From: Differential Evolution

Stan, Price

1.1 Introduction to Parameter Optimization

1.1.1 Overview

In simple terms, optimization is the attempt to maximize a system's desirable properties while simultaneously minimizing its undesirable characteristics. What these properties are and how effectively they can be improved depends on the problem at hand. Tuning a radio, for example, is an attempt to minimize the distortion in a radio station's signal. Mathematically, the property to be minimized, distortion, can be defined as a function of the tuning knob angle, x:

$$f(x) = \frac{\text{noise power}}{\text{signal power}}. \tag{1.1}$$

Because their most extreme value represents the optimization goal, functions like Eq. 1.1 are called *objective functions*. When its minimum is sought, the objective function is often referred to as a *cost function*. In the special case where the minimum being sought is zero, the objective function is sometimes known as an *error function*. By contrast, functions that describe properties to be maximized are commonly referred to as *fitness functions*. Since changing the sign of an objective function transforms its maxima into minima, there is no generality lost by restricting the following discussion to function minimization only.

Tuning a radio involves a single variable, but properties of more complex systems typically depend on more than one variable. In general, the objective function, $f(\mathbf{x}) = f(x_0, x_1, \ldots, x_{D-1})$, has D parameters that influence the property being optimized. There is no unique way to classify objective functions, but some of the objective function attributes that affect an optimizer's performance are:

- *Parameter quantization.* Are the objective function's variables continuous, discrete, or do they belong to a finite set? Additionally, are all variables of the same type?

Parameter dependence. Can the objective function's parameters be optimized independently (separable function), or does the minimum of one or more parameters depend on the value of one or more other parameters (parameter dependent function)?
Dimensionality, D. How many variables define the objective function?
Modality. Does the objective function have just one local minimum (uni-modal) or more than one (multi-modal)?
Time dependency. Is the location of optimum stationary (e.g., static), or non-stationary (dynamic)?
Noise. Does evaluating the same vector give the same result every time (no noise), or does it fluctuate (noisy)?
Constraints. Is the function unconstrained, or is it subject to additional equality and/or inequality constraints?
Differentiability. Is the objective function differentiable at all points of interest?

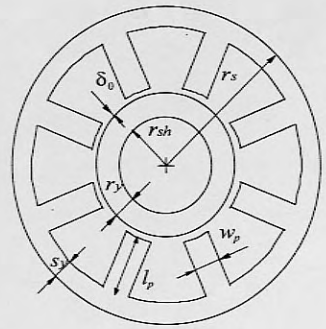
In the radio example, the tuning angle is real-valued and parameters are ontinuous. Neither mixed-variable types, nor parameter dependence is an ssue because the objective function's dimension is one, i.e., it depends on single parameter. The objective function's modality, however, depends n how the tuning knob angle is constrained. If tuning is restricted to the icinity of a single radio station, then the objective function is *uni-modal* ecause it exhibits just one (local) optimum. If, however, the tuning knob cans a wider radio band, then there will probably be several stations. If he goal is to find the station with least distortion, then the problem beomes *multi-modal*. If the radio station frequency does not drift, then the bjective function is not time dependent, i.e., the knob position that yields he best reception will be the same no matter when the radio is turned on. n the real world, the objective function itself will have some added noise, ut the knob angle will not be noisy unless the radio is placed on some viorating device like a washing machine. The objective function has no obious constraints, but the knob-angle parameter is certainly restricted.

Even though distortion's definition (Eq. 1.1) provides a mathematical lescription of the property being minimized, there is no computable objecive function – short of simulating the radio's circuits – to determine the listortion for a given knob angle. The only way to estimate the distortion at a given frequency is to tune in to it and listen. Instead of a well-defined, computable objective function, the radio itself is the "black box" that transforms the input (knob angle) into output (station signal). Without an adequate computer simulation (or a sufficiently refined actuator), the ob-

Tuning a radio is a trivial exercise primarily because it involves a single parameter. Most real-world problems are characterized by partially non-differentiable, nonlinear, multi-modal objective functions, defined with both continuous and discrete parameters and upon which additional constraints have been placed. Below are three examples of challenging, real-world engineering problems of the type that DE was designed to solve. Chapter 7 explores a wide range of applications in detail.

Optimization of Radial Active Magnetic Bearings

The goal of this electrical/mechanical engineering task is to maximize the bearing force of a radial active magnetic bearing while simultaneously minimizing its mass (Štumberger et al. 2000). As Fig. 1.1 shows, several constraints must be taken into account.



Objectives:
- maximal bearing force
- minimum mass

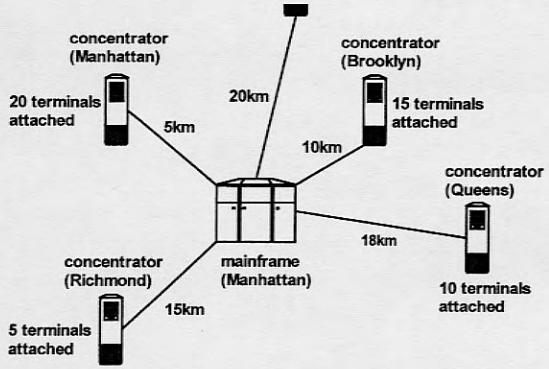
Parameters:
- stator yoke $s_y > 0$
- rotor yoke width $r_y > 0$
- pole width $w_p > 0$
- axial length $l > 0$

Constraints:
- air gap $\delta_0 = 0.4$mm
- stator radius $r_s = 52.5$mm
- shaft radius $r_{sh} = 35$mm
- $r_s = r_{sh} + r_y + \delta_0 + l_p + s_y$

Fig. 1.1. Optimizing a radial active magnetic bearing

Capacity Assignment Problem

Figure 1.2 shows a computer network that connects terminals to concentrators, which in turn connect to a large mainframe computer. The cost of a line depends nonlinearly on the capacity. The goal is to satisfy the data delay constraint of 4 ms while minimizing the cost of the network. A more detailed discussion appears in Schwartz (1977).



Objectives:
- **minimize network cost**

Parameters:
- **line capacities**

Constraints:
- **average data delay between terminals < 4s**
- **line capacities > 0**
- **cost of line nonlinearly depending on capacity**
- **Terminals transmit at 64kbps on average**
- **Average message length is 1000 bits long**

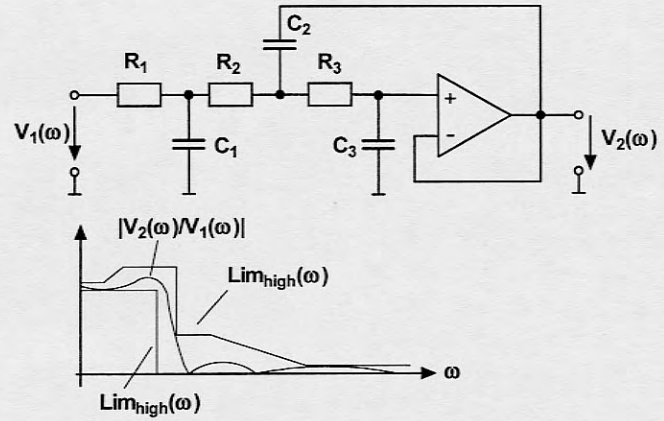
Fig. 1.2. Optimizing a computer network

Filter Design Problem

The goal here is to design an electronic filter consisting of resistors, capacitors and an operational amplifier so that the magnitude of the ratio of output to input voltages, $|V_2(\omega)/V_1(\omega)|$ (a function of frequency ω), satisfies the tolerance scheme depicted in the lower half of Fig. 1.3.

Classifying Optimizers

Once a task has been transformed into an objective function minimization problem, the next step is to choose an appropriate optimizer. Table 1.1 classifies optimizers based, in part, on the number of points (vectors) that they track through the D-dimensional problem space. This classification does not distinguish between multi-point optimizers that operate on many points in parallel and multi-start algorithms that visit many points in sequence. The second criterion in Table 1.1 classifies algorithms by their reliance on objective function derivatives.



Objectives:
- **Fit $|V_2(\omega)/V_1(\omega)|$ between $Lim_{high}(\omega)$ and $Lim_{low}(\omega)$**

Parameters:
- **Resistors R_i, Capacitors C_i**

Constraints:
- **$0 < C_i < C_{max}$**
- **$0 < R_i < R_{max}$**
- **R_i, C_i from E24 norm series (discrete set)**

Fig. 1.3. Optimizing an electronic filter

Table 1.1. A classification of optimization approaches and some of their representatives

	Single-point	Multi-point
Derivative-based	Steepest descent Conjugate gradient Quasi-Newton	Multi-start and clustering techniques
Derivative-free (direct search)	Random walk Hooke–Jeeves	Nelder–Mead Evolutionary algorithms Differential evolution

Not all optimizers neatly fit into these categories. Simulated annealing (Kirkpartick et al. 1983; Press et al. 1992) does not appear in this classification scheme because it is a meta-strategy that can be applied to any derivative-free search method. Similarly, clustering techniques are general strategies, but because they are usually combined with derivative-based optimizers (Janka 1999) they have been assigned to the derivative-based, multi-point category. As Table 1.1 indicates, differential evolution (DE) is a multi-point, derivative-free optimizer.

The following section outlines some of the traditional optimization algorithms that motivated DE's development. Methods from each class in Table 1.1 are discussed, but their many variants and the existence of other novel methods (Corne et al. 1999; Onwubolu and Babu 2004) make it impossible to survey all techniques. The following discussion is primarily focused on optimizers designed for objective functions with continuous and/or discrete parameters. With a few exceptions, combinatorial optimization problems are not considered.

