1

An Introduction to Optimization

1.1 A General Optimization Problem

1.1.1 Definition

1.1.1.1 Qualitative Description

Throughout human being history, mankind has faced optimization problems and made great efforts to solve them. Loosely speaking, optimization is the process of finding the best way to use available resources, while at the same time not violating any of the constraints that are imposed. More accurately, we may say that we wish to define a system mathematically, identify its variables and the conditions they must satisfy, define properties of the system, and then seek the state of the system (values of the variables) that gives the most desirable (largest or smallest) properties. This general process is referred to as optimization.

It is not our purpose here to define a system. This is the central problem of various disciplines which are sciences or are struggling to become sciences. Our concern here is, given a meaningful system, what variables will make the system have the desirable properties.

1.1.1.2 Mathematical Formulation

The general optimization problem is mathematically defined as
Find $\mathbf{x}^* = [x_1^* \quad x_2^* \quad \cdots \quad x_N^*] \in D^N = D_1 \cap D_2 \cap \cdots \cap D_N$
where

$$f_i^{\min}(\mathbf{x}^*) \leq f_i^{\min}(\mathbf{x}) \, \forall \mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_N] \in D^N, \quad 1 \leq i \leq N_{f^{\min}},$$

$$f_i^{\max}(\mathbf{x}^*) \geq f_i^{\max}(\mathbf{x}) \, \forall \mathbf{x} \in D^N, \quad 1 \leq i \leq N_{f^{\max}},$$

$$c_i^{=}(\mathbf{x}^*) = 0, \quad 1 \leq i \leq N_{c^{=}},$$

$$c_i^{+}(\mathbf{x}^*) > 0, \quad 1 \leq i \leq N_{c^{+}},$$

$$c_i^{-}(\mathbf{x}^*) < 0, \quad 1 \leq i \leq N_{c^{-}}.$$

Differential Evolution Anyong Qing

$\mathbf{x}^*$ is the optimal solution in the N-dimensional search space D^N. N is the number of optimization parameters, or the dimension of the optimization problem. D_i, either continuous or discrete, is the search space of x_i, the ith optimization parameter. D_i and D_j are not necessarily identical, from the point of view of either type or size. $\mathbf{x}$ is an N-dimensional vector of optimization parameters. $f_i^{\min}(\mathbf{x})$ is the ith objective function to be minimized, $N_{f^{\min}}$ is the number of objective functions to be minimized. $f_i^{\max}(\mathbf{x})$ is the ith objective function to be maximized, $N_{f^{\max}}$ is the number of objective functions to be maximized. $c_i^{=}(\mathbf{x})$ is the ith equality constraint function, $N_{c^{=}}$ is the number of equality constraint functions. $c_i^{+}(\mathbf{x})$ is the ith positive constraint function, $N_{c^{+}}$ is the number of positive constraint functions. $c_i^{-}(\mathbf{x})$ is the ith negative constraint function, $N_{c^{-}}$ is the number of negative constraint functions.

An optimization problem is thus made up of three essential ingredients: optimization parameters $\mathbf{x}$; objective functions, $f_i^{\min}(\mathbf{x})$ and $f_i^{\max}(\mathbf{x})$; and constraint functions, $c_i^{=}(\mathbf{x})$, $c_i^{+}(\mathbf{x})$, and $c_i^{-}(\mathbf{x})$.

Objective and constraint functions in many real-world application problems can be formulated in many ways. There might be better formulation of objective and constraint functions to describe a particular optimization problem. Any knowledge about the optimization problem should be worked into the objective and constraint functions. Good objective and constraint functions can make all the difference.

1.1.1.3 Some Examples

1.1.1.3.1 Investment Fund Management An investment fund manager is authorized to invest a sum of money A in N investment funds which claim to offer return rates r_i, $1 \leq i \leq N$. The rate of administration charge for fund i is c_i. Suppose he invests an amount x_i in the ith fund. How can he plan his investment so that he will get maximum return?

1.1.1.3.2 Experimental Data Fitting Suppose a mathematical model $h(\mathbf{x})$ with unknown physical parameters $\mathbf{x}$ has been built for a physical phenomenon. A total of m experiments have been done during which experimental data $\mathbf{y}$ have been collected. It is then required to fit the mathematical model $h(\mathbf{x})$ to the collected experimental data $\mathbf{y}$.

1.1.1.3.3 Radome Design Radomes are used to protect antennas. Materials for a radome can only be chosen from the available materials database. A radome has to meet certain electromagnetic and mechanic demands. It is also subject to economic constraints and fabrication limitations.

1.1.2 Optimization Parameters

Optimization parameters $\mathbf{x}$ are critical for an optimization problem. They affect the value of objective and constraint functions. If there are no optimization parameters, we cannot define the objective and constraint functions. In the investment fund management problem, the optimization parameters are the amounts of money invested in each fund. In experimental data fitting problems, the optimization parameters are the parameters that define the model. In the radome design problem, the optimization parameters might include the material index in the materials database, material thickness, and some other parameters.

An optimization parameter can be continuous, discrete, or even symbolic.

1.1.3 Objective Functions

An objective function $f(\mathbf{x})$ is what we want to optimize. It is either $f_i^{\min}(\mathbf{x})$ or $f_i^{\max}(\mathbf{x})$, depending on the desirable properties of the optimization problem. For instance, in the investment fund management problem, the fund manager wants to maximize the return. In fitting experimental data to a user-defined model, we might minimize the total deviation of observed data from predictions based on the model. In the radome design problem, we have to maximize the strength and minimize the distortion and cost.

Almost all optimization problems have objective functions. However, in some cases, such as the design of integrated circuit layouts, the goal is to find optimization parameters that satisfy the constraints of the model. The user does not particularly want to optimize anything, so there is no reason to define an objective function. This type of problem is usually called a feasibility problem. On the other hand, in some optimization problems, there is more than one objective function. For instance, in the radome design problem, it would be nice to minimize weight and maximize strength simultaneously.

An objective function has at least one global optimum, and may have multiple local optima as shown in Figure 1.1. In the remainder of this book, optimum stands for the global optimum of an objective function, $f(\mathbf{x}^*)$, the objective function value at the optimal solution point $\mathbf{x}^*$. For an optimization problem with multiple objective functions, optimal solution points corresponding to different objective functions may be inconsistent.

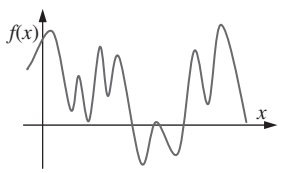

Figure 1.1 A one-dimensional continuous multimodal objective function

An objective function has some characteristic features. These features are very important for choosing optimization algorithms to solve the optimization problem of interest.

1.1.3.1 Continuity

An objective function can be continuous as shown in Figure 1.1, or discontinuous, as shown in Figure 1.2.

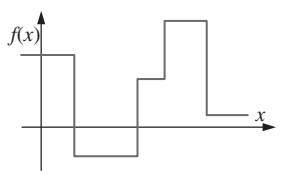

Figure 1.2 A one-dimensional discontinuous objective function

1.1.3.2 Decomposability

Decomposability is sometimes also referred to as nonlinearity, separability, or epistasis. If each optimization parameter x_i is independent of the other parameters x_j ($j \neq i$) in $f(\mathbf{x})$, it is decomposable and is relatively easy to optimize, since the landscape can be decomposed into simpler components. The optimization process of a decomposable objective function can be performed in a sequence of N independent optimization processes, where each parameter is optimized independently.

As an example, consider minimizing the two-dimensional sphere function

$$f(\mathbf{x}) = x_1^2 + x_2^2. \tag{1.1}$$

Regardless of the value of x_2, we can find the solution $x_1 = 0$ which minimizes $f(\mathbf{x})$, and vice versa for x_1.

However, many objective functions are not decomposable in this manner. The two-dimensional Chung–Reynolds function

$$f(\mathbf{x}) = (x_1^2 + x_2^2)^4 \tag{1.2}$$

is not decomposable but is very easy to optimize, since its first-order derivative is a product

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = 8x_i(x_1^2 + x_2^2)^3. \tag{1.3}$$

Such a product yields a solution for $x_1 = 0$ that is independent of x_2, and vise versa.

From this observation, the following general condition was developed by Salomon [1] in order to determine whether an objective function $f(\mathbf{x})$ is easy to optimize or not:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = g(x_i)h(\mathbf{x}), \tag{1.4}$$

where $g(x_i)$ means any function of only x_i and $h(\mathbf{x})$ means any function of $\mathbf{x}$. If this condition is satisfied, $f(\mathbf{x})$ is partially decomposable and easy to optimize, because solutions for each x_i can still be obtained independently of all other parameters $x_j, j \neq i$. The Chung–Reynolds function is partially decomposable and is therefore easy to optimize.

On the other hand, the Rosenbrock saddle function

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2 \tag{1.5}$$

is not decomposable since the equations below do not satisfy the above condition:

$$\frac{\partial f(\mathbf{x})}{\partial x_1} = 400x_1(x_1^2 - x_2) + 2(x_1 - 1), \tag{1.6}$$

$$\frac{\partial f(\mathbf{x})}{\partial x_2} = 200(x_2 - x_1^2). \tag{1.7}$$

1.1.3.3 Differentiability

A continuous objective function may be non-differentiable or differentiable of order n (≥ 1). A continuous non-differentiable function is shown in Figure 1.3.

Figure 1.3 A continuous non-differentiable one-dimensional objective function

1.1.3.4 Modality

An objective function $f(\mathbf{x})$ is unimodal if there is some path from every point $\mathbf{x}$ to the optimal solution point $\mathbf{x}^*$ along which it is monotonous ([2], p. 106). Otherwise, it is multimodal. Figure 1.1 shows a one-dimensional multimodal objective function and Figure 1.4 shows a two-dimensional multimodal objective function where the dot is the optimal point $\mathbf{x}^*$.

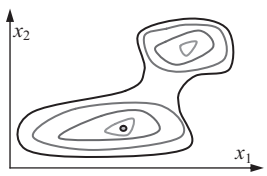

Figure 1.4 A two-dimensional multimodal objective function

A one-dimensional unimodal objective function is drawn in Figure 1.5 while a two-dimensional unimodal objective function is drawn in Figure 1.6.

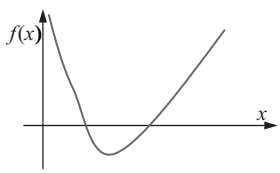

Figure 1.5 A one-dimensional unimodal objective function

1.1.3.5 Noise

Some optimization problems are dynamic or even noisy. In another word, their objective and/or constraint functions are random to a certain extent. Noise is most often used to represent randomness.

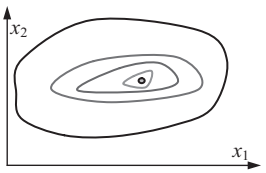

Figure 1.6 A two-dimensional unimodal function

1.1.3.6 Scalability

An objective function $f(\mathbf{x})$ is not scalable if its number of parameters, or dimension, is fixed. On the other hand, scalable objective functions can be scaled to any dimension. As dimension increases, the search space size also increases exponentially. The difficulty of finding an optimal solution increases accordingly.

1.1.3.7 Symmetry

A symmetric objective function $f(\mathbf{x})$ does not change its value if $\mathbf{x}$ is replaced by any of its $N!$ permutations. The aforementioned sphere function and Chung–Reynolds function are symmetric while the Rosenbrock saddle function is non-symmetric or asymmetric.

1.1.3.8 Uniqueness

An optimal solution point $\mathbf{x}^*$ is unique if there is only one optimal solution. Some objective functions, for example, $f(x) = \sin(x),\ x \in [-2\pi,\ 2\pi]$, may have multiple optimal solution points. Objective function values at all optimal solution points are equal. In this regard, no optimal solution is distinguishable from others.

Some people confuse uniqueness with modality.

1.1.4 Constraint Functions

Constraints allow the optimization parameters to take on certain values but exclude others. For the investment fund management problem, the amount of money invested must not exceed the available money.

Constraints are not absolutely necessary for an optimization problem. In fact, the field of unconstrained optimization is a large and important one for which a lot of algorithms and software are available. However, it has been argued that almost all problems really do have constraints. For example, any optimization parameter denoting the number of objects in a system can only be meaningful if it is less than the number of elementary particles in the known universe! Nevertheless, in practice, answers that make good sense in terms of the underlying physics can often be obtained without putting constraints on the optimization problems.

