3.2 Genetic algorithms

We shall start our study of EAs by describing a basic GA as well as its practical use in a simple function optimization example. Next, the various operators will be discussed in some detail. Consider the problem of finding the *maximum* of a function $f(x_1, \ldots, x_n)$ of n variables. The set of allowed values for the variables $\mathbf{x} = (x_1, \ldots, x_n)^T$ is referred to as the **search space**. Now, in order to apply a GA to solve this problem, the variables must be encoded in strings of digits referred to as **chromosomes**. The digits constituting the chromosome are referred to as **genes**, in accordance with the biological terminology introduced above. Thus, the genes encode the information stored in the chromosome, and there exists different **encoding schemes**. In the original GAs, introduced by Holland [32] and others in the 1970s, a **binary encoding scheme**[3] was employed in which the genes take the values 0 or 1.

When the algorithm is initialized, a **population** (i.e. a set) of N chromosomes c_i, $i = 1, \ldots, N$ is generated by assigning random values, normally with equal probability for the two alleles[4] 0 and 1, to the genes. The chromosomes thus formed constitute the first **generation**. Note that in the biological counterpart a population is defined as a set of individuals, rather than a set of chromosomes. However, in GAs the distinction between the chromosomes and the corresponding individuals is less relevant than in biological organisms, because of the simple mapping (illustrated in Example 3.1) that generates individuals from chromosomes. The complex developmental programmes seen in biological organisms are largely absent in GAs, even though exceptions exist [4, 19].

After initialization, each of the N chromosomes is decoded to form the corresponding individual, in this case consisting of the n variables[5] x_j, $j = 1, \ldots, n$. The procedure for obtaining the variables from the chromosomes can be implemented in various ways. One of the simpler ways is to divide the chromosome (of length m) into n equal parts consisting of $k = m/n$ bits, and considering each such part as a binary number which can then be converted to a decimal number representing the variable in question, as described in Example 3.1. Clearly, the accuracy of a variable is determined by the number of bits used.

---

[3] The main motivation for using binary encoding is, in fact, historical and, to some extent, aesthetic: the theoretical analysis of GAs becomes more compact and clear if a binary alphabet is used. However, in applications of GAs, other encoding methods such as real-number encoding described in Section 3.2.1 work just as well as binary encoding methods.

[4] In connection with GAs, the terms *allele* and *value* (of a gene) will be used interchangeably.

[5] When referring to the generic variables of a problem, the notation x_j, with a single subscript, will be used in order to conform with the notation used in Chapter 2. However, when referring to the variables obtained for individual i, two indices will be used, and the variables will then be written x_{ij}, where the first index enumerates the individuals, from 1 to N, and the second index enumerates the actual variables, from 1 to n.

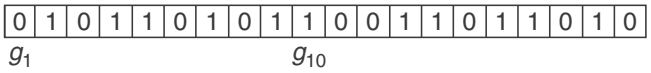

Figure 3.4: *A chromosome encoding two variables, x_1 and x_2, with 10-bit accuracy.*

Example 3.1
Consider the simple problem of finding, using a GA, the maximum of some function $f(x_1, x_2)$ of two variables that are to be encoded using 10 bits each. The search range, i.e. the allowed values for each variable, is in this example taken as $[-3, 3]$. At initialization, N strings of $m = 2 \times 10 = 20$ bits are thus formed randomly. In the decoding step for each chromosome, the first 10 bits are used to form x_1 and the remaining bits form x_2 (see also Fig. 3.4). When forming x_1, the first 10 bits of the chromosome are converted to a decimal number in the range [0, 1] according to

$$x_{1,\text{tmp}} = \sum_{j=1}^{10} 2^{-j} g_j. \tag{3.1}$$

This value is then scaled to the required range using the transformation

$$x_1 = -3 + \frac{2 \times 3}{1 - 2^{-10}} x_{1,\text{tmp}}. \tag{3.2}$$

The second variable is obtained using the last 10 genes

$$x_{2,\text{tmp}} = \sum_{j=1}^{10} 2^{-j} g_{j+10} \tag{3.3}$$

and, scaling the result in the same way as for x_1, one obtains:

$$x_2 = -3 + \frac{2 \times 3}{1 - 2^{-10}} x_{2,\text{tmp}}. \tag{3.4}$$

In the particular case shown in the figure, one gets

$$x_{1,\text{tmp}} = 2^{-2} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-10} = \frac{363}{1024} \approx 0.3545 \tag{3.5}$$