1.1.2 Single-Point, Derivative-Based Optimization

Derivative-based methods embody the classical approach to optimization. Before elaborating, a few details on notation are in order. First, a D-dimensional parameter vector is defined as:

$$\mathbf{x}=\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{D-1} \end{pmatrix}=\begin{pmatrix} x_0 & x_1 & \dots & x_{D-1} \end{pmatrix}^T. \tag{1.2}$$

Letters in lowercase italic symbolize individual parameters; bold lowercase letters denote vectors, while bold uppercase letters represent matrices. Introducing several special operator symbols further simplifies formulation of the classical approach. For example, the *nabla* operator is defined as

$$\nabla=\begin{pmatrix} \partial/\partial x_0 \\ \partial/\partial x_1 \\ \dots \\ \partial/\partial x_{D-1} \end{pmatrix} \tag{1.3}$$

in order to simplify the expression for the *gradient vector*:

$$\mathbf{g}(\mathbf{x})=\nabla\cdot f(\mathbf{x})=\begin{pmatrix} \dfrac{\partial f(\mathbf{x})}{\partial x_0} \\ \dfrac{\partial f(\mathbf{x})}{\partial x_1} \\ \dots \\ \dfrac{\partial f(\mathbf{x})}{\partial x_{D-1}} \end{pmatrix}. \tag{1.4}$$

It is also convenient to define the *Hessian matrix*:

$$\mathbf{G}(\mathbf{x})=\nabla^2\cdot f(\mathbf{x})=\begin{pmatrix} \partial f/\partial x_0\partial x_0 & \partial f/\partial x_0\partial x_1 & \dots & \partial f/\partial x_0\partial x_{D-1} \\ \partial f/\partial x_1\partial x_0 & & & \\ \dots & & & \\ \partial f/\partial x_{D-1}\partial x_0 & \dots & \dots & \partial f/\partial x_{D-1}\partial x_{D-1} \end{pmatrix}. \tag{1.5}$$

The symbol ∇^2 is meant to imply second-order (partial) differentiation, not that the nabla operator, ∇, is squared.

Using these notational conveniences, the Taylor series for an arbitrary objective function becomes

$$\begin{aligned} f(\mathbf{x}) &= f(\mathbf{x}_0)+\frac{\nabla f(\mathbf{x}_0)}{1!}\cdot(\mathbf{x}-\mathbf{x}_0)+(\mathbf{x}-\mathbf{x}_0)^T\cdot\frac{\nabla^2 f(\mathbf{x}_0)}{2!}\cdot(\mathbf{x}-\mathbf{x}_0)+\dots \\ &= f(\mathbf{x}_0)+\mathbf{g}(\mathbf{x}_0)\cdot(\mathbf{x}-\mathbf{x}_0)+(\mathbf{x}-\mathbf{x}_0)^T\cdot\frac{1}{2}\mathbf{G}(\mathbf{x}_0)\cdot(\mathbf{x}-\mathbf{x}_0)+\dots, \end{aligned} \tag{1.6}$$

where $\mathbf{x}_0$ is the point around which the function $f(\mathbf{x})$ is developed. For a point to be a minimum, elementary calculus (Rade and Westergren 1990) demands that

$$\mathbf{g}(\mathbf{x}_{\text{extr}})=\mathbf{0}, \tag{1.7}$$

i.e., *all* partial derivatives at $\mathbf{x}=\mathbf{x}_{\text{extr}}$ must be zero. In the third term on the right-hand side of Eq. 1.6, the difference between $\mathbf{x}$ and $\mathbf{x}_0$ is squared, so in order to avoid a negative contribution from the Hessian matrix, $\mathbf{G}(\mathbf{x}_0)$ must be positive semi-definite (Scales 1985). In the immediate neighborhood about $\mathbf{x}_0$, higher terms of the Taylor series expansion make a negligible contribution and need not be considered.

Applying the chain rule for differentiation to the first three terms of the Taylor expansion in Eq. 1.6 allows the gradient about the arbitrary point $\mathbf{x}_0$ to be expressed as

$$\mathbf{x}_{extr} = -\mathbf{g}(\mathbf{x}_0) \cdot \mathbf{G}^{-1}(\mathbf{x}_0) + \mathbf{x}_0. \tag{1.9}$$

where $\mathbf{G}^{-1}$ is the inverse of the Hessian matrix.

If the objective function, $f(\mathbf{x})$, is quadratic, then Eq. 1.9 can be applied directly to obtain its true minimum. Figure 1.4 shows how Eq. 1.9 computes the optimum of a (uni-modal) quadratic function independent of where the starting point, $\mathbf{x}_0$, is located.

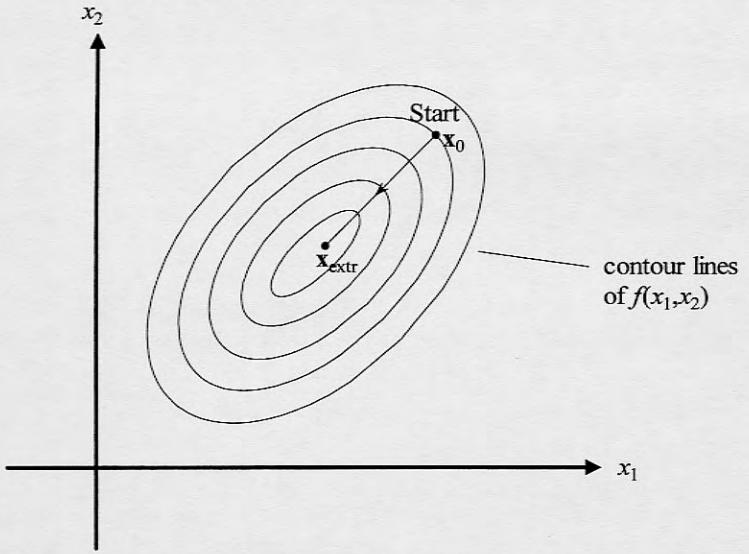


Fig. 1.4. If the objective function is quadratic and differentiable, then Eq. 1.9 can determine its optimum.

Even though there are applications, e.g., acoustical echo cancellation in speakerphones, where the objective function is a simple quadratic (Glentis et al. 1999), the majority of optimization tasks lack this favorable property. Even so, classical derivative-based optimization can be effective as long the objective function fulfills two requirements:

R1 The objective function must be *two-times differentiable.*

R2 The objective function must be *uni-modal*, i.e., have a single minimum.

$$f(x_1,x_2) = 10 - e^{-(x_1^2+x_2^2)}. \tag{1.10}$$

Figure 1.5 graphs the function defined in Eq. 1.10.

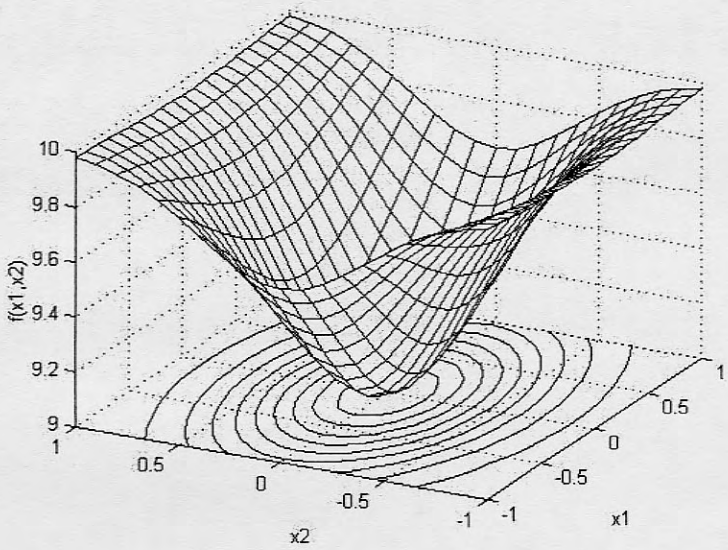


Fig. 1.5. An example of a uni-modal objective function

The method of *steepest descent* is one of the simplest gradient-based techniques for finding the minimum of a uni-modal and differentiable function. Based on Eq. 1.9, this approach assumes that $\mathbf{G}^{-1}(\mathbf{x}_0)$ can be replaced with the identity matrix:

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}. \tag{1.11}$$

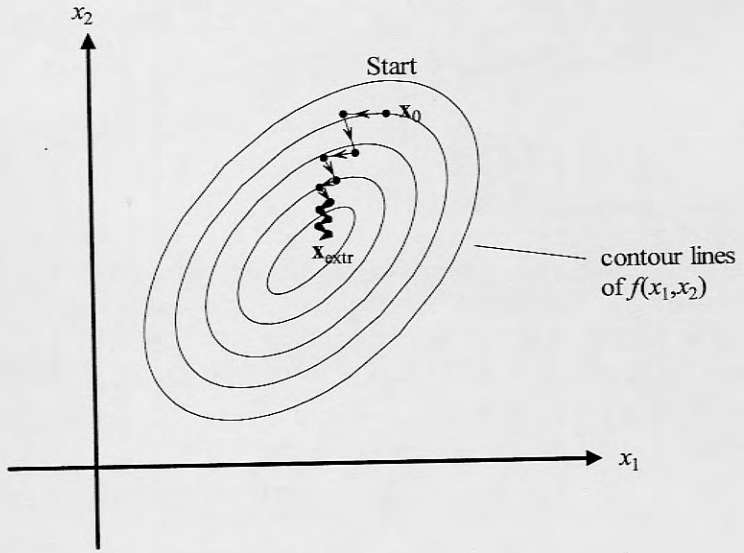
This crude replacement does not lead directly to the minimum, but to the point

$$\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{g}(\mathbf{x}_0). \tag{1.12}$$

ince the negative gradient points downhill, $\mathbf{x}_1$ will be closer to the imum than $\mathbf{x}_0$ unless the step was too large. Adding a step size, γ, to the eral recursion relation that defines the direction of steepest descent vides a measure of control:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \cdot \mathbf{g}(\mathbf{x}_n) \tag{1.13}$$

ure 1.6 shows a typical pathway from the starting point, $\mathbf{x}_0$, to the optim $\mathbf{x}_{extr}$. Additional details of the classical approach to optimization can found in Bunday and Garside (1987), Pierre (1986), Scales (1985) and ss et al. (1992). The point relevant to DE is that the classical approach eals the existence of a *step size problem* in which the best step size deids on the objective function.



g. 1.6. The method of steepest descent first computes the negative gradient, then ces a step in the direction indicated.