Sometimes, constraint functions and objective functions for an optimization problem are exchangeable, depending on the priority of the desirable properties. In fact, later in this book, we avoid distinguishing objective functions and constraint functions whenever possible. Moreover, without loss of generality, in the rest of this book, we are exclusively concerned

with minimization problems unless explicitly stated otherwise. As a matter of fact, it is very easy to convert a maximization problem into minimization problem.

Many people regard the search space for an optimization problem as a constraint. However, in this book, we do not take this approach.

1.1.5 Applications

Optimization has a consistent track record across a wide range of science, engineering, industry and commerce. In fact, many optimization problems come directly from real-world applications.

A simple search on Google with the keywords "optimization" and "application" will get numerous hits. Publications on optimization that do not mention applications are very rare. There is no need for us to prove the usefulness of optimization by presenting a long list of fields of application in which optimization has been involved. Space considerations also do not permit us to give an exhaustive and in-depth review on applications of optimization. Therefore, no further discussion on applications of optimization will be given here.

1.1.6 Optimization Algorithms

Optimization has long been the subject of intensive study. Numerous optimization algorithms have been proposed. In general, these algorithms can be divided into two major categories, deterministic and stochastic. Hybrid algorithms which combine deterministic and stochastic features are stochastic in essence and regarded as such. However, it is acceptable to treat them as a third category from the point of view of purity.

1.2 Deterministic Optimization Algorithms

A deterministic optimization algorithm will always get the same solution with the same number of objective function evaluations regardless of the time it is started, if the search space, starting-point, and termination conditions are unchanged. If the algorithm is run multiple times on the same computer, the search time for each run will be exactly the same. In other words, deterministic optimization is clonable.

Dimension is a good criterion for classifying deterministic optimization algorithms. Deterministic optimization algorithms are accordingly divided into one-dimensional and multi-dimensional deterministic optimization algorithms. Some multi-dimensional deterministic algorithms need the help of one-dimensional deterministic optimization algorithms.

1.2.1 One-Dimensional Deterministic Optimization Algorithms

Use of the derivative is a good choice for distinguishing one-dimensional optimization algorithms.

1.2.1.1 Zeroth-Order One-Dimensional Deterministic Optimization Algorithms

Zeroth-order algorithms involve objective function evaluation and comparison only. Prominent algorithms include the exhaustive search algorithm, dichotomous algorithms, the parabolic

interpolation algorithm, and the Brent algorithm. If the minimum of the objective function $f(x)$ is known, a nonlinear equation can be formulated. In this case, the Secant algorithm for nonlinear equations is applicable.

1.2.1.1.1 Exhaustive Search Algorithm The exhaustive search algorithm ([2], pp. 137–138) samples the search space $[a, b]$ at m points. Usually, the sample points are equally spaced within $[a, b]$. The minimum value of the objective function at each and every sample point is regarded as the optimum and the corresponding sample point is regarded as the optimal solution.

The exhaustive search algorithm is also known as enumeration algorithm or brute force algorithm ([3], p. 13).

1.2.1.1.2 Dichotomous Algorithms For minimizing the objective function $f(x)$, the Fortran-style pseudo-code for dichotomous algorithms ([2], p. 141) is shown in Figure 1.7.

```
n = 0
b_L = a
b_U = b
do while (termination conditions not satisfied)
  n = n + 1
  get y1 within [b_L, b_U]
  get y2 within [b_L, b_U]
  f1 = f(y1)
  f2 = f(y2)
  if (f2 < f1) then
    b_L = y1
  else
    b_U = y2
  end if
end do
```

Figure 1.7 Fortran-style pseudo-code for dichotomous algorithms

There are many different schemes for obtaining y_1 and y_2 within $[b_L, b_U]$, such as the equal interval, Fibonacci, and golden section schemes.

1.2.1.1.3 Parabolic Interpolation Algorithm A local quadratic approximation to the objective function $f(x)$ is useful because the minimum of a quadratic is easy to compute. The parabolic interpolation algorithm ([4], p. 185, [5], pp. 51, 72) interpolates the objective function by a quadratic polynomial which fits the objective function values at three points. The minimum point of the parabola, a new estimate of the minimum point of the objective function, will replace one of the three previous points. This process is repeated until termination conditions are fulfilled.

1.2.1.1.4 Brent Algorithm The Brent algorithm [6] is a hybrid of the parabolic interpolation algorithm and the golden section algorithm. The objective function in each iteration is approximated by an interpolating parabola through three existing points. The minimum point of the parabola is taken as a guess for the minimum point. It is accepted and used to generate a smaller interval if it lies within the bounds of the current interval. Otherwise, the algorithm falls back to an ordinary golden section step.

1.2.1.1.5 Secant Algorithm for Nonlinear Equation For an objective function $f(x)$ with known optimum, it is straightforward to formulate a nonlinear equation $f(x) = 0$. Its truncated Taylor series $f(x + h) \approx f(x) + f'(x)\, h$ is a linear function of h that approximates $f(x)$ near a given x. Assuming $f'(x) \neq 0$ and $f(x + h) = 0$, we get $h = -f(x)/f'(x)$. Because the truncated Taylor series is only an approximation to the nonlinear function $f(x)$, its root $x + h$ does not equal to the root of $f(x)$. Therefore, this process has to be repeated until an acceptable root is located. This motivates the update scheme of the Newton algorithm for nonlinear equation ([4], pp. 156–158) as $x^{n+1} = x^n - f(x^n)/f'(x^n)\, f'(x^n)$. However, the derivative $f'(x^n)$ may not be available. In this case, it is approximated by finite difference. The Newton update scheme accordingly becomes the Secant update scheme ([4], p. 158), expressed as

$$x^{n+1} = x^n - f(x^n)\frac{x^n - x^{n-1}}{f(x^n) - f(x^{n-1})}. \tag{1.8}$$

1.2.1.2 Higher-Order One-Dimensional Deterministic Optimization Algorithms

Besides objective function evaluation and comparison, higher-order algorithms directly make use of derivatives. Three algorithms in this category are commonly used.

1.2.1.2.1 Newton Algorithm Another way to obtain a local quadratic approximation to the objective function $f(x)$ is to use a truncated Taylor series expansion ([4], pp. 185–186)

$$f(x+h) \approx f(x) + f'(x)h + \frac{f''(x)}{2}h^2. \tag{1.9}$$

The minimum point of this quadratic function of h is given by

$$h = -\frac{f'(x)}{f''(x)}. \tag{1.10}$$

This result motivates the iteration scheme for the Newton algorithm

$$x^{n+1} = x^n - \frac{f'(x^n)}{f''(x^n)}. \tag{1.11}$$

Obviously, the Newton algorithm here is equivalent to the Newton algorithm for the nonlinear equation $f'(x) = 0$.

1.2.1.2.2 Secant Algorithm The Secant algorithm ([7], p. 55) here is equivalent to the Secant algorithm for nonlinear equation $f'(x) = 0$. Finite difference is applied to approximate the second-order derivative, $f''(x_k)$, in Equation 1.11. Its update scheme is

$$x^{n+1} = x^n - f'(x^n)\frac{x^n - x^{n-1}}{f'(x^n) - f'(x^{n-1})}. \tag{1.12}$$

1.2.1.2.3 Cubic Interpolation Algorithm This is another polynomial approximation algorithm in which the objective function $f(x)$ is approximated by a local third-order polynomial $p_3(x)$ ([7], pp. 56–58). The basic logic is similar to that of the parabolic interpolation algorithm. However, in this instance, evaluation of both objective function and its derivative at each point is required. Consequently, the approximation polynomial can be constructed using fewer points.

A third-order polynomial fitting the objective function $f(x)$ and its derivative $f'(x)$ at points a and b is given by $p_3(x) = c_0 + c_1(x-a) + c_2(x-a)(x-b) + c_3(x-a)^2(x-b)$, where $f(a) = c_0$, $f(b) = c_0 + c_1(b-a)$, $f'(a) = c_1 + c_2(a-b)$, $f'(b) = c_1 + c_2(b-a) + c_3(b-a)^2$. The minimum point of $p_3(x)$ is

$$x^* = \begin{cases} b & \lambda < 0, \\ b - \lambda(b-a) & 0 \le \lambda \le 1, \\ a & \lambda > 1. \end{cases} \tag{1.13}$$

where $\lambda = [f'(b) + \alpha - \beta]/[f'(b) - f'(a) + 2\alpha]$, $\alpha = \sqrt{\beta^2 - f'(a)f'(b)}$, $\beta = 3[f(a) - f(b)]/(b-a) + f'(a) + f'(b)$. Either a or b will be replaced by x^*, depending on both function values and the values of derivatives. The Fortran-style pseudo-code for one implementation is given in Figure 1.8 ([5], p. 82).

1.2.2 Multi-Dimensional Deterministic Optimization Algorithms

Similarly, the use of gradient, Hessian matrix or even higher-order partial derivatives is a good way to distinguish multi-dimensional optimization algorithms.

1.2.2.1 Zeroth-Order Multi-Dimensional Deterministic Optimization Algorithms

Likewise, zeroth-order algorithms here also involve objective function evaluation and comparison only. Prominent algorithms include the grid search algorithm, univariate search algorithms, the pattern search algorithm, Powell's conjugate direction algorithm, and the downhill simplex algorithm. If the minimum of the objective function $f(\mathbf{x})$ is known, a nonlinear equation can be formulated. In this case, Broyden's algorithm for nonlinear equation is applicable.

1.2.2.1.1 Grid Search Algorithm The grid search algorithm ([2], pp. 156–158) is the simplest algorithm for finding the minimum and the corresponding minimum solution point of objective function $f(\mathbf{x})$. D_i, the search space of x_i, is sampled at m_i points to form a grid of $\prod_{i=1}^{N} m_i$ points. The minimum value of the objective function at each and every grid point is regarded as the minimum and the corresponding grid point is regarded as the optimal

compute $f(a)$
compute $f(b)$
compute $f'(a)$
compute $f'(b)$
do while (termination conditions not satisfied)
$\beta = 3[f(a) - f(b)] / (b - a) + f'(a) + f'(b)$
$\alpha = \sqrt{\beta^2 - f'(a)f'(b)}$
$\lambda = [f'(b) + \alpha - \beta] / [f'(b) - f'(a) + 2\alpha]$

$$x^* = \begin{cases} a & \lambda < 0 \\ b - \lambda(b-a) & 0 \le \lambda \le 1 \\ b & \lambda > 1 \end{cases}$$

compute $f(x^*)$
compute $f'(x^*)$
if ($f'(a) < 0$ and ($f'(x^*) > 0$ or $f(x^*) \ge f(a)$)) then
b = x^*
$f(b) = f(x^*)$
$f'(b) = f'(x^*)$
else
a = x^*
$f(a) = f(x^*)$
$f'(a) = f'(x^*)$
end if
end do

Figure 1.8 Fortran-style pseudo-code for one implementation of the cubic interpolation algorithm

solution point. Obviously this approach soon becomes prohibitive as the dimension N and the number of sample points m_i for each dimension increase. A more efficient approach starts from a grid point, evaluates the objective function values at the surrounding $3^N - 1$ grid points, selects the grid point with the smallest objective function value as the new starting-point, and repeats this process until termination conditions are fulfilled.

1.2.2.1.2 Univariate Search Algorithms The guiding idea behind univariate search algorithms ([2], pp. 158–159) is to change one optimization parameter at a time so that the function is optimized in each of the coordinate directions $\mathbf{e}^i$, $1 \le i \le N$. The Fortran-style pseudo-code is given in Figure 1.9.

Univariate search algorithms are regarded as zeroth-order optimization algorithms for multi-dimensional optimization problems because the gradient of the objective function is not explicitly involved, although the derivative of $f(\mathbf{x}^{n-1} + \lambda \mathbf{e}^i)$ with respect to λ might be involved in the inner one-dimensional optimization algorithm.

```
choose starting point x^0
n = 1
do while (termination conditions not fulfilled)
  i = mod(n, N)
  if (i = 0) i = N
  find λ^n to minimize f(x^(n-1) + λ^n e^i)
  x^n = x^(n-1) + λ^n e^i
  n = n + 1
end do
```

Figure 1.9 Fortran-style pseudo-code for univariate search algorithms

1.2.2.1.3 Pattern Search Algorithm The pattern search algorithm is also known as the Hooke–Jeeves algorithm [8]. The essential idea behind it is to move from one solution point to the next.

