1

Optimization-Based Design

We will begin our study of power magnetic device design with a general consideration of the design process. A case will be made to approach the design process rather formally by converting the design problem into an optimization problem. Next, single-objective optimization is discussed, with particular emphasis on optimization using genetic algorithms (GAs). This is followed by a discussion of multi-objective optimization. Practical aspects of formulating design problems as optimization problems are then considered. The chapter concludes with a design example that focuses on a UI-core inductor.

COPYRIGHTED MATERIAL

1.1 Design Approach

It is appropriate to begin this work by considering the design process. Clearly, there are a myriad of different approaches by which components may be designed. For example, a possible manual design process is illustrated in Figure 1.1. In order to consider this process in a more concrete way, suppose that the component we are designing is an electromagnet and that we wish to design an electromagnet so that a certain set of specifications are met.

Using the design process in Figure 1.1, our first step would be to perform a detailed mathematical analysis of the device. Typically, when we analyze a device, our analysis predicts device performance (mass, loss, force) in terms of the device parameters (geometry, materials) rather than directly addressing the design problem by deriving expressions for what the device parameters should be in terms of the device specifications (allowed loss, required force). Therefore, we must manipulate our detailed analysis into a set of design equations that are used to calculate the design parameters as a function of device specifications. However, going from detailed analysis to design equations invariably requires numerous assumptions and approximations, even beyond the ones found in our original "detailed" analysis. As a result, we check our design, either against our original analysis or using some numerical tool such as a finite element analysis. Based on the results from the numerical analysis, we will revise the design and repeat the numerical analysis until specifications are met, at which point we have arrived at a final design. Of course, we often use a more involved design process; for example, another iteration of the design may be made based on physical prototypes.

The manual design process we have been considering involves an engineer in the iteration process. Variations of this process are successfully used ubiquitously throughout the engineering community. However, the process has some significant drawbacks. First, it requires a great deal of engineering time. Second, it requires a great deal of engineering experience. This experience

Power Magnetic Devices: A Multi-Objective Design Approach, Second Edition. Scott D. Sudhoff.
© 2022 The Institute of Electrical and Electronics Engineers, Inc. Published 2022 by John Wiley & Sons, Inc.
Companion website: www.wiley.com/go/sudhoff/Powermagneticdevices

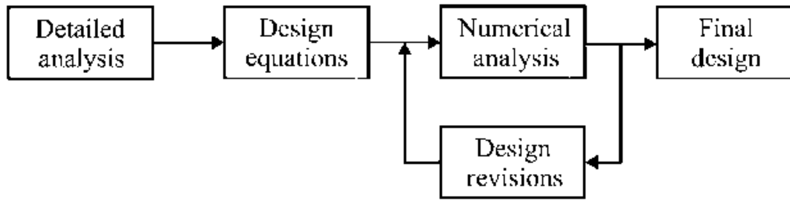

Figure 1.1 A manual design process.

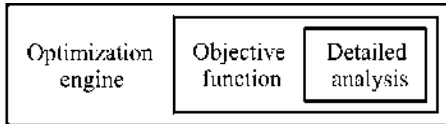

Figure 1.2 Optimization-based design process.

comes into play in the development of the design equations, which often take the form of rules-of-thumb based at least partially on experience. Experience is also a factor in making changes to the design based on the numerical analysis. Finally, while the process has been very successful in yielding working designs, it may not lead to the best design.

An alternate design process is illustrated in Figure 1.2. Therein, an optimization-based design process is shown. In this case, the process is not illustrated in a sequential manner as in Figure 1.1, but rather in an organizational manner. The process again starts with a detailed analysis of the device or component. However, unlike the manual design process, in the optimization-based process, the detailed analysis is not used to formulate design equations. Instead, the detailed analysis is used to calculate design metrics such as mass, cost, and loss. The detailed analysis is also used to check constraints such as achieving some minimum acceptable level of performance. The metrics and constraints are combined into an objective or fitness function. This function is defined so that its optimization results in optimization of the design metrics subject to all design constraints being met.

At the outermost level of this design process, an optimization engine will select the parameters of the design (geometry, materials, etc.) so as to maximize the objective function. In terms of computational algorithm, Figure 1.2 depicts an optimization engine at the outer level. This engine operates on an objective function that is calculated based on the detailed analysis.

There are several advantages of this approach. First, it is unnecessary to formulate design equations. This is beneficial in that it reduces the number of approximations and assumptions made and reduces the amount of design experience needed for a good design. Second, the design is formally optimized with regard to the design metrics, potentially leading to better designs, at least in terms of the design metrics. Third, since the engineer is out of the optimization loop, less engineering time is generally required. There are some disadvantages of the procedure. First, the process can be numerically intense and require significant computing time, sometimes on the order of hours and, in extreme cases, days. Fortunately, computer time is significantly less expensive than engineering time. Second, the quality of the result depends upon the quality of the detailed analysis. In this regard, design experience is still valuable, though not as critical as in the manual design approach.

In order to utilize the optimization-based design process, it is clearly necessary to be able to optimize mathematical functions. For design purposes, we will be optimizing the objective function, which we will also refer to as a fitness function. Optimization is a broad subject, which has been

the subject of a strong and sustained interest of a host of researchers over the years. The purpose of this chapter is to introduce the subject to an extent sufficient to enable the reader to utilize an optimization-based design process for power magnetic devices. More thorough study of optimization methods will serve every engineer well; for a good textbook devoted to the subject the reader is referred to Chong and Żak [1].

1.2 Mathematical Properties of Objective Functions

Before discussing optimization algorithms, it is appropriate to discuss some properties of objective functions that are relevant to their optimization, as these properties determine the effectiveness of one optimization approach relative to another.

As we proceed to do this, note that throughout this work, scalar variables are normally in italic font (for example, x) while vector and matrices are bold nonitalic (for example, $\mathbf{x}$). Functions of all dimensionalities are denoted by nonitalic nonbold font (for example, $\mathrm{x}(\theta)$). Brackets in equations are associated with iteration number in iterative methods.

In considering the properties of the objective function, it is appropriate to begin by defining our parameter vector, which will be denoted as $\mathbf{x}$. The domain of $\mathbf{x}$ is referred to as the search space and will be denoted Ω, which is to say we require $\mathbf{x} \in \Omega$. The elements of parameter vector $\mathbf{x}$ will include those variables of a design that we are free to select. In general, some elements of $\mathbf{x}$ will be discrete in nature while others will be continuous. An example of a discrete element might be one that designates a material type from a list of available materials. A geometrical parameter such as the length of a motor would be an example of an element that can be selected from a continuous range. If all members of the parameter vector are discrete, the search space is described as being discrete. If all members of the search space are continuous (in the set of real numbers), the search space is said to be continuous. If the elements of $\mathbf{x}$ include both discrete and continuous elements, the search space is said to be mixed. It is assumed that the function that we wish to optimize is denoted $\mathrm{f}(\mathbf{x})$. We will assume that $\mathrm{f}(\mathbf{x})$ returns a vector of dimension m of real numbers, that is, $\mathrm{f}(\mathbf{x}) \in \mathbb{R}^m$, where m is the number of objectives we are considering. For most of this chapter, we will merely consider $\mathrm{f}(\mathbf{x})$ to be a mathematical function for which we wish to identify the optimizer of; however, in Section 1.9, and in the rest of this book for that matter, we will focus on how to construct $\mathrm{f}(\mathbf{x})$ so as to serve as an instrument of engineering design.

For this section, let us focus on the case where all elements of $\mathbf{x}$ are real numbers so that $\mathbf{x} \in \mathbb{R}^n$, where $\mathbb{R}^n$ denotes the set of real numbers of dimension n and where the number of objectives is one (that is, $m = 1$) so that $\mathrm{f}(\mathbf{x})$ is a scalar function of a vector argument. Finally, let us suppose we wish to minimize $\mathrm{f}(\mathbf{x})$. A point $\mathbf{x}^*$ is said to be the global minimizer of f over Ω provided that

$$\mathrm{f}(\mathbf{x}^*) \leq \mathrm{f}(\mathbf{x}) \quad \forall x \in \Omega \backslash \{x^*\} \tag{1.2-1}$$

where $\forall$ is read as "for all" and $\Omega\backslash\{\mathbf{x}^*\}$ denotes the set Ω less the point $\mathbf{x}^*$. If the $\leq$ is replaced by $<$, then $\mathbf{x}^*$ is referred to as the strict global minimizer.

As stated previously, the function $\mathrm{f}(\mathbf{x})$ can have properties that make it easier or more difficult to find the global minimizer. Some of these properties are depicted in Figure 1.3. An example of a feature that makes it more difficult to find the global minimizer is a discontinuity as shown in Figure 1.3(a). Therein $\Omega = [x_{mn}, x_{mx}]$ and the discontinuity is at $x = x_a$. In this case, the discontinuity results in a point where the function's derivative is undefined. Since many optimization algorithms use the derivative of the function as part of the algorithm, such behavior can be

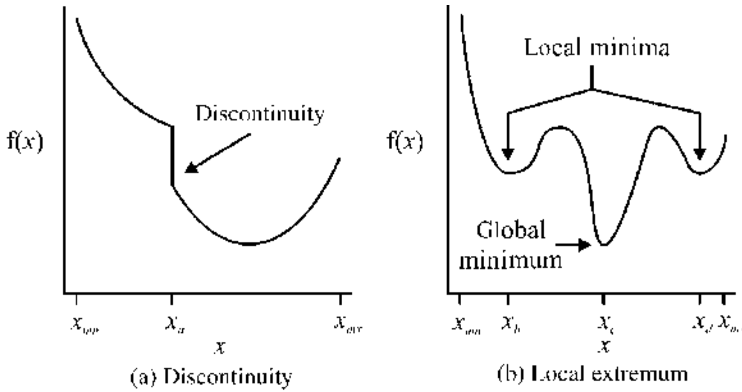

Figure 1.3 Function properties.

problematic. In general, any problem with a discrete or mixed search space will have a discontinuous objective.

Another property that can be problematic is the existence of local minima. These are illustrated in Figure 1.3(b). Therein $x = x_b$, $x = x_c$, and $x = x_d$ are all local minima; however, only the point $x = x_c$ is a global minimizer. Many minimization algorithms can converge to local minimizers and fail to find the global minimizer.

Related to the existence of local extrema is the convexity (or lack thereof) of a function. It is appropriate to begin the discussion of function convexity by considering the definition of a convex set. Let $\Theta \subset \mathbb{R}^n$ denote a set. A set is considered convex if all the points on the line connecting any two points in the set are also in the set. In other words, if

$$\alpha \mathbf{x}_a + (1 - \alpha)\mathbf{x}_b \in \Theta \quad \forall \alpha \in [0, 1] \tag{1.2-2}$$

for any two points $\mathbf{x}_a$, $\mathbf{x}_b \in \Theta$ then Θ is convex. This is illustrated in Figure 1.4 for $\Theta \subset \mathbb{R}^2$.

In order to determine if a function is convex, it is necessary to consider its epigraph. The epigraph of $\mathrm{f}(\mathbf{x})$ is simply the set of points greater than or equal to $\mathrm{f}(\mathbf{x})$. A function is considered convex if its epigraph is a convex set, as is shown in Figure 1.5. Note that this set will be in $\mathbb{R}^{n+1}$, where n is the dimension of $\mathbf{x}$.

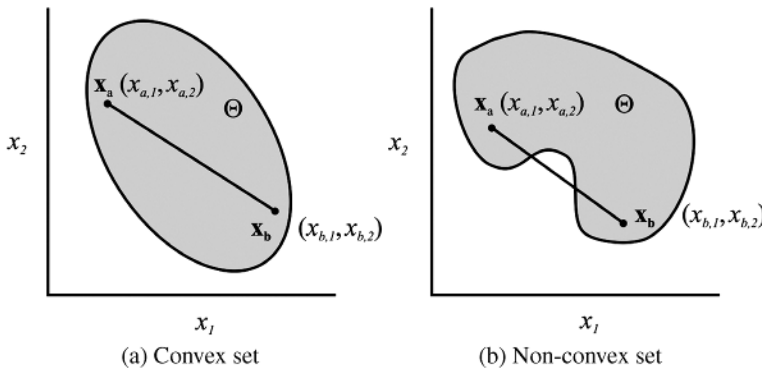

Figure 1.4 Definition of a convex set.

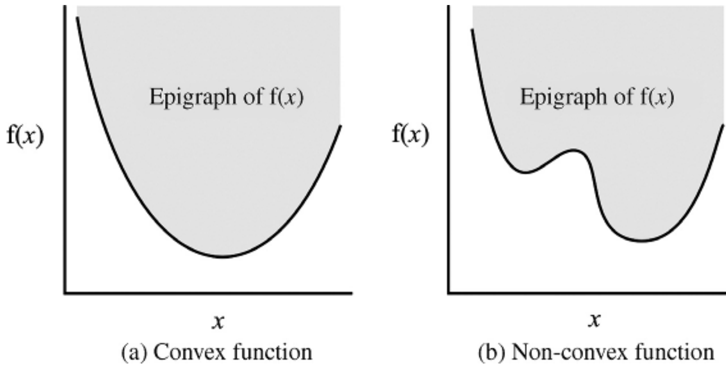

Figure 1.5 Definition of a convex function.

If the function being optimized is convex, the optimization process becomes much easier. This is because it can be shown that any local minimizer of a convex function is also a global minimizer. Therefore the situation shown in Figure 1.3(b) cannot occur. As a result, the minimization of continuous convex functions is straightforward and computationally tractable.

1.3 Single-Objective Optimization Using Newton's Method

Let us consider a method to find the extrema of an objective function $f(\mathbf{x})$. Let us focus our attention on the case where $f(\mathbf{x}) \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^n$. Algorithms to solve this problem include gradient methods, Newton's method, conjugate direction methods, quasi-Newton methods, and the Nelder–Mead simplex method, to name a few. Let us focus on Newton's method as being somewhat representative.

In order to set the stage for Newton's method, let us first define some operators. The first derivative or gradient of our objective function is denoted $\nabla f(\mathbf{x})$ and is defined as

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}^T \tag{1.3-1}$$

The second derivative or Hessian of $f(\mathbf{x})$is defined as

$$F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial^2 x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial^2 x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial^2 x_n} \end{bmatrix} \tag{1.3-2}$$

If $\mathbf{x}^*$ is a local minimizer of f, and if $\mathbf{x}^*$ is in the interior of Ω, it can be shown that

$$\nabla f(\mathbf{x}^*) = \mathbf{0} \tag{1.3-3}$$

and that

$$\mathrm{F}(\mathbf{x}^*) \geq 0 \tag{1.3-4}$$

Note that the statement $\mathrm{F}(\mathbf{x}^*) \geq 0$ means that $\mathrm{F}(\mathbf{x}^*)$ is positive semi-definite, which is to say that $\mathbf{y}^T\mathrm{F}(\mathbf{x}^*)\mathbf{y} \geq 0 \; \forall \, \mathbf{y} \in \mathbb{R}^n$. This second condition verifies that a minimum rather than a maximum has been found. It is important to understand that the conditions (1.3-3) and (1.3-4) are necessary but not sufficient conditions for $\mathbf{x}^*$ to be a mimimizer, unless f is a convex function. If f is convex, (1.3-3) and (1.3-4) are necessary and sufficient conditions for $\mathbf{x}^*$ to be a global minimizer.

