligned}\right\},\ i = 1,\ldots,N_H \qquad [1.66]$$

where α > 0 is the size of the displacement step, which depends on the problem considered and such that $\alpha \approx O\left(L/10\right)$, where *L* is the characteristic scale of the problem considered. The product ⊗ is an element-wise product.

In order to know where/if the new *i*th cuckoo will deposit an egg, it randomly chooses a host nest X_j ($j \neq i$) with the cost function F_j.

The new cuckoo lays an egg if:

$$F_{new\,i} < F_j \qquad [1.67]$$

In this case, solution $X_{new\,i}$ replaces solution X_j in the population.

– Step 2: local search

The new cuckoo eggs have a probability p_a of being discovered.

In this case ($rnd(0,1) < p_a$), new eggs are generated. Several methods can be envisaged for this. For CS, we will retain the following, where we proceed to a hybridization between elements randomly chosen in the population:

$$\begin{aligned} x'_{new\,ij} &= x_{ij} + rnd(0,1)\left(x_{pj} - x_{mj}\right),\ j = 1,\ldots,n,\ i = 1,\ldots,N_H \\ F'_{new\,i} &= f\left(X'_{new\,i}\right) \end{aligned} \qquad [1.68]$$

where $m \in \{1,2,3,\ldots,N_H\}$ and $p \in \{1,2,3,\ldots,N_H\}$, such that $p \neq m$ are randomly chosen.

A selection identical to that of the previous step is then made.

Pseudo-code of the algorithm

The algorithm's simplified pseudo-code is given below:

```
Generate an initial population of $N_H$ host nests $X_i$
While the stop criterion is not satisfied, repeat:
    Generate a cuckoo using Levy Flight [1.66] and evaluate its cost function $F_i$ ,
    Choose a nest, $X_j$, at random from among the host nests $N_H$,
    If $F_i < F_j$                                                    [1.69]
        Replace $F_j$ with this new solution,
    End If
    A fraction $p_a$ of the worst nests is abandoned and others are generated instead
    [1.68], and make a selection.
    Identify the best solution $X_{best}$
End while
```

Setting the algorithm

We will retain the following standard settings for CS [YAN 09a]:

– The eggs' probability of discovery: $p_a = 0.25$

– Size of the displacement step: $\alpha = \dfrac{\left\| X_{\min} - X_{\max} \right\|}{10}$

– Sizing of the Levy flight: $\beta = 3/2$

There are no empirical rules for choosing the number of nests. However, following the same principles as PSO, the algorithm has given its best results with a number of nests equal to:

$$N_H = floor\left(10 + \sqrt{n}\right) \qquad [1.70]$$

The stop criterion of the algorithm

Several stop criteria can be used. We will retain:

– the maximum number of iterations, knowing that, at each iteration, the fitness function is evaluated P times;

– a maximum number of fitness evaluations;

– the algorithm's premature convergence, which is detected when all the cuckoos tend to be identical, in other words when the following relation is satisfied:

$$\exists X_{best} : \frac{\max\limits_{i=1,\ldots,P} \left\| X_i^{(k)} - X_{best} \right\|}{\left\| X_{max} - X_{min} \right\|} < \varepsilon \qquad [1.71]$$

In most cases, we will choose $\varepsilon = 10^{-5}$.

Variants of the algorithm

Various existing versions consist of modifying the local and/or the global exploration phase. We will retain the CS/DE version [WAN 12], which consists of hybridizing the algorithm with DE. More precisely, the principle consists of mutation and cross-over operations of DE/Rand/1 in the CS global search phase to respectively generate new solutions instead of the Levy flight and potentially to select them. This generation/selection mechanism is given by [1.72].

```
For i = 1 is $N_H$, do
    Randomly select $r_1 \neq r_2 \neq r_3 \neq i$,
    $X_v = X_{r1} + F \times (X_{r2} - X_{r3})$,
    $r_4 = floor(1 + rnd(0,1) \times N_H)$
    For $j$ = 1 to $n$, do
If $rnd(0,1) \leq C_r$ or j == $r_4$ then
            $X_u(j) = X_v(j)$
        Else        $X_u(j) = X_i(j)$
        End If                                              [1.72]
    End for
    If $F(X_u) < F(X_{r4})$ then
        $X_{r4} = X_u$
        $F(X_{r4}) = F(X_u)$
    End If
End For
```

We find again the DE mutation and cross-over operators. We will retain the following standard settings: $F = 0.75$ and $C_r = 0.8$.

The CS/DE pseudo-code is given below:

```
Generating an initial population of $N_H$ host nests $X_i$
While the stop criterion is not satisfied, repeat:
    Generate a new cuckoo population by [1.72]                    [1.73]
    Proceed with the local search (stage 2)
    Identify the best solution $X_{best}$
End while
```

The versions retained

We will retain the two versions of CS that have proven to be most effective in our study: *CS* and *CS/DE*.