There are two kinds of moves in the pattern search algorithm: the exploratory move and the pattern move. An exploratory move consists of a series of univariate searches. Each univariate search moves a little along its coordinate direction. The pattern move is larger in size.

Fortran-style pseudo-code is given in Figure 1.10.

```
choose starting point x
choose exploratory move steps δ_i, 1 ≤ i ≤ N
choose initial pattern move steps Δ_i, 1 ≤ i ≤ N
do while (termination conditions not fulfilled)
  do i = 1, N
    if( f(x + δ_i e^i) < f(x))
      x = x + δ_i e^i
      Δ_i = Δ_i + δ_i
    else if( f(x - δ_i e^i) < f(x))
      x = x - δ_i e^i
      Δ_i = Δ_i - δ_i
    end if
  end do
  if( f(x + Δ) < f(x))
    x = x + Δ
  end if
end do
```

Figure 1.10 Fortran-style pseudo-code for pattern search algorithms

1.2.2.1.4 Powell's Conjugate Direction Algorithms The univariate search algorithm is very appealing due to its simplicity. However, it may not converge to the optimal solution point in the case of non-decomposable objective functions. Powell proposed a simple but vastly superior variation. It gets the name Powell's conjugate direction algorithm ([2], pp. 162–163) because it chooses conjugate directions to move when applied to an objective function of quadratic form, $f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$, where $\cdot$ denotes the dot product of vectors. Directions $\mathbf{p}$ and $\mathbf{q}$ are conjugate if $\mathbf{p} \cdot \mathbf{A} \cdot \mathbf{q} = 0$.

The Fortran-style pseudo-code for Powell's conjugate direction algorithm is given in Figure 1.11.

```
choose starting point x^0
do i = 1, N
   p^i = e^i
end do
do while (termination conditions not fulfilled)
   do i = 1, N
      find λ^i to minimize f(x^(i-1) + λ_i p^i)
      x^i = x^(i-1) + λ_i p^i
   end do
   do i = 1, N – 1
      p^i = p^(i+1)
   end do
   p^N = x^N – x^0
   find λ to minimize f(x^N + λ (x^N – x^0))
   x^0 = x^N + λ (x^N – x^0)
end do
```

Figure 1.11 Fortran-style pseudo-code for powell's conjugate direction algorithm

1.2.2.1.5 Downhill Simplex Algorithm The downhill simplex algorithm for minimizing $f(\mathbf{x})$ is due to Nelder and Mead [9]. A non-degenerate N-dimensional simplex is a geometrical figure consisting of $N + 1$ distinct vertices. For example, a two-dimensional simplex is a triangle and three-dimensional simplex is a tetrahedron.

Suppose we have obtained an N-dimensional simplex with vertices $\mathbf{x}^i$, $i = 1, \ldots, N + 1$. Identify the vertices with the highest, second highest, and lowest objective function value, $\mathbf{x}^{\mathrm{H}}$, $\mathbf{x}^{\mathrm{S}}$, and $\mathbf{x}^{\mathrm{L}}$. Let $\mathbf{x}^{\mathrm{G}}$ be the geometric centroid of all the vertices except $\mathbf{x}^{\mathrm{H}}$, that is,

$$\mathbf{x}^{\mathrm{G}} = \frac{1}{n}\left(\sum_{i=1}^{N+1} \mathbf{x}^i - \mathbf{x}^{\mathrm{H}}\right). \tag{1.14}$$

Several key operations, including reflection, contraction, and expansion, are involved in determining a new vertex to replace $\mathbf{x}^{\mathrm{H}}$. If a vertex better than $\mathbf{x}^{\mathrm{H}}$ is not generated by all

these operations, the simplex will contract toward $\mathbf{x}^{\mathrm{L}}$ through a simultaneous contraction $\mathbf{x}^{i} = \mathbf{x}^{\mathrm{L}} + \eta\,(\mathbf{x}^{i} - \mathbf{x}^{\mathrm{L}})$ where $0 < \eta < 1$, $1 \leq i \leq N + 1$, $i \neq \mathrm{L}$, as shown in Figure 1.12.

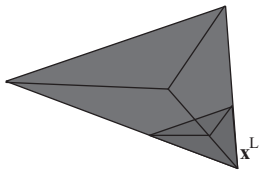

Figure 1.12 Simultaneous contraction

The mirror vertex $\mathbf{x}^{\mathrm{R}}$ of $\mathbf{x}^{\mathrm{H}}$ with respect to $\mathbf{x}^{\mathrm{G}}$ is obtained through reflection $\mathbf{x}^{\mathrm{R}} = \mathbf{x}^{\mathrm{G}} + \alpha\,(\mathbf{x}^{\mathrm{G}} - \mathbf{x}^{\mathrm{H}})$ where $\alpha > 0$. A schematic three-dimensional reflection is drawn in Figure 1.13.

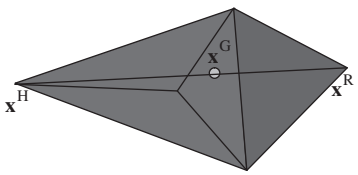

Figure 1.13 Reflection

An expansion as shown in Figure 1.14, $\mathbf{x}^{\mathrm{E}} = \mathbf{x}^{\mathrm{R}} + \beta\,(\mathbf{x}^{\mathrm{R}} - \mathbf{x}^{\mathrm{G}})$, where $\beta > 0$, will be made to search for better point if $f(\mathbf{x}^{\mathrm{R}}) < f(\mathbf{x}^{\mathrm{L}})$. $\mathbf{x}^{\mathrm{E}}$ will be accepted to replace $\mathbf{x}^{\mathrm{H}}$ if $f(\mathbf{x}^{\mathrm{E}}) < f(\mathbf{x}^{\mathrm{R}})$. Otherwise, $\mathbf{x}^{\mathrm{R}}$ will replace $\mathbf{x}^{\mathrm{H}}$.

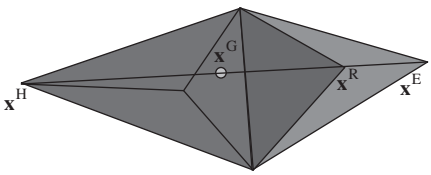

Figure 1.14 Expansion

$\mathbf{x}^{\mathrm{R}}$ will also replace $\mathbf{x}^{\mathrm{H}}$ if $f(\mathbf{x}^{\mathrm{R}}) < f(\mathbf{x}^{\mathrm{S}})$. Otherwise, a contraction as shown in Figure 1.15 is made to obtain a new point $\mathbf{x}^{\mathrm{C}} = \mathbf{x}^{\mathrm{H}} + \lambda\,(\mathbf{x}^{\mathrm{R}} - \mathbf{x}^{\mathrm{H}})$, where $0 < \lambda < 1$. $\mathbf{x}^{\mathrm{C}}$ will be accepted to replace $\mathbf{x}^{\mathrm{H}}$ if $f(\mathbf{x}^{\mathrm{C}}) < f(\mathbf{x}^{\mathrm{H}})$.

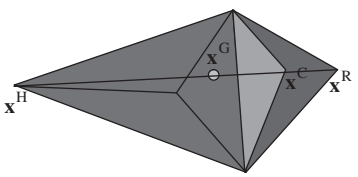

Figure 1.15 Contraction

The Fortran-style pseudo-code for the above downhill simplex algorithm is given in Figure 1.16.

```
set α, β, λ, and η
choose starting simplex x^i, 1 ≤ i ≤ N + 1
do while (termination conditions not fulfilled)
  get x^H, x^S, x^L, and x^G
  x^R = x^G + α (x^G − x^H)
  if (f(x^R) < f(x^L))
    x^E = x^R + β (x^R − x^G)
    if (f(x^E) < f(x^R))
      x^H = x^E
    else
      x^H = x^R
    end if
  else
    if (f(x^R) < f(x^S))
      x^H = x^R
    else
      x^C = x^H + λ (x^R − x^H)
      if (f(x^C) < f(x^H))
        x^H = x^C
      else
        do i = 1, N + 1
          if( i ≠ L) x^i = x^L + η (x^i − x^L)
        end do
      end if
    end if
  end if
end do
```

Figure 1.16 Fortran-style pseudo-code for downhill simplex algorithm

1.2.2.1.6 Broyden's Algorithm for Nonlinear Equation It is straightforward to generalize the one-dimensional Newton algorithm for nonlinear equations to multiple dimensions as $\mathbf{x}^{n+1} = \mathbf{x}^n + \mathbf{s}^n$, where $\mathbf{J}(\mathbf{x}^n) \cdot \mathbf{s}^n = -f(\mathbf{x}^n)$ and $J_i(\mathbf{x}^n) = \partial f(\mathbf{x}^n)/\partial x_i$ is the Jacobian (partial derivative) with respect to x_i. It is also straightforward to generalize the corresponding one-dimensional Secant algorithm for nonlinear equation to multiple dimensions ([4], pp. 167–168). One of the simplest and most effective Secant algorithms for multi-dimensional nonlinear equations is Broyden's algorithm, whose Fortran-style pseudo-code is given in Figure 1.17.

```
choose starting point x^0
choose starting Jacobian vector J^0
n = 0
do while (termination conditions not fulfilled)
   get s^n from J^n (x^n) · s^n = –f(x^n)
   x^(n+1) = x^n + s^n
   J^(n+1) = J^n + f(x^(n+1)) / (s^n · s^n) s^n
   n = n + 1
end do
```

Figure 1.17 Fortran-style pseudo-code for broyden's algorithm for multi-dimensional nonlinear equation

1.2.2.2 Higher-Order Multi-Dimensional Deterministic Optimization Algorithms

Besides objective function evaluation and comparison, higher-order algorithms make direct use of partial derivatives. Four algorithms in this category are commonly used.

1.2.2.2.1 Steepest Descent Algorithm It is well known that the negative gradient $-\nabla f(\mathbf{x})$ at a given point $\mathbf{x}$ is locally the steepest descent direction in the sense that the function value decreases more rapidly along it than along any other direction. This fact leads to one of the oldest algorithms for multi-dimensional optimization, the steepest descent algorithm ([2], pp. 166–168, [4], pp. 187–188, [5], pp. 100–101) whose update scheme is $\mathbf{x}^{n+1} = \mathbf{x}^n - \alpha^n \nabla f(\mathbf{x}^n)$, where the step α^n is usually obtained by applying any of the aforementioned one-dimensional minimization algorithms for the function $f(\mathbf{x}^n - \alpha \nabla f(\mathbf{x}^n))$ of α.

The steepest descent algorithm is also referred to as the Cauchy algorithm.

1.2.2.2.2 Newton Algorithm The Newton algorithm ([7], pp. 103–105, [4], pp. 189–190) for minimizing the multi-dimensional function $f(\mathbf{x})$ is a straightforward generalization of its one-dimensional counterpart. The straightforward generalization of the update scheme of the Newton algorithm for one-dimensional minimization problems would be $\mathbf{x}^{n+1} = \mathbf{x}^n - \mathbf{H}^{-1}(\mathbf{x}^n) \nabla f(\mathbf{x}^n)$, where $\mathbf{H}(\mathbf{x})$ is the Hessian matrix and $H_{ij}(\mathbf{x}) = \partial^2 f(\mathbf{x}) / \partial x_i \partial x_j$. However, to avoid matrix inversion, the update scheme is slightly modified as $\mathbf{x}^{n+1} = \mathbf{x}^n + \mathbf{s}^n$, where $\mathbf{H}(\mathbf{x}^n) \cdot \mathbf{s}^n = -\nabla f(\mathbf{x}^n)$.

Sometimes, the Newton algorithm is modified ([7], p. 105) by introducing a line search as $\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha^n \mathbf{s}^n$ where the step α^n is obtained by applying any of the aforementioned one-dimensional minimization algorithms for the function $f(\mathbf{x}^n + \alpha\, \mathbf{s}^n)$ of α.

Marquardt ([7], p. 105) even combines the steepest descent algorithm and the Newton algorithm as $\mathbf{x}^{n+1} = \mathbf{x}^n + \mathbf{s}^n$, where $[\mathbf{H}(\mathbf{x}^n) + \alpha^n \mathbf{I}] \cdot \mathbf{s}^n = -\nabla f(\mathbf{x}^n)$ and $\mathbf{I}$ is the identity matrix.