so that

$$x_1 = -3 + 6 \times \frac{1024}{1023} \times \frac{363}{1024} = -\frac{27}{31} \approx -0.8710. \tag{3.6}$$

x_2 can be obtained in a similar way. ■

Once the variables have been obtained, the individual is evaluated, a procedure that differs from problem to problem. The aim of the evaluation is to assign a **fitness value**, which can later be used when selecting individuals for reproduction. The fitness of individual i is denoted F_i. Normally, in GAs as in the biological counterpart, the fitness value is a goodness measure rather than an error measure, i.e. the aim is to *maximize* fitness. However, this does not, of course, exclude

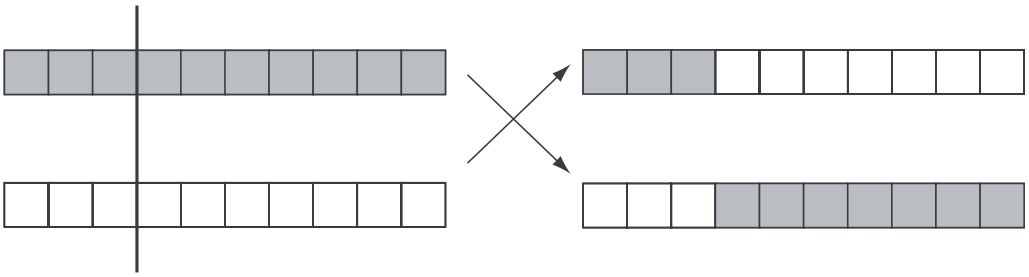

Figure 3.5: *The standard crossover procedure used in GAs. Each square corresponds to a gene. The crossover point, which is indicated by a thick line, is chosen randomly.*

minimization: minimizing a given measure is equivalent to maximizing its inverse. In the case of function maximization the evaluation is simple: the function value itself can be taken as the fitness value.

The procedure of decoding the chromosome, evaluating the corresponding individual and assigning a fitness measure is repeated until all N individuals have been evaluated. The next step is to form the second generation. Returning briefly to the biological counterpart, we can list the steps required to do so. First of all, there must be a process of **selection** in which the most fit individuals are selected as progenitors. However, we may also appreciate that the selection procedure should not be fully deterministic, i.e. it should not *always* favour the most fit individuals. Such a greedy procedure would easily lead to stagnation, especially in view of the fact that the initial population is generated randomly: an individual that happens to be better than the others, but still far from the global optimum may, if deterministic selection is applied, come to dominate the population, thus preventing the GA from finding the global optimum. In some cases, a less fit individual may contain a sequence of genes (or, to be strict, alleles) that will generate a highly fit individual when combined with genetic material from another individual, assuming that sexual reproduction is applied. Thus, the selection process is normally stochastic. A common approach is to select individuals (from the entire population) in direct proportion to their fitness, even though other methods exist as well, as we shall see below.

After selection, new individuals are formed through **reproduction**. In sexual reproduction, the genetic material of two individuals is combined. Individuals are therefore selected in pairs, using the selection procedure introduced above. In a standard GA, the genetic material is contained in a single chromosome of given length, with only one strand (unlike the double-stranded DNA molecules in biological organisms). Thus, a simple procedure for combining the genetic material of two individuals, a process referred to as **crossover**, consists of cutting the chromosomes at a randomly selected **crossover point** and then assembling the first part of the first chromosome with the second part of the second chromosome, and vice versa, as illustrated in Fig. 3.5.

The next step in the formation of new individuals is mutation. Of course, in a computer the copying errors referred to as mutations in biological organisms can

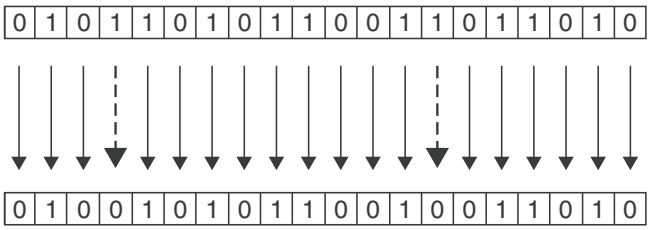

Figure 3.6: *The mutation procedure. Each gene is mutated with a small probability* p_{mut}. *In this case two mutations, marked with dashed arrows, occurred.*

easily be avoided. However, these errors play a crucial role in providing evolution with new material to work with. Thus, in GAs, once the new chromosomes have been generated through crossover, they are subjected to mutations in the form of random variation (bit flipping) of some, randomly selected, genes. Typically, mutations are carried out on a gene-by-gene basis in which the probability of mutation of any given gene equals a pre-specified **mutation probability** p_{mut}. In practice, this is done by generating, for each gene, a random number $r \in [0, 1]$, and mutating the gene if $r < p_{\text{mut}}$. The procedure is illustrated in Fig. 3.6.