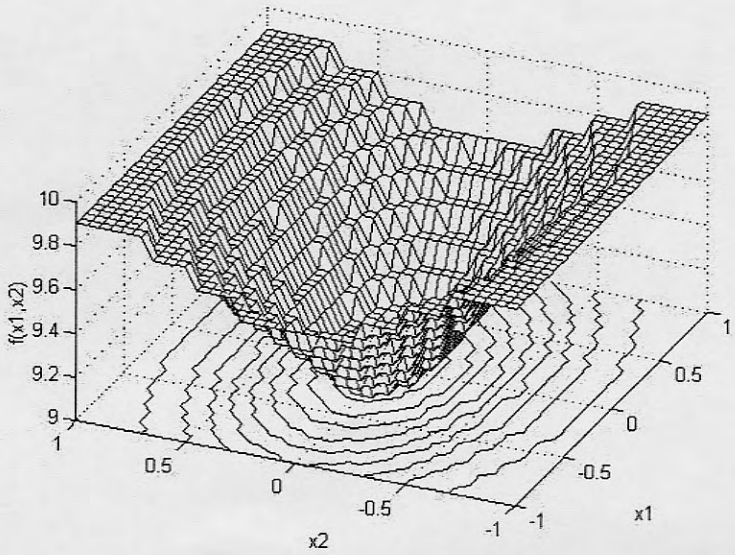
Replacing the inverse Hessian, $\mathbf{G}^{-1}(\mathbf{x}_0)$, with the identity matrix introices its own set of problems and more elaborate techniques like Gauss–ewton, Fletcher–Reeves, Davidon–Fletcher–Powell, Broyden–Fletcher–oldfarb–Shanno and Levenberg–Marquardt (Scales 1985; Pierre 1986) ive been developed in response. These methods roughly fall into two ategories. *Quasi-Newton methods* approximate the inverse Hessian by a

By contrast, *conjugate gradient* methods dispense with the Hessian matrix altogether, opting instead to use line optimizations in conjugate directions to avoid computing second-order derivatives. In addition to Quasi-Newton and conjugate gradient methods, mixtures of the two approaches also exist. Even so, all these methods require the objective function to be one-time or two-times differentiable. In addition, their fast convergence on quadratic objective functions does not necessarily transfer to non-quadratic functions. Numerical errors are also an issue if the objective function exhibits singularities or large gradients. Methods that do not require the objective function to be differentiable provide greater flexibility.

1.1.3 One-Point, Derivative-Free Optimization and the Step Size Problem

There are many reasons why an objective function might not be differentiable. For example, the "floor" operation in Eq. 1.14 quantizes the function in Eq. 1.10, transforming Fig. 1.5 into the stepped shape seen in Fig. 1.7. At each step's edge, the objective function is non-differentiable:

$$f(x_1,x_2) = \text{floor}\left(10 \cdot \left(10 - \exp\left(-x_1^2 - 3x_2^2\right)\right)\right)/10 \tag{1.14}$$



- Constraining the objective function may create regions that are non-differentiable or even forbidden altogether.
- If the objective function is a computer program, conditional branches make it non-differentiable, at least for certain points or regions.
- Sometimes the objective function is the result of a physical experiment (Rechenberg 1973) and the unavailability of a sufficiently precise actuator can make computing derivatives impractical.
- If, as is the case in evolutionary art (Bentley and Corne 2002), the objective function is "subjective", an analytic formula is not possible.
- In co-evolutionary environments, individuals are evaluated by how effectively they compete against other individuals. The objective function is not explicit.

When the lack of a computable derivative causes gradient-based optimizers to fail, reliance on derivative-free techniques known as *direct search* algorithms becomes essential. Direct search methods are "generate-and-test" algorithms that rely less on calculus than they do on heuristics and conditional branches. The meta-algorithm in Fig. 1.8 summarizes the direct search approach.

```
Initialization();          //choose the initial base point
                           //(introduces starting-point problem)
while (not converged)      //decide the number of iterations
{                          //(dimensionality problem)
   vector_generation(); //choose a new point
                           //(introduces step size problem)
   selection();            //determine new base point
}
```

Fig. 1.8. Meta-algorithm for the direct search approach

The meta-algorithm in Fig. 1.8 reveals that the direct search has a selection phase during which a proposed move is either accepted or rejected. Selection is an acknowledgment that in all but the simplest cases, not all proposed moves are beneficial. By contrast, most gradient-based optimizers accept each point they generate because base vectors are iterates of a recursive equation. Points are rejected only when, for example, a line

Enumeration or Brute Force Search

As their name implies, one-point, direct search methods are initialized with a single starting point. Perhaps the simplest one-point direct search is the *brute force method*. Also known as *enumeration*, the brute force method visits all grid points in a bounded region while storing the current best point in memory (see Fig. 1.9). Even though generating a sequence of grid points is trivial, the enumerative method still faces a step size problem because if nothing is known about the objective function, it is hard to decide how fine the grid should be. If the grid is too coarse, then the optimum may be missed. If the grid becomes too small, computing time explodes exponentially because a grid with N points in one dimension will have N^D points in D dimensions. Because of this "curse of dimensionality", the brute force method is very rarely used to optimize objective functions with a significant number of continuous parameters. The curse of dimensionality demonstrates that better sampling strategies are needed to keep a search productive.

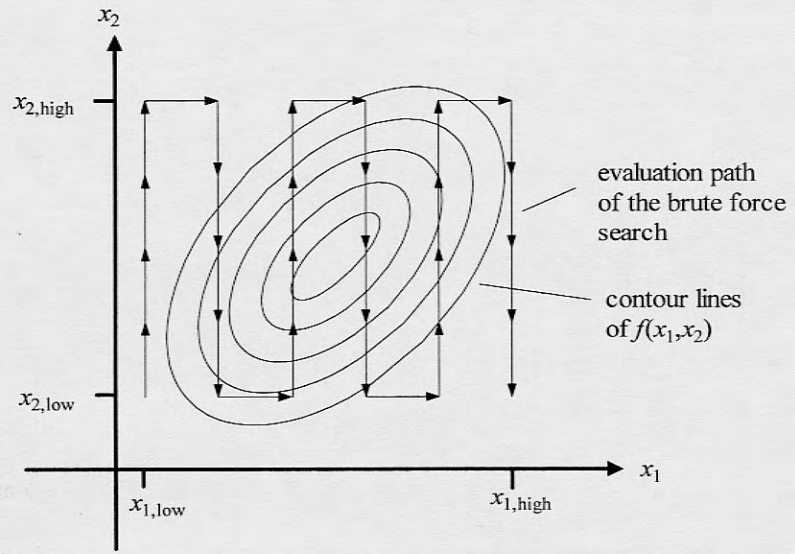


Fig. 1.9. The brute force search tries all grid points in a given region.

Random Walk

The *random walk* (Gross and Harris 1985) circumvents the curse of dimensionality inherent in the brute force method by sampling the objective function value at randomly generated points. New points are generated by adding a random deviation, $\Delta\mathbf{x}$, to a given base point, $\mathbf{x}_0$. In general, each coordinate, Δx_i, of the random deviation follows a Gaussian distribution

$$p(\Delta x_i) = \frac{1}{\sigma_i \cdot \sqrt{2\pi}} \exp\left(-0.5 \cdot \frac{(\Delta x_i - \mu_i)^2}{\sigma_i^2} \right), \tag{1.15}$$

where σ_i and μ_i are the standard deviation and the mean value, respectively, for coordinate i. The random walk's selection criterion is "greedy" in the sense that a trial point with a lower objective function value than that of the base point is always accepted. In other words, if $f(\mathbf{x}_0 + \Delta\mathbf{x}) \leq f(\mathbf{x}_0)$, then $\mathbf{x}_0 + \Delta\mathbf{x}$ becomes the new base point; otherwise the old point, $\mathbf{x}_0$, is retained and a new deviation is applied to it. Figure 1.10 illustrates how the random walk operates.

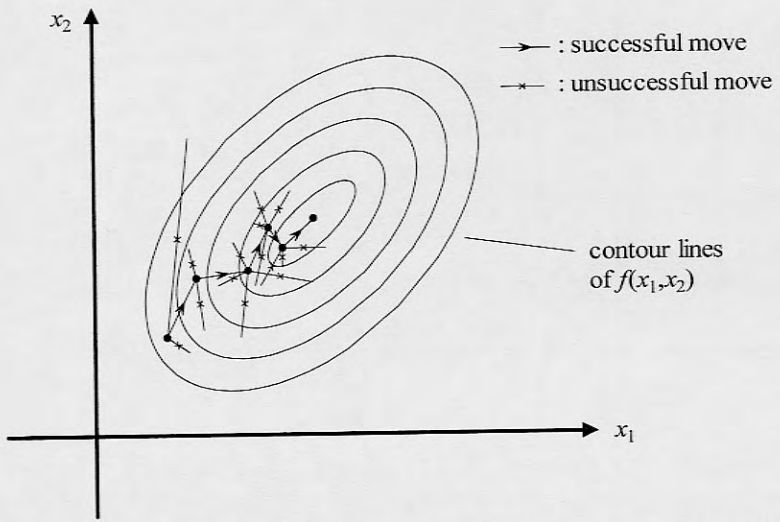


Fig. 1.10. The random walk samples the objective function by taking randomly generated steps from the last accepted point.