1.2.2.2.3 Quasi-Newton Algorithm Quasi-Newton algorithms ([7], pp. 112–114, [4], pp. 191–193, [5], pp. 151–157) are used to improve the reliability and reduce the overhead of the Newton algorithm. In general, an update scheme $\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha^n \mathbf{s}^n$ is used, where

$\mathbf{B}(\mathbf{x}^n) \cdot \mathbf{s}^n = -\nabla f(\mathbf{x}^n)$. Here $\mathbf{B}(\mathbf{x}^n)$ is an $N \times N$ matrix approximating $\mathbf{H}(\mathbf{x}^n)$. $\mathbf{B}(\mathbf{x}^n)$ is called a metric. Thus, they are often called variable metric algorithms. The Davidon–Fletcher–Powell (DFP) algorithm and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithms are two of the most famous quasi-Newton algorithms.

The Fortran-style pseudo-code for BFGS algorithm is given in Figure 1.18, where the symbol $\otimes$ stands for the outer or direct product of two vectors $\mathbf{u}$ and $\mathbf{v}$. $\mathbf{u} \otimes \mathbf{v}$ is a matrix. Its element $(\mathbf{u} \otimes \mathbf{v})_{ij}$ is $= u_i v_j$.

```
choose starting point x^0
choose starting Jacobian vector B^0
n = 0
do while (termination conditions not fulfilled)
  get s^n from B_n(x^n) · s^n = –∇f(x^n)
  x^(n+1) = x^n + α^n s^n
  y^n = ∇f(x^(n+1)) – ∇f(x^n)
  B^(n+1) = B^n + (y^n ⊗ y^n) / (y^n · s^n) – (B^n · s^n) ⊗ (B^n · s^n) / (s^n · B^n · s^n)
  n = n + 1
end do
```

Figure 1.18 Fortran-style pseudo-code for BFGS algorithm

1.2.2.2.4 Conjugate Gradient Method The conjugate gradient algorithm ([7], pp. 107–110, [4], pp. 193–195, [5], pp. 109–115) was initially proposed by Hestenes and Stiefel [10]. Later, Fletcher and Reeves [11] proved its quadratic convergence and extended it to nonquadratic functions. Therefore, it is also referred to as the Fletcher–Reeves algorithm ([2], pp. 169–173).

The conjugate gradient algorithm is another alternative to Newton's algorithm that does not require explicit second derivatives. In fact, the conjugate gradient does not even store an approximation to the Hessian matrix. As its name suggests, the conjugate gradient algorithm also uses the gradient, but it searches for the solution along a sequence of conjugate (i.e., orthogonal in some inner product) directions. The Fortran-style pseudo-code for the conjugate gradient algorithms is given in Figure 1.19.

1.2.3 Randomizing Deterministic Optimization Algorithms

Some deterministic algorithms can be randomized. Random multi-start is the simplest and most common approach. It is applicable to all deterministic algorithms requiring starting-points. The other approach is to randomize quantities such as the line search step. It applies to most of multi-dimensional deterministic optimization algorithms.

Some people regard randomized deterministic optimization algorithms as stochastic algorithms. This approach causes unnecessary confusion and should be abandoned. The original algorithms are strictly followed here unless explicitly stated otherwise.

choose starting point $\mathbf{x}^0$

$\mathbf{g}^0 = \nabla f(\mathbf{x}^0)$

$\mathbf{s}^0 = -\mathbf{g}^0$

$n = 0$

do while (termination conditions not fulfilled)

get step α^n by minimizing function $f(\mathbf{x}^n + \alpha^n \mathbf{s}^n)$ of α

$\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha^n \mathbf{s}^n$

$\mathbf{g}^{n+1} = \nabla f(\mathbf{x}^{n+1})$

$\mathbf{B}^{n+1} = \mathbf{B}^n + (\mathbf{g}^{n+1} \otimes \mathbf{g}^{n+1}) / (\mathbf{g}^n \cdot \mathbf{g}^n)$

$\mathbf{S}^{n+1} = -\mathbf{g}^{n+1} + \mathbf{B}^{n+1} \cdot \mathbf{s}^n$

$n = n + 1$

end do

Figure 1.19 Fortran-style pseudo-code for conjugate gradient algorithm

1.3 Stochastic Optimization Algorithms

1.3.1 Motivation

Deterministic optimization algorithms have undergone many decades of intensive development. Nowadays, many deterministic optimization algorithms have appeared in textbooks of different levels. Some of them have even been implemented as intrinsic subroutines in commercial software packages. They have enjoyed considerable success.

Nevertheless, deterministic optimization algorithms have been facing more and more serious challenges arising from diverse real-world applications. Failure cases have been rapidly accumulating. Their inherent weaknesses have been frequently exposed.

Most deterministic optimization algorithms are mathematically elegant. However, they are never user-friendly. A user may be required to provide not only the objective and constraint functions but also their derivatives. This requirement is invariably tedious and may prove prohibitive. Computation of derivatives may impose serious overhead. The overhead becomes even worse when computation of derivatives has to be done through approximation such as finite difference.

Most deterministic optimization algorithms imply excessive restrictions on optimization parameters, objective functions, and constraint functions. Most deterministic optimization algorithms apply to real continuous optimization parameters only. Convexity, continuity, and differentiability of objective and constraint functions are the most common implications assumed by deterministic optimization algorithms. Unfortunately, many real-world application problems do not satisfy even one of these assumptions.

It is obvious that most deterministic optimization algorithms demand one or more starting-points. Good starting-points are critical for the success of deterministic optimization algorithms. Poor starting-points may have a significant adverse effect on deterministic optimization algorithms' efficiency, or even cause them to fail.

Most often, good starting-points cannot be chosen by chance. A good choice relies heavily on a priori knowledge which may require years of experience. When such a priori knowledge is not available, people usually have to use trial and error to locate starting-points before starting the optimization process. The aforementioned random multi-start approach is the other choice for this problem.

Computational efficiency is the most promising feature of most deterministic optimization algorithms. However, as computer technology advances speedily with each passing day, people's expectations with regard to efficiency have risen significantly.

As such, in recent decades, people have increasingly turned their attention to stochastic optimization algorithms, especially the evolutionary algorithms. An exponential growth in the use of stochastic optimization has been seen.

1.3.2 Outstanding Features

Stochastic optimization algorithms have many interesting features. Some of these features are controversial. Nevertheless, stochastic optimization has been gaining more and more popularity and acceptance.

1.3.2.1 Randomness

As mentioned earlier, deterministic optimization is clonable. In contrast, as the name indicates, results obtained from a stochastic optimization algorithm are in general unpredictable due to randomness. In practice, one may never be able to get identical optimal solutions, although the solutions obtained may differ only very slightly.

However, from the point of view of practical application, two mathematically different results are regarded as identical if both of them meet the tolerance requirement imposed by the practical application.

A controversy accompanying stochastic optimization algorithms is their proof of absolute success, either theoretically or numerically. No stochastic optimization algorithm guarantees absolute success, although the failure percentage might be very small. A search by a stochastic optimization algorithm may miss the optimal solution. This constitutes a major challenge for the entire stochastic optimization community. Mathematicians working on it are a long way from a successful conclusion.

1.3.2.2 Simplicity

Stochastic optimization algorithms are in general mathematically simpler than deterministic algorithms. Usually, neither an exact nor an approximate derivative is involved. Most stochastic optimization algorithms generate their initial solution through an inherent initialization process, and thus avoid being troubled by choosing a starting-point. Relief from heavy reliance on trial and error or a priori knowledge in guessing starting-points is a tremendous advantage in the eyes of many optimization practitioners.

Another controversy as regards stochastic optimization algorithms is their rigorous mathematical foundation. Most of the stochastic optimization algorithms are inspired by natural phenomena which mankind has been struggling to understand.

Each stochastic optimization algorithm has at least one intrinsic control parameter. The performance of stochastic optimization algorithms more or less depends on these intrinsic control parameters. It is well known that tuning these intrinsic control parameters for better performance is usually very hard. In this sense, stochastic optimization algorithms are not as simple as people have believed.

1.3.2.3 Efficiency

Stochastic optimization algorithms usually require more objective function evaluations to find the optimal solution than deterministic optimization algorithms, given that both of them succeed. In other words, they are computationally more expensive or less efficient.

1.3.2.4 Robustness

This is the third controversy as far as stochastic optimization algorithms are concerned. Stochastic optimization algorithms may occasionally miss the optimal solution even if everything is favorable. On the other hand, stochastic optimization algorithms are usually capable of bracketing a quasi-optimal solution within a wide search space, while deterministic optimization algorithms usually fail to do so in the same situation. In most practical applications, a quasi-optimal solution is welcome. Very often, it is immediately accepted. In the event that it is not acceptable, certain measures can be taken to refine it.

1.3.2.5 Versatility

Most stochastic optimization algorithms do not impose restrictions on optimization problems. In addition, many stochastic optimization algorithms apply to discrete or even symbolic optimization parameters as well as real ones. In this regard, stochastic optimization optimizations are versatile.

1.3.3 Classification

Stochastic optimization algorithms can be divided into two major categories according to their origins: physical algorithms and evolutionary algorithms. Some people regard artificial neural networks and artificial immune systems as stochastic optimization algorithms. Indeed, they can be used to solve certain optimization problems. However, before solving the optimization problem at hand, training has to be carried out. These algorithms cannot accomplish the optimization by themselves without the help of training sets. For this reason, this author personally would not regard them as stochastic optimization algorithms.

Once again, we do not regard hybrids of two or more stochastic optimization algorithms as a separate category.

1.3.4 Physical Algorithms

Stochastic optimization algorithms in this category are inspired by physical phenomena. The Monte Carlo algorithm [12] and the simulated annealing algorithm [13,14] are two of the most prominent algorithms in this category.

1.3.4.1 Monte Carlo Algorithm

The Monte Carlo algorithm, named for a famous casino city in Monaco, relies on repeated random sampling to find the optimal solution. The use of randomness and the repetitive nature of the process are analogous to the activities conducted at a casino. It was originally practiced under more generic names such as statistical sampling. In the 1940s, physicists working on nuclear weapons projects at the Los Alamos National Laboratory coined the present name.

The Fortran-style pseudo-code for the Monte Carlo algorithm for minimizing $f(\mathbf{x})$ is given in Figure 1.20.

```
n = 1
do while (termination conditions not fulfilled)
    randomly generate x^n
    compute f(x^n)
    n = n + 1
 end do
```

Figure 1.20 Fortran-style pesudo-code for the Monte Carlo algorithm

1.3.4.2 Simulated Annealing Algorithm

The simulated annealing algorithm imitates the annealing process in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce defect. By analogy with this physical process, each step of the simulated annealing algorithm replaces the current solution by a new solution with a probability that depends on the difference of objective function values at the two solution points and the temperature. The most often implemented probability distribution function is the Boltzmann probability distribution which is $p(\mathbf{x}, \mathbf{y}, \mathrm{T}) = \exp\{-[f(\mathbf{y}) - f(\mathbf{x})]/(k\,T)\}$ where k is the Boltzmann constant.

The Fortran-style pseudo-code for a generic simulated annealing algorithm is given in Figure 1.21. Here $p(\mathbf{x}^{n-1}, \mathbf{y}^n, T^{n-1})$ is positive value when $f(\mathbf{y}^n) > f(\mathbf{x}^{n-1})$. This means that the new solution $\mathbf{y}^n$ may be accepted even if it is worse than $\mathbf{x}^{n-1}$. It is this feature that prevents the simulated annealing algorithm from being trapped in a locally minimum solution point.

1.4 Evolutionary Algorithms

Evolutionary algorithms were inspired by Darwin's theory of evolution. Natural selection is the foundation of Darwin's theory of evolution.

The study of evolutionary algorithms began in the 1960s. A number of creative researchers independently came up with the idea of mimicking the biological evolution mechanism and developed three mainstream evolutionary algorithms, namely, genetic algorithms [15–17], evolutionary programming [18,19], and evolution strategies [20,21]. Other evolutionary algorithms include memetic algorithms [22,23], scatter search ([22,24], [25], pp. 183–193, [26], pp. 519–537, [27]), self-organizing migrating [28], and tabu search [29–32].

```
generate x^0
set initial temperature T^0
n = 1
do while (termination conditions not fulfilled)
  get new solution y^n
  get f(y^n)
  if(p(x^{n-1}, y^n, T^{n-1}) > rand(0, 1))
    x^n = y^n
    f(x^n) = f(y^n)
  end if
  get new temperature T^n
  n = n + 1
end do
```

Figure 1.21 Fortran-style pseudo-code for simulated annealing algorithm

In recent decades, swarm algorithms [33] including ant colony optimization [22,34], bees algorithm [35], cultural algorithm [22,36], particle swarm optimization [22,37], have emerged within the evolutionary computation community. The swarm algorithms, as their name implies, imitate human or animal behaviors.