The final step is **replacement**: after selection, crossover and mutation, there are now two populations available, each with N individuals. In **generational replacement**, the individuals of the first generation are all discarded, and the N new individuals thus form the second generation, which is evaluated in the same way as the first generation. The procedure is then repeated, generation after generation, until a satisfactory solution to the problem has been found. The basic GA just described is summarized in Algorithm 3.1. Before describing the various genetic operators in detail, we shall briefly consider a specific example.

Example 3.2
Use a GA to find the maximum of the function

$$f(x_1, x_2) = \mathrm{e}^{-x_1^2 - x_2^2}, \tag{3.7}$$

in the range $x_1, x_2 \in [-2, 2]$. The function is illustrated in the top left panel of Fig. 3.7.

Solution It is elementary to see that, in the given interval, the function reaches its maximum (=1) at $x_1 = x_2 = 0$. In this simple case, one obviously does not need to apply a sophisticated optimization method, but the problem is appropriate for illustrating the basic operation of a GA, particularly because a function of two variables can easily be visualized. Assuming that a binary encoding scheme is used, the first step is to select the accuracy, i.e. the number of bits per variable (k) in the chromosomes. The overhead involved in extending the chromosome length a bit is almost negligible, so in a purely mathematical problem such as this one, where there is no measurement error, one might as well choose a rather large k. Let us set $k = 25$ so that the smallest allowed increment (see also eqn (3.9) below) equals

$$\frac{4}{1 - 2^{-k}} 2^{-k} \approx 1.19 \times 10^{-7}. \tag{3.8}$$

1. Initialize the population by randomly generating N binary strings (chromosomes) $c_i, i = 1, \ldots, N$ of length $m = kn$, where k denotes the number of bits per variable.
2. Evaluate the individuals:
   - 2.1. Decode chromosome c_i to form the corresponding variables $x_{ij}, j = 1, \ldots, n$ (or, in vector form, $\mathbf{x}_i$).
   - 2.2. Evaluate the objective function f using the variable values obtained in the previous step, and assign a fitness value $F_i = f(\mathbf{x}_i)$.
   - 2.3. Repeat steps 2.1 and 2.2 until the entire population has been evaluated.
3. Form the next generation:
   - 3.1. Select two individuals i_1 and i_2 from the evaluated population, such that individuals with high fitness have a greater probability of being selected than individuals with low fitness.
   - 3.2. Generate two new chromosomes by crossing the two selected chromosomes c_{i_1} and c_{i_2}.
   - 3.3. Mutate the two chromosomes generated in the previous step.
   - 3.4. Repeat steps 3.1–3.3 until N new individuals have been generated. Then replace the N old individuals by the N newly generated individuals.
4. Return to step 2, unless the termination criterion has been reached.

Algorithm 3.1: *Basic genetic algorithm. See the main text for a complete description of the algorithm.*

The initial step of the GA thus comprises generating random strings consisting of $m = 2 \times 25 = 50$ bits. The choice of the population size N normally affects the results. Typical values of the population size range from around 30 to 1,000. For this simple problem, however, we shall choose $N = 10$.

The initial population is shown in the top right panel of Fig. 3.7. Evaluating the 10 individuals, it was found, in the run illustrated in the figure, that the best individual of the initial generation had a fitness of around 0.5789. Next, the second generation was formed. Selection was carried out stochastically in direct proportion to fitness (using the roulette-wheel method described in Section 3.2.1 below). The selection procedure was carried out 10 times, with replacement, meaning that individuals could be selected more than once. From each of the five pairs thus generated, new chromosomes were formed using the crossover procedure shown in Fig. 3.5, with randomly selected crossover points. Next, the newly formed chromosomes were mutated, as shown in Fig. 3.6, with the mutation rate $p_{\text{mut}} = 0.02$ so that, on average, one bit per chromosome was changed. The best individual in the second generation obtained a fitness of around 0.8552. The progress of the GA in a typical run is shown in the two lower panels of Fig. 3.7. In this run, a value of $f(x_1, x_2) \approx 0.9987$ was obtained after 25 generations, i.e. after the evaluation of 250 individuals.