At this point, we are posed to set forth Newton's method of finding function minimizers. This method is iterative and is based on a kth estimate of the solution denoted $\mathbf{x}[k]$. Then an update formula is applied to generate a (hopefully) improved estimate $\mathbf{x}[k+1]$ of the minimizer. The update formula is derived by first approximating f as a Taylor series about the current estimated solution $\mathbf{x}[k]$. In particular,

$$\begin{aligned} \mathrm{f}(\mathbf{x}) = \mathrm{f}(\mathbf{x}[k]) + \nabla\mathrm{f}(\mathbf{x}[k])(\mathbf{x}-\mathbf{x}[k]) + \ldots \\ \frac{1}{2}(\mathbf{x}-\mathbf{x}[k])^T\mathrm{F}(\mathbf{x}[k])(\mathbf{x}-\mathbf{x}[k]) + H \end{aligned} \tag{1.3-5}$$

where H denotes higher-order terms. Neglecting these higher-order terms and finding the gradient of $\mathrm{f}(\mathbf{x})$ based on (1.3-5), we obtain

$$\nabla\mathrm{f}(\mathbf{x}) \approx \nabla\mathrm{f}(\mathbf{x}[k]) + \mathrm{F}(\mathbf{x}[k])(\mathbf{x}-\mathbf{x}[k]) \tag{1.3-6}$$

From the necessary condition (1.3-3), we next take the right-hand side of (1.3-6) and equate it with zero; then we replace $\mathbf{x}$, which will be our improved estimate, with $\mathbf{x}[k+1]$. Manipulating the resulting expression yields

$$\mathbf{x}[k+1] = \mathbf{x}[k] - \mathrm{F}(\mathbf{x}[k])^{-1}\nabla\mathrm{f}(\mathbf{x}[k]) \tag{1.3-7}$$

which is Newton's method.

Clearly, Newton's method requires $\mathrm{f} \subset C^2$, which is to say that f is in the set of twice differentiable functions. Note that the selection of the initial solution, $\mathbf{x}[1]$, can have a significant impact on which (if any) local solution is found. Also note that method is equally likely to yield a local maximizer as a local minimizer.

Example 1.3A Let us apply Newton's method to find the minimizer of the function

$$\mathrm{f}(\mathbf{x}) = 2(x_1-2)^4 + 3(e^{x_2}-x_1)^2 + 8 \tag{1.3A-1}$$

This example is an arbitrary mathematical function; we will consider how to construct $\mathrm{f}(\mathbf{x})$ so as to serve a design purpose in Section 1.9. Inspection of (1.3A-1) reveals that the global minimum is at $x_1 = 2$ and $x_2 = \ln(2)$. However, let us apply Newton's method to find the minimum. Our first step is to obtain the gradient and the Hessian. From (1.3A-1), we have

$$\nabla\mathrm{f}(\mathbf{x}) = \begin{bmatrix} 8(x_1-2)^3 - 6(e^{x_2}-x_1) \\ 6(e^{x_2}-x_1)e^{x_2} \end{bmatrix} \tag{1.3A-2}$$

and

$$\mathrm{F}(\mathbf{x}) = \begin{bmatrix} 24(x_1-2)^2 + 6 & -6e^{x_2} \\ -6e^{x_2} & 12e^{2x_2} - 6x_1e^{x_2} \end{bmatrix} \tag{1.3A-3}$$

Table 1.1 Newton's Method Results

k	x[*k*]	f(x[*k*])	∇f(x[*k*])	F(x[*k*])
1	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	43	$\begin{bmatrix} -70 \\ 6 \end{bmatrix}$	$\begin{bmatrix} 102 & -6 \\ -6 & 12 \end{bmatrix}$
2	$\begin{bmatrix} 0.677 \\ -0.162 \end{bmatrix}$	14.2	$\begin{bmatrix} -19.6 \\ 0.888 \end{bmatrix}$	$\begin{bmatrix} 48.0 & -5.10 \\ -5.10 & 5.23 \end{bmatrix}$
3	$\begin{bmatrix} 1.11 \\ 0.0938 \end{bmatrix}$	9.25	$\begin{bmatrix} -5.53 \\ -0.0938 \end{bmatrix}$	$\begin{bmatrix} 24.9 & -6.58 \\ -6.58 & 7.13 \end{bmatrix}$
10	$\begin{bmatrix} 1.95 \\ 0.666 \end{bmatrix}$	8.00	$\begin{bmatrix} -2.23 \cdot 10^{-3} \\ 2.00 \cdot 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 6.07 & -11.7 \\ -11.7 & 22.7 \end{bmatrix}$

Let us arbitrarily take our initial estimate of the solution to be $\mathbf{x}[1] = [0 \quad 0]^T$. Table 1.1 lists the-numerical results from the repeated application of (1.3-7). As can be seen, during the first three iterations, the value of the function decreases rapidly. However, then the rate of reduction of the function slows. Observe that on the 10th iteration the value of the objective function is the minimum value to three significant digits, though there is still some discrepancy in the estimate of the minimizer. In this problem, the minimum is quite shallow, which reduces the speed of convergence.

Newton's method can be extremely effective on some problems, but prove problematic on others. For example, if $f(\boldsymbol{x})$ is not twice differentiable for some $\boldsymbol{x}$, difficulties arise since Newton's method requires the function, its gradient, and its Hessian. Many optimization methods require similar information and share similar drawbacks. There are optimization methods that do not require derivative information. One example is the Nelder–Mead simplex method. Even so, this algorithm can still become trapped at local minimizers if the function is not convex.

One feature that makes these methods susceptible to becoming trapped at a local minimum is that they take the approach of starting with a single estimated solution and attempt to refine that estimate. If the single estimate is close to a local extrema, it will tend to converge to that extrema. There is another class of optimization methods that are not based on a single estimate of the solution but on a large number (a population) of estimates. These population-based methods are not as susceptible to convergence to a nonglobal local extrema because there are a multitude of candidate optimizers.

Genetic algorithms (GAs) are a population-based optimization algorithms that have proven very effective in solving design optimization problems. Other population-based optimization methods, such as particle swarm optimization, have also been used successfully. While one can engage in a lengthy debate over which algorithm is superior, such a debate is unlikely to be fruitful. The focus of this text is on posing the design problem as a formal optimization problem; once the problem is so posed, any optimization algorithm can be used. A discussion of GAs is included herein in order to provide the reader with a background in at least one method that can be used for the optimization process.

1.4 Genetic Algorithms: Review of Biological Genetics

In this section, we will set the stage for the use of GAs as optimization engines by reviewing some principles of biological genetics. All living things have a set of instructions on how they are constructed. These instructions are written in the deoxyribonucleic acid (DNA) contained within each

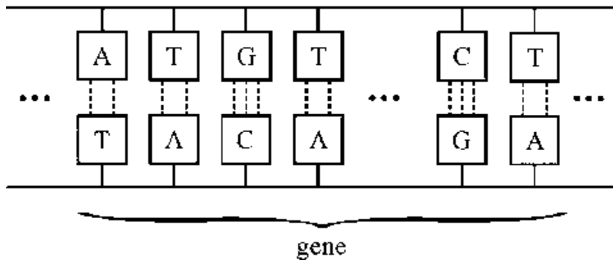

Figure 1.6 Deoxyribonucleic acid (DNA).

cell of that living being. The structure of this molecule was first determined by James Watson and Francis Crick in the 1950s and is depicted in Figure 1.6. Therein, the horizontal strands are made of phosphate and a sugar called deoxyribose. These strands are wound into a helix structure. The short vertical dashed lines in Figure 1.6 indicate weak hydrogen bonds that are instrumental in the duplication of DNA. The letters A and G stand for adenine and guanine, respectively, which are compounds known as purines. The letters T and C designate thymine and cytosine, which are pyrimidines. The combinations AT, TA, GC, and CG form a four-letter alphabet. A sequence of letters from this alphabet forms a gene of a living being. In terms of our discussion on design, we may view the gene as a design parameter of a living organism.

Each DNA molecule in a living organism is known as a chromosome. Living organisms generally have multiple chromosomes. For example, humans have 46 chromosomes per cell. These chromosomes are arranged into 22 pairs (one of each pair contributed by the father and one of each pair contributed by the mother). In addition, there are the two sex chromosomes denoted as X and Y. In humans and many other organisms, the existence of chromosomes in pairs leads to dominant and recessive genes, as discovered by Gregory Mendel, a Roman Catholic monk and botanist, who studied the propagation of traits in pea plants. However, not all living creatures have chromosomes organized in pairs; ants, wasps, and bees are haploid and have only one occurrence of each chromosome, while strawberries are octaploid with eight occurrences of each chromosome. This provides something to contemplate while eating strawberry pie.

In addition to some understanding of genes, chromosomes, and DNA, it is important to consider sexual reproduction and, in particular, the formation of gametes (sperm and egg cells). The formation of gametes is through a process known as meiosis, which is illustrated in Figure 1.7. Let us consider a diploid organism with two pairs of chromosomes, a long set and a short set as shown in Figure 1.7. This is consistent with chromosomes in cells that vary in length. The production of gametes begins with the chromosomes lengthening out within the cell as shown in Figure 1.7 (a). Note there are two copies of this chromosome, one contributed by the father (darkly shaded) and one contributed by the mother (lightly shaded). Next replication of the chromosomes occurs as seen in Figure 1.7(b). The two copies of a chromosome are referred to as chromatids, and they are connected at a point called the centromere. At this point, meiosis starts with the pairing of the chromosomes given by mother and father. In this pairing process, it is possible for the arms of the chromosomes to interchange, thereby leading to a new chromosome that consists of some genes from the father and some genes from the mother. Note that the crossover point is normally between genes; this is because of the large amount of (apparently) nonfunctional DNA in most chromosomes. While one crossover of one chromosome is shown, multiple crossovers in all chromosomes are possible.

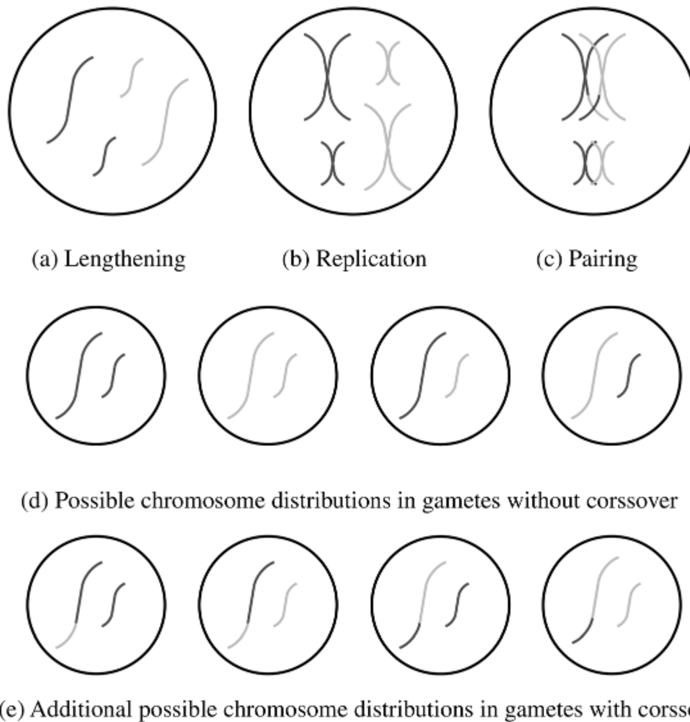

Figure 1.7 Meiosis.

After the chromosome pairing and crossover, the cell, which now contains four versions of each chromosome, splits into four cells. The four chromosomes segregate into these four cells. If crossover did not occur, this two-chromosome organism could produce four genotypes (genetically unique) of gametes; therefore, a single mother–father pair could produce sixteen genotypes. However, because of the crossover, many additional genotypes of gametes can be produced. In fact, because of crossover, the number of genotypes that can be produced becomes related to the number of genes, not just the number of chromosomes. Thus, crossover is very important in achieving genetic diversity.

Of course, in the case of humans, chromosome distribution alone with 23 pairs of chromosomes, yields $2^{23} = 8{,}388{,}608$ genotypes for the gametes. A set of parents could thus produce $8{,}388{,}608^2$ genetically different children, which would seem to be an impressively diverse set, even without crossover. However, in artificial GAs, the number of chromosomes is much smaller, often consisting of a single chromosome.

Beyond increasing the sheer number of genotypes of the gametes, crossover plays another critical role because it allows beneficial genes (traits) on a given chromosome to be decoupled from detrimental genes (traits). Crossover will play a very important role in the operation of GAs.

In addition to gamete diversity due to genetic crossover and chromosomal segregation into gametes, additional diversity is brought about because of mutation. Mutation arises from errors in copying DNA. In mitosis, or cell division, mutation often has little effect, since the mutated cell will often die. However, in meiosis, mutation can have a significant impact since the mutated genetic code will propagate into the genetic code of every cell in the child. Even then, many

mutations are not noticeable because they are a part of the genetic code that is unused. When mutation has a noticeable effect, it is generally for the worse. However, occasionally beneficial mutations occur which improve the ability of an individual (and eventually a species) to survive.

A final concept from biology that will serve our needs for an optimization engine is the idea of natural selection and the survival of the fittest, an idea stemming from Charles Darwin's voyages of the H.M.S Beagle, during a period of time roughly contemporary with the work of Mendel and the American Civil War. The idea that the most fit individuals of a population survive to reproduce is directly used in GAs. These algorithms are based on an explicit fitness function, which will be used to determine which individuals "survive" and will be placed into a mating pool.

Clearly, the discussion in this section is at a high level and has been greatly simplified. The interested reader is referred to Crow [2] for a more thorough introduction to the topic.

1.5 The Canonical Genetic Algorithm

A century after the work of Mendel and Darwin, but a mere decade after the work of Watson and Crick, John Holland, a professor at the University of Michigan, proposed using the principles of biological genetics as a computation algorithm for optimization, a concept instantiated by a GA [3]. In this section, we will begin our consideration of GAs with a canonical GA similar to Holland's original vision.

GAs are quite different from traditional optimization algorithms. First of all, GAs operate not on the argument of the function being optimized, but rather on an encoding of the argument. Second, rather than iterating to improve an estimate for an optimizer, GAs iterate to improve a large number of different estimates of the optimizer. This collection of estimates will be referred to as a population. The use of a population of estimated solutions improves the chances of finding a global optimum. Third, GA operations are based only on the values of the objective function—gradients and Hessians are not used, nor even estimated. This property is useful in function with discontinuities or with a discrete or mixed search space. Finally, GA operations are based on probabilistic rather than deterministic computations.

The first concept that must be set forth in a GA is that it, like evolution, operates on a population, not on an individual. We will denote the population within the GA as $\mathbf{P}[k]$, where k is the generation number. The kth generation consists of a number of individuals, that is,

$$\mathbf{P}[k] = \left\{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \cdots \boldsymbol{\theta}^{N_p}\right\} \tag{1.5-1}$$

where $\boldsymbol{\theta}^i$ is the genetic code for the ith individual in the kth generation of the population and where N_p denotes the number of individuals in the population, which should be an even number. The genetic code for the ith individual may be organized as

$$\boldsymbol{\theta}^i = \begin{bmatrix} \text{chromosome } 1 & \left\{\begin{matrix} \boldsymbol{\theta}_1^i \\ \boldsymbol{\theta}_2^i \end{matrix}\right. \\ \text{chromosome } 2 & \left\{\begin{matrix} \boldsymbol{\theta}_3^i \\ \boldsymbol{\theta}_4^i \\ \boldsymbol{\theta}_5^i \end{matrix}\right. \\ & \vdots \\ \text{chromosome } N_c & \left\{ \boldsymbol{\theta}_{N_g}^i \right. \end{bmatrix} \tag{1.5-2}$$

where N_c is the number of chromosomes, N_g is the number of genes, and $\boldsymbol{\theta}_j^i$ is the jth gene of the ith individual (and it is understood that we are referring to the kth generation). Each gene is a string sequence. Recall that DNA consists of alphabet AT, TA, CG, and GC. In the case of the canonical GA, the string is most typically a binary sequence. Thus, $\boldsymbol{\theta}^i$ takes the form of a binary number. The significance of the chromosome organization will come into play when we consider reproduction.