1.4.1 Evolutionary Terminologies

For the convenience of the following description, several essential terminologies frequently encountered in evolutionary computation are defined.

1.4.1.1 Gene

The gene is the basic building block of all evolutionary algorithms. There are usually two classes of genes: real, where a gene is a real number; and alphabetic, where a gene takes a value from an alphabet set. Common alphabet sets are the binary, octal, decimal, and hexadecimal sets.

1.4.1.2 Chromosome

The chromosome is another essential building block of all evolutionary algorithms. It is a symbolic representation of optimization parameters **x**.

Genes are concatenated into chromosomes in the following form

$$\mathbf{g} = \overbrace{g_{11} \cdots g_{1L_1}}^{x_1} \quad \overbrace{g_{21} \cdots g_{1L_2}}^{x_2} \quad \cdots \quad \overbrace{g_{N1} \cdots g_{NL_N}}^{x_N}$$

where g_{ij} is the jth gene representing x_i, and L_i is the number of genes in the substring for x_i.

A two-way mapping exists between an actual optimization parameter and the corresponding substring of genes in a chromosome. Such mapping is usually referred to as an encoding/decoding operation in evolutionary computation.

Prominent mappings include the following:

- Natural mapping. The optimization parameter itself is the gene, that is, $\mathbf{x} = \mathbf{g}$. In this case, optimization parameters and chromosome will be used interchangeably.
- Digitizer. Digitize an optimization parameter as follows

$$x_i \approx b_i^L + (b_i^U - b_i^L) \sum_{j=1}^{L_i} g_{ij} B_i^{-j}, \tag{1.15}$$

where B_i is the base of the alphabet set for x_i, $D_i = [b_i^L, b_i^U]$ is the search space of x_i.

1.4.1.3 Fitness

Fitness is the measure of goodness of a chromosome. It is directly related to objective and constraint function values through a scaling operation. It is not absolutely necessary for some evolutionary algorithms such as differential evolution.

1.4.1.4 Individual

In a real society, an individual is a living creature. However, in the community of evolutionary computation, an individual $\mathbf{p}$ is an aggregate of a chromosome $\mathbf{g}$, optimization parameter values $\mathbf{x}$, and objective function (including constraint) values $\mathbf{f}$. Additional attributes of an individual may include fitness value, generation, velocity, age, gender, and even memory.

The similarity of two individuals $\mathbf{p}$ and $\mathbf{q}$ is quantitatively defined as

$$s(\mathbf{p}, \mathbf{q}) = e(\mathbf{x}^p, \mathbf{x}^q) + \sum_{i=1}^{N_f} e[f_i(\mathbf{x}^p), f_i(\mathbf{x}^q)], \tag{1.16}$$

where

$$e(\mathbf{x}, \mathbf{y}) = \begin{cases} ||\mathbf{x}-\mathbf{y}||/\min(||\mathbf{x}||, ||\mathbf{y}||), & ||\mathbf{x}|| \neq 0 \cap ||\mathbf{y}|| \neq 0, \\ ||\mathbf{x}-\mathbf{y}||, & \text{otherwise}, \end{cases}$$

$$||\mathbf{x}|| = \sqrt{\sum_{j=1}^{N} x_j^2},$$

$$e(f, g) = \begin{cases} |f-g|/\min[|f|, |g|], & |f| \neq 0 \cap |g| \neq 0, \\ |f-g|, & \text{otherwise}. \end{cases}$$

To quantitatively compare two individuals, a logic dominance function is defined as

$$d(\mathbf{p},\mathbf{q}) = \begin{cases} \text{true}, & \begin{array}{l} \forall 1 \le i \le N_{o^{\min}} : o_i^{\min}(\mathbf{x}^p) \le o_i^{\min}(\mathbf{x}^q) \cap \\ \forall 1 \le i \le N_{o^{\max}} : o_i^{\max}(\mathbf{x}^p) \ge o_i^{\max}(\mathbf{x}^q) \cap \\ \forall 1 \le i \le N_{c^=} : |c_i^=(\mathbf{x}^p)| \le |c_i^=(\mathbf{x}^q)| \cap \\ \forall 1 \le i \le N_{c^-} : c_i^-(\mathbf{x}^p) \le c_i^-(\mathbf{x}^q) \cap \\ \forall 1 \le i \le N_{c^+} : c_i^+(\mathbf{x}^p) \ge c_i^+(\mathbf{x}^q), \end{array} \\ \text{false}, & \text{otherwise}. \end{cases} \tag{1.17}$$

1.4.1.5 Population

A population is a congregation of individuals. An important characteristic feature of a population is its age, expressed in terms of generation. It is the object all evolutionary algorithms work on. On average, statistically speaking, a later population is better than an earlier population.

The most important feature of a population $\mathbf{P}^n$ of generation n containing N_p individuals is its diversity which can be defined as

$$d(\mathbf{P}^n) = \min_{1 \le i \ne j \le N_p} s(\mathbf{p}^{n,i}, \mathbf{p}^{n,j}). \tag{1.18}$$

If the best individual of population $\mathbf{P}^n$, $\mathbf{p}^{n,\text{best}}$, and the worst individual, $\mathbf{p}^{n,\text{worst}}$, are identifiable, it is computationally more efficient to redefine the population diversity as

$$d(\mathbf{P}^n) = s(\mathbf{p}^{n,\text{best}}, \mathbf{p}^{n,\text{worst}}). \tag{1.19}$$

In practice, both diversity functions must be applied with care. Mathematically, the former definition is more reasonable. It is also more beneficial for diversity preservation. However, it is computationally much more expensive. On the other hand, the latter definition may overemphasize an individual's performance and discriminate an individual's genetic features.

1.4.2 Prominent Evolutionary Algorithms

1.4.2.1 Genetic Algorithms

Genetic algorithms, a class of search techniques inspired by evolutionary biology, were originated by J.H. Holland. The general flow chart of genetic algorithms is presented in Figure 1.22.

As seen in Figure 1.22, genetic algorithms involve two stages: initialization and evolution. Initialization generates an initial population $\mathbf{P}^0$. Then $\mathbf{P}^0$ evolves into $\mathbf{P}^1$, $\mathbf{P}^1$ evolves into $\mathbf{P}^2$, . . ., until the termination conditions are fulfilled. While evolving from $\mathbf{P}^n$ to $\mathbf{P}^{n+1}$, the three evolutionary operations – selection, crossover, and mutation – are executed in sequence.

Originally, optimization parameters were represented by binary chromosomes regardless of their actual types. Accordingly, crossover and mutation are Boolean operations. This applies to the following description unless otherwise explicitly stated. In addition, here, maximizing

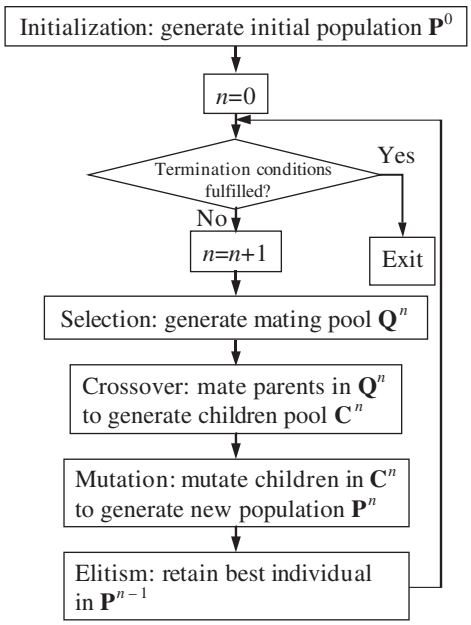

Figure 1.22 General flow chart of genetic algorithms

$f(\mathbf{x})$ is assumed for convenience and consistency with the original standard binary genetic algorithm. In this case, we use the objective function $f(\mathbf{x})$ directly as a fitness function.

1.4.2.1.1 Initialization Initialization generates an initial population $\mathbf{P}^0$ which contains N_p individuals $\mathbf{p}^{0,i}$, $1 \le i \le N_p$. Individual $\mathbf{p}^{0,i}$ is generated in three steps. A binary chromosome $\mathbf{g}^{0,i}$ is first generated. Then the binary chromosome is decoded to obtain the actual optimization parameter values $\mathbf{x}^{0,i}$. The optimization parameters are finally used to evaluate the objective function value $f(\mathbf{x}^{0,i})$. The Fortran-style pseudo pseudo-code for this process is given in Figure 1.23.

```
do i = 1, N_p
  do j = 1, N
    do k = 1, L_j
      g_{j,k}^{0,i} = int(2 * rand(0, 1))
    end do
    x_j^{0,i} = b_j^L + (b_j^U - b_j^L) Σ_{k=1}^{L_j} g_{j,k}^{0,i} 2^{-k}
  end do
  evaluate f(x^{0,i})
end do
```

Figure 1.23 Fortran-style pseudo-code for initialization of genetic algorithms

1.4.2.1.2 Selection The selection operator is believed to be responsible for the convergence of genetic algorithms. It selects good individuals on the basis of their fitness values and produces a temporary population, namely, the mating pool $\mathbf{Q}^n$ with member individuals $\mathbf{q}^{n,i}$. The mating pool is usually, but not necessarily, the same size as $\mathbf{P}^{n-1}$. This can be achieved by many different schemes, but the most common methods are roulette-wheel, ranking and tournament selection.

1.4.2.1.2.1 Roulette-Wheel Selection The essential idea behind roulette-wheel selection is that the probability of selection of an individual $\mathbf{p}^{n-1,i}$ in the parent population $\mathbf{P}^{n-1}$ is proportional to its fitness value, that is, $p^{n-1,i} \propto f(\mathbf{x}^{n-1,i}) / \sum_{i=1}^{N_p} f(\mathbf{x}^{n-1,i})$. It gets its name due to its analogy to the roulette-wheel in a casino.

1.4.2.1.2.2 Ranking Selection The idea of ranking selection is very simple. All individuals $\mathbf{p}^{n-1,i}$ in the parent population $\mathbf{P}^{n-1}$ are ranked according to their fitness. Individuals with higher rank (higher fitness value) are admitted into $\mathbf{Q}^n$. Usually, the better half of the individuals in $\mathbf{P}^{n-1}$ are copied and duplicated to fill $\mathbf{Q}^n$.

1.4.2.1.2.3 Tournament Selection Tournament selection implements a tournament to decide the membership of an individual $\mathbf{p}^{n-1,i}$ in parent population $\mathbf{P}^{n-1}$ in $\mathbf{Q}^n$. The winner of an m-participant tournament is selected. The Fortran-style pseudo-code for the binary, that is, two-participant, tournament selection procedure is given in Figure 1.24.

```
do i = 1, N_p
  j = int(N_p * rand(0, 1)) + 1
  k = int(N_p * rand(0, 1)) + 1
  do while (k = j)
    k = int(N_p * rand(0, 1)) + 1
  end do
  if(f(x^(n-1,j)) > f(x^(n-1,k)))
    q^(n,i) = p^(n-1,j)
  else
    q^(n,i) = p^(n-1,k)
  end if
end do
```

Figure 1.24 Fortran-style pseudo-code for binary tournament selection

1.4.2.1.3 Crossover The crossover operator has been believed to be the main search tool of genetic algorithms. It mates individuals $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$ in the mating pool by pairs and generates children by crossing over the mated pairs with probability p_c, one of the intrinsic control parameters of genetic algorithms. Analogously to human society, the two mating partners are defined as parents, among which individual $\mathbf{q}^{n,i}$ is the mother and individual $\mathbf{q}^{n,j}$ is the father. Usually, but not necessarily, every pair of parents delivers two children.

Many crossover schemes have been developed. Four of them are introduced here.

1.4.2.1.3.1 One-Point Crossover One-point crossover randomly selects a single crossover point r ($1 < r \leq N$). The two children, $\mathbf{c}^{n,a}$ and $\mathbf{c}^{n,b}$, are delivered by swapping genes of the two mating partners, $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$, as shown in Figure 1.25. More exactly, child $\mathbf{c}^{n,a}$ inherits genes of $\mathbf{q}^{n,i}$ to the left of the crossover point and genes of $\mathbf{q}^{n,j}$ from the crossover point (including) to the right end of the chromosomes. $\mathbf{c}^{n,b}$ inherits the remaining genes of $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$.