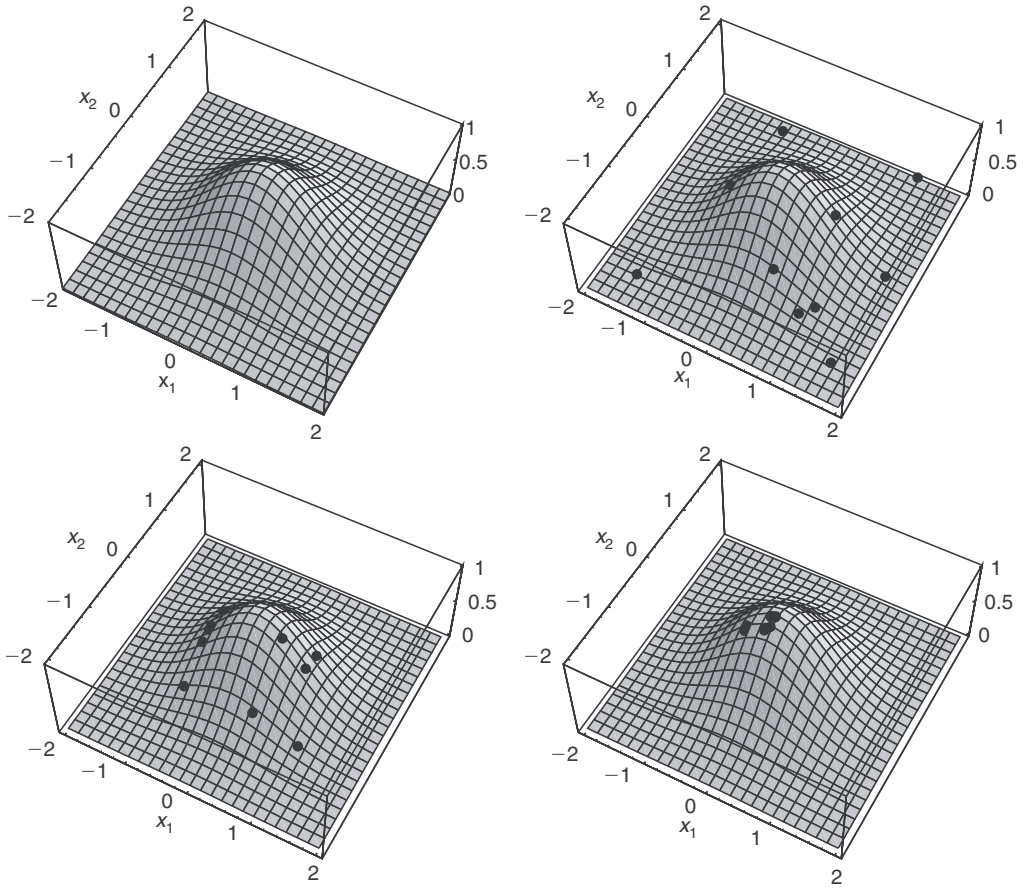

Figure 3.7: *The progress of a GA searching for the maximum of the function* $f(x_1, x_2) = \mathrm{e}^{-x_1^2 - x_2^2}$. *The upper right panel shows the initial population, whereas the lower left and lower right panels show the population after 2 and 25 generations, respectively.*

Note that the results shown in Fig. 3.7 are taken from a single run. However, because of the stochastic nature of the GA, different runs take different paths towards the optimum, a fact that can be illustrated by letting the GA run many times, starting from different random initial populations. Such an analysis was carried out, repeating the run described above 100 times, in each run evaluating 250 individuals. The result was an average best fitness in the final generation (over the 100 runs) of 0.9810, with a span from a worst result of 0.7522 to a best result of 0.9999. The median of the best fitness in the final generation was 0.9935.