The stopping criterion for a random walk might be a preset maximum number of iterations or some other problem-dependent criterion. With a brute force search. Like both the classical and the brute force methods, the random walk suffers from the step size problem because it is very difficult to choose the right standard deviations when the objective function is not sufficiently well known.

Hooke and Jeeves

The Hooke–Jeeves method is a one-point direct search that attacks the step size problem (Hooke and Jeeves 1961; Pierre 1986; Bunday and Garside 1987; Schwefel 1994). Also known as a *direction* or *pattern search*, the Hooke–Jeeves algorithm starts from an initial base point, $\mathbf{x}_0$, and explores each coordinate axis with its own step size. Trial points in all D positive and negative coordinate directions are compared and the best point, $\mathbf{x}_1$, is found. If the best new trial point is better than the base point, then an attempt is made to make another move in the same direction, since the step from $\mathbf{x}_0$ to $\mathbf{x}_1$ was a good one. If, however, none of the trial points improve on $\mathbf{x}_0$, the step is presumed to have been too large, so the procedure repeats with smaller step sizes. The pseudo-code in Fig. 1.11 summarizes the Hooke–Jeeves method. Figure 1.12 plots the resulting search path.

```
...
while (h > h_min)  //as long as step length is still not small enough
{
    x_1 = explore(x_0,h); //explore the parameter space
    if (f(x_1) < f(x_0))  //if improvement could be made
    {
       x_2 = x_1 + (x_1 - x_0);   //make differential pattern move
       if (f(x_2) < f(x_1)) x_0 = x_2;
       else                 x_0 = x_1;
    }
    else h = h*reduction_factor;
}
...

function explore(vector x_0, vector h)
{  //---note that e_i is the unit vector for coordinate i---
   for (i=0; i<D; i++) //for all D dimensions
   {
      if (f(x_0+e_i*h) < f(x_0)) x_0 = x_0 + e_i*h; //check coordinate i
      else if (f(x_0-e_i*h) < f(x_0)) x_0 = x_0 - e_i*h;
   }
   return(x_0);
}
```

Fig. 1.11. Pseudo-code for the Hooke–Jeeves method

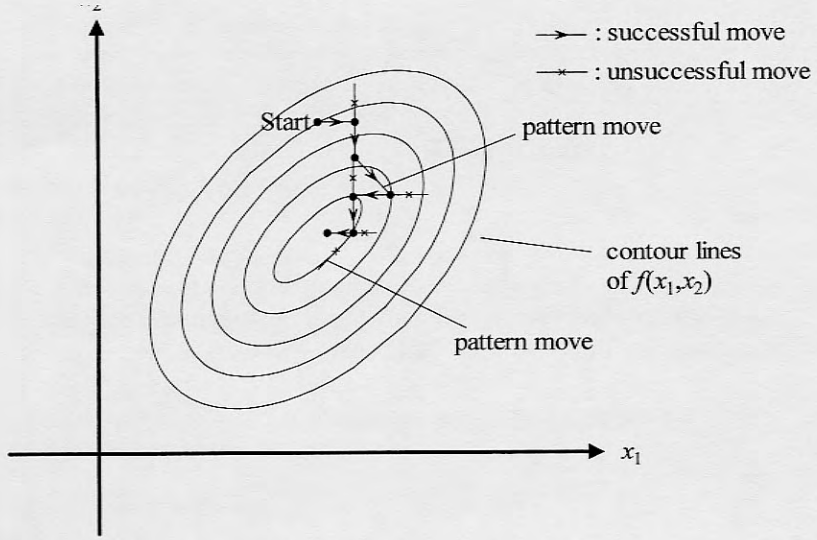


Fig. 1.12. A search guided by the Hooke–Jeeves method. Positive axis directions are always tried first.

On many functions, its adaptive step sizes make the Hooke–Jeeves search much more effective than either the brute force or random walk algorithms, but step sizes that shrink and never increase can be a drawback. For example, if steps are forced to become small because the objective function contains a "valley", then they will be unable to expand to the appropriate magnitude once the valley ends.

1.2 Local Versus Global Optimization

Both the step size problem and objective function non-differentiability can make even uni-modal functions a challenge to optimize. Additional obstacles arise once requirement R2 is dropped and the objective function is allowed to be multi-modal. Equation 1.16 is an example of a multi-modal function. As Fig. 1.13 shows, the "peaks" function in Eq. 1.16 has more than one local minimum:

$$\cdot \exp\left(x_1^2 + x_2^2\right) - \frac{1}{3} \cdot \exp\left((x_1+1)^2 + x_2^2\right)$$

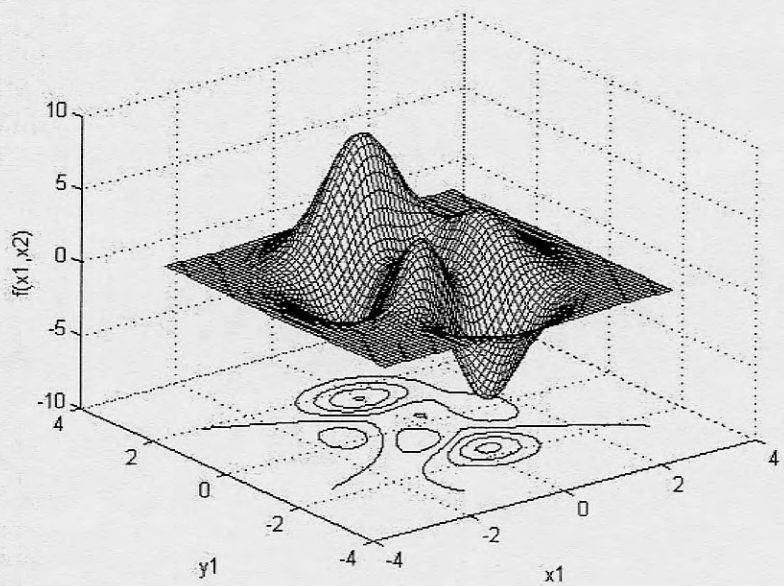


Fig. 1.13. The "peaks" function defined by Eq. 1.16 is multi-modal.

Because they exhibit more than one local minimum, multi-modal functions pose a *starting point problem*. Mentioned briefly in the direct search meta-algorithm (Fig. 1.8), the starting point problem refers to the tendency of an optimizer with a greedy selection criterion to find only the minimum of the basin of attraction in which it was initialized. This minimum need not be the global one, so sampling a multi-modal function in the vicinity of the global optimum, at least eventually, is essential. Because the Gaussian distribution is unbounded, there is a finite probability that the random walk will eventually generate a new and better point in a basin of attraction other than the one containing the current base point. In practice, successful inter-basin jumps tend to be rare. One method that increases the chance that a point will travel to another basin of attraction is simulated annealing.

.2.1 Simulated Annealing

imulated annealing (SA) (Kirkpatrick et al. 1983; Press et al. 1992), thorughly samples the objective function surface by modifying the greedy criɔrion to accept some uphill moves while continuing to accept all downhill ıoves. The probability of accepting a trial vector that lies uphill from the urrent base point decreases as the difference in their function values inreases. Acceptance probability also decreases with the number of function valuations, i.e., after a reasonably long time, SA's selection criterion beomes greedy. The random walk has traditionally been used in conjunction vith SA to generate trial vectors, but virtually any search can be modified ɔ incorporate SA's selection scheme. Figure 1.14 describes the basic SA lgorithm.

```
...
fbest = f(x0);//start with some base point
T     = T0;   //and some starting temperature
while (convergence criterion not yet met)
{
    Δx = generate_deviation(); //e.g., a Gaussian distribution
    if (f(x0+Δx) < f(x0))  //if improvement can be made
    {
       fbest = f(x0+Δx);
       x0 = x0+Δx; //new, improved base point
    }
    else
    {
       d = f(x0+Δx)-f(x0);//positive value
       r = rand(); //generate uniformly distr. variable ex [0,1]
       if (r < exp(-d*beta/T)) //Metropolis algorithm
       {
          x0 = x0+Δx; //new base point derived from uphill move
       }
    }
    T = T*reduction_factor;
}
...
```

Fig. 1.14. The basic simulated annealing algorithm. In this implementation, the ɾandom walk generates trial points.

The term "annealing" refers to the process of slowly cooling a molten substance so that its atoms will have the opportunity to coalesce into a minimum energy configuration. If the substance is kept near equilibrium at temperature T, then atomic energies E are distributed according to the

$$P(E) \sim \exp\left(\frac{E}{k \cdot T}\right), \tag{1.17}$$

where k is the Boltzmann constant.

By equating energy with function value, SA attempts to exploit nature's own minimization process *via* the Metropolis algorithm (Metropolis et al. 1953). The Metropolis algorithm implements the Boltzmann equation as a selection probability. While downhill moves are always accepted, uphill moves are accepted only if a uniformly distributed random number from the interval [0,1] is smaller than the exponential term:

$$\Theta = \exp\left(-\frac{d \cdot \beta}{T}\right). \tag{1.18}$$

The variable, d, is the difference between the uphill objective function value and the function value of the current base point, i.e., their "energy difference". Equation 1.18 shows that the acceptance probability, Θ, decreases as d increases and/or as T decreases. The value, β, is a problem-dependent control variable that must be empirically determined.