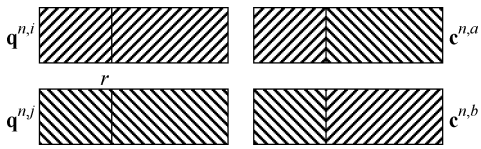

Figure 1.25 One-point crossover

1.4.2.1.3.2 Multi-Point Crossover In the multi-point crossover scheme, the chromosomes of both $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$ are cut into $m + 1$ partitions. Child $\mathbf{c}^{n,a}$ inherits odd-partition genes of $\mathbf{q}^{n,i}$ and even-partition genes of $\mathbf{q}^{n,j}$. Similarly, $\mathbf{c}^{n,b}$ inherits the remaining genes of $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$. An example of a two-point crossover ($m = 2$) is shown in Figure 1.26.

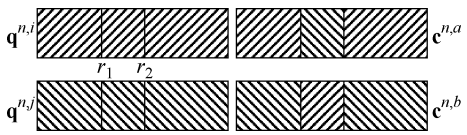

Figure 1.26 Two-point crossover

1.4.2.1.3.3 Binomial Crossover The Fortran-style pseudo-code for the binomial crossover scheme is given in Figure 1.27, where L is the chromosome length and p_c is the probability of crossover. One extreme case is that $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$ do not exchange any genes. In this case, they will be forced to exchange one gene at a randomly selected site.

```
do k = 1, L
  if(rand(0, 1) < p_c)
    g_k^{n, c, a} = g_k^{n, q, j}
    g_k^{n, c, b} = g_k^{n, q, i}
  else
    g_k^{n, c, a} = g_k^{n, q, i}
    g_k^{n, c, b} = g_k^{n, q, j}
  end if
end do
```

Figure 1.27 Fortran-style pseudo-code for binomial crossover

1.4.2.1.3.4 Exponential Crossover Exponential crossover is a cyclic two-point crossover. The Fortran-style pseudo-code is given in Figure 1.28, where p_e is the probability of exponential crossover which controls the crossover length. A possible crossover result is given in Figure 1.29, where $r + M > L$.

```
do k = 1, L
    g_k^{n,c,a} = g_k^{n,q,i}
    g_k^{n,c,b} = g_k^{n,q,j}
end do
M = 1
β = rand(0, 1)
do while (β ≤ p_c and M < L –2)
  M = M + 1
  β = rand(0, 1)
end do
r = L * rand(0, 1) + 1
do m = 0, M – 1
  k = mod(r + m, L)
  if (k = 0) k = L
  g_k^{n,c,a} = g_k^{n,q,j}
  g_k^{n,c,b} = g_k^{n,q,i}
end do
```

Figure 1.28 Fortran-style pseudo-code for exponential crossover

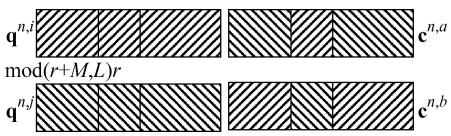

Figure 1.29 Exponential crossover

1.4.2.1.4 Mutation After crossover, some of the genes in the child chromosome are inverted with probability p_m, another intrinsic control parameter of genetic algorithms. The mutation operator is included to prevent premature convergence by ensuring the population diversity.

1.4.2.1.5 Elitism Obviously, there is no direct comparison between $\mathbf{P}^{n-1}$ and $\mathbf{P}^n$. All children are unconditionally admitted to $\mathbf{P}^n$ and all parents in $\mathbf{P}^{n-1}$ are unconditionally abandoned. Consequently, the best individual (in terms of fitness) in $\mathbf{P}^n$ may be worse than that in $\mathbf{P}^{n-1}$. In this case, elitism will be implemented by replacing the worst individual in $\mathbf{P}^n$ with the best individual in $\mathbf{P}^{n-1}$.

1.4.2.1.6 Real-Coded Genetic Algorithm The standard binary genetic algorithm encodes optimization parameters into binary chromosomes regardless of the nature of the optimization parameters. This makes the standard binary genetic algorithm flexible in that it can handle almost all kinds of optimization parameters. However, staircase error will inevitably be introduced when encoding a real number. The encoding and decoding operations also make the algorithm more computationally expensive for problems with real optimization parameters. A less often mentioned serious pitfall is the increased difficulty in controlling the algorithm due to the introduction of at least one more intrinsic control parameter, the chromosome length. Working directly on real optimization parameters can remove all these overheads. This leads to the real-coded genetic algorithm [38], in which natural real code is adopted, that is, $\mathbf{x} = \mathbf{g}$. Besides the aforementioned Boolean crossover operators, other schemes are also applicable. Arithmetic crossover is most often applied.

1.4.2.1.6.1 Arithmetic One-Point Crossover Arithmetic one-point crossover randomly selects a single crossover point r $(1 < i < N)$. Child $\mathbf{c}^{n,a}$ inherits genes of $\mathbf{q}^{n,i}$ to the left of the crossover point while $\mathbf{c}^{n,b}$ inherits genes of $\mathbf{q}^{n,j}$ to the left of the crossover point. The remaining genes of $\mathbf{c}^{n,a}$ and $\mathbf{c}^{n,b}$ are generated by arithmetic swapping

$$\left.\begin{aligned} x_k^{n,c,a} &= h x_k^{n,q,i} + (1-h) x_k^{n,q,j} \\ x_k^{n,c,b} &= (1-h) x_k^{n,q,i} + h x_k^{n,q,j} \end{aligned}\right\} \quad k \geq r. \tag{1.20}$$

where h is the arithmetic swapping intensity. An example of schematic crossover is shown in Figure 1.30.

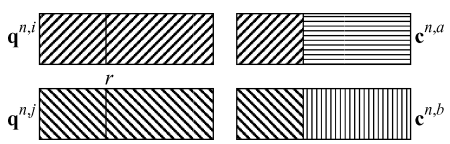

Figure 1.30 Arithmetic one-point crossover

1.4.2.1.6.2 Arithmetic Multi-Point Crossover In arithmetic multi-point crossover the optimization parameters of both $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$ are likewise cut into $m + 1$ partitions. Child $\mathbf{c}^{n,a}$ inherits odd-partition genes from $\mathbf{q}^{n,i}\mathbf{c}^{n,b}$ inherits odd-partition genes from $\mathbf{q}^{n,j}$. The remaining genes of $\mathbf{c}^{n,a}$ and $\mathbf{c}^{n,b}$ are generated by arithmetic swapping. As a example, an arithmetic two-point crossover $(m = 2)$ is shown in Figure 1.31.

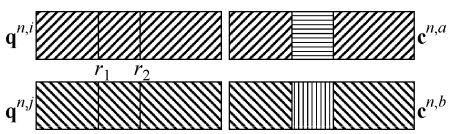

Figure 1.31 Arithmetic two-point crossover

1.4.2.1.6.3 Arithmetic Binomial Crossover The Fortran-style pseudo-code for arithmetic binomial crossover is given in Figure 1.32. One extreme case is where $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$ do not exchange any parameters. In this case, they will be forced to exchange one parameter at a randomly selected site.

```
do k = 1, N
  if(rand(0, 1) < p_c)
    x_k^{n, c, a} = h x_k^{n, q, i} + (1 – h) x_k^{n, q, j}
    x_k^{n, c, b} = (1 – h) x_k^{n, q, i} + h x_k^{n, q, j}
  else
    x_k^{n, c, a} = x_k^{n, q, i}
    x_k^{n, c, b} = x_k^{n, q, j}
  end if
end do
```

Figure 1.32 Fortran-style pseudo-code for arithmetic binomial crossover

1.4.2.1.6.4 Arithmetic Exponential Crossover The Fortran-style pseudo-code for arithmetic exponential crossover is given in Figure 1.33. The only difference with exponential crossover happens at lines 15 and 16 as highlighted. A possible crossover result is given in Figure 1.34, where $r + M > N$.

```
do k = 1, N
  x_k^{n, c, a} = x_k^{n, q, i}
  x_k^{n, c, b} = x_k^{n, q, j}
end do
M = 1
β = rand(0, 1)
do while (β ≤ p_c and M < N –2)
  M = M + 1
  β = rand(0, 1)
end do
r = N * rand(0, 1) + 1
do m = 0, M – 1
  k = mod(r + m, N)
  if (k = 0) k = N
    x_k^{n, c, a} = h x_k^{n, q, i} + (1 – h) x_k^{n, q, j}
    x_k^{n, c, b} = (1 – h) x_k^{n, q, i} + h x_k^{n, q, j}
end do
```

Figure 1.33 Fortran-style pseudo-code for arithmetic exponential crossover

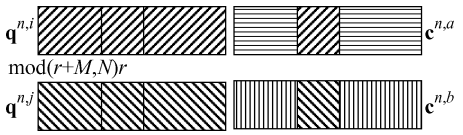

Figure 1.34 Arithmetic exponential crossover

1.4.2.1.6.5 Non-Uniform Arithmetic One-Point Crossover Non-uniform arithmetic one-point crossover randomly selects a single crossover point r ($1 < i < N$). Child $\mathbf{c}^{n,a}$ inherits genes of $\mathbf{q}^{n,i}$ to the left of the crossover point while $\mathbf{c}^{n,b}$ inherits genes of $\mathbf{q}^{n,j}$ to the left of the crossover point. Then, a crossover intensity, $h_k^{n,i}, k \geq r$, is randomly generated for each gene from the crossover point (including) to the right end of chromosomes. $h_k^{n,i}$ usually but not necessarily lies in [0,1]. Finally, the remaining genes of $\mathbf{c}^{n,a}$ and $\mathbf{c}^{n,b}$ are generated by non-uniform arithmetic swapping

$$\left.\begin{aligned} x_k^{n,i} &= h_k^{n,i} x_k^{n,q,i} + (1-h_k^{n,i}) x_k^{n,q,j} \\ x_k^{n,i+1} &= (1-h_k^{n,i}) x_j^{n,q,i} + h_k^{n,i} x_k^{n,q,j} \end{aligned}\right\} \quad k \geq r. \tag{1.21}$$

A schematic crossover result is shown in Figure 1.35.

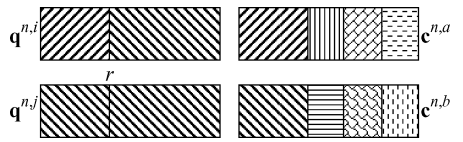

Figure 1.35 Non-uniform arithmetic one-point crossover

It is apparent that non-uniform arithmetic one-point crossover looks very similar with arithmetic one-point crossover. However, the crossover intensity $h_k^{n,i}$ for each child parameter $h_k^{n,i}$ to the right of the crossover point ($k \geq r$) is in general different. In this regard, it is non-uniform.

1.4.2.1.6.6 Non-Uniform Arithmetic Multi-Point Crossover In non-uniform arithmetic multi-point crossover the optimization parameters of both $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$ are likewise cut into $m + 1$ partitions. Child $\mathbf{c}^{n,a}$ inherits odd-partition genes from $\mathbf{q}^{n,i}$, while $\mathbf{c}^{n,b}$ inherits odd-partition genes from $\mathbf{q}^{n,j}$. The remaining genes of $\mathbf{c}^{n,a}$ and $\mathbf{c}^{n,b}$ are generated by non-uniform arithmetic swapping.

A schematic crossover result is shown in Figure 1.36.

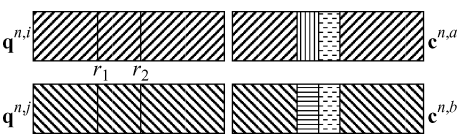

Figure 1.36 Non-uniform arithmetic multi-point crossover

1.4.2.1.6.7 Non-Uniform Arithmetic Binomial Crossover The Fortran-style pseudo-code for non-uniform arithmetic binomial crossover is given in Figure 1.37. The difference with arithmetic binomial crossover is highlighted.

```
do k = 1, N
  if(rand(0, 1) < p_c)
    h_k^{n,i} = rand(0, 1)
    x_k^{n,c,a} = h_k^{n,i} x_k^{n,q,i} + (1 − h_k^{n,i}) x_k^{n,q,j}
    x_k^{n,c,b} = (1 − h_k^{n,i}) x_k^{n,q,i} + h_k^{n,i} x_k^{n,q,j}
  else
    x_k^{n,c,a} = x_k^{n,q,i}
    x_k^{n,c,b} = x_k^{n,q,j}
  end if
end do
```

Figure 1.37 Fortran-style pseudo-code for non-uniform arithmetic binomial crossover

One extreme case is where $\mathbf{q}^{n,i}$ and $\mathbf{q}^{n,j}$ do not exchange any parameters. In this case, they will be forced to exchange one parameter at a randomly selected site.