It should be noted that these results are an underestimate of the performance of the GA, because there is no guarantee, following Algorithm 3.1, that the best individual of a given generation is not eliminated during selection or as a result of crossover or mutation. Thus, the results in the final generation are not necessarily the best results obtained. For example, in the run depicted in Fig. 3.7, an individual with fitness 0.9995 was found in generation 22, but it was eliminated in the subsequent selection stage. In order to avoid losing good individuals, the best individual of any given generation is normally copied unchanged to the next generation, a procedure called elitism that will be studied towards the end of Section 3.2.1. ■

3.2.1 Components of genetic algorithms

As illustrated above, GAs involve several steps, and each component or operator[6] (e.g. selection, crossover, etc.) can be implemented in various ways. In addition, most components are associated with one or more parameters that, of course, must be set to appropriate values. In this section, we shall consider the components and parameters of GAs, giving a few alternative implementations for each component. However, just knowing of the existence of different choices of components is not sufficient: one must also be able to *choose* wisely among the available options, and to set the corresponding parameters correctly. Alas, there is no single choice of components and parameters that does better than other choices over all problems. However, by considering the performance of different GAs on benchmark problems, one may at least draw some conclusions regarding the performance of various (combinations of) components and the appropriate range of their parameters. A set of benchmark functions, some of which will be used in the examples below, is given in Appendix D.

3.2.1.1 Encoding schemes

In Example 3.1, standard binary encoding was used, in which the allowed alleles are 0 and 1, and where a generic variable x is formed from genes $g_1, \ldots, g_k$ as

$$x = -d + \frac{2d}{1 - 2^{-k}}(2^{-1}g_1 + \cdots + 2^{-k}g_k), \tag{3.9}$$

giving a value in the range $[-d, d]$. Binary encoding schemes were employed in the original GAs introduced by Holland [32] and are still frequently used. However, even though a binary representation simplifies the analytical treatment of GAs (see Appendix B, Section B.2), in practical applications the decoding step introduced in eqn (3.9) may be perceived as unnecessary, and it can be avoided by using **real-number encoding** in which each gene g holds a floating-point number in the range $[0, 1]$ from which the corresponding variable x is simply obtained as[7]

$$x = -d + 2dg, \tag{3.10}$$

again resulting in a value in the range $[-d, d]$.

---

[6] The terms *component* and *operator* will be used interchangeably in connection with GAs.

[7] It is, of course, possible to let the range of the genes be $[-d, d]$ so that even the rescaling in eqn (3.10) becomes unnecessary. However, in some problems, different genes should have different ranges. Thus, by using the range $[0, 1]$ throughout the chromosome, and then rescaling according to eqn (3.10), only the decoding procedure of the GA must be adapted to the problem at hand, whereas, for example, the initialization and mutation procedures can be applied without any particular adaptation.

While there is no systematic difference in the performance of GAs as a result of choosing either binary or real-number encoding, it should be noted that in real-number encoding each gene carries more information than in binary encoding. Therefore, a random change in one gene in a binary chromosome generally leads to a smaller change in the corresponding variable x than if real-number encoding is used. For this reason, specialized mutation operators (creep mutations), which are discussed later in this section, are commonly used in connection with real-number encoding schemes. One should also note that the number of potential crossover points is reduced significantly if real-number encoding is used. Thus, if the problem under consideration involves only a small number of variables (2–3, say), the positive effects of crossover may be strongly diminished if real-number encoding is used. However, because GAs are typically applied in problems involving many variables, this problem rarely occurs in practice.

Returning to binary encoding schemes, we note that a small change in a variable x may require flipping many bits which, in turn, is an unlikely event. Thus, the algorithm may get stuck simply as a result of the encoding scheme. Consider, as an example, a 10-bit binary encoding scheme, applied in a case with a single variable x, and assume that the best possible chromosome is 1000000000. Now, if the population happens to converge to 0111111111, the decoded value x will be near the optimal one, but taking the final step to the very best chromosome will require flipping all 10 bits in the chromosome. An alternative representation which avoids this problem is the **Gray code** [25]. A Gray code is simply a binary representation of the integers in the range $[0, 2^k - 1]$ such that going from an integer i to $i+1$ requires only 1 bit to change in the representation. Letting $\gamma(g_1, \ldots, g_k)$ denote the integer i obtained from a set of genes $g_1, \ldots, g_k$, the corresponding variable x (for use in a GA) can be obtained as