1.3.5. *Firefly algorithm*

Principle

The firefly algorithm (firefly – FF) [YAN 09b] is inspired by fireflies' brilliant phosphorescence and their behavior of attraction. In particular, bright fireflies attract one another (the brightest attracts the others), and the further apart they are, the less they attract. Some hypotheses are realized to simplify the algorithm and improve its efficiency:

– each firefly is assumed to be unisex, which means that it can be attracted by any other firefly;

– the fireflies are only attracted by other fireflies that are brighter and this attraction decreases with the distance that separates them;

– luminosity is determined by the cost function to be minimized.

At iteration *k*, the position of the *i*th firefly attracted by the *j*th firefly is revealed by the following evolution law:

$$X_i^{(k+1)} = X_i^{(k)} + \beta_{ij}\left(X_j^{(k)} - X_i^{(k)}\right) + \alpha_k \varepsilon_i^{(k)} \qquad [1.74]$$

where β_{ij} is the attraction coefficient of the *i*th firefly to the *j*th firefly and α_k is a parameter. In most cases:

$$\alpha_k = \alpha_0^{\ k} \qquad [1.75]$$

ε_i is usually a normal distribution but, as in *CS*, the *Levy flight* principle can also be used.

The attraction is assumed to follow the following law:

$$\beta_{ij} = \begin{cases} \beta_0 e^{-\gamma r_{ij}^{\ 2}} & \text{si } f(X_i) \geq f(X_j) \\ 0 & \text{si } f(X_i) < f(X_j) \end{cases} \qquad [1.76]$$

where ε is a parameter (absorption coefficient) and r_{ij} is the distance between the fireflies X_i and X_j, defined as the Euclidean norm:

$$r_{ij} = \left\| X_i - X_j \right\| = \sqrt{\sum\nolimits_k \left(x_{ik} - x_{jk} \right)^2} \qquad [1.77]$$

Pseudo-code of the algorithm

The algorithm's pseudo-code is given below:

```
Generate randomly an initial population of $N_L$ fireflies $X_i$ ($i$=1,…,$N_L$) in the search space,
The intensity of light $I_i$ at $X_i$ is determined by $f(X_i)$,
define the light absorption coefficient ε,
While the stop criterion is not satisfied, repeat:
    For $i$ = 1 to $N_L$ fireflies
        For $j$ = 1 to $N_L$ fireflies
            if ($I_i > I_j$) then
                Move firefly $i$ towards firefly $j$ with [1.74]                [1.78]
            End If
            The attraction varies with the distance $r$ via exp($-r^2$),
            Evaluate the new solutions and reveal the intensities.
        End For j
    End For i
    Identify the best solution $X_{best}$
End while
```

Setting the algorithm

We will retain the following standard settings:

– Sizing of the Levy flight: $\beta = 3/2$

– Absorption coefficient: $\gamma = \dfrac{1}{d_{\max}^{\ 2}}$, $d_{\max} = \left\| X_{\max} - X_{\min} \right\|$

– Initialization: $\beta_0 = 0{,}01L$, $L = \dfrac{1}{D}\sum_k \left(X_{\max k} - X_{\min k} \right)$

– Initialization: $\alpha_0 = 0{,}9$

It is shown that, in some conditions, *FF* tends towards particular variants of *PSO*. Thus, we can use the same rules for setting the number of fireflies:

$$N_L = floor\left(10 + \sqrt{D}\right) \qquad [1.79]$$

Stop criterion of the algorithm

Several stop criteria can be used. We will retain:

– the maximum number of iterations, knowing that, at each iteration, the fitness function is evaluated *P* times;

– a maximum number of fitness evaluations;

– the algorithm's premature convergence, which is detected when all the fireflies tend to be identical, in other words when the following relationship is met:

$$\exists X_{best} : \frac{\max\limits_{i=1,\dots,P} \left\| X_i^{(k)} - X_{best} \right\|}{\left\| X_{max} - X_{min} \right\|} < \varepsilon \qquad [1.80]$$

In most cases, $\varepsilon = 10^{-5}$ will be chosen.

1.4. Summary

The algorithms and the variants retained can be summarized in the following table:

Family	Name of the algorithm	Description
Swarm	Std_PSO	Particle swarm algorithm, *LocalBest* version, canonical settings
	TVRndIW_PSO	Particle swarm algorithm, *LocalBest* version, canonical settings with stochastic inertia factor around the canonical value
	StaticCEC_QPSO	Classical quantum particle swarm algorithm with a constant contraction–expansion coefficient
	TVDecCEC_QPSO	Classical quantum particle swarm algorithm with a contraction–expansion coefficient that decreases over time
	FF	Firefly algorithm
	ABC	Standard bee colony algorithm
	CS	Classical cuckoo search algorithm
Evolutionary	Rnd_DE	Differential evolution algorithm in DE/Rand/1 version
	Rnd2Bst_DE	Differential evolution algorithm in DE/Rand-2-best/1 version
	Rand2Bst_PDE	Differential evolution algorithm in PDE/Rand-2-best/1 version
Swarm/evolutionary hybrid	CS/DE	Cuckoo search algorithm CS hybridized with DE/Rand/1

Table 1.1. *Tested algorithms*

Other algorithms have been tested but without real success (either because they have not converged, or because they are too complex to set). For example, we cite:

– Bat algorithm (BA) [YAN 10b];

– Electromagnetic-like algorithm (ElA) [BIR 03];

– Harmony search (HS) [LEE 05].