The fact that the genes are encoded results in a limitation of the domain of the parameter vector. In other words, the domain of possible values of each element of the parameter vector is inherently limited. In some cases, this property is very convenient, but in other cases this limitation on the domain of the parameter vectors is disadvantageous.

Associated with the genetic code for each population member, we will have a decoding function that translates the genetic code into a parameter vector. In particular,

$$\mathbf{x}^i = \mathrm{d}\left(\boldsymbol{\theta}^i\right) \tag{1.5-3}$$

where $\mathbf{x}^i$ is the parameter vector of the ith member of the population and is structured as

$$\mathbf{x}^i = \begin{bmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_{N_g}^i \end{bmatrix} \tag{1.5-4}$$

As can be seen, $\mathbf{x}^i$ has one element (denoted with a subscript) for each gene. However, it is not partitioned into chromosomes.

Based on the parameter vector of ith population member, the objective function can be evaluated. In particular,

$$f^i = \mathrm{f}\left(\mathbf{x}^i\right) \tag{1.5-5}$$

In the case of a GA, the objective function is referred to as a fitness function. It will be used in a "survival of the fittest" sense to determine which members of the population will mate to form the next generation. In the context of a GA, fitness is viewed in a positive sense, thus it is assumed that we wish to maximize the fitness function. Fortunately, it is a straightforward matter to convert between maximization of a function and the minimization of a function. At this point, we have enough background to discuss the computational aspects of a GA. However, before doing this, it is appropriate to briefly pause in our development and consider an example.

Example 1.5A Suppose the 13th member of the population has the genetic code

$$\boldsymbol{\theta}^{13} = \Big[\underbrace{0\ \ 1\ \ 0}_{\text{gene1}}\ \ \underbrace{1\ \ 1\ \ 0\ \ 1}_{\text{gene2}}\ \ \underbrace{0\ \ 1\ \ 1}_{\text{gene3}}\Big]^T \tag{1.5A-1}$$

The decoding algorithm is on a gene-by-gene basis and is of the form

$$x_i = x_{mn,i} + \frac{x_{mx,i} - x_{mn,i}}{2^{l_i} - 1} \sum_{m=1}^{l_i} b_m 2^{m-1} \tag{1.5A-2}$$

where l_i is the number of bits of the ith gene, m is an index ranging from least significant (rightmost) bit to most significant (leftmost) bit for a given gene, b_m is the value of that bit (0 or 1), and $x_{mn,i}$ and $x_{mx,i}$ are the minimum and maximum values of the ith element of the parameter vector. For the

problem at hand, we assume $x_{mn,1} = 5$, $x_{mx,1} = 10$, $x_{mn,2} = -2$, $x_{mx,2} = 0$, $x_{mn,3} = 0$, and $x_{mx,3} = 1$. In Section 1.9, we will formally consider the construction of fitness functions, and in Section 1.10, we will consider an engineering example. However, for the moment we will assume a purely mathematical fitness function given by

$$\mathrm{f}(\mathbf{x}) = \frac{1}{(x_1 - 1)^2 + (x_2 + 2)^2 + x_3^2 + 1} \tag{1.5A-3}$$

Our goal is to compute the fitness of the 13th member of the population.

The solution to this problem is straightforward. Using (1.5A-2), we obtain $\mathbf{x}_1^{13} = 6.43$, $\mathbf{x}_2^{13} = -0.267$, and $\mathbf{x}_3^{13} = 0.429$. Substitution of these values into (1.5A-3) yields $f^{13} = 0.0297$. Note that while the division of the bits of (1.5A-1) into genes is very important to determine the fitness, the organization of genes into chromosomes is irrelevant for fitness evaluation and so has not been specified in this example.

At this point, we can now consider the primary aspects of a GA. These are illustrated in Figure 1.8. Therein, the first step is initialization that yields an initial population denoted $\mathbf{P}[1]$. Next, the fitness of every member of the population is evaluated. Based on the fitness, a mating pool $\mathbf{M}[k]$ is determined by the selection process. The individuals in this population will mate and genetic operators such as crossover, segregation, and mutation will be used to produce children who will form the next generation $\mathbf{P}[k + 1]$. A stopping criterion is then checked; this can be as simple as checking a generation number. Once the stopping criterion is met, the algorithm concludes by selection of the most fit individual of the final population to be the optimizer. This process is implemented by the argmax() operator, which returns the argument that maximizes its objective (which, in this case, is carried about by inspection of the finite population).

It is now appropriate to consider each of these operations in more detail, beginning with the initialization step. The genetic code of every member of the population is initialized at random. This yields an initial population of designs, denoted $\mathbf{P}[1]$. The next step in the algorithm is to compute

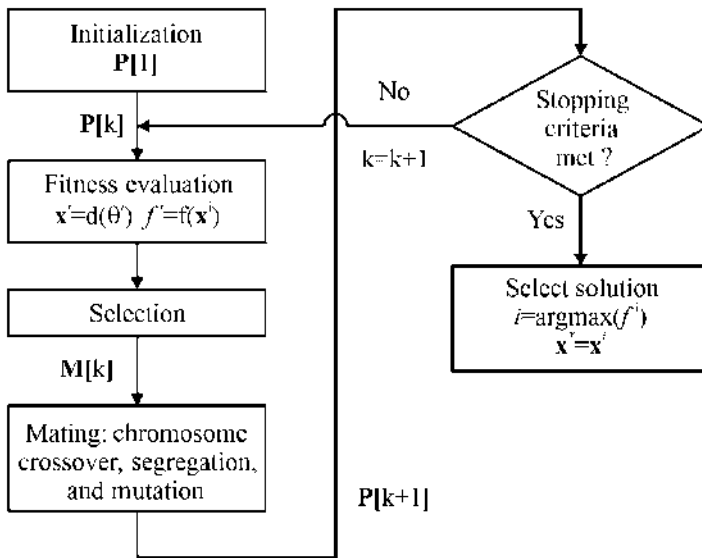

Figure 1.8 Canonical genetic algorithm.

the fitness of every member of the population. This is accomplished by applying (1.5-3) and (1.5-5) to every member of the population. Example 1.5A illustrates this step for a single member of the population.

The next step in the process is selection. In this step, members of the population $\mathbf{P}[k]$ are placed into the mating pool $\mathbf{M}[k]$. Two algorithms to do this are roulette wheel selection and tournament selection. In both methods, the mating pool is initially empty and is filled one member at a time by repeatedly applying the selection mechanism. In roulette wheel selection, members of the population are drawn into the mating pool with a probability proportional to their fitness. In particular, the probability of individual i being drawn into the mating pool on a given draw is given by

$$p^i = \frac{f^i}{\sum_{i=1}^{N_p} f^i} \tag{1.5-6}$$

When applying this particular algorithm, it is important that the fitness function be constructed so that $f^i \geq 0$. If this is not the case, it is possible to scale/adjust the fitness so that the condition is satisfied (for example, by adding a constant). Note that once a population member has been copied to the mating pool, it is not removed from the population. Thus, it can be copied to the mating pool multiple times.

In n-way tournament selection, n members of the population are selected at random, and the member of this subset with the highest function is put into the mating pool. Here again, the member placed into the mating pool is not removed from the population. In tournament selection, there is no restriction on the range of the fitness function, which provides a slight simplification.

The next step in process is mating, which is comprised of chromosome crossover, segregation, and mutation. In this step, pairs of parents are used to create pairs of children. Pseudo-code for this step appears in Table 1.2. Therein, underlined text denotes comments. Referring to the

Table 1.2 Pseudo-Code for Mating, Crossover, Segregation, and Mutation

```
for i =1 to Np/2

      compute element indices
      i1 = 2i - 1
      i2 = 2i
      get genetic codes of parents
      θp1 = i1th individual in M[k]
      θp2 = i2th individual in M[k]

      apply genetic operators
      apply crossover to {θp1,θp2} yielding {θa1,θa2}
      segregate chromosomes of {θa1,θa2}yielding{θb1,θb2}
      apply mutation to {θb1,θb2} yielding {θc1,θc2}

      place children into next population
      θc1 becomes the i1th individual of P[k + 1]
      θc2 becomes the i2th individual of P[k + 1]
end
```

pseudo-code, $\boldsymbol{\theta}^{p1}$ and $\boldsymbol{\theta}^{p2}$ denote the genetic code from parents taken from the mating pool. A crossover operator is applied to the codes to form intermediate genetic codes $\boldsymbol{\theta}^{a1}$ and $\boldsymbol{\theta}^{a2}$. Crossover occurs at random points within a given chromosome. If it occurs, then all elements of the portion of the chromosome strings of the two parents past the crossover point are interchanged. This is similar to biological crossover but not identical: In the case of biological crossover, this process occurs in the formation of the gametes. The net result is the same. Next, the chromosomes of $\boldsymbol{\theta}^{a1}$ and $\boldsymbol{\theta}^{a2}$ are randomly segregated to form the next stage of intermediate codes $\boldsymbol{\theta}^{b1}$ and $\boldsymbol{\theta}^{b2}$, much as the chromosome pairs of a cell are segregated in forming gametes. Genetic codes $\boldsymbol{\theta}^{b1}$ and $\boldsymbol{\theta}^{b2}$ are then mutated to form $\boldsymbol{\theta}^{c1}$ and $\boldsymbol{\theta}^{c2}$, which will be the children. The mutation operator consists of random interchanges of 0 and 1. The final step in the process is that the children $\boldsymbol{\theta}^{c1}$and $\boldsymbol{\theta}^{c2}$ are placed into the next generation $\mathbf{P}[k+1]$. It should be noted that in many GAs there is only a provision to have one chromosome, in which case the chromosome segregation process does not come into play and so genetic diversity is brought about solely by crossover and mutation.

Example 1.5B Let us consider the crossover, segregation, and mutation processes on a numerical example. Figure 1.9 illustrates these operators. Therein $g_1 - g_4$ denote genes 1–genes 4 and c_1 and c_2 denote chromosomes 1 and 2, respectively. We start at the left of the diagram with the two parents $\boldsymbol{\theta}^{p1}$ and $\boldsymbol{\theta}^{p2}$. The first operator applied is crossover. Assuming the crossover point is between the first and second bits in the first chromosome, and that no crossover occurs in the second chromosome, we arrive at $\boldsymbol{\theta}^{a1}$ and $\boldsymbol{\theta}^{a2}$. The portions of the genetic code which have been acted on by the crossover operator are shown in bold. Next, chromosome segregation occurs. In this case, by a random choice the first child inherits the first chromosome from $\boldsymbol{\theta}^{a2}$ and the second chromosome from $\boldsymbol{\theta}^{a1}$. The second child inherits the remaining chromosomes. Next, let us assume that mutation acts to change the status of the first bit of gene 3 in $\boldsymbol{\theta}^{b2}$. This yields $\boldsymbol{\theta}^{c1}$ and $\boldsymbol{\theta}^{c2}$, where the affected bit is again indicated in boldface.

At this point, the main points of a canonical GA have been described. This canonical form is the type of algorithm first developed by Holland. The canonical GA can also be used to explain the effectiveness of GAs using schema theory. A schema is a pattern of bits; one reason GAs are so effective is that they can be shown to exhibit an implicit parallelism in which all schema that result in higher than average fitness tend to propagate. The interested reader is referred to textbooks on GAs such as those by Goldberg [4,5].

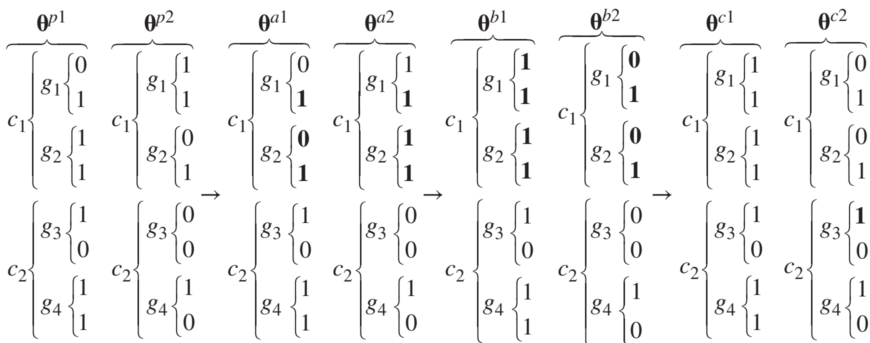

Figure 1.9 Chromosome crossover, segregation, and mutation in Example 1.5B.

At this point, an example would prove useful. Before embarking on such an example, however, it will prove useful to address one weakness of the GA. This weakness is the encoding algorithm. Although the representation of a member of the population as a string similar to that in DNA is intellectually appealing, it is also inconvenient. Further, a binary representation of real numbers suffers from finite resolution and the fact that sometimes large numbers of bits may need to change to accomplish a small change unless, for example, a Gray code scheme is used. As it turns out, it is possible to create a GA without encoding genes as strings and, instead, representing them with real numbers. Real-coded GAs are simpler to write than their canonical counterparts and very effective. They will be our next topic.

1.6 Real-Coded Genetic Algorithms

Real-coded GAs are very similar to canonical GAs except that instead of each gene being represented as a binary string, each gene is represented by a real number. This proves convenient for coding purposes and makes representing each gene to the numerical precision of floating-point numbers for a given machine (computer) straightforward. Beyond the change of the way in which a gene is represented, the algorithm presented in Figure 1.8 is still applicable, though we will need to modify the encoding, crossover, and mutation operators.

Encoding

Let us begin our development by considering the form of the genetic code for the ith individual. In the case of the canonical GA, this was given by (1.5-2), where each element of $\boldsymbol{\theta}^i$ was represented by a string. In the real-coded GA, (1.5-2) still applies. However, instead of each element of $\boldsymbol{\theta}^i$ being represented by a string, in a real-coded GA each element is represented by a real number. In fact, a real-coded GA could be written such that $\boldsymbol{\theta}^i = \mathbf{x}^i$. However, it will be convenient to provide a mapping of $\mathbf{x}^i$ to $\boldsymbol{\theta}^i$ so that the gene values of $\boldsymbol{\theta}^i$ fall into the domain [0,1].

The mapping between $\mathbf{x}^i$ and $\boldsymbol{\theta}^i$ is accomplished on a gene-by-gene basis. A simple choice is a linear map. Let x and θ denote a gene (element) of the $\mathbf{x}^i$ and $\boldsymbol{\theta}^i$. For a linear mapping, we have

$$x = \mathrm{d}_j(\theta) \tag{1.6-1}$$

where j denotes the gene number and

$$\mathrm{d}_j(\theta) = x_{mn,j} + \left(x_{mx,j} - x_{mn,j}\right)\theta \tag{1.6-2}$$

where $x_{mn,j}$ and $x_{mx,j}$ denote the minimum and maximum values of the parameter.

Closely related to linear mapping is integer mapping, wherein x, $x_{mn,j}$, and $x_{mx,j}$ are integers. In this case, (1.6-2) still applies, but we require

$$\theta \in \left\{ \frac{0}{x_{mx,j} - x_{mn,j}}, \frac{1}{x_{mx,j} - x_{mn,j}}, \cdots, \frac{x_{mx,j} - x_{mn,j}}{x_{mx,j} - x_{mn,j}} \right\} \tag{1.6-3}$$

This mapping is useful in choosing, for example, between different types of steel in a design. If the third gene represented the type of steel used, and five types of steels were being considered, then $x_{mn,3} = 1$, $x_{mx,3} = 5$, and $\theta \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$.