1.4.2.1.6.8 Non-Uniform Arithmetic Exponential Crossover Non-uniform arithmetic exponential crossover is a cyclic two-point crossover. The Fortran-style pseudo-code is given in Figure 1.38, where the difference with arithmetic exponential crossover is highlighted.

A possible crossover result is given in Figure 1.39, where $r + M > N$. However, the Boolean inversion mutation operator is not applicable any more. Schemes working on real optimization parameters directly have to be implemented. Random perturbation mutation is most often applied.

Random perturbation mutation alters an optimization parameter of child $\mathbf{c}^{n,a}$ by adding a random perturbation term as $x_k^{n,c,a} = x_k^{n,c,a} + \alpha_k^{n,a} B_k$ where $-1 \leq \alpha_k{}^{n,a} \leq 1$ is a random number and B_k is the perturbation amplitude of the kth optimization parameter. B_k is usually no more than 10% of the corresponding search space.

1.4.2.2 Evolution Strategies

Evolution strategies were proposed by I. Rechenberg and H.P. Schwefel. There are two types: the (μ, λ)-strategy and the $(\mu + \lambda)$-strategy. Both generate a child population $\mathbf{C}^n$ of size λ through mutation from parent population $\mathbf{P}^{n-1}$ of size μ. The (μ, λ)-strategy generates $\mathbf{P}^n$ of size μ by selecting μ individuals from $\mathbf{C}^n$, while the $(\mu + \lambda)$-strategy generates $\mathbf{P}^n$ of size μ by selecting μ individuals from the union of $\mathbf{C}^n$ and $\mathbf{P}^{n-1}$. The general flow chart for evolution strategies is shown in Figure 1.40.

There are two notable differences with genetic algorithms. One is the reversed order of selection and mutation. The other one is the disappearance of crossover.

An individual in evolution strategies has an additional attribute: strategy parameters, that is, variances and covariances of a generalized N-dimensional normal distribution for mutation,

```
do k = 1, N
   x_k^{n, c, a} = x_k^{n, q, i}
   x_k^{n, c, b} = x_k^{n, q, j}
end do
M = 1
β = rand(0, 1)
do while (β ≤ p_c and M < N –2)
   M = M + 1
   β = rand(0, 1)
end do

r = N * rand(0, 1) + 1
do m = 0, M – 1
   k = mod(r + m, N)
   if (k = 0) k = N
      h_k^{n, i} = rand(0, 1)
      x_k^{n, c, a} = h_k^{n, i} x_k^{n, q, i} + (1 – h_k^{n, i}) x_k^{n, q, j}
      x_k^{n, c, b} = (1 – h_k^{n, i}) x_k^{n, q, i} + h_k^{n, i} x_k^{n, q, j}
end do
```

Figure 1.38 Fortran-style pseudo-code for non-uniform arithmetic exponential crossover

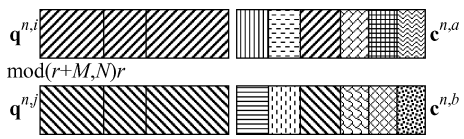

Figure 1.39 Non-uniform arithmetic exponential crossover

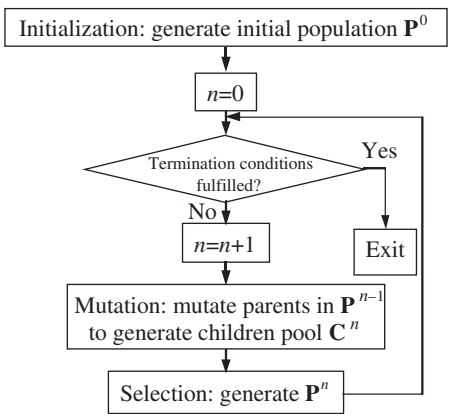

Figure 1.40 General flow chart of evolution strategies

$\mathbf{p} = (\mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\eta})$. $\boldsymbol{\sigma}$ is an N_σ-dimensional variance vector and $\boldsymbol{\eta}$ is an N_η-dimensional covariance vector, $\eta_j \in [-\pi, \pi]$, $1 \leq j \leq N_\eta$, $N_\sigma \geq 0$, $N_\eta \geq 0$.

N_σ and N_η play an essential role in mutation. Therefore, mutation schemes of evolution strategies are expressed as (N_σ, N_η)-mutation. In what follows we consider three representative mutation schemes.

1.4.2.2.1 (1, 0)-Mutation In this scheme, a child individual **c** is generated from its parent individual **p** as follows:

$$\begin{aligned} \sigma^c &= \sigma^p e^{\tau N(0,1)}, \\ x_j^c &= x_j^p + \sigma^c N_j(0,1), \end{aligned} \tag{1.22}$$

where $\tau \propto \sqrt{1/N}$, $N(0, 1)$ denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0, 1)$ is also a normally distributed one-dimensional random number with mean zero and standard deviation one but applies to σ_j^c only.

1.4.2.2.2 (N, 0)-Mutation In this scheme, a child individual **c** is generated from its parent individual **p** as follows:

$$\begin{aligned} \sigma_j^c &= \sigma_j^p e^{\tau N(0,1) + \tau' N_j(0,1)}, \\ x_j^c &= x_j^p + \sigma_j^c N_j(0,1), \end{aligned} \tag{1.23}$$

where $\tau \propto \sqrt{1/(2N)}, \tau' \propto \sqrt{1/(2\sqrt{N})}$, and $N(0, 1)$ applies to all σ_j^c, $1 \leq j \leq N$.

1.4.2.2.3 (N, N (N − 1)/2)-Mutation In this scheme, a child individual **c** is generated from its parent individual **p** as follows:

$$\begin{aligned} \sigma_j^c &= \sigma_j^p e^{\tau N(0,1) + \tau' N_j(0,1)}, \\ \eta_j^c &= \eta_j^p + \beta N_j(0,1), \\ x_j^c &= x_j^p + N_j(\mathbf{0}, \boldsymbol{\sigma}^c, \boldsymbol{\eta}^c), \end{aligned} \tag{1.24}$$

where $\tau \propto \sqrt{1/(2N)}, \tau' \propto \sqrt{1/(2\sqrt{N})}$, $\beta \approx 0.0873$, $N_j(\mathbf{0}, \boldsymbol{\sigma}, \boldsymbol{\eta})$ is the jth element of the N-dimensional correlated mutation vector $\mathbf{N}(\mathbf{0}, \boldsymbol{\sigma}, \boldsymbol{\eta})$ with mean zero, variances $\boldsymbol{\sigma}$, and covariances $\boldsymbol{\eta}$.

1.4.2.3 Evolutionary Programming

Evolutionary programming was proposed by L.J. Fogel. The original evolutionary programming was intended to operate on finite-state machines and the corresponding discrete representations. However, the present variants developed by L.J. Fogel are utilized for optimization problems with real optimization parameters. These variants have much in common with evolution strategies. A minor difference between them lies in selection. Each individual in the union of $\mathbf{C}^n$ and $\mathbf{P}^{n-1}$ is compared with q (>1) randomly chosen individuals, or opponents, which are also from the union. The individual is assigned a score which is the number of wins, or the number of individuals among its opponents dominated by it. The μ individuals with the highest scores survive.

1.4.2.4 Particle Swarm Optimization

Particle swarm optimization was introduced by Kennedy and Eberhart in 1995 [37]. As its name implies, it was inspired by the movement and intelligence of swarms. A swarm is a structured collection of interacting organisms such as bees, ants, or birds. Each organism in a swarm is a particle, or agent. Particles and swarms in particle swarm optimization are equivalent to individuals and populations in other evolutionary algorithms. The position, or site, of a particle in a swarm is the vector of optimization parameters $\mathbf{x}$ in particle swarm optimization.

Besides the attributes of an individual in other evolutionary algorithms, a particle in a swarm has two additional attributes: its velocity $\mathbf{v}^i$ and its memory $\mathbf{p}^{i,\,\text{best}}$ for the best site it has ever visited. Usually the velocity of $\mathbf{p}^{i,\,\text{best}}$ is not part of the memory. Accordingly, a swarm also has in its memory $\mathbf{p}^{\text{best}}$, the best site all the particles in the swarm have ever visited. Similarly, the velocity of $\mathbf{p}^{\text{best}}$ is not part of the memory.

Particles in a swarm cooperate by sharing knowledge. This has been shown to be the critical idea behind the success of the particle swarm optimization algoirthm. The block diagram for particle swarm optimization is shown in Figure 1.41.

The movement of a particle is accomplished in four steps: velocity update; position update; memory update; and swarm memory update.

1.4.2.4.1 Velocity Update The velocity $\mathbf{v}^{n,i}$ of particle $\mathbf{p}^{n,i}$ is updated by

$$v_j^{n+1,i} = wv_j^{n,i} + c_p\alpha_j(p_j^{i,\text{best}} - x_j^{n,i}) + c_s\beta_j(g_j^{\text{best}} - x_j^{n,i}), \tag{1.25}$$

where w, c_p, and c_s are the inertial weight, cognitive rate, and social rate which are intrinsic control parameters of particle swarm optimization.

1.4.2.4.2 Position Update The position $\mathbf{x}^{n,i}$ is updated by

$$x_j^{n+1,i} = x_j^{n,i} + \Delta t v_j^{n,i}, \tag{1.26}$$

where Δt is the time step. It usually is assumed 1 since it can always be absorbed by w, c_p, and c_s.

1.4.2.4.3 Memory Update The personal memory $\mathbf{p}^{i,\,\text{best}}$ of particle $\mathbf{p}^{n,i}$ will be updated if the updated position $\mathbf{x}^{n,i}$ outperforms the current best position $\mathbf{x}^{i,\,\text{best}}$ in its memory.

1.4.2.4.4 Swarm Memory Update The swarm memory $\mathbf{p}^{\text{best}}$ will be updated if the updated position of particle $\mathbf{p}^{n,i}$ outperforms the best position $\mathbf{x}^{\text{best}}$ in the swarm's memory.

1.4.3 Evolutionary Crimes

After years of involvement in the evolutionary computation community, this author has noticed various acts of misconduct within it. Some of these acts have led to serious misconceptions. By analogy with inverse crimes in the inverse problems community, this author refers these acts of misconduct as evolutionary crimes.

In fact, although it is embarrassing to admit, this author is both a criminal and a victim of evolutionary crimes.

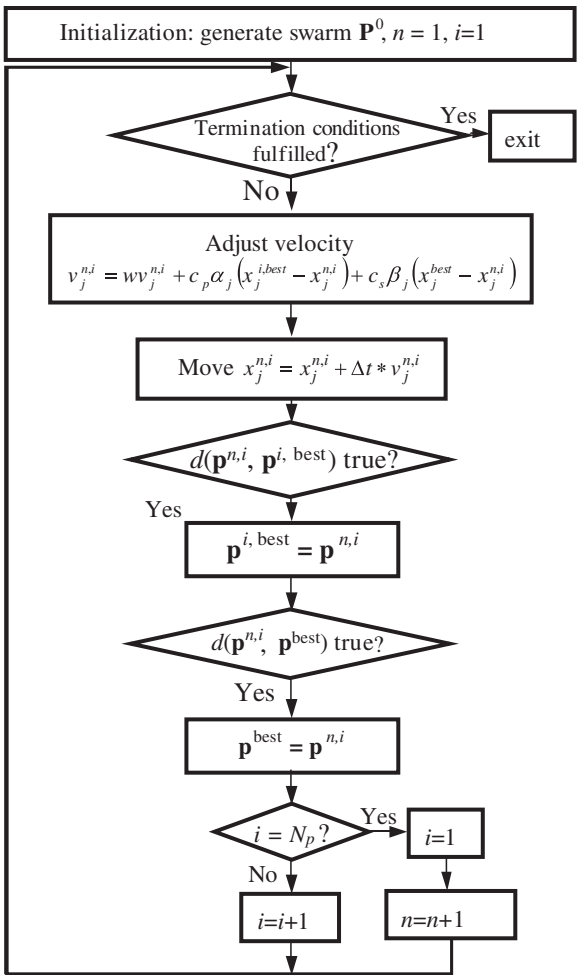

Figure 1.41 Block diagram for particle swarm optimization

1.4.3.1 Definition

An evolutionary crime is the exploitation of inadequate numerical evidence to claim advantage of an optimization algorithm over an evolutionary algorithm and/or the goodness of certain intrinsic control parameters of an evolutionary algorithm. Four kinds of evolutionary crime have been observed: ignorance of randomness (type A); baseless generalization (type B); biased comparison (type C); and transfer of problem (type D).