One of annealing's drawbacks is that special effort may be required to find an *annealing schedule* that lowers T at the right rate. If T is reduced too quickly, the algorithm will behave like a local optimizer and become trapped in the basin of attraction in which it began. If T is not lowered quickly enough, computations become too time consuming. There have been many improvements to the standard SA algorithm (Ingber 1993) and SA has been used in place of the greedy criterion in direct search algorithms like the method of Nelder–Mead (Press et al. 1992). The step size problem remains, however, and this may be why SA is seldom used for continuous function optimization. By contrast, SA's applicability to virtually any direct search method has made it very popular for combinatorial optimization, a domain where clever, but greedy, heuristics abound (Syslo et al. 1983; Reeves 1993).

1.2.2 Multi-Point, Derivative-Based Methods

Multi-start techniques are another way to extensively sample an objective function landscape. As their name implies, multi-start techniques restart the optimization process from different initial points. Typically, each sample point serves as the initial point for a greedy, local optimization method (Boender and Romeijn 1995). Often, the local search is derivative-based

local minimum because they all initially fell within the perimeter of the same basin of attraction.

Clustering methods (Törn and Zelinkas 1989; Janka 1999) refine the multi-start method by applying a clustering algorithm to identify those sample points that belong to the same basin of attraction, i.e., to the same cluster. Ideally, each cluster yields just one point to serve as the base point for a local optimization routine. *Density clustering* (Boender and Romeijn 1995; Janka 1999) is based on the assumption that clusters are shaped like hyper-ellipsoids and that the objective function is quadratic in the neighborhood of a minimum. Other methods, like the one described in Locatelli and Schoen (1996), use a proximity criterion to decide if a local search is justified. Because this determination often requires that all previously visited points be stored, highly multi-modal functions of high dimension can strain computer memory capacity. As a result, clustering algorithms are typically limited to problems with a relatively small number of parameters.

1.2.3 Multi-Point, Derivative-Free Methods

Evolution Strategies and Genetic Algorithms

Evolution strategies (ESs) were developed by Rechenberg (1973) and Schwefel (1994), while genetic algorithms (GAs) are attributed to Holland (1962) and Goldberg (1989). Both approaches attempt to evolve better solutions through recombination, mutation and survival of the fittest. Because they mimic Darwinian evolution, ESs, GAs, DE and their ilk are often collectively referred to as *evolutionary algorithms*, or EAs. Distinctions, however, do exist. An ES, for example, is an effective continuous function optimizer, in part because it encodes parameters as floating-point numbers and manipulates them with arithmetic operators. By contrast, GAs are often better suited for combinatorial optimization because they encode parameters as bit strings and modify them with logical operators. Modifying a GA to use floating-point formats for continuous parameter optimization typically transforms it into an ES-type algorithm (Mühlenbein and Schlierkamp-Vosen 1993; Salomon 1996). There are many variants to both approaches (Bäck 1996; Michalewicz 1996), but because DE is primarily a numerical optimizer, the following discussion is limited to ESs.

one another by means of recombination. Beginning with a population of μ parent vectors, the ES creates a child population of $\lambda \geq \mu$ vectors by recombining randomly chosen parent vectors. Recombination can be *discrete* (some parameters are from one parent, some are from the other parent) or *intermediate* (e.g., averaging the parameters of both parents) (Bäck et al. 1997; Bäck 1996). Once parents have been recombined, each of their children is "mutated" by the addition of a random deviation, $\mathbf{\Delta x}$, that is typically a zero mean Gaussian distributed random variable (Eq. 1.15).

After mutating and evaluating all λ children, the (μ, λ)-ES selects the best μ children to become the next generation's parents. Alternatively, the $(\mu + \lambda)$-ES populates the next generation with the best μ vectors from the combined parent and child populations. In both cases, selection is greedy within the prescribed selection pool, but this is not a major drawback because the vector population is distributed. Figure 1.15 summarizes the meta-algorithm for an ES.

```
Initialization();      //choose starting population of μ members
while (not converged)  //decide the number of iterations
{
   for (i=0; i<λ; i++) //child vector generation: λ > μ
   {
     p1(i) = rand(μ);     //pick a random parent from μ parents
     p2(i) = rand(μ);     //pick another random parent  p2(i) != p1(i)
     c1(i) = recombine(p1(i),p2(i)); //recombine parents
     c1(i) = mutate(c1(i));          //mutate child
     save(c1(i));         //save child in an intermediate population
   }
   selection();          //μ new parents out of either λ, or λ+μ
}
```

Fig. 1.15. Meta-algorithm for evolution strategies (ESs)

While ESs are among the best global optimizers, their simplest implementations still do not solve the step size problem. Schwefel addressed this issue in Schwefel (1981) where he proposed modifying the Gaussian mutation distribution with a matrix of adaptive covariances, an idea that Rechenberg suggested in 1967 (Fogel 1994). Equation 1.19 generalizes the multi-dimensional Gaussian distribution to include a covariance matrix, $\mathbf{C}$ (Papoulis 1965):

$$p(\Delta\mathbf{x}) = \frac{1}{\det(\mathbf{C}) \cdot \sqrt{(2\pi)^D}} \exp\left(-0.5(\Delta\mathbf{x} - \mu)^T \cdot \mathbf{C}^{-1} \cdot (\Delta\mathbf{x} - \mu)\right) \tag{1.19}$$

In Eq. 1.19, $\boldsymbol{\mu}$ is the *mean vector* and $\mathbf{C}$ is the *covariance matrix*

$$\mathbf{C} = \begin{pmatrix} \sigma_{0,0}^2 & \sigma_{0,1} & \cdots & \sigma_{0,D-1} \\ \sigma_{1,0} & \sigma_{1,1}^2 & \cdots & \sigma_{1,D-1} \\ \cdots & \cdots & \cdots & \cdots \\ \sigma_{D-1,0} & \sigma_{D-1,1} & \cdots & \sigma_{D-1,D-1}^2 \end{pmatrix} = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{S} \tag{1.20}$$

with the *scatter matrix*

$$\mathbf{S} = \begin{pmatrix} \sigma_0^2 & 0 & \cdots & 0 \\ 0 & \sigma_1^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_{D-1}^2 \end{pmatrix} \tag{1.21}$$

and the *correlation matrix*

$$\mathbf{R} = \begin{pmatrix} 1 & \rho_{0,1} & \cdots & \rho_{0,D-1} \\ \rho_{1,0} & 1 & \cdots & \rho_{1,D-1} \\ \cdots & \cdots & \cdots & \cdots \\ \rho_{D-1,0} & \rho_{D-1,1} & \cdots & 1 \end{pmatrix} \tag{1.22}$$

By permitting the otherwise symmetrical Gaussian distribution to become ellipsoidal, the ES can assign a different step size to each dimension. In addition, the covariance matrix allows the Gaussian mutation ellipsoid to rotate in order to adapt better to the topography of non-decomposable objective functions. A *decomposable* function (Salomon 1996) can always be written as

$$f(\mathbf{x}) = \sum_{i=0}^{D-1} f_i(x_i). \tag{1.23}$$

Because decomposable functions lack cross-terms, their parameters can be optimized independently. Thus, decomposability replaces the task of optimizing one function having D dimensions with the much simpler task of optimizing D one-dimensional functions. The hyper-ellipsoid is a simple

$$f(\mathbf{x}) = \sum_{i=0}^{D-1} \alpha_i x_i^2. \tag{1.24}$$

If, however, the hyper-ellipsoid is rotated in all dimensions, it becomes impossible to optimize one parameter independent of the others. This parameter dependence is often referred to as *epistasis*, an expression from biology (www 01). Salomon (1996) shows that unless an optimizer addresses the issue of parameter dependence, its performance on epistatic objective functions will be seriously degraded. This important issue is discussed extensively in Sect. 2.6.2.

Adapting the components of $\mathbf{C}$ requires additional "strategy parameters", i.e., the variances and position angles of the D-dimensional hyper-ellipsoids for which $\mathbf{C}$ is positive definite (Sprave 1995). Thus, the ES with correlated mutations increases a problem's dimensionality because it characterizes each individual by not only a vector of D objective function parameters, but also an additional vector of up to $D \cdot (D - 1)/2$ strategy parameters. For problems having many variables, the time and memory needed to execute these additional (matrix) calculations may become prohibitive.

Nelder and Mead

The Nelder–Mead polyhedron search (Nelder and Mead 1965; Bunday and Garside 1987; Press et al. 1992; Schwefel 1994), tries to solve the step size problem by allowing the step size to expand or contract as needed. The algorithm begins by forming a $(D + 1)$-dimensional polyhedron, or *simplex*, of $D + 1$ points, $\mathbf{x}_i$, $i = 0, 1, \ldots, D$, that are randomly distributed throughout the problem space. For example, when $D = 2$, the simplex is a triangle. Indices of the points are ordered according to ascending objective function value so that $\mathbf{x}_0$ is the best point and $\mathbf{x}_D$ is the worst point. To obtain a new trial point, $\mathbf{x}_r$, the worst point, $\mathbf{x}_D$, is reflected through the opposite face of the polyhedron using a weighting factor, $F1$:

$$\mathbf{x}_r = \mathbf{x}_D + F1 \cdot (\mathbf{x}_m - \mathbf{x}_D). \tag{1.25}$$

The vector, $\mathbf{x}_m$, is the *centroid* of the face opposite $\mathbf{x}_D$:

$$\mathbf{x}_m = \frac{1}{D}\left(\sum_{i=0}^{D-1} \mathbf{x}_i\right). \tag{1.26}$$

Figure 1.16 illustrates the *reflection operation* defined in Eq. 1.25.