In some cases, the domain of a parameter may span many decades in magnitude. If the domain of the parameter is always positive, then a logarithmic mapping is appropriate. In this case,

$$\mathrm{d}_j(\theta) = x_{mn,j}\left(\frac{x_{mx,j}}{x_{mn,j}}\right)^{\theta} \tag{1.6-4}$$

Crossover

In the case of the canonical GA, crossover is accomplished by breaking the subject chromosomes of the parents at the same point and interchanging them to form the corresponding chromosomes of the children. This interchange substantially alters the gene where the chromosome is interchanged, and it results in an interchange of genes falling after the interchange point.

There are several algorithms to achieve crossover in real-coded GAs. One of them is single-point crossover, which can be readily generalized to n-point crossover. This method is straightforward and is readily illustrated by the example shown in Figure 1.10. Therein, the genetic code for two parents, $\boldsymbol{\theta}^{p1}$ and $\boldsymbol{\theta}^{p2}$, is shown. Observe that the elements of these vectors are real numbers in the domain [0,1]. In this algorithm, a crossover point is chosen at random and the genes after the crossover points are interchanged (these genes are in bold) to form two new genetic codes, $\boldsymbol{\theta}^{a1}$ and $\boldsymbol{\theta}^{a2}$, which will become children after the application of additional genetic operators such as mutation.

Single-point crossover is very similar to biological crossover. However, note that gene values cannot become altered using this operator. This limits the amount of genetic diversity that can be brought about.

Simple-blend crossover can be used to increase the genetic diversity. Let us consider the jth gene of parents $\boldsymbol{\theta}^{p1}$ and $\boldsymbol{\theta}^{p2}$. If the jth gene is being crossed over, the new gene values are determined by first computing a random number υ given by

$$\upsilon = \mathrm{U}(\cdot) \tag{1.6-5}$$

where $\mathrm{U}(\cdot)$ denotes a random number generator that generates a uniformly distributed random number in the range [0,1]. Next, the gene values of the children are set to

$$\boldsymbol{\theta}_j^{a1} = \frac{1}{2}\left(\boldsymbol{\theta}_j^{p1} + \boldsymbol{\theta}_j^{p2}\right) + \alpha\left(\boldsymbol{\theta}_j^{p2} - \boldsymbol{\theta}_j^{p1}\right)\upsilon \tag{1.6-6}$$

$$\boldsymbol{\theta}_j^{a2} = \frac{1}{2}\left(\boldsymbol{\theta}_j^{p1} + \boldsymbol{\theta}_j^{p2}\right) - \alpha\left(\boldsymbol{\theta}_j^{p2} - \boldsymbol{\theta}_j^{p1}\right)\upsilon \tag{1.6-7}$$

where α is an algorithm constant often taken to be 0.5. Observe that in this algorithm the average gene value for the children is the same as for the parents. Canonical GAs also have this property.

Another commonly used crossover algorithm is simulated binary crossover. This algorithm is designed to yield results that would be similar to those obtained by crossover within a gene using

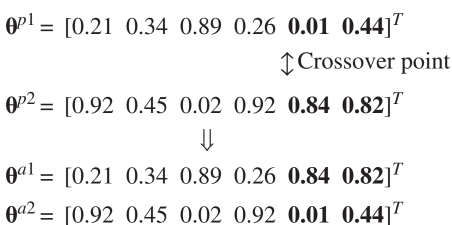

Figure 1.10 Single-point crossover.

the canonical GA. These properties include the fact that the sum of the gene values is the same for the children as the parents and that the children tend to be similar to the parents (which is also the case in simple-blend crossover). In this algorithm, first a random number in [0,1) is computed as in (1.6-5) from which a factor β is computed as

$$\beta = \begin{cases} (2\upsilon)^{\frac{1}{\eta_c+1}} & \upsilon \leq \frac{1}{2} \\ \left(\frac{1}{2(1-\upsilon)}\right)^{\frac{1}{\eta_c+1}} & \upsilon > \frac{1}{2} \end{cases} \tag{1.6-8}$$

In (1.6-8), η_c is an algorithm parameter. Once β is found, the genes of the children are computed as

$$\boldsymbol{\theta}_j^{a1} = \frac{1}{2}\left[(1+\beta)\boldsymbol{\theta}_j^{p1} + (1-\beta)\boldsymbol{\theta}_j^{p2}\right] \tag{1.6-9}$$

$$\boldsymbol{\theta}_j^{a2} = \frac{1}{2}\left[(1-\beta)\boldsymbol{\theta}_j^{p1} + (1+\beta)\boldsymbol{\theta}_j^{p2}\right] \tag{1.6-10}$$

In both simple blend and simulated binary crossover, it is possible to generate gene values outside of [0,1]. For example, suppose the parent gene values are 0.5 and 1, and we use simple-blend crossover with $\alpha = 0.8$. Further, suppose $\upsilon = 0.9$. The resulting gene values for the children would be 0.39 and 1.11, the latter of which is outside the allowed set of values. In this case, gene repair becomes necessary. There are several approaches to this. Hard limiting the gene values is one approach. For example, a gene value calculated as 1.2 is simply represented as 1.0. Similarly, a gene value of −2.1 would be set to 0. Alternately, gene values can be ring mapped, which is to say corrected using a modulus 1 operator. In this case, 1.2 would be repaired to 0.2 and −2.1 would be repaired to 0.9. For integer-coded genes, additional repair is necessary in order to make sure that the gene values take on allowed values.

Our discussion of simple blend and simulated binary crossover has focused at the gene level, and so it is now appropriate to consider the application of these processes at the chromosome level. There are several approaches that could be taken. In scalar crossover, each gene in a chromosome that is being operated upon undergoes a crossover operation independently. Each gene in a chromosome uses a different υ. In vector crossover, υ is the same for all genes in a crossover. This leads to scalar simple-blend crossover, vector simple-blend crossover, scalar simulated binary crossover, and vector simulated binary crossover.

As can be seen, it is perhaps too easy to create new genetic operators. However, before leaving the topic, one final combination will be considered, namely, single-point simple-blend crossover. In this algorithm, a single gene of the chromosome is selected as a crossover point. Unlike single-point crossover, however, the crossover point is a gene, not a position between genes. That gene is operated upon with the simple blend operator. The remaining genes are interchanged as in single-point crossover. The process is illustrated in Figure 1.11. Therein, the third gene is randomly selected as

Figure 1.11 Single-point simple-blend crossover.

$$\boldsymbol{\theta}^{p1} = [0.21\ \ 0.34\ \ 0.89\ \ 0.26\ \ 0.01\ \ 0.44]^T$$
$$\boldsymbol{\theta}^{p2} = [0.92\ \ 0.45\ \ 0.02\ \ 0.92\ \ 0.84\ \ 0.82]^T$$

↕ Crossover point

$$\boldsymbol{\theta}^{a1} = [0.21\ \ 0.34\ \ 0.10\ \ 0.92\ \ 0.84\ \ 0.82]^T$$
$$\boldsymbol{\theta}^{a2} = [0.92\ \ 0.45\ \ 0.81\ \ 0.26\ \ 0.01\ \ 0.44]^T$$

the crossover point, α is taken as 0.5, and υ was 0.81 (determined at random). Single-point simple-blend crossover is somewhat similar to the inheritance of quantitative traits wherein multiple genes are involved in determining the extent of an attribute. One could also devise single-point simulated binary crossover or multipoint simple-blend crossovers in a straightforward fashion.

Mutation

In the process of the duplication of chromosomes to form gametes, occasionally errors in DNA duplication occur, which are referred to as mutations. Normally, particularly in mature species, these mutations have either little effect or a harmful effect. However, occasionally mutations yield beneficial traits. This is particularly true in the early evolution of a species. Mutation in GAs helps to explore the parameter space; in doing so, many mutations are harmful, but occasionally mutations occur which are very beneficial.

Referring to the pseudo-code in Table 1.2, genes are selected at random for mutation. Many chromosomes will have no mutations, and others may have several. There are several approaches that can be applied to mutate a gene. One approach, referred to as total mutation, is to simply reinitialize the mutated gene at random in the interval [0,1]. Figure 1.12 illustrates the total mutation operator applied to the third gene of a chromosome (which could be either *a1* or *a2* with respect to Table 1.2).

A second method is partial absolute mutation. In this method, if the jth gene is selected for mutation, the mutated value is given by

$$\boldsymbol{\theta}_j^b = \boldsymbol{\theta}_j^a + \sigma \mathrm{N}(\cdot) \tag{1.6-11}$$

where $\mathrm{N}(\cdot)$ denotes a zero-mean, unity variance Gaussian random number generator and σ is a desired standard deviation. A related method is partial relative mutation wherein the modified gene value may be expressed as

$$\boldsymbol{\theta}_j^b = \boldsymbol{\theta}_j^a(1 + \sigma \mathrm{N}(\cdot)) \tag{1.6-12}$$

In partial absolute mutation, the size of the mutation is independent of the value of the gene; in relative mutation, the size of the perturbation tends to increase with the magnitude of the coded gene value. In both of these methods, gene repair is necessary because either method could result in a gene value outside of [0,1].

There are also vector-based mutation operators. In absolute vector mutation, we have

$$\boldsymbol{\theta}^b = \boldsymbol{\theta}^a + \sigma \mathrm{N}(\cdot)\mathbf{V}(\cdot) \tag{1.6-13}$$

where $\mathbf{V}(\cdot)$ denotes a unit vector with the same dimensionality as the chromosome in a random direction. For relative vector mutation, we have

$$\boldsymbol{\theta}_j^b = \boldsymbol{\theta}_j^a\left(1 + \sigma \mathrm{N}(\cdot)\mathbf{V}_j(\cdot)\right) \quad \forall j \in \left[1, 2, \ldots, N_g\right] \tag{1.6-14}$$

where N_g is the number of genes in the chromosome in question. Both of these operators require gene repair.

$$\boldsymbol{\theta}^a = [0.21\ \ 0.34\ \ \mathbf{0.89}\ \ 0.26\ \ 0.84\ \ 0.82]^T$$
$$\Downarrow \text{Mutation}$$
$$\boldsymbol{\theta}^b = [0.92\ \ 0.45\ \ \mathbf{0.25}\ \ 0.25\ \ 0.84\ \ 0.82]^T$$

Figure 1.12 Total mutation.

A final mutation operator we will consider is integer mutation. Recall that in integer coding, we are representing integers as real numbers and mapping them to a discrete set of values within [0,1]. In the case of integer-coded genes, the mutation operators just described are not appropriate. Thus, it is convenient to use a total mutation of these genes with the result discretized to the allowed values.

The mutation operators just described all have uniform and nonuniform versions. In uniform versions, the parameters of the algorithms (the mutation rates, and standard deviations of mutation amount) are constant with respect to generation number. In nonuniform mutation, these rates vary. Normally, high mutation rates and large standard deviations are used at the beginning of evolution while smaller rates and standard deviations are used toward the end of the study when the population is mature from an evolutionary point of view.

Example 1.6A Let us illustrate the use of an elementary GA based on a real-coded version of Figure 1.8. In particular, let us attempt to find the value of $\mathbf{x}$ that maximizes the function

$$f(\mathbf{x}) = \frac{1}{(x_1x_2 - 6)^2 + 4(x_2 - 3)^2 + 1} \tag{1.6A-1}$$

For the purposes of solving this problem, we will use linear coding with $x_{mn,1} = x_{mn,2} = 0$ and $x_{mx,\,1} = x_{mx,\,2} = 5$. In order to form the mating pool, we will use two-way tournament selection. We will also assume simple-blend crossover with $\alpha = 0.5$ and a probability of crossover of 0.5. We will use total mutation with the probability of a gene mutation of 0.1. In this example, we will consider three generations with a population size of 8. The code for this example can be obtained from Sudhoff [6].

Table 1.3 lists the data for the evolution. The first population's parameters were determined by random initialization. Next, the parameter vector for each member of the population was decoded yielding $\mathbf{x}^i$ for every member of the population. This is then evaluated by calculating $f^i = \mathrm{f}(\mathbf{x}^i)$, where the fitness is given by (1.6A-1). At this point, the maximum, median, and mean fitness values of the population are 0.2213, 0.04123, and 0.06940, respectively. The next step is the creation of the mating pool using tournament selection. Applying simple-blend crossover, mutation, and associated gene repair operators yields the second population. Decoding to obtain the parameter vector for each element and then evaluating the fitness, it can be seen that maximum and mean fitness values of the population have increased to 0.4886 and 0.09160 while the median fitness is decreased to 0.03661. Repeating the process a third time, the maximum, median, and mean fitness increase to 0.7719, 0.08486, and 0.1774, respectively. After the third generation, taking the individual with the highest fitness yields $x^* = [1.942 \quad 3.233]^T$. This is starting to approach the correct solution, which is $x = [2 \quad 3]^T$. Note, however, that a different run will produce different results since a population size of 8 over three generations is inadequate to find a consistent solution.

Before concluding this example, it is interesting to repeat it with a more substantial population size and over a larger number of generations. Figure 1.13 illustrates an optimization run over eight generations with a population size of 25 individuals. In Figure 1.13(a), the best fitness in the population, median fitness of the population, and mean fitness of the population are shown versus generation. All three of these quantities can be seen to increase with generation number, though not monotonically. Figure 1.13(b) depicts a plot of the individuals in terms of their parameter vectors. Therein, an o, x, +, ∗, star, diamond, triangle, and pentagram depict members of the first through eighth generations, respectively. In addition, the darkness of the shading of each point is

Table 1.3 Example of Real-Coded Genetic Algorithm Evolution

				Generation 1				
P[1]	0.3210 0.8296	0.8222 0.5707	0.5718 0.2860	0.6991 0.7963	0.4416 0.4462	0.4657 0.2790	0.6754 0.9037	0.9085 0.7472
$\mathbf{x}^i$	1.6051 4.1478	4.1109 2.8534	2.8591 1.4301	3.4957 3.9813	2.2079 2.2311	2.3283 1.3952	3.3769 4.5183	4.5426 3.7360
f^i	0.1492	0.0295	0.0689	0.0148	0.2213	0.0530	0.0104	0.0081
M[1]	0.5718 0.2860	0.4657 0.2790	0.8222 0.5707	0.3210 0.8296	0.4657 0.2790	0.8222 0.5707	0.4416 0.4462	0.3210 0.8296
				Generation 2				
P[2]	0.5718 0.2860	0.1355 0.3321	0.8975 0.7424	0.6534 0.6579	0.4657 0.2790	0.5279 0.0321	0.3627 0.6971	0.8467 0.5786
$\mathbf{x}^i$	2.8591 1.4301	0.6774 1.6606	4.4874 3.7118	3.2668 3.2895	2.3283 1.3952	2.6394 0.1604	1.8133 3.4857	4.2336 2.8932
f^i	0.0689	0.0313	0.0086	0.0419	0.0530	0.0155	0.4886	0.0249
M[2]	0.1355 0.3321	0.5718 0.2860	0.3627 0.6971	0.5718 0.2860	0.8467 0.5786	0.8467 0.5786	0.5718 0.2860	0.6534 0.6579
				Generation 3				
P[3]	0.1355 0.3321	0.5718 0.2860	0.5462 0.3365	0.3883 0.6467	0.8467 0.5786	0.8467 0.5786	0.6235 0.5216	0.6017 0.4223
$\mathbf{x}^i$	0.6774 1.6606	2.8591 1.4301	2.7308 1.6824	1.9417 3.2334	4.2336 2.8932	4.2336 2.8932	3.1174 2.6082	3.0085 2.1114
f^i	0.0313	0.0689	0.1008	0.7719	0.0249	0.0249	0.1625	0.2335

proportional to generation number. Thus, it can be seen that as the evolution progresses, the population moves toward the objective function maximizer. At the end of the run, the estimate of the solution is $x^* = [2.133 \quad 2.755]^T$, which yields $f(x^*) = 0.797$. With the given limited number of generations and small population size, the results vary from run to run. This variance could be

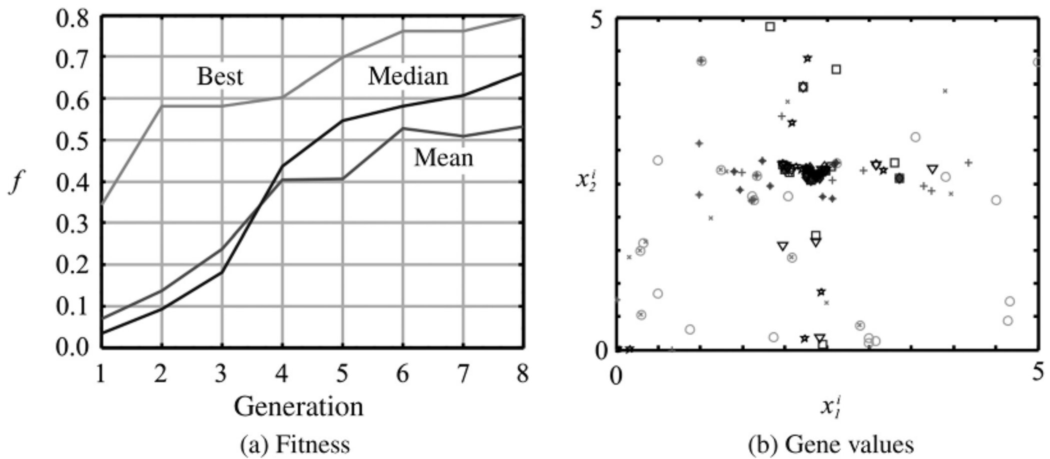

Figure 1.13 Fitness and gene values for Example 1.6A.

eliminated with a larger population and more generations, but it would make the resulting figures (particularly Figure 1.13(b)) hard to trace.