$$x = -d + \frac{2d}{1 - 2^{-k}} 2^{-k} \gamma(g_1, \ldots, g_k). \tag{3.11}$$

A Gray code representation of the numbers 0, 1, 2, 3 is given by $\gamma(00) = 0$, $\gamma(01) = 1$, $\gamma(11) = 2$, $\gamma(10) = 3$. Clearly, several different Gray code representations can be generated for the 2-bit case (another example is $10 \leftrightarrow 0$, $11 \leftrightarrow 1$, $01 \leftrightarrow 2$, $00 \leftrightarrow 3$). However, those representations differ from the original one only in that the binary numbers have been permuted or inverted. An interesting question is whether the Gray code is unique if permutations and inversions are disregarded. The answer turns out to be negative for $k > 3$ (see Exercise 3.3).

A further alternative to the standard binary encoding scheme is **messy encoding**, which generates a less position-dependent representation by associating each gene with a number determining its position in the chromosome. Thus, a messy chromosome of length m is represented as

$$c = ((p_1, g_{p_1}), (p_2, g_{p_2}), \ldots, (p_j, g_{p_j}), \ldots, (p_m, g_{p_m})), \tag{3.12}$$

where p_j denotes the position in the chromosome, and g_{p_j} the corresponding allele. Thus, for example, the messy chromosome ((1,0),(5,1),(3,0),(4,1),(2,1)) represents the chromosome 01011. When messy encoding is used, the initialization procedure consists of generating, for each messy chromosome, a random permutation of the numbers $1, \ldots, m$ to be used as the position markers p_j as well as a sequence of random bits to be used as the alleles g_{p_j}. Two obvious problems that may occur in connection with messy encoding schemes are missing positions and duplicates. In the latter case, a messy chromosome may (as a result of crossover) contain several copies of a given gene position p_j, e.g. ((1,0),(3,1),(3,0),(2,1),...). One way of resolving such conflicts is simply to use the first occurrence of a given gene position and thus discard all other occurrences. The case of missing gene positions can be mitigated simply by extending the length of the messy chromosomes beyond the required length m, thus reducing (but not eliminating) the probability of missing gene positions as a result of crossover. If some positions are still missing, one may, for example, set those missing positions to a given value, i.e. either 0 or 1.

The encoding schemes encountered thus far are all applicable in problems involving, say, function optimization, where the chromosomes are required to generate a vector $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$ of real-valued variables. In addition, other encoding schemes, tailored to particular problems, exist as well. An example is **permutation encoding** in which each chromosome holds a permutation (i.e. an ordering) of the integers $1, \ldots, n$. Permutation encoding is applicable, for example, in the travelling salesman problem (TSP) considered in Chapter 4.

3.2.1.2 Selection

Selection, the process of choosing the chromosomes that will be used when forming new individuals, can be carried out in many different ways; the two most common are **roulette-wheel selection** and **tournament selection**. In roulette-wheel selection, individuals are selected from the population using a fitness-proportional procedure, which can be visualized as a roulette-wheel (hence the name) such that each individual occupies a slice proportional to its fitness. When this imaginary wheel is turned and eventually comes to a halt, the probability of an individual being selected will be proportional to its fitness. The roulette wheel is, of course, just a metaphor. In practice, the procedure is carried out by forming the cumulative relative fitness values ϕ_j, defined as

$$\phi_j = \frac{\sum_{i=1}^{j} F_i}{\sum_{i=1}^{N} F_i}, \quad j = 1, \ldots, N, \tag{3.13}$$

where F_i denotes the fitness of individual i. Next, a random number $r \in [0, 1]$ is drawn, and the selected individual is taken as the one with the smallest j that satisfies

$$\phi_j > r. \tag{3.14}$$

The procedure is illustrated in Example 3.3.

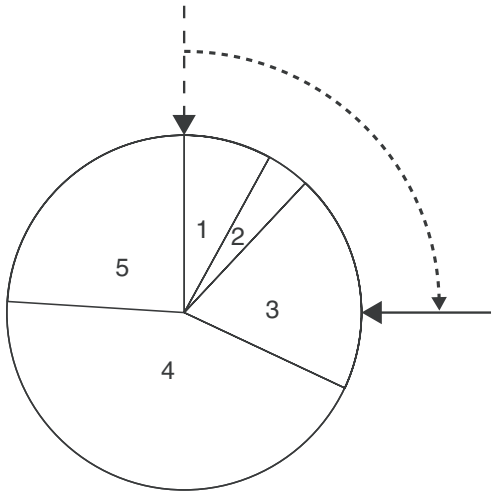

Figure 3.8: *Roulette-wheel selection in a population consisting of five individuals, described in detail in Example 3.3.*

Example 3.3
A roulette-wheel selection step is to be carried out on a population consisting of $N = 5$ individuals, with fitness values $F_1 = 2.0$, $F_2 = 1.0$, $F_3 = 5.0$, $F_4 = 11.0$ and $F_5 = 6.0$. The random number $r = 0.25$ is drawn. Which individual is selected?