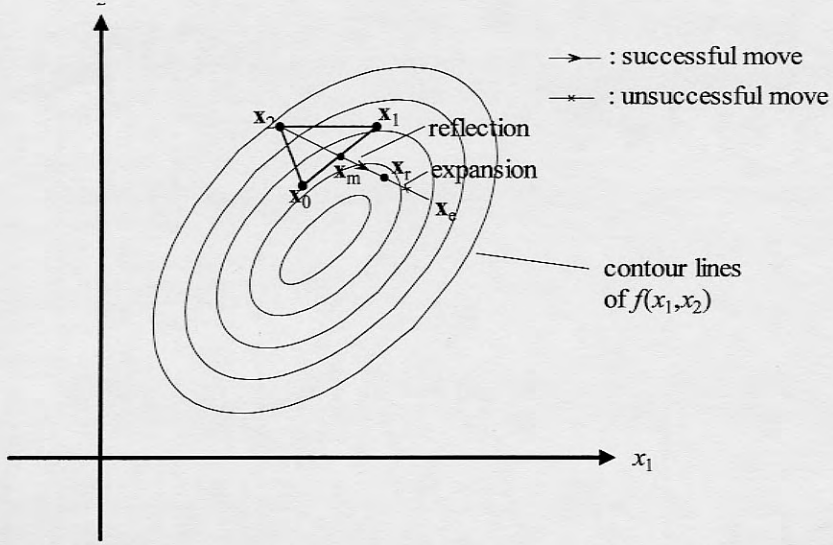


Fig. 1.16. Reflection and expansion in the Nelder–Mead method where $D = 2$

If a reflection through the centroid improves on the best point, $\mathbf{x}_0$, i.e., if $f(\mathbf{x}_r) < f(\mathbf{x}_0)$, then the Nelder–Mead algorithm takes another step in the same direction based on the assumption that still further improvement may be possible. When weighted by a second scale factor, $F2$, this *expansion step* generates a new trial point, $\mathbf{x}_e$:

$$\mathbf{x}_e = \mathbf{x}_r + F2 \cdot (\mathbf{x}_m - \mathbf{x}_D) \tag{1.27}$$

If this expansion step also improves on $\mathbf{x}_0$, then $\mathbf{x}_e$ replaces $\mathbf{x}_D$. This new set of $D + 1$ points becomes the next simplex and the procedure begins again by ordering points based on their objective function value. If, however, $\mathbf{x}_e$ did not improve upon $\mathbf{x}_0$, then $\mathbf{x}_r$ replaces $\mathbf{x}_D$. If $\mathbf{x}_r$ did not improve upon $\mathbf{x}_0$ in the first place, then $\mathbf{x}_r$ is compared to the next worst point, $\mathbf{x}_{D-1}$. If $\mathbf{x}_r$ is better than $\mathbf{x}_{D-1}$, then $\mathbf{x}_r$ replaces $\mathbf{x}_D$. If, however, $\mathbf{x}_r$ is worse than $\mathbf{x}_{D-1}$, a third scaling constant, $F3$, *shrinks* the entire simplex. Pseudo-code for the Nelder–Mead algorithm appears in Fig. 1.17. Figures 1.18–1.21 illustrate how the simplex moves in a two-dimensional parameter space.

```
//---ascending objective function value------------------
sort(x_i,D+1);
//---compute centroid-------------------------------------
x_m = 0;
for (i=0; i<D; i++) x_m = x_m + x_i;
x_m = x_m/D;
//---start exploration of surface------------------------
x_r = x_m + F1(x_m - x_D);//reflection
if (f(x_r) < f(x_0))   //if best point is improved
{
   x_e = x_r + F2(x_m - x_D);//expansion
   if (f(x_e) < f(x_0)) x_D = x_e;
   else                 x_D = x_r;
}
else if (f(x_r) < f(x_D-1))//if next worst point is improved
{
   x_D = x_r;
}
else//if best and next worst point are not improved
{
   if (f(x_r) < f(x_D))
   {
      x_D = x_r;//replace worst point with reflected point
      x_c = x_m + F3(x_m - x_D);//contract around centroid
   }
   else
   {
      x_c = x_m - F3(x_m - x_D);//contract around centroid
   }
   if (f(x_c) < f(x_D))//if contraction was successful
   {
      x_D = x_c;
   }
   else //contract around the best point
   {
      for (i=1; i<=D; i++) x_i = 0.5*(x_0 + x_i);
   }
}
}//end while
...
```

Fig. 1.17. Pseudo-code for the Nelder–Mead algorithm

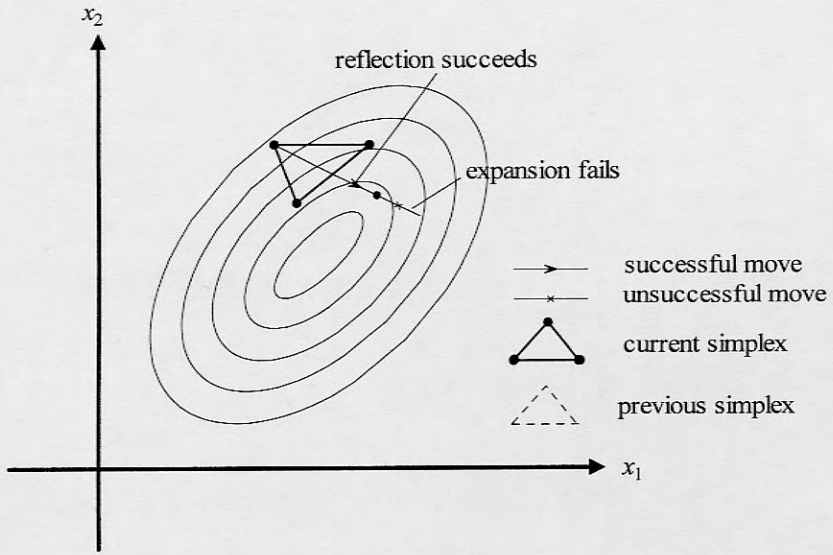


Fig. 1.18. Evolution of the Nelder–Mead simplex: first iteration. The reflection succeeds but the following expansion fails.

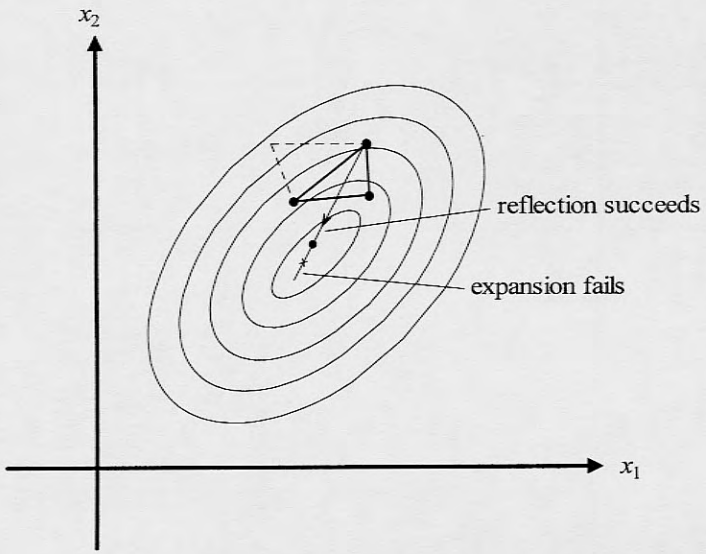


Fig. 1.19. Evolution of the Nelder–Mead simplex: second iteration. Again the reflection succeeds but the expansion fails.

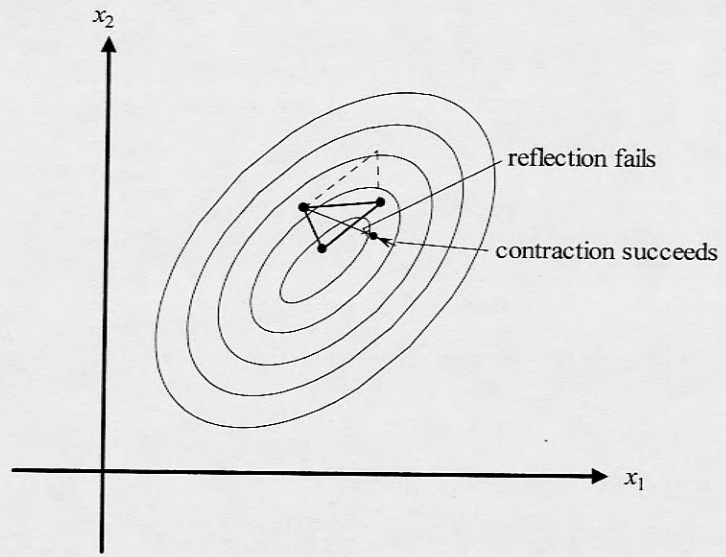


Fig. 1.20. Evolution of the Nelder–Mead simplex: third iteration. This time, even the reflection fails so a contraction must be tried. The contraction is successful.