While Example 1.6A helped to illustrate the operation of a real-coded GA, practical GAs typically have several additional features to increase their efficacy. We will now briefly consider some of these additional operators.

Scaling

Scaling the fitness values of a population can result in improved algorithm performance. Scaling algorithms are only used in the context of roulette wheel selection. Consider a situation early in an evolution. Suppose individual A is significantly more fit than the remainder of the population. In this case, many copies of individual A will become part of the mating pool. In fact, without scaling, copies of individual A can rapidly dominate the population, leading to premature convergence and a failure to fully explore the search space. In this case, the scaling algorithm could be used to reduce the fitness of individual A so that it does not become as common as quickly, permitting the evolution to explore other avenues.

Conversely, late in the evolution, suppose that individual B is slightly better than the rest of the population. Because the fitness of most individuals is quite good, the chances of individual B being put into the mating pool are not much more than that of an average individual. In this case, it is appropriate to scale the fitness of population member B so as to increase its likelihood of being put into the mating pool. Another purpose of scaling is that for roulette wheel selection, all fitness values need to be positive.

Many scaling algorithms take the form of (1.6-15). Therein, a and b are coefficients, which vary by algorithm, and f' denotes the scaled fitness. Expressions for a and b are listed by method in Table 1.4. In this table, f_{min}, f_{max}, f_{avg}, and f_{med} denote the minimum, maximum, average, and median fitness of the population.

$$f' = \max(0, af + b) \tag{1.6-15}$$

Another scaling method approach is quadratic scaling. In this algorithm, the scaled fitness is calculated as

$$f' = af^2 + bf + c \tag{1.6-16}$$

Table 1.4 Linear Scaling Methods

Method	***a***	***b***	**Comments**
Offset scaling	1	$-f_{min}$	Ensures positive fitness
Standard linear scaling	$\frac{(k-1)f_{avg}}{f_{max}-f_{avg}}$	$f_{avg}(1-a)$	Most fit individual k times more likely to be in mating pool than average
Modified linear scaling	$\frac{(k-1)f_{med}}{f_{max}-f_{med}}$	$f_{med}(1-a)$	Most fit individual k times more likely to be in mating pool than median
Mapped linear scaling	$\frac{(k-1)f_{avg}}{f_{max}-f_{\min}}$	$-af_{min}+1$	Minimum fitness mapped to 1; maximum fitness to k
Sigma truncation	1	$-(f_{avg}-kf_{std})$	Average fitness maps to kf_{std}

where a, b, and c are given by

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} f_{max}^2 & f_{max} & 1 \\ f_{avg}^2 & f_{avg} & 1 \\ f_{min}^2 & f_{min} & 1 \end{bmatrix}^{-1} \begin{bmatrix} k_{max} \\ 1 \\ k_{min} \end{bmatrix} \tag{1.6-17}$$

In (1.6-17), k_{max} and k_{min} are algorithm constants selected such that $k_{max} > 1$ and $0 < k_{min} < 1$. In this approach, an individual with a fitness equal to the population average is scaled to 1, the most fit individual has a scaled fitness of k_{max}, and the least fit individual has a scaled fitness of k_{min}. Thus, the most fit individual is k_{max} more likely as an average fit individual to go into the mating pool, and the least fit individual is k_{min} times more likely as the average fit individual to be put in the mating pool (which is less likely than average since $k_{min} < 1$).

Diversity Control

The point of a population-based optimization is to search for the minimizer or maximizer using a large number of candidate solutions. Having large numbers of identical members of the population defeats the purpose of having a population in the first place. Diversity control algorithms examine the closeness of the solutions and compute a penalty for identical or nearly identical individuals; the penalized fitness is then used in the selection process.

Elitism

In biological systems, there is no guarantee that the most fit individual in a population will survive long enough to procreate because of environmental influences. For example, the most fit grizzly bear in Alaska could be killed by a hunter. While examples of this occur every day in the natural world, as engineers we may not want to lose the best induction motor design in our population. This has led to the elitism operator. This operator compares the most fit individual in a new population to the most fit individual in the previous population. If the best fitness in the new population is lower than the previous population, the most fit member of the previous population replaces a member of the new population. In this way, the best fitness in the population cannot go down. While this operator is distinctly nonbiological, it normally results in improved algorithm performance.

Migration

In biological populations, it sometimes occurs that a population becomes geographically dispersed, and then members of these geographically isolated regions continue to evolve on separate evolutionary tracks. Sometimes, because of storms or random events, a few members of one region migrate into another region. These individuals then have children with members of the region which they have migrated into, often yielding very fit offspring.

It is possible to create GAs where such migration occurs. In such a GA, each individual is associated with a region, and for the most part, only individuals within a given region interact. Each region acts as a separate population. Occasionally, however, a few individuals are transferred between regions.

At first consideration, one might conclude that incorporating such behavior would have little effect on the algorithm. However, in Sudhoff [7] it is shown that at least in the case of one motor design example, the migration operator had a profound effect on the performance of the algorithm.

That paper included a comparison between GA optimization and particle swarm optimization, and it found that choice of algorithm was less important than the use of migration in the GA or a somewhat analogous feature known as neighborhoods in particle swarm optimization.

Death

In the canonical GA, the entire population is replaced by children. Those individuals who are selected in the mating pool but do not undergo any crossover, segregation, or mutation "survive" to the next generation unchanged. However, in some GAs, rather than forming a mating pool of the same size as the population, a mating pool that is a fraction of the size of the population is formed, thereby generating a population of children a fraction of the size of the population. The children replace members of the existing population, so that a new population is formed which has the children, plus some members of the previous population. In order to make room for the children (to hold the population size constant), it becomes necessary to decide which members of the current population will "die" to make room for the introduction of children. The selection of those individuals to be replaced can be made on criteria other than those used for selection—for example, based on diversity (or more particularly lack thereof) or some other attribute. In some sense, these approaches are reminiscent of biological populations, where the boundaries between generations are gradual.

Local Search

After the initial generations of a GA, it is often the case that the most fit individuals are near a solution. In this case, some GAs will initiate local searches around the most fit individual. One way to accomplish this is to create a set of slight mutations of the most fit individual. Let us refer to the most fit individual as individual A, and refer to the population of mutations of this individual as $\mathbf{P}_A$. If the most fit individual in $\mathbf{P}_A$ is more fit than individual A, then the most fit individual of $\mathbf{P}_A$ replaces individual A in the population.

Deterministic Search

Many classical optimization methods are very effective if they initialized to be close to the solution. This is because many functions are "locally" convex. At the same time, while GAs are often very effective at getting close to a global optimum, they may not always converge rapidly from a good approximation of a solution to the exact solution. This suggests a combination of the two approaches, wherein the best individual of the final population is used to initialize a classical optimization method. To this end, the Nelder–Mead simplex method [1] is particularly attractive because it does not require gradients or Hessians. In performing such an optimization, it is normally the case that in problems with a mixed search space, those genes that represent discrete choices are held fixed.

Enhanced Real-Coded Genetic Algorithm

Before concluding this section, it is interesting to place all of the aforementioned operators into a single block diagram so that the relationships between the operators can be made clear. This is illustrated in Figure 1.14. Therein, initialization and the first fitness evaluation are as in the canonical GA diagramed in Figure 1.8. Next, the scaling algorithm is (optionally) employed. Note that the input and output of the scaling operator is the population $\mathbf{P}[k]$; the same symbol for the input and output is used since the scaling operator does not operate on the population; it merely adjusts

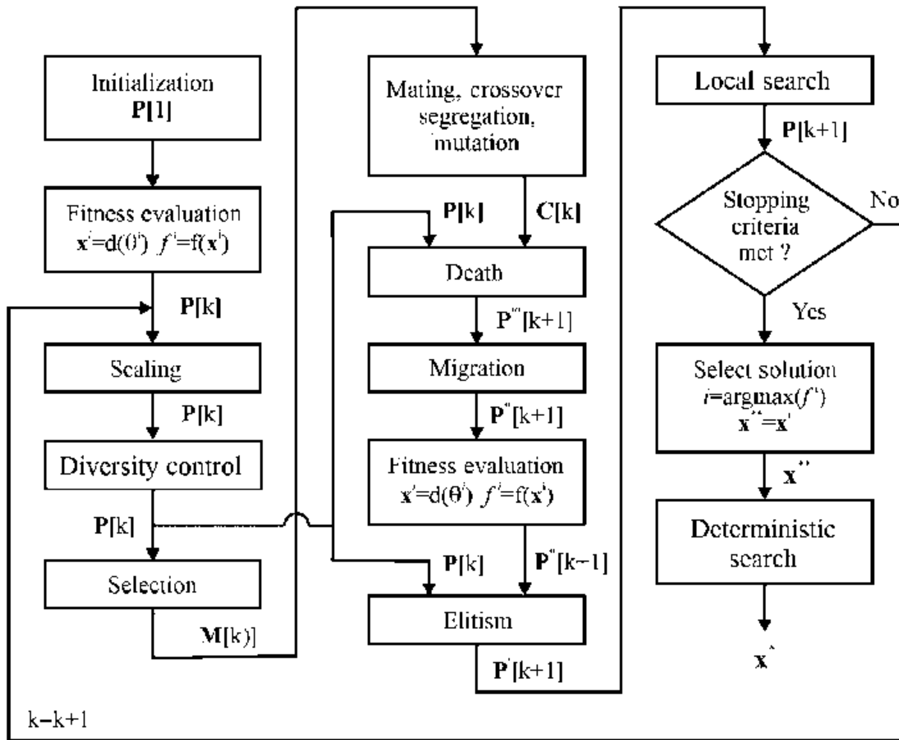

Figure 1.14 Enhanced real-coded genetic algorithm.

fitness values. The same is true of the diversity control operator, if utilized. Next, the selection operator creates a mating pool $\mathbf{M}[k]$. Based on the mating pool, the mating, crossover, segregation, and mutation operators yield a population of children $\mathbf{C}[k]$.

The death algorithm selects a member of the current population to be replaced by the children, yielding the beginnings of the next generation, $\mathbf{P}'''[k + 1]$. The primes are used because this population will be modified before becoming the next generation. First the migration operator will be applied, which may occasionally move individuals from one region to another. This yields $\mathbf{P}''[k + 1]$. Next, the gene values are decoded and the fitness evaluated. The result is again denoted $\mathbf{P}''[k + 1]$ since the population itself does not change. The elitism operator is then applied, which compares the most fit individual of the previous population $\mathbf{P}[k]$ to the most fit individual in $\mathbf{P}''[k + 1]$ and replaces that individual if appropriate. This yields population $\mathbf{P}'[k + 1]$. The local search operator is employed to generate $\mathbf{P}[k + 1]$.

Once the next generation is formed, the stopping criterion is checked. Often, the stopping criterion is a check to see if a sufficient number of generations have passed. If the stopping criterion has not been met, the process repeats, starting with scaling. If the stopping criterion has been met, then an estimate of the solution $\mathbf{x}^{**}$ is selected as the most fit individual of the population. This estimate can then be passed to a deterministic optimization algorithm for further refinement, yielding x^*.

The algorithm depicted in Figure 1.14 is probably more involved than is typical, because many optional operators have been used. For example, diversity control does not have to be used. Scaling is not necessary when using tournament or when the fitness values are always positive. A separate death algorithm is not necessary if the entire population is replaced by children. Random and

deterministic search routines are often not used. Of the optional algorithms, elitism is fairly important. In addition, although not as commonly employed as elitism, the migration operator has been shown to have a significant impact on performance. Clearly, there is a wide variety of GA operators, and many books have been written on this subject. The reader is referred to references [4,5,8,9] for just a few of these. The good news is that almost all variations are effective optimization engines—they vary primarily in how quickly they converge and their probability of finding global solutions.

1.7 Multi-Objective Optimization and the Pareto-Optimal Front

Much of our focus in this chapter has focused on the single-objective optimization of an objective or fitness function f(x). However, in the case of design problems, it is normally the case that there are multiple objectives of interest. In this section, we begin our consideration of the multi-objective optimization problem.

Let us begin our discussion by the consideration of some designs of an electric motor. Suppose we are designing a motor and wish to minimize material cost and also to minimize loss. Further, suppose that we had eight designs available, each of which met all specifications for the machine, but each of which had a different cost and a different loss, as illustrated in Figure 1.15. Therein, each number listed is an index of a design. Thus, the x above point 1 has coordinates given by $f(\mathbf{x}^1)$.

Now let us consider the question of which design is the best. Let us first compare design 5 to design 7 (again using our shorthand notation of using a 5 to designate $\mathbf{x}^5$). Design 7 has lower loss and lower cost than design 5, and so it is clearly better than design 5. Using similar reasoning, we can see that design 8 is better than designs {4,5,7}. Now let us compare design 8 to design 6. Design 8 has lower loss, but design 6 has lower cost. At this point, it is difficult to say which is better. This leads to the concept of dominance.

Consider two parameter vectors $\mathbf{x}^a$ and $\mathbf{x}^b$. The parameter vector $\mathbf{x}^a$ is said to dominate $\mathbf{x}^b$ if $f(\mathbf{x}^a)$ is no worse than $f(\mathbf{x}^b)$ in all objectives and $f(\mathbf{x}^a)$ is strictly better than $f(\mathbf{x}^b)$ in at least one objective. The statement "$\mathbf{x}^a$ dominates $\mathbf{x}^b$" is equivalent to the statement that "$\mathbf{x}^b$ is dominated by $\mathbf{x}^a$." Returning to the example of the previous paragraph, we can say that design 8 dominates designs {4,5,7}.