Solution The roulette-wheel is illustrated in Fig. 3.8. The random number $r = 0.25$ corresponds to a rotation of 90 degrees (i.e. a quarter of a circle) of the wheel or, equivalently, rotation of the stopping point (indicated by an arrow in the figure) by 90 degrees. In this case, the arrow points to individual 3, which is thus selected. Mathematically, one would form the sum $\sum_{i=1}^{5} F_i = 25$ and then compute ϕ_j according to eqn (3.13). In this case (check!), $\phi_1 = 0.08$, $\phi_2 = 0.08 + 0.04 = 0.12$, $\phi_3 = 0.32$, $\phi_4 = 0.76$, and $\phi_5 = 1.00$. Thus, the smallest j for which $\phi_j > r = 0.25$ is $j = 3$, resulting in the selection of individual 3. ■

Evidently, roulette-wheel selection is not very plausible from a biological point of view. An alternative procedure that better resembles the typical contests between pairs of animals (usually males) so often seen in nature is tournament selection. In its simplest form, tournament selection consists of picking two individuals randomly (i.e. with equal probability for all individuals, and, for simplicity, with replacement[8]) from the population, and then selecting the best individual of the pair, i.e. the one with higher fitness, with probability p_{tour}. Thus, the worse individual is selected with probability $1 - p_{\text{tour}}$. p_{tour} is referred to as the **tournament selection**

[8] If one wishes to eliminate the possibility of picking a pair consisting of two copies of the same individual, one may simply avoid using replacement, i.e. pick the second individual of the pair from the remaining $N - 1$ individuals. However, the probability of picking the same individual twice equals $1/N^2$ and, therefore, doing so has no systematic adverse effect on the progress of a typical GA run where N is of order 10^2 or larger.

parameter, and it typically takes values around 0.7–0.8. In practice, tournament selection is achieved by drawing a random number r in [0, 1], and selecting the better of the two individuals if $r < p_{\text{tour}}$. Otherwise, the worse individual is selected. Tournament selection can be generalized to involve more than two individuals. In the general case, j individuals are selected randomly from the population and the best individual is selected with probability p_{tour}. If this individual is not selected, the next step is to repeat the procedure for the remaining $j - 1$ individuals, again with probability p_{tour} of selecting the best individual. j is referred to as the **tournament size**. Example 3.4 illustrates the selection of a pair of individuals using tournament selection.

Example 3.4
Consider again the population of five individuals introduced in Example 3.3. An individual is to be selected using tournament selection with tournament size equal to two, and $p_{\text{tour}} = 0.8$. What is the probability p_1 of selecting individual 1 with fitness $F_1 = 2.0$?

Solution Because the selection takes place with replacement, all pairs of individuals may occur, with equal probability, in this case equal to $1/(5 \times 5) = 1/25$. The pairs can be represented on a grid, as shown in Fig. 3.9. Now, for any given pair $(1, k)$ or $(k, 1)$ involving individual 1 (shown as shaded grid cells in the figure), the probability of selecting that individual equals p_{tour} if $F_1 > F_k$ and $1 - p_{\text{tour}}$ otherwise, except for the pair (1, 1) consisting of two copies of individual 1, for which selection of that individual obviously occurs with probability 1. Thus, with the fitness values F_i given in Example 3.3, one obtains

$$p_1 = \frac{1}{25}(1 + 2p_{\text{tour}} + 6(1 - p_{\text{tour}})) = \frac{1}{25}(7 - 4p_{\text{tour}}) = 0.152. \tag{3.15}$$

■

It should be noted that in tournament selection negative fitness values are allowed in principle, whereas in roulette-wheel selection all fitness values must be non-negative. Furthermore, in tournament selection it is only the relative fitness values that matter, not their absolute difference