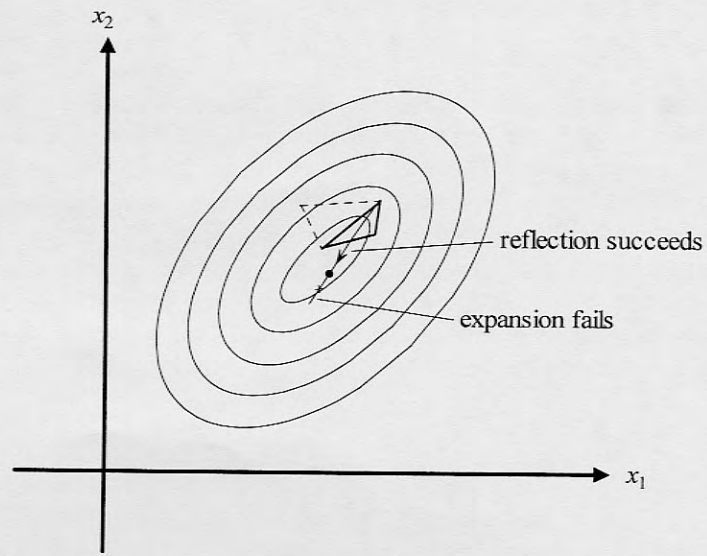


Fig. 1.21. Evolution of the Nelder–Mead simplex: fourth iteration. The reflection succeeds, but the expansion does not

can shrink as well as expand to adapt to the current objective function surface. This makes the step size a variable that depends on the topography of the objective function landscape. Like the Nelder–Mead method, DE also exploits vector differences but without the positional bias inherent in simplex reflections. Section 2.6.3 explores this distinction in detail.

Unlike DE, the Nelder–Mead algorithm restricts the number of sample points to $D + 1$. This limitation becomes a drawback for complicated objective functions that require many more points to form a clear model of the surface topography. Box (Box 1965; Bunday and Garside 1987; Schwefel 1994) suggested using a geometrical entity called a *complex* that, unlike a simplex, contains $2D$ points. Box also exploited the difference vectors formed by the centroid and all other points except for the worst one, but for multi-modal functions in particular, excessive reliance on the centroid as a reference point is meaningless, or, worse, the cause of premature convergence.

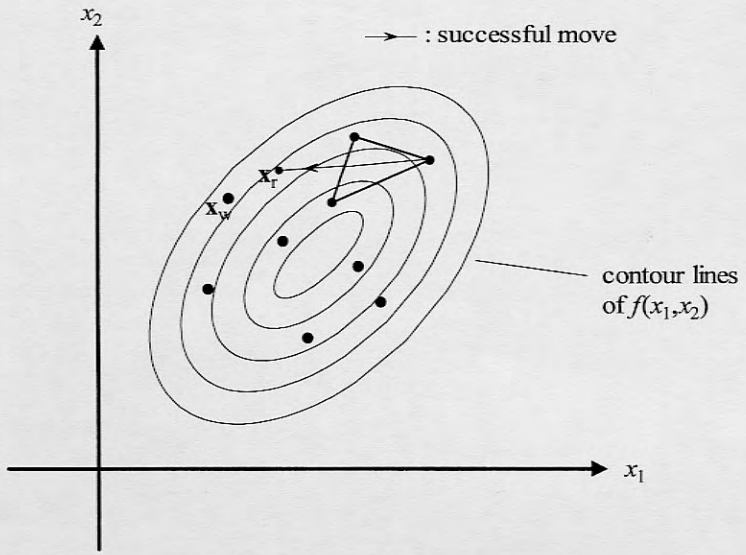


Fig. 1.22. The CRS method applies Nelder–Mead's reflections to a population of points.

```
//---ascending objective function value-----------------
sort(x_i,N_P);  //x_0 is best, x_D is worst point x_w
//---compute centroid (Points of centroid should be------
//---all different. Code to achieve that is not shown.)--
j = rand(N_P); //pick a random point from the population
               //as a variant pick j=0 (best point)
x_m = x_j;
for (i=1; i<D; i++)
{
   j = rand(N_P); //pick a random point from the population
   x_m = x_m + x_j;
}
x_m = x_m/D;
//---start exploration of surface-----------------------
x_r = x_m + F1(x_m - x_j);//reflection from last j, usually F1=1
if (bounds_ok(x_r) == TRUE)//if inside the region of interest
{
   if (f(x_r) < f(x_D))   //if worst point is improved
   {
      x_D = x_r;
   }
}
//---optionally there follows a local search------------
//---starting from the best points----------------------
...
}//end while
```

Fig. 1.23. Pseudo-code for the CRS-type algorithms

Controlled Random Search

Price's (no relation to the author of this book) *controlled random search* (CRS) also uses difference vectors for reflection operations (Price 1978). CRS employs a Nelder–Mead-like simplex consisting of $D + 1$ points drawn at random from a population of $Np > D + 1$ vectors as shown in Figure 1.22. A reflection through the centroid generates a new point $\mathbf{x}_r$. If this point is better than the current worst point $\mathbf{x}_w$, $\mathbf{x}_r$ replaces $\mathbf{x}_w$. Figure 1.23 presents pseudo-code for the CRS.

CRS resembles DE because the population size is a control variable and because vector differences generate new points. Like the Nelder–Mead algorithm, though, CRS's reflection operations are a form of arithmetic recombination (see Sect. 2.6.3), whereas DE's vector operations more closely resemble a mutation operation (see Sect. 2.5).

One drawback of the CRS algorithm is that continually replacing the current worst point exerts high selective pressure that may force the population to prematurely converge. Even though it is a multi-point strategy, this "replace worst" selection strategy also makes it difficult to implement the CRS algorithm in parallel. Conflicts can arise because the current worst point can change after every reflection. There have been several improvements to the CRS algorithm, most notably by Ali et al. (1997) and Ali and Törn (2004).

1.2.4 Differential Evolution – A First Impression

Price and Storn developed DE to be a reliable and versatile function optimizer that is also easy to use. The first written publication on DE appeared as a technical report in 1995 (Storn and Price 1995). Since then, DE has proven itself in competitions like the IEEE's International Contest on Evolutionary Optimization (ICEO) in 1996 and 1997 and in the real world on a broad variety of applications. Recently, Mathematica® added DE to its numerical optimizer package.

Like nearly all EAs, DE is a population-based optimizer that attacks the starting point problem by sampling the objective function at multiple, randomly chosen initial points. Preset parameter bounds define the domain from which the *Np* vectors in this initial population are chosen (Fig. 1.24). Each vector is indexed with a number from 0 to *Np* – 1. Like other population-based methods, DE generates new points that are perturbations of existing points, but these deviations are neither reflections like those in the CRS and Nelder–Mead methods, nor samples from a predefined probability density function, like those in the ES. Instead, DE perturbs vectors with the scaled difference of two randomly selected population vectors (Fig. 1.25). To produce the trial vector, $\mathbf{u}_0$, DE adds the scaled, random vector difference to a third randomly selected population vector (Fig. 1.26). In the selection stage, the trial vector competes against the population vector of the same index, which in this case is number 0. Figure 1.27 illustrates the select-and-save step in which the vector with the lower objective function value is marked as a member of the next generation. Figures 1.28–1.29 indicate that the procedure repeats until all *Np* population vectors have competed against a randomly generated trial vector. Once the last trial vector has been tested, the survivors of the *Np* pairwise competitions become parents for the next generation in the evolutionary cycle.