Consider a set of parameter vectors denoted $\mathbf{X}$. The nondominated set $\mathbf{X}_{nd} \subset \mathbf{X}$ is a set of parameter vectors that are not dominated by any other member parameter vector in $\mathbf{X}$. In our example, $\mathbf{X} = \{1, 2, ..., 8\}$ and $\mathbf{X}_{nd} = \{1, 6, 8\}$. The nondominated set can be viewed as the set of best designs.

If we consider the set of parameters to be the entire domain for the parameter vector (i.e., $\boldsymbol{\Omega}$) then the set of nondominated designs is referred to as the Pareto-optimal set. The boundary defined by the objectives over this set is known as the Pareto-optimal front. Pareto was an Italian engineer,

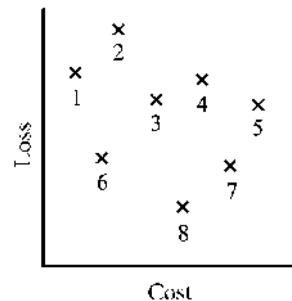

Figure 1.15 Motor performance objective space.

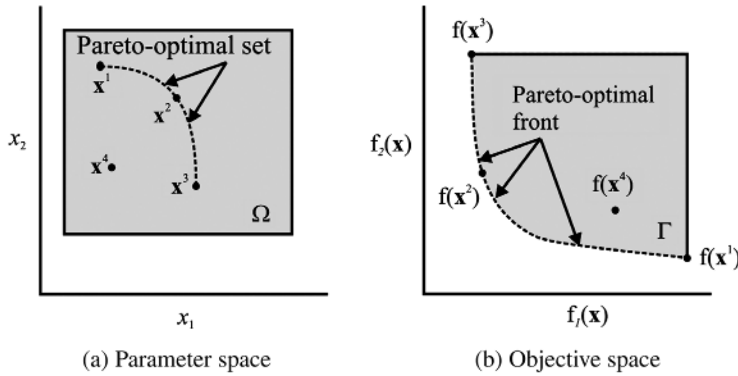

Figure 1.16 Pareto-optimal set and front.

economist, and philosopher who lived in the period of 1848–1923 (overlapping Mendel and Darwin) and who, besides his work in multi-objective optimization, formulated the Pareto principle that 80% of effects stem from 20% of causes (and that 80% of wealth is owned by 20% of the population).

The concepts of a Pareto-optimal set and Pareto-optimal front are illustrated in Figure 1.16 for a system with a two-dimensional parameter space (i.e., a two-dimensional parameter vector) and a two-dimensional objective space. As an example, if we were designing an inductor, the variables of the parameter space may consist of the number of turns and diameter of the wire, variables in the objective space might be inductor volume and inductor power loss. Returning to Figure 1.16(a) we see the parameter space defined over a set Ω (shaded). Members of this set include $\mathbf{x}^1$, $\mathbf{x}^2$, $\mathbf{x}^3$, and $\mathbf{x}^4$. Every point in Ω is mapped to a point in objective space Γ (also shaded) shown in Figure 1.16(b). For example, $\mathbf{x}^4$ in parameter space maps to $f(\mathbf{x}^4)$ in objective space. Points along the dashed line in **Ω** in Figure 1.16(a) are nondominated and form the Pareto-optimal set. They map to the dashed line in Figure 1.16(b) in objective space where they form the Pareto-optimal front.

The goal of multi-objective optimization is to calculate the Pareto-optimal set and the associated Pareto-optimal front. One may argue that in the end, only one design is chosen for a given application. This is often the case. However, to make such a decision requires knowledge of what the tradeoff between competing objectives is—that is, the Pareto-optimal front.

There are many approaches to calculating the Pareto-optimal set and the Pareto-optimal front. These approaches include the weighted sum method, the ε-constraint method, and the weighted metric method, to name a few [10]. In each of these methods, the multi-objective problem is converted to a single-objective optimization, and that optimization is conducted to yield a single point on the Pareto-optimal front. Repetition of the procedure while varying the appropriate parameters yields the Pareto-optimal front.

In order to illustrate this procedure, let us consider the ε-constraint method for a system with two objectives, $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, which we wish to minimize. In order to determine the Pareto-optimal set and front, the problem

$$\begin{aligned} &\text{minimize} \quad f_2(\mathbf{x}) \\ &\text{subject to} \quad f_1(\mathbf{x}) \leq \varepsilon \end{aligned} \tag{1.7-1}$$

is repeatedly solved for different values of ε. This minimization can be carried out numerically using any single-objective minimization method. Solving (1.7-1) with $\varepsilon = \varepsilon_{1a}$ yields $\mathbf{x}^{1a}$ with objective

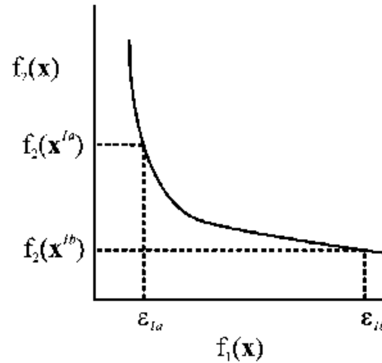

Figure 1.17 Calculation of Pareto-optimal front with ε-constraint method.

function values $(\varepsilon_{1a},f_2(\mathbf{x}^{1a}))$ which is a point on the Pareto-optimal front, as shown in Figure 1.17. Solving (1.7-2) with $\varepsilon = \varepsilon_{1b}$ yields $\mathbf{x}^{1b}$ with objective function values $(\varepsilon_{1b},f_2(\mathbf{x}^{1b}))$. Repeating the procedure over a range of ε values, the Pareto-optimal set and front can be determined.

In more concrete terms, suppose we wished to minimize volume and loss for some electromagnetic device. If our first objective was volume and our second objective was loss, we would repeatedly minimize loss with different constraints on the volume. In this case, each ε value would correspond to a different volume. For each numerically different volume constraint, we would get the corresponding loss and thereby—one single-objective optimization at a time—build up a Pareto-optimal front between volume and loss.

As it turns out, GAs are well-suited to compute Pareto-optimal sets. This is because they operate on a population. In multi-objective optimizations, the population of designs can be made to conform to the Pareto-optimal set. This allows a GA to determine the Pareto-optimal set and Pareto-optimal front in a single analysis without requiring the solution of a separate optimization for every point on the front as is needed in the weighted sum method, ε-constraint method, and weighted metric methods.

1.8 Multi-Objective Optimization Using Genetic Algorithms

GAs are well-suited for multi-objective optimization, and there are a large number of approaches that can be taken. The goal in all of these methods is to evolve the population so that it becomes a Pareto-optimal set.

Multi-objective GAs fall into two classes, nonelitist and elitist. The elitist strategies are particularly effective because they explicitly identify and preserve, when possible, the nondominated individuals. Elitist strategies include the elitist nondominated sorting GA (NSGA-II), distance-based Pareto GA, and the strength Pareto GA [11]. In order to keep the present discussion limited that we might soon start discussing device design, we will somewhat arbitrarily focus our attention on the elitist NSGA-II.

Before setting forth this algorithm, we will first consider the problem of finding the nondominated individuals in a population. Since finding the Pareto-optimal front is the goal of any multi-objective optimization, being able to identify those candidate solutions which are nondominated will be an important task. In a generic sense, the nondominated solutions will be given a favored status in many multi-objective methods. To identify the nondominated solutions, we will consider Kung's method. Consider a population **P**. We wish to find the nondominated subset of **P**,

Table 1.5 Pseudo-Code for Kung's Method

```
function R=front(S)
   if |S|=1
      R=S
   else
      i=floor(|S|/2)
      T=front(S^{1:i})
      B=front(S^{i+1|S|})
      N=solutions of B not dominated
        by any solution of T
      R=T ∪ N
   end
end
```

which we will denote **E**. The first step in Kung's method is to sort the members of **P** from best to worst in terms of the first objective. In the event that two or more members have identical values of the first objective, this subset is ordered by the second objective (and by subsequent objectives if necessary). This sorted population will be denoted $\mathbf{P}_s$. The nondominated set **E** is then calculated as

$$\mathbf{E} = \text{front}(\mathbf{P}_s) \tag{1.8-1}$$

where front() is a recursively defined function. In order to describe this function, let $|\cdot|$ denote the number of elements in a set, and let $\mathbf{P}^{a:b}$ denote the ath to bth individuals in a population or set. With this notation, the pseudo-code for Kung's algorithm is listed in Table 1.5. Therein, **S** and **R** denote the input and output of the routine. The terms **T**, **B**, and **N** refer to intermediate variables that represent sets. The variable i is a scalar integer index.

In order to illustrate this algorithm, let us apply Kung's algorithm to the set of designs illustrated in Figure 1.15. For this problem, $\mathbf{P} = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Considering cost to be our first objective, $\mathbf{P}_s = \{1, 6, 2, 3, 8, 4, 7, 5\}$. Figure 1.18 illustrates the calls to the front routine. Therein, the order of the calls and the input and output arguments to each call are indicated. The basis of this method is that whenever the front is called, its return argument **R** will never have any dominated members. This is a result of the following: (a) The sorting of the first objective ensures that no member of a return argument **T** can be dominated by a member of **B**, (b) every member of **B** is checked against every member of **T** before returning to the next higher level, and (c) a set of one is inherently a nondominated set. As a result, **T**, **B**, and **R** are all nondominated sets.

In executing this example, calls 4, 5, 7, 8, 11, 12, 14, and 15 return nondominated sets because their input and output argument set size is 1. In calls 3, 6, 10, and 13, no member of an input set **T** can be dominated by the corresponding set **B** because of the ordering with respect to the first objective. The return arguments of calls 3, 6, 10, and 13 cannot have dominated members because the input sets to these calls (**T** and **B**) were nondominated, no member of **T** can be dominated by a member of **B**, and every member of **B** is checked against every member of **T**. Thus, proceeding from the bottom of Figure 1.18 to the top as subroutine calls are made, we see that the process involves a gradual combining and sifting of smaller nondominated sets to yield a larger nondominated set.

One question that arises in this example is why we need function calls beyond call 1 as it generates the nondominated set. It must be remembered that the calls are recursive in nature and so the algorithm of call 1 cannot be completed without calls 2–15. Another question that is often asked is if

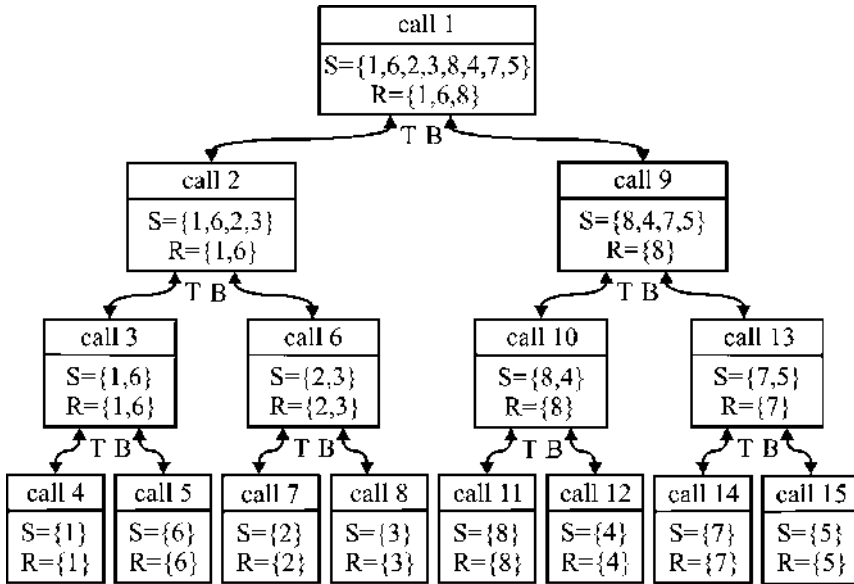

Figure 1.18 Application of Kung's method.

we really need to make calls to front when the number of elements in the input set is one. With some additional if-then constructs, calls to front with an input set size of one could in fact be avoided.

In addition to Kung's method, two more processes will be needed in order to set forth the elitist NSGA-II. The first of these is nondominated sorting (NDS). In order to understand NDS, let us consider again Figure 1.15 with population $\mathbf{P} = \{1, 2, 3, 4, 5, 6, 7, 8\}$. As we have already discussed, the nondominated set is {1,6,8}, which was found using Kung's algorithm. Let us consider this set front 1, denoted $\mathbf{F}_1$. Now suppose we remove the members of $\mathbf{F}_1$ from $\mathbf{P}$, and find the nondominated set of the remaining population by again applying Kung's algorithm. This will yield the second front, given by $\mathbf{F}_2 = \{2, 3, 7\}$. Now let us once more consider the population $\mathbf{P}$ but now less the members in $\mathbf{F}_1$ and $\mathbf{F}_2$. Finding the nondominated set of the remaining population by again applying Kung's algorithm yields the third front, $\mathbf{F}_3 = \{4, 5\}$. In this way, it is possible to associate with every member of the population a rank with regard to which front they are associated. This ranking will be used to compare different members of the population in order to determine which members will become a part of the mating pool.

Although a population member on the first front can be said to be superior to a member on the second front, the question arises as to how to compare solutions on the same front. This issue will be resolved using the concept of crowding distance. The crowding distance associated with a solution $\mathbf{x}^i$ is defined as

$$c^i = \sum_{o=1}^{O} \frac{f_o^{\mathrm{N}_L(o,i,F_n)} - f_o^{\mathrm{N}_S(o,i,F_n)}}{\max\limits_{j \in P}\left(f_o^j\right) - \min\limits_{j \in P}\left(f_o^j\right)} \quad i \in \mathbf{F}_n \tag{1.8-2}$$

where $\mathbf{F}_n$ is the set of indices in front n, we assume i is in front $\mathbf{F}_n$, o is an index of objective number, O is the number of objectives, $\mathrm{N}_L(o, i, \mathbf{F}_n)$ returns the index of the individual with the next largest fitness (greater than individual i) in the oth objective in front $\mathbf{F}_n$, and $\mathrm{N}_S(o, i, \mathbf{F}_n)$ returns the index of

the individual with the next smallest fitness (less than individual i) in the oth objective in front F_n. In the case where there is no next larger or smaller individual in some objective—that is, it is a maximum or minimum in some objective—the crowding distance is taken as a very large number (pseudo-infinity).

Crowding tournament selection uses the concepts of front rank and crowding distance in order to decide which individuals to put into the mating pool. In this method, individuals $\mathbf{x}^{c1}$ and $\mathbf{x}^{c2}$ are randomly drawn from the current population. If one of these solutions has a better front rank than the other, it is copied into a mating pool. If the two individuals have the same front rank, then the one with the better (larger) crowding distance goes into the mating pool, as it has greater diversity in terms of objectives.