1.4.3.1.1 Ignorance of Randomness The stochastic nature of evolutionary algorithms means that the result of a single simulation run is much less trustworthy because of randomness. Instead, comparison should be based on averaged results over multiple runs. The higher the number of runs, the more trustworthy the average simulation result is.

Babu and co-workers reported two studies on intrinsic control parameters of differential evolution in 2000 ([39,40]). Evolutionary crime mentioned here is committed.

1.4.3.1.2 Baseless Generalization Many people have realized the sensitivity of differential evolution to its intrinsic control parameters. Therefore, they have carried out case studies on intrinsic control parameters. These case studies are conducted on very limited cases of intrinsic control parameters. The span of the intrinsic control parameters studied only partially covers the intrinsic control parameter domain. However, the researchers claim that the observed best intrinsic control parameters are the optimal intrinsic control parameters of differential evolution. Such a claim is unconvincing and constitutes an evolutionary crime.

Another baseless generalization is concerned with the test bed. Sometimes, the test bed implemented is not well designed. Problem features are not carefully taken care of. Only limited problem features are addressed. However, the final conclusion is unintentionally or purposely generalized.

1.4.3.1.3 Biased Comparison When a new optimization algorithm is developed, its proposer needs to prove its capability. Comparison with other existent optimization algorithms is the most common approach. This crime is usually evident when at least one evolutionary algorithm is chosen as competitor. It can be found throughout the evolutionary computation literature.

Every evolutionary algorithm has its specific intrinsic control parameters. For example, population size, mutation intensity, and crossover probability are the intrinsic control parameters for some strategies of differential evolution, while population size, crossover probability, and mutation probability are the three intrinsic control parameters for the real-coded genetic algorithm.

It is well known that the performance of an evolutionary algorithm is more or less sensitive to its intrinsic control parameters. Inappropriate choice of intrinsic control parameter values may make its performance deteriorate significantly or even cause it to fail.

Bias is due to fixing intrinsic control parameters of the competing evolutionary algorithms so as claim that the new optimization algorithm has advantages over competing evolutionary algorithms. It is appropriate to claim advantage only if the best intrinsic control parameters of the competing evolutionary algorithm are known. In this case, the new optimization algorithm would be remarkable because it absolutely outperforms its competitors. Unfortunately, very often, the best intrinsic control parameter values and the corresponding best performance of the competing evolutionary algorithms are not known.

In fact, bias, and thus evolutionary crime, may still be evident even if more sets of intrinsic control parameter values are investigated, if the sets are not carefully chosen.

1.4.3.1.4 Transfer of Problem This crime has been committed by most practitioners of evolutionary algorithms working on adaptation of intrinsic control parameters. They have realized the sensitivity of evolutionary algorithms to their intrinsic control parameters and the difficulty of choosing optimal intrinsic control parameters which implies that they do not know the best intrinsic control parameter values of the evolutionary algorithms concerned. They have made various proposals to adapt the intrinsic control parameters and usually claimed performance improvement over the original evolutionary algorithm. However, these proposals have their own specific intrinsic control parameters. Therefore, the problem changes merely from choosing optimal intrinsic control parameters for the evolutionary algorithms

concerned to choosing intrinsic control parameters for a specific proposal to adapt those intrinsic control parameters. In the worst scenario, a proposal to adapt intrinsic control parameters for an evolutionary algorithm introduces more intrinsic control parameters and builds a hierarchy.

1.4.3.2 Damages

Evolutionary crimes have caused serious damages, most notoriously the overestimation of crossover and the underestimation of mutation in genetic algorithms. One aspect of the damage to differential evolution is the three misconceptions widely circulated within the differential evolution community.

1.4.3.2.1 There is No Dramatic Difference in Performance Between the One- and Two-Array Methods More exactly, the one-array method is the classic differential evolution while the two-array method is its dynamic counterpart. We will show later that dynamic differential evolution significantly outperforms classic differential evolution.

1.4.3.2.2 DE/best/1/* is More Prone to Being Trapped in a Local Optimum than DE/rand/1/* Indeed, fixing the best individual in a population as the base for mutant generation is greedier than using a random base. However, differential evolution has its own balance mechanism.

In fact, this misconception is against the secular administration. It is a kind of anarchism in differential evolution. At all levels of secular administration, a leader guides the whole community. A good national leader will bring stability and prosperity to the whole country while anarchism brings disastrous suffering in the form of poverty, starvation, or even war.

1.4.3.2.3 Crossover is Not so Important The role of crossover in differential evolution has been underestimated since its inception. The success of the innovative idea of differential mutation may be to blame for this conception. However, evolutionary crime is definitely responsible for it too.

1.4.3.3 Remedies

Evolutionary crimes have caused huge damage to the evolutionary computation community. It is expected to cause more damage, at least in the near future, since many evolutionary computation practitioners have not even realized that they are committing evolutionary crimes. It is therefore imperative to promote the concept within the whole community. Awareness of evolutionary crimes is only the first step. Measures have to be taken to avoid future commitment.

1.4.3.3.1 Application Instead of Comparison It is understandable that the computational cost of multiple runs may not be affordable, especially for computationally expensive application problems. In this case, most often, the first priority is to prove the potential of the evolutionary algorithm concerned by successfully capturing the solution of the problem. Hence, it may not be absolutely necessary to compare the performance of the evolutionary algorithm concerned with other optimization algorithms, which requires averaged performance over multiple runs.

1.4.3.3.2 Comprehensive Parametric Study A proposer of a new optimization algorithm might want to claim advantages over its competing evolutionary algorithms. In this case, besides multiple runs, a comprehensive parametric study of intrinsic control parameters of at least the competitors has to be conducted to locate the best intrinsic control parameter values and the corresponding best performance.

As already observed, combinations of intrinsic control parameters are infinite. It is computationally neither practical nor necessary to try all combinations of intrinsic control parameters. A more reasonable approach is to replace the infinite set of intrinsic control parameters with a finite set containing representative values of intrinsic control parameters.

References

[1] Salomon, R. (1996) Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions: a survey of some theoretical and practical aspects of genetic algorithms. *Bio Systems*, **39**(3), 263–278.

[2] Cooper, L. and Steinberg, D. (1970) *Introduction to Methods of Optimization*, W.B. Saunders, Philadelphia.

[3] Price, K.V., Storn, R.M. and Lampinen, J.A. (2005) *Differential Evolution: A Practical Approach to Global Optimization*, Springer, Berlin.

[4] Heath, M.T. (1997) *Scientific Computing: An Introductory Survey*, McGraw-Hill, Boston.

[5] Joshi, M.C. and Moudgalya, K.M. (2004) *Optimization: Theory and Practice*, Alpha Science International, Harrow.

[6] Brent, R.P. (1973) *Algorithms for Minimization without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ.

[7] Reklaitis, G.V., Ravindran, A. and Ragsdell, K.M. (1983) *Engineering Optimization: Methods and Applications*, John Wiley & Sons, Inc., New York.

[8] Hooke, R. and Jeeves, T.A. (1966) Direct search of numerical and statistical problems. *Journal of the ACM*, **8**, 212–229.

[9] Nelder, J.N. and Mead, R. (1964) A simplex method for function minimization. *Computer Journal*, **7**(4), 308–313.

[10] Hestenes, M.R. and Stiefel, E. (1952) Methods of conjugate gradients for solving linear systems. *NBS Research Journal*, **49**, 409–436.

[11] Fletcher, R. and Reeves, C.M. (1964) Function minimization by conjugate grdients. *Computer Journal*, **7**(4), 149–154.

[12] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E. (1953) Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**(6), 1087–1092.

[13] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) Optimization by simulated annealing. *Science*, **220**(4598), 671–680.

[14] Davis, L. (ed.) (1987) *Genetic Algorithms and Simulated Annealing*, Pitman, London.

[15] Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

[16] Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

[17] Davis, L. (ed.) (1991) *The Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

[18] Fogel, L.J. (1962) Autonomous automata. *Industrial Research*, **4**, 14–19.

[19] Fogel, L.J., Owens, A.J. and Walsh, M.J. (1966) *Artificial Intelligence through Simulated Evolution*, John Wiley & Sons, Inc., New York.

[20] Schwefel, H.P. (1995) *Evolution and Optimum Seeking*, John Wiley & Sons, Inc., New York.

[21] Beyer, H.G. and Schwefel, H.P. (2002) Evolution strategy – a comprehensive introduction. *Natural Computing*, **1**(1), 3–52.

[22] Corne, D., Dorigo, M. and Glover, F. (eds) (1999) *New Ideas in Optimization*, McGraw-Hill, London.

[23] Caorsi, S., Massa, A., Pastorino, M. and Randazzo, A. (2004) Detection of PEC elliptic cylinders by a memetic algorithm using real data. *Microwave and Optical Technology Letters*, **43**(4), 271–273.

[24] Glover, F., Laguna, M. and Marti, R. (2000) Fundamentals of scatter search and path relinking. *Control Cybernetics*, **39**(3), 653–684.

[25] Pardalos, P.M. and Resende, M.G.C. (eds) (2002) *Handbook of Applied Optimization*, Oxford University Press, Oxford.

[26] Ghosh, A. and Tsutsui, S. (eds) (2003) *Advances in Evolutionary Computation: Theory and Applications*, Springer, New York.
[27] Laguna, M. and Marti, R. (2003) *Scatter Search: Methodology and Implementations in C*, Kluwer Academic, Boston.
[28] Onwubolu, G.C. and Babu, B.V. (eds) (2004) *New Optimization Techniques in Engineering*, Springer, Berlin.
[29] Glover, F. (1989) Tabu search, part I. *ORSA Journal on Computing*, **1**(3), 190–206.
[30] Glover, F. (1990) Tabu search, part II. *ORSA Journal on Computing*, **2**(1), 4–32.
[31] Cvijovic, D. and Klinowski, J. (1995) Taboo search – an approach to the multiple minima problem. *Science*, **267**, 664–666.
[32] Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer, Norwell, MA.
[33] Kennedy, J., Eberhart, R.C. and Shi, Y. (2001) *Swarm Intelligence*, Morgan Kaufmann, San Francisco.
[34] Dorigo, M. (1992) Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy.
[35] Pham, D.T., Ghanbarzadeh, A., Koc, E. *et al.* (2006) The bees algorithm – a novel tool for complex optimisation problems, in *Intelligent Production Machines and Systems: 2nd I*PROMS Virtual Conference* (eds. D.T. Pham, E.E. Eldukhri and A.J. Soroka), Elsevier, Oxford, pp. 454–461.
[36] Reynolds, R.G. (1994) An introduction to cultural algorithms, in *Proceedings of the Third Annual Conference on Evolutionary Programming*, World Scientific, Singapore, pp. 131–139.
[37] Kennedy, J. and Eberhart, R.C. (1995) Particle swarm optimization. IEEE International Conference on Neural Networks, Perth, WA, 27 Nov.–1 Dec. 1995, vol. 4, pp. 1942–1948.
[38] Qing, A., Lee, C.K. and Jen, L. (2001) Electromagnetic inverse scattering of two-dimensional perfectly conducting objects by real-coded genetic algorithm. *IEEE Transactions on Geoscience and Remote. Sensing*, **39**(3), 665–676.
[39] Babu, B.V. and Chaturvedi, G. (2000) Evolutionary computation strategy for optimization of an alkylation reaction. International Symposium 53rd Annual Session IIChE, Science City, Calcutta, December 18–21.
[40] Babu, B.V. and Munawar, S.A. (2000) Differential evolution for the optimal design of heat exchangers. All India Seminar Chemical Engineering Progress Resource Development: A Vision 2010 Beyond, Orissa State Centre Bhuvaneshwar, March 13.