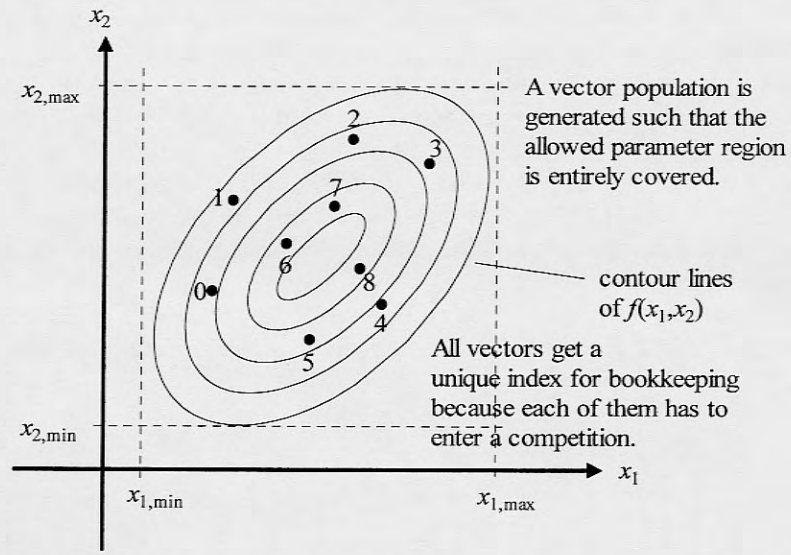


Fig. 1.24. Initializing the DE population

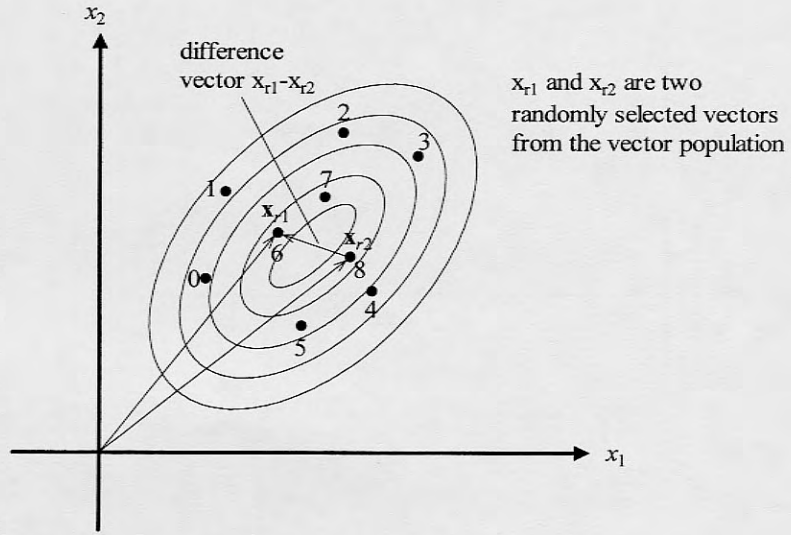


Fig. 1.25. Generating the perturbation: $\mathbf{x}_{r1} - \mathbf{x}_{r2}$

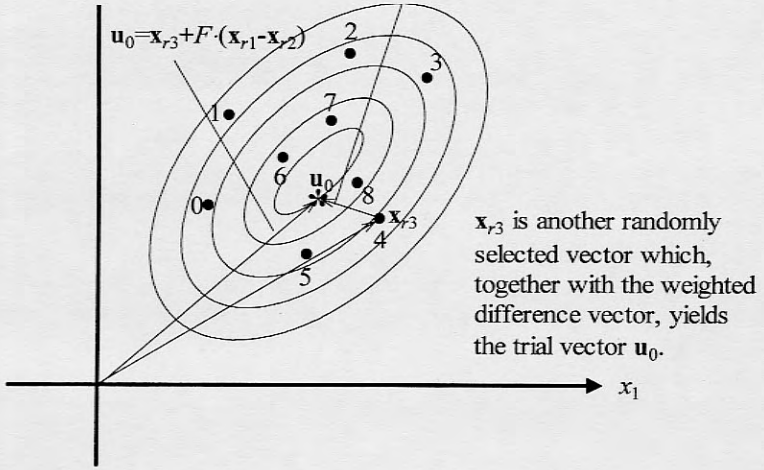


Fig. 1.26. Mutation

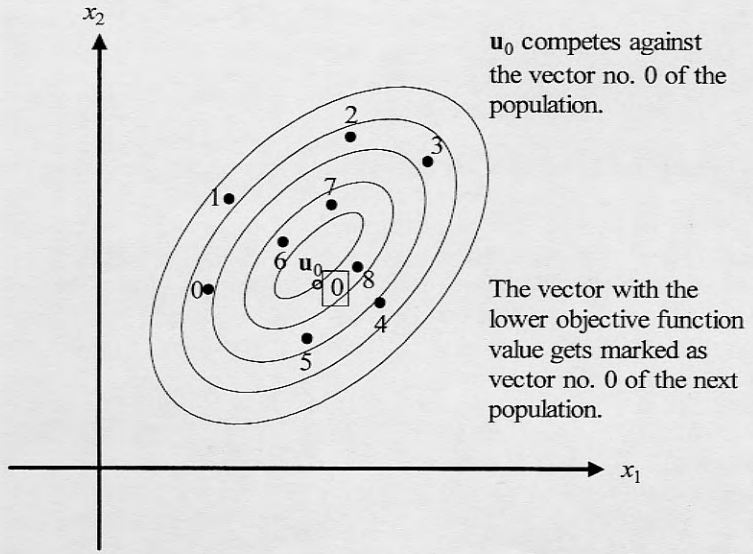


Fig. 1.27. Selection. Because it has a lower function value, $\mathbf{u}_0$ replaces the vector with index 0 in the next generation.

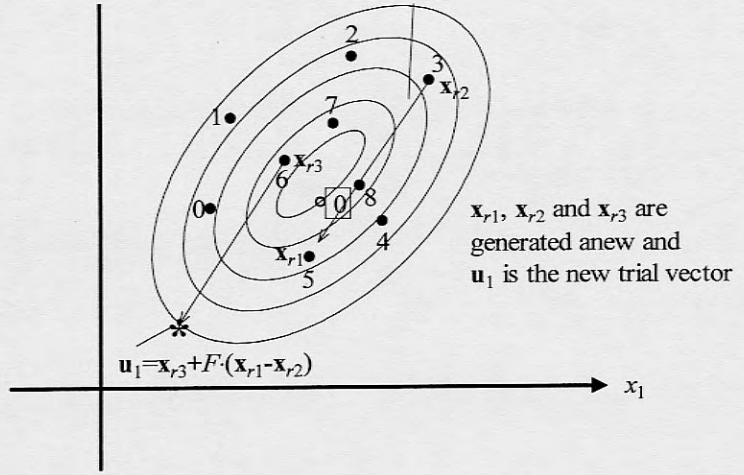


Fig. 1.28. A new population vector is mutated with a randomly generated perturbation.

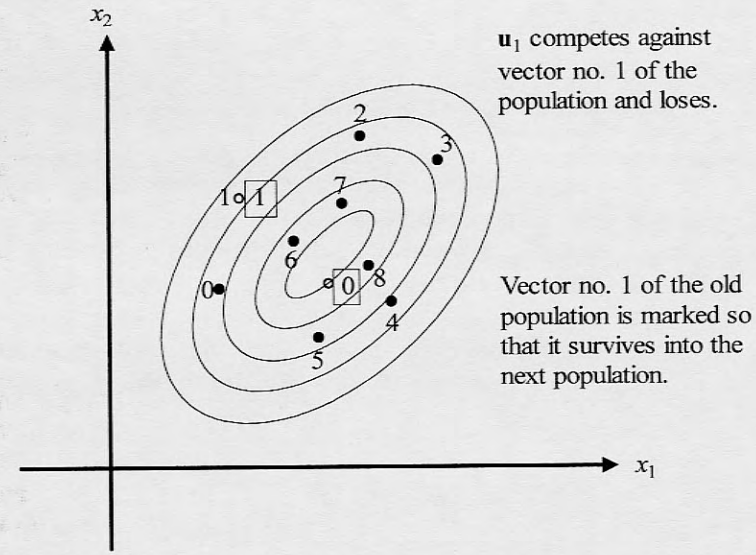


Fig. 1.29. Selection. This time, the trial vector loses.

Figure 1.30 presents pseudo-code for DE's most basic idea.

```
...
while (convergence criterion not yet met)
{
    //---x_i defines a vector of the current vector population-------
    //---y_i defines a vector of the new vector population-----------
    for (i=0; i<N_P; i++)
    {
       r1 = rand(N_P); //select a random index from 1, 2, ..., N_P
       r2 = rand(N_P); //select a random index from 1, 2, ..., N_P
       r3 = rand(N_P); //select a random index from 1, 2, ..., N_P
       u_i = x_r3 + F*(x_r1 - x_r2);
       if (f(u_i) <= f(x_i))
       {
         y_i = u_i;
       }
       else
       {
         y_i = x_i;
       }
    }
}//end while
...
```

Fig. 1.30. Pseudo-code for a simplified form of DE's generate-and-test operations

Even though the scheme described above already works remarkably well, DE's performance can be improved and its methodology adapted to a wide variety of optimization scenarios. The following chapters provide additional insight into how and why DE works, including a convergence proof, performance comparisons with other global optimization algorithms, practical applications, and computer code for solving real-world tasks.

References

Ali MM, Törn A, Viitanen S (1997) A numerical comparison of some modified controlled random search algorithms. Journal of Global Optimization 11:377–385

Ali MM, Törn A (2004) Population set based global optimization algorithms: some modifications and numerical studies. Computers and Operations Research 31(10):1703–1725

Bäck T (1996) Evolutionary algorithms in theory and practice. Oxford University Press

Bäck T, Hammel U, Schwefel H-P (1997) Evolutionary computation: comments on the history and current state. IEEE Transactions on Evolutionary Computation 1(1):3–17

Bentley PJ, Corne DW (2002) Creative evolutionary systems. Morgan Kaufmann, San Francisco

Boender C, Romeijn H (1995) Stochastic methods. In: Horst R, Pardalos P (eds) Handbook of global optimization. Kluwer, Dordrecht

Box MJ (1965) A new method of constrained optimization and a comparison with other methods. Computer Journal 8:42–52

Bunday BD, Garside GR (1987) Optimisation methods in PASCAL. Edward Arnold, London

Corne D, Dorigo M, Glover F (1999) New ideas in optimization. McGraw-Hill, London

Fogel DB (1994) Guest editorial on evolutionary computation. IEEE Transactions on Neural Networks 5(1):1–14

Glentis GO, Berberidis K, Theodoridis S (1999) Efficient least squares adaptive algorithms for FIR transversal filtering. IEEE Signal Processing Magazine July:13–41

Goldberg DE (1989) Genetic algorithms in search optimization and machine learning. Addison-Wesley, Reading, MA

Gross D, Harris CM (1985) Fundamentals of queuing theory. Wiley, New York

Holland JH (1962) Outline for a logical theory of adaptive systems. Journal of the Association for Computing Machinery 3:297–314

Hooke R, Jeeves TA (1961) Direct search solution of numerical and statistical problems. Journal of the Association for Computing Machinery 8:212–229

Ingber L (1993) Simulated annealing: practice versus theory. Journal of Mathematical and Computer Modeling 18(11): 29–57

Janka E (1999) Vergleich stochastischer Verfahren zur globalen Optimierung. Diplomarbeit, University of Wien

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

Locatelli M, Schoen F (1996) Simple linkage: analysis of a threshold-accepting global optimization method. Journal of Global Optimization 9:95–111

Metropolis N, Rosenbluth AE, Rosenbluth NM, Teller AN, Teller E, (1953) Equation of state calculation by fast computing machines, Journal of Chemical Physics 21:1087-1091

Michalewicz Z (1996) Genetic algorithms + data structures = evolution programs, 3rd ed. Springer, Berlin Heidelberg New York

Mühlenbein H, Schlierkamp-Vosen D (1993) Predictive models for the breeder genetic algorithm I. Evolutionary Computation 1(1):25–49

Nelder JA, Mead R (1965) A simplex method for function minimization. Computer Journal 7:308–313

Onwubolu GC, Babu BV (eds) (2004) New optimization techniques in engineering. Studies in Fuzziness and Soft Computing, vol 141. Springer, Berlin Hei-