Example 1.8A The concept of crowding distance is illustrated in Figure 1.19. Suppose we wish to calculate the crowding distance of population member 7. Using (1.8-2), we have

$$c^7 = \frac{6-2}{8-1} + \frac{7-3}{11-3} = 1.07 \tag{1.8A-1}$$

At this point, the remaining elements of the NSGA-II can be set forth. The process is illustrated in Figure 1.20. Therein, we start with a population $\mathbf{P}[k]$. Using crowded tournament selection (CTS), a mating pool $\mathbf{M}$ is created which is in turn used to create a population of children $\mathbf{C}$. Next, the current population $\mathbf{P}[k]$ is combined with the children $\mathbf{C}$ to form an enlarged population $\mathbf{R}$. Non NDS is used to divide this population into groups based on front, yielding sets $\mathbf{F}_1$, $\mathbf{F}_2$, and so on. Next, these sets are used to create the next generation $\mathbf{P}[k+1]$. This is done by filling $\mathbf{P}[k+1]$with sets of fronts from first to last. At some point, shown as $\mathbf{F}_3$ in Figure 1.20, the full front will have more members

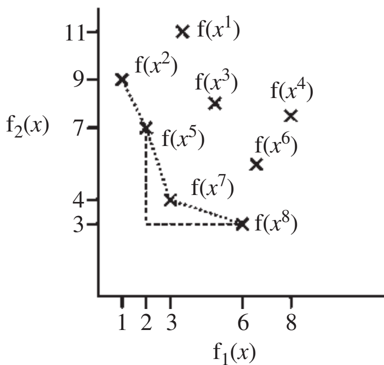

Figure 1.19 Crowding distance.

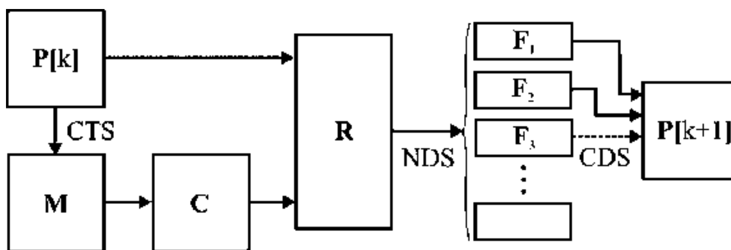

Figure 1.20 Elitist nondominated sorting genetic algorithm (NSGA-II).

than remain empty in $\mathbf{P}[k+1]$. At this point, crowding distance selection (CDS) is used to determine which members of the final included front are included in $\mathbf{P}[k+1]$. In particular, the members of $\mathbf{F}_3$ with the best crowding distance are used to complete the next population.

Using this algorithm, as the population evolves, it will come closer and closer to approaching the Pareto-optimal set, which will lead to a family of designs. We will use multi-objective optimization extensively in this book for the design of power magnetic devices.

1.9 Formulation of Fitness Functions for Design Problems

The previous sections of this chapter have focused on the design process, and some general-purpose single- and multi-objective optimization techniques. In this section, we consider methodologies to construct fitness functions.

In constructing the fitness function, we will have a variety of metrics, such as mass and loss, and also a number of constraints related to the appropriate operation and construction of the device of interest. It will often be the case that we will have to perform multiple analyses in order to evaluate metrics, and that some of these analyses may be computationally expensive. The fact that a variety of analyses of varying computational intensity will be required will be a significant consideration in the construction of the fitness function.

Let us begin our discussion of the construction of the fitness function with consideration of the constraints. Let us assume that we have C constraints, and use c_i to denote the status of the ith constraint. If the constraint is satisfied, we will set $c_i = 1$. If the constraint is not satisfied, we will have $0 \le c_i < 1$. It is convenient to define c_i so that it approaches 1 as the constraint becomes closer to becoming satisfied.

In order to test constraints, it is convenient to define the less-than-or-equal-to and greater-than-or-equal-to functions as

$$\text{lte}(x, x_{mx}) = \begin{cases} 1 & x \le x_{mx} \\ \dfrac{1}{1 + x - x_{mx}} & x > x_{mx} \end{cases} \tag{1.9-1}$$

$$\text{gte}(x, x_{mn}) = \begin{cases} 1 & x \ge x_{mn} \\ \dfrac{1}{1 + x_{mn} - x} & x < x_{mn} \end{cases} \tag{1.9-2}$$

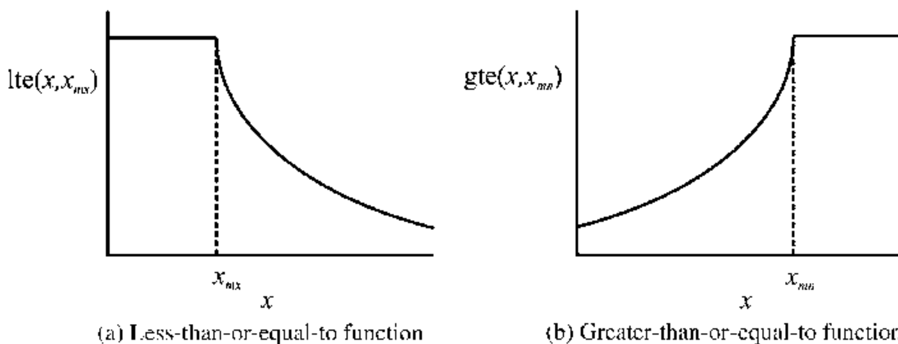

Figure 1.21 Constraint functions.

These functions are illustrated in Figure 1.21.

As an example, we may require the height of a transformer to be less than a maximum value h_{mx}. If this is the first constraint and if we use $h(\mathbf{x})$ to represent the height in terms of our design parameters, our first constraint could be calculated as

$$c_1 = \text{lte}(h(\mathbf{x}), h_{mx}) \tag{1.9-3}$$

Next, let us consider our design metrics. The design metrics will also be a function of the parameter vector $\mathbf{x}$. In this text, common metrics will be mass and loss. Let the number of metrics be denoted M, and the value of the ith metric be denoted m_i. It will henceforth be assumed that all metric values are greater than zero.

As we will see, metrics and objectives are closely related but not synonymous. Metrics will be based on attributes of interest. Objectives will be based on the metrics, but also be influenced by constraints.

Let us now discuss the definition of the objective or fitness function. In keeping with the usual practice of GAs, we will assume that our fitness function (which is synonymous with the objective function) is of a form to be maximized. One approach to creating a fitness function begins with first forming a combined constraint. This can be done by averaging the constraints as

$$\bar{c} = \frac{1}{C}\sum_{i=1}^{C} c_i \tag{1.9-4}$$

Next, the elements of the fitness function are defined as either

$$f_i = \begin{cases} \varepsilon(\bar{c}-1) & \bar{c} < 1 \\ m_i & \bar{c} = 1 \end{cases} \quad \text{(maximize metric)} \tag{1.9-5}$$

or

$$f_i = \begin{cases} \varepsilon(\bar{c}-1) & \bar{c} < 1 \\ \dfrac{1}{m_i} & \bar{c} = 1 \end{cases} \quad \text{(minimize metric)} \tag{1.9-6}$$

where ε is a small positive number. The value has no impact on results but is convenient for plotting the fitness versus generation. It is commonly chosen to be on the order of 10^{-10}.

With this definition, designs that do not meet all constraints have negative fitness, while designs that do meet all constraints have positive fitness. If the constraints are not met, the fitness increases (becomes less negative) as the constraints become closer to being met. Note that the metric in the denominator in (1.9-6) does not pose a risk of a singularity because for most cases the metrics cannot be zero if all constraints are met. For example, a workable inductor design will have nonzero mass. Also, observe that all elements of the fitness function are negative (and equal) in the case where one or more constraints are not met; all elements of the fitness function are positive (and generally speaking not equal) if all of the constraints are met.

While the use of (1.9-5) and (1.9-6) is convenient for many design problems, there are cases when certain calculations are nonsensical unless constraints are met. There are also cases where certain calculations are computationally expensive, and there is a desire to avoid them if possible. For example, if predicting loss were computationally expensive, and that it is already known that not all constraints are passed, then there is little motivation to proceed with the loss calculation. Table 1.6 contains pseudo-code to treat this case. Therein, C_I and C_s denote the number of constraints imposed and satisfied, respectively, thus far in the current evaluation of the fitness function.

Table 1.6 Approach for Treating Constraints

calculate constraint c_i
$C_I = C_I + 1$
$C_S = C_S + c_i$

if $(C_S < C_I)$

$$\mathbf{f} = \varepsilon\left(\frac{C_S - C}{C}\right)[1 \quad 1 \quad \cdots \quad 1]^T$$

return
end

Both of these quantities are initialized to zero. After the constraint in question is calculated, the number of constraints imposed and number of constraints satisfied are updated. If the number of constraints satisfied falls below the number of constraints imposed, the fitness is calculated based on the number of constraints satisfied, and no further action (calculation of further constraints or metrics) is taken. This can lead to a significant computational savings for some problems.

The code block shown in Table 1.6 is repeated sequentially for every constraint. After all constraints are tested, the metrics may be calculated and the fitness assigned as in (1.9-5) and (1.9-6) for the case when all constraints are met. There are also variations of this procedure. For example, it may be convenient to calculate and test several constraints at a time.

1.10 A Design Example

In this section, we will conclude this chapter with a design example. In particular, we will endeavor to design a UI-core inductor using an optimization-based design process. At this point, we have not studied any magnetics or magnetic devices. This will be the subject of the remainder of this book. Since we are not yet in a position to derive or understand the relationships needed, an elementary analysis will be provided and the reader is asked to simply accept these relationships at face value for the time being. It should be observed that the analysis used to derive the needed relationships is very simplistic, but this does not matter because our purpose here is only to look at the design process. We will conduct a much more detailed analysis and design in subsequent chapters.

A UI-core inductor is depicted in Figure 1.22. Therein, the dark region is the magnetic core. The magnetic core conducts magnetic flux and is made of a U-shaped piece (the U-core) and an I-shaped piece (the I-core). Conductors (wires) pass through the middle of the U, a region called the slot, and also around the outside of the U-core to form a winding. The slot has a width denoted w_s and a height denoted d_s. The width of the core (both U- and I-cores) is denoted the core width w_c, and the length of the core pieces is l_c. The two cores are separated by an air gap g. The cross-sectional drawing (a) is the view that one would obtain by looking to the right from the left side of the side view (b) if the side view were cut in half (in the direction into the page). The light region indicates a winding comprised of N turns of wire.

Our goal in this design will be to design an inductor that has an inductance of at least L_{mn}, a flux density below B_{mx}, and a current density below J_{mx} at rated dc current i_{rt}. It is desirable to minimize

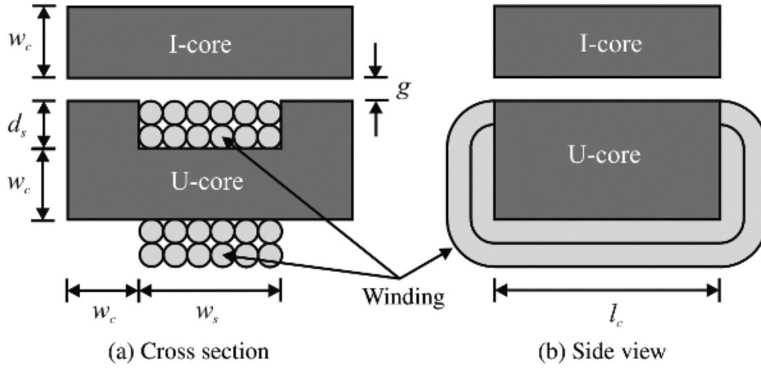

Figure 1.22 UI-core inductor.

the mass M of the inductor and to minimize the power loss at rated current, denoted P_{rt}. We will also constrain our designs to have a mass below M_{mx} and a power loss less than P_{mx}.

The free parameters in our design are the number of turns N, the slot depth d_s, the slot width w_s, the core thickness w_c, the core depth l_c, and the air gap g. Thus, our parameter vector may be expressed as

$$\mathbf{x} = [N^* \quad d_s \quad w_s \quad w_c \quad l_c \quad g]^T \tag{1.10-1}$$

In (1.10-1), N^* is the desired number of turns rather than the actual number of turns N because, as a design parameter, we will let the number of turns be represented as a real number rather than an integer. This is because for large N this variable acts in a more continuous rather than discrete fashion. The actual number of turns is calculated from the desired number as

$$N = \text{round}(N^*) \tag{1.10-2}$$

In order to perform the optimization, we will need to analyze the device. It is assumed that windings occupy the entire slot, that the core is infinitely permeable, and that the fringing and leakage flux components are negligible. Again, for the reader unfamiliar with these terms, the analysis can be taken as a set of arbitrary mathematical equations; we will spend the rest of the book defining and developing more accurate expressions for these quantities.

With these assumptions, the mass of the design may be expressed as

$$M = 2(2w_c + w_s + d_s)l_c w_c \rho_{mc} + (2l_c + 2w_c + \pi d_s)d_s w_s k_{pf} \rho_{wc} \tag{1.10-3}$$

In (1.10-3), ρ_{mc} and ρ_{wc} denote the mass density of the magnetic core and wire conductor, respectively, and k_{pf} is the fraction of the U-core window occupied by conductor. Ideally, it would be 1, but 0.7 is a very high number in practice.

The next step is the computation of loss. The power dissipation of the winding at rated current may be expressed as

$$P_{rt} = \frac{(2l_c + 2w_c + \pi d_s)N^2 i_{rt}^2}{d_s w_s k_{pf} \sigma_{wc}} \tag{1.10-4}$$

In (1.10-4), σ_{wc} denotes the conductivity of the wire conductor.

There are constraints both on the inductance, flux density at rated current, and current density at rated current. These quantities may be expressed as

$$L = \frac{\mu_0 l_c w_c N^2}{2g} \tag{1.10-5}$$

$$B_{rt} = \frac{\mu_0 N i_{rt}}{2g} \tag{1.10-6}$$

$$J_{rt} = \frac{Ni}{w_s d_s k_{pf}} \tag{1.10-7}$$

In (10.1-5) and (1.10-6), μ_0 is the magnetic permeability of free space, a constant equal to $4\pi 10^{-7}$ H/m.

In order to formulate a fitness function, expressions (1.10-1)–(1.10-7) can be sequentially evaluated. Then constraint functions can be evaluated as

$$c_1 = \text{gte}(L, L_{mn}) \tag{1.10-8}$$

$$c_2 = \text{lte}(B_{rt}, B_{mx}) \tag{1.10-9}$$

$$c_3 = \text{lte}(J_{rt}, J_{mx}) \tag{1.10-10}$$

$$c_4 = \text{lte}(P_{rt}, P_{mx}) \tag{1.10-11}$$

$$c_5 = \text{lte}(M, M_{mx}) \tag{1.10-12}$$

Keeping with (1.9-4), we find the aggregate constraint

$$\bar{c} = \frac{1}{5}(c_1 + c_2 + c_3 + c_4 + c_5) \tag{1.10-13}$$

We will consider both single- and multi-objective optimization. For the single-objective case, we will minimize mass and our fitness is given by

$$f = \begin{cases} \varepsilon(\bar{c} - 1) & \bar{c} < 1 \\ \dfrac{1}{M} & \bar{c} = 1 \end{cases} \tag{1.10-14}$$

For the multi-objective case, the fitness function will be taken as

$$f = \begin{cases} \varepsilon(\bar{c} - 1)[1 \quad 1]^T & \bar{c} < 1 \\ \left[\dfrac{1}{M} \quad \dfrac{1}{P_{rt}}\right]^T & \bar{c} = 1 \end{cases} \tag{1.10-15}$$

In (1.10-14) and (1.10-15), we will take $\varepsilon = 10^{-10}$.

For our design, let us consider a ferrite material for the core with $B_{mx} = 0.617$ T and $\rho_{mc} = 4680$ kg/m^3, and consider copper for the wire with $\rho_{wc} = 8890$ kg/m^3 and $J_{mx} = 7.5$ A/mm^2. We will take

Table 1.7 Domain of Design Parameters

Parameter	N	d_s **(m)**	w_s **(m)**	w_c **(m)**	l_c **(m)**	g **(m)**
Min. value	1	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-5}
Max. value	10^3	10^{-1}	10^{-1}	10^{-1}	10^{-1}	10^{-2}
Encoding	log	log	log	log	log	log
Chromosome	1	1	1	1	1	1

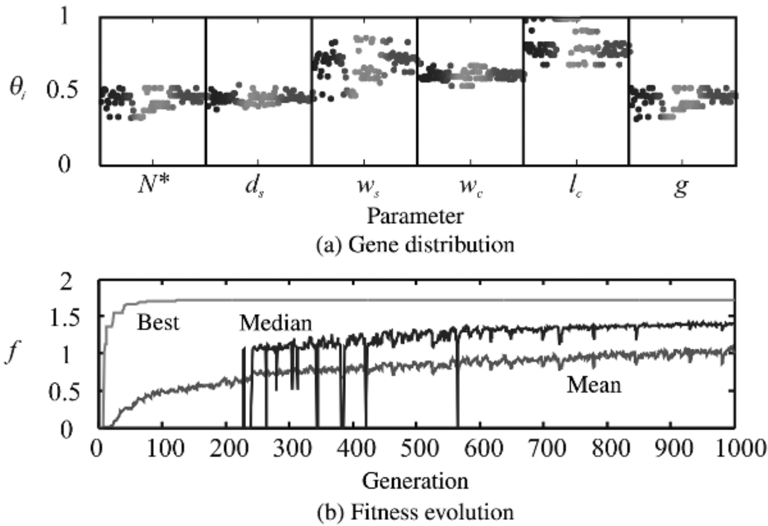

Figure 1.23 Single-objective optimization study.

rated current to be 10 A and take the minimum inductance L_{mn} to be 1 mH. Finally, let us take the maximum allowed mass as $M_{mx} = 1\text{kg}$, and the maximum allowed loss to be $P_{mx} = 1\text{W}$.

The next step in the design process is to determine the parameter space $\mathbf{\Omega}$. This is tabulated in Table 1.7. Some level of engineering estimation is required to select a reasonable range. However, situations where a range is incorrectly set are usually easy to detect by looking at the population distribution. We will return to this point.

We have now set forth a fitness function and a domain for the parameter vector, and so we can proceed to conduct an optimization. We will begin with a single-objective case. To conduct this study, a MATLAB-based genetic optimization toolbox known as GOSET was used. This open-source code and the code for this particular example are available at no cost in Sudhoff [6].

Figure 1.23 illustrates the progression of the study, which was conducted with a population size of 1000 over 1000 generations. Therein, Figure 1.23(a) shows the gene distribution at the end of the optimization. Recall that θ^i is the normalized value of the ith gene. Each design is shown in encoded parameter space as a series of dots, each with its own shade (for example, a certain dark shade may correspond to design 37 of the population). Because of the large numbers of designs, it is not possible to pick one design among all the designs. However, a sense of the distribution of the gene (parameter) values in the population can readily be obtained. The horizontal coordinate of each design within its parameter window is proportional to its ranked fitness—with lower ranks toward the left side of the window for a given parameter and higher ranks toward the right side of a given window. Considerable information can be discerned from the distribution plot. For example, there seems to be more sensitivity to d_s (which is tightly clustered) than to w_s (which is less tightly clustered). A distribution of a gene (parameter) value at the bottom or top of the range indicates that it may be appropriate to adjust the domain of that parameter.

Figure 1.23(b) depicts the fitness versus generation. The best fitness in the population, the median fitness of the population, and the mean fitness of the population are shown. Note that for a few generations, the best fitness is zero (actually slightly < 0), but then the best fitness increases rapidly until generation 150 or so, after which the fitness climes more slowly. The median and mean fitness rise more slowly than the fitness of the best individual. Observe that there are large rapid changes in

Turns $N = 25$ ($N^* = 25.3$)
Slot depth d_s (mm) = 8.39
Slot width w_s (mm) = 25.5
Core width w_c (mm) = 15.0
Core length l_c (mm) = 43.3
Air gap g (mm) = 0.255

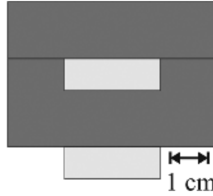

Figure 1.24 UI-core design.

the median fitness because this is a fitness of the median individual which changes from generation to generation. The mean fitness of the population is more stable. As can be seen, the mean fitness of the population occasionally goes down; this does not happen in the case of the fitness of the most fit individual in the population because of the elitism operator.

The most fit individual in the final population is illustrated in Figure 1.24, which lists the design parameters as well as a cross-sectional diagram. Note that $N^* = 25.3$ maps to $N = 25$ from (1.10-2). The design's mass is 0.578 kg, and the power loss at rated current is $0.9\overline{9}$ W. The inductance is right at 1 mH and the flux density is at 0.617 T. The current density of the design is 1.67 A/mm^2. It would appear that our design is against all constraint limits except those on mass and current density.

At this point, the question arises regarding how we know that our design is optimal. Unfortunately, we do not. There is not an optimization algorithm known that can guarantee convergence to the global optimum for a generic problem without known mathematical properties. However, in the GOSET code used for this example, a traditional optimization method (Nelder–Mead simplex) is used to optimize the design starting from the endpoint of the GA run, and this helps to ensure a local optimum. Still, there is no guarantee that a global optimum is obtained. Therefore, the prudent designer will re-run the optimization several times in order to gain confidence in the results. The runs can then be inspected to see if all runs converged to the same fitness. If significant variation in fitness has occurred, the use of more generations and/or a larger population size is indicated.

For our single-objective optimization problem, the optimization was re-run a multitude of times in order to investigate the variability of the design obtained from one run to the next. We will view variation of parameters and metrics in terms of normalized standard deviations. For example, the normalized standard deviation of the number of turns is the standard deviation of the number of turns divided by the median value of the number of turns for each design, interpreted as a percentage. Conducting the optimization process 100 times yielded the following normalized standard deviations: N with a 11%, d_s with a 4.4%, w_s with a 17%, w_c with a 6.4%, l_c with a 14%, and g with a 11% standard deviation. These may seem relatively large. However, it is interesting that normalized standard deviation in mass is only 1.0%. This indicates that there is a family of designs with equally good performance. It is interesting to observe that while appreciable design variation was found, every solution determined was viable (and not that different in terms of performance metrics).

It may seem objectionable to the reader that the results are not repeatable, which arises from the use of a random set of initial designs, and stochastic operators in the GA. However, even Newton's method will generate random variation in the solution of an optimization problem if the initial condition is selected at random. In Newton's method, providing a consistent initial condition will of course produce a consistent final answer; however, being consistent can merely mean being

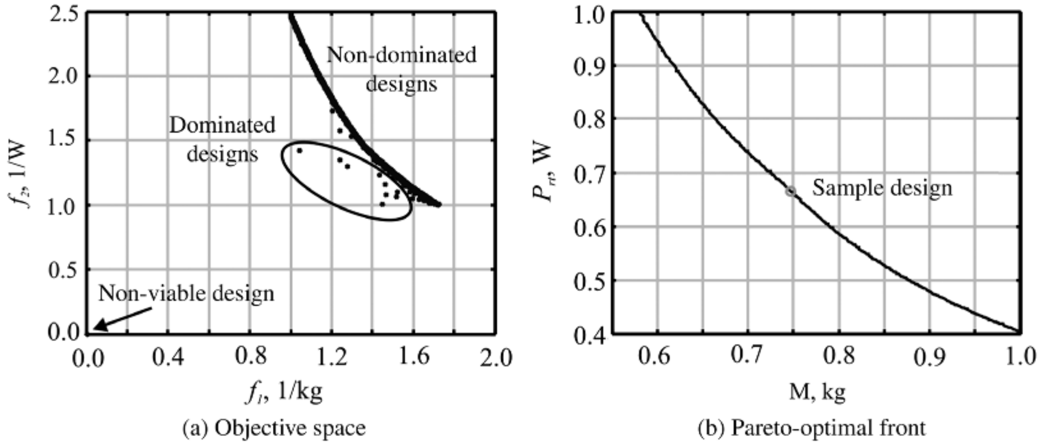

Figure 1.25 Multi-objective optimization results.

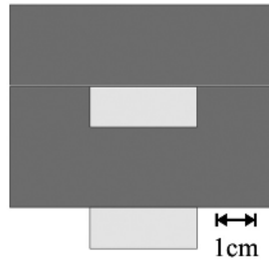

Figure 1.26 Sample design from Pareto-optimal front.

consistently incorrect—which can happen if the algorithm becomes consistently trapped at a same local minimizer while missing the global minimizer.

Let us now turn our attention to a multi-objective optimization of the UI-core inductor. The only change in our approach is the replacement of (1.10-14) by (1.10-15). Using 2000 generations with a population size of 1000 yields the results shown in Figure 1.25. Figure 1.25(a) illustrates the objective space at the end of the optimization run. Each point in Figure 1.25(a) represents the objectives of a complete design. Nonviable designs have fitness values close to the origin (and are slightly negative in each axis). Viable, but dominated, designs are also apparent. The remaining designs are nondominated. In Figure 1.25(b) only the nondominated designs are shown, and the fitness elements are reciprocated so that mass and loss can be plotted directly. Again, each point represents the mass and loss of an individual design. Note the tradeoff between mass and loss. This will be a recurrent theme in this text. A sample design on the front is also indicated; this design is illustrated in more detail in Figure 1.26. The sample design has a mass of 0.75 kg, and a loss at rated current of 0.67 W. The inductance is just over 1 mH, and flux density at rated current is 0.617 T. The current density at rated current is 1.3 A/mm^2.

Figure 1.27 illustrates the gene distribution of the final population of designs. In Figure 1.27(a), the genes are sorted by objective 1. This means that the genes of designs with higher mass are toward the left of the parameter window, and genes of designs with lower mass are toward the right.

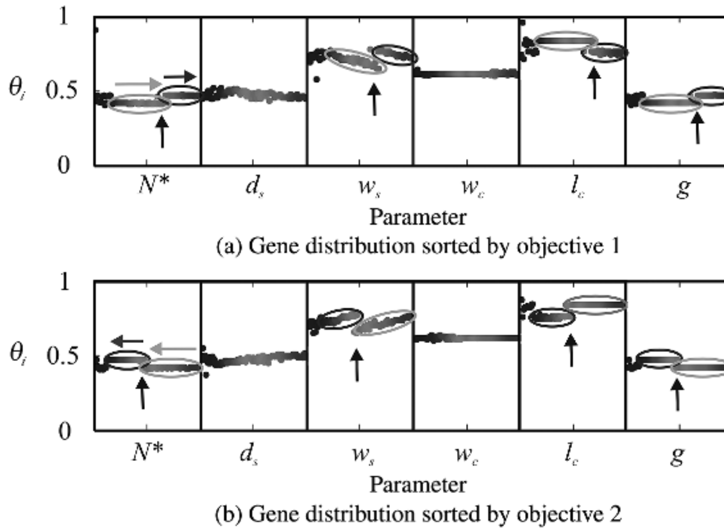

Figure 1.27 Sample design from Pareto-optimal front.

In Figure 1.27(b), the genes are sorted by objective 2, so that designs with the most loss are toward the left, and designs with the least loss are toward the right.

Unlike the case of single-objective optimization, the clustering of all values of a gene to approximately the same value is not expected in multi-objective optimization because the parameters will vary along the front. In order to illustrate this, consider the slot depth d_s. Observe that in Figure 1.27 (a) it has a slightly downward slope while in Figure 1.27(b) it has a slightly upward slope. This is because as we move from a low-mass high-loss design to a high-mass low-loss design the slot depth decreases. The core depth w_c can be seen to be approximately constant.

The remaining parameters undergo more interesting behaviors. Consider N^*, for example. Observe that in Figure 1.27(a), the nondominated solutions fall into two groups, which are indicated with a darker shaded and lighter shaded ellipses for lower and higher mass, respectively. The direction of decreasing mass is indicated with an arrow. Observe that the designs undergo a bifurcation indicated by a black vertical arrow. This can also be seen in Figure 1.27(b), wherein the sets are again circled. Note that the direction of decreasing mass is now to the left. The designs that are not in the two groups are dominated solutions. The bifurcation of the design space is also readily apparent in the slot width w_s, core length l_c, and air gap g. Such bifurcations in the design space can be the result of the change of a discrete variable, or the result of the design space moving into or out of a constraint. In this example, if we replace (10.1-1) with $N = N^*$, the bifurcation disappears. Of course, in doing this, our problem becomes strictly mathematical in nature since N must be an integer in practice.

References

1 E. K. Chong and S. H. Zak, *An Introduction to Optimization*, third edition. Hoboken: John Wiley & Sons, Inc., 2008.

2 J. F. Crow, *Genetics Notes*. New York: Macmillan Publishing Company, 1983.
3 J. H. Holland, *Adaptation in Natural and Artificial Systems*, Boston: MIT Press, 1992.
4 D. E. Goldberg, *Genetic Algorithms in Search, Optimization, & Machine Learning*. Boston: Addison Wesley Longman, Inc., 1989.
5 D. E. Goldberg, *The Design of Innovation*. Boston: Kluwer Academic Publishers, 2002.
6 S. D. Sudhoff, MATLAB codes for Power Magnetic Devices: A Multi-Objective Design Approach, second edition. Available: http://booksupport.wiley.com.
7 B. N. Cassimere and S. D. Sudhoff, Population based design of a permanent magnet synchronous machine, *IEEE Transactions on Energy Conversion*, vol. **24**, no. 2, pp. 347–357, 2009.
8 Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, third, revised and extended edition. Berlin: Springer-Verlag, 1999.
9 A. Osyczka, *Evolutionary Algorithms for Single and Multicriteria Design Optimization*. Heidelberg: Physica-Verlag, 2002.
10 G. P. Liu, J. B. Yang and J. F. Whidborne, *Multiobjective Optimization and Control*. Exeter: Research Studies Press Ltd., 2003.
11 K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Chichester: John Wiley & Sons, Ltd., 2001.

Problems

1 It is desired to minimize the function

$$g(x_1, x_2) = (x_1 - 1)^2 + 3(x_2 - 4)^2 - 200$$

What is a possible fitness function (the answer is not unique) if using a canonical GA?

2 The fitness values of the members of a population are 23, 96, 42, 12, 8, 7, and 47. What is the expected number of times the individual with a fitness of 42 will appear in the mating pool? Use roulette wheel selection.

3 Consider two parents, with $x_1 = 0.2$ and $x_2 = 0.5$. Consider simulated binary crossover with $\eta_c = 1$. Form 10^5 children, and plot a histogram of the children arranged in 20 equally spaced bins on the interval 0 to 1. Implement gene repair using ring mapping.

4 Repeat Problem 3 with $x_2 = 0.25$.

5 Suppose a design problem was coded with five genes on a single chromosome, and single-point crossover was used. During the crossover between two parents, how many different children (in terms of genotype) could be produced?

6 A logarithmically mapped gene has a range from 10^{-3} to 10^6. If the value for this gene for a particular individual is 37.6, what is its' normalized value? What would its' normalized value be if it were linearly mapped.

7 During an evolution, the minimum, maximum, and average fitness of a population is 1.2, 270, and 40, respectively. If the most fit individual is to be three times more likely than the average fit individual to be selected, and the least fit individual is 1/3 as likely as the average fit individual to be selected, what is the scaled fitness of an individual whose raw fitness is 58.

8 Below are the objective function values for an inductor design. It is desired to minimize both objectives.

Individual	**1**	**2**	**3**	**4**	**5**	**6**
Mass (kg)	5	3	1	4	2	4
Loss (kW)	2	3	6	5	4	1

Use Kung's algorithm to identify the nondominated individuals.

9 Consider Problem 8. Compute the crowding distance of all nondominated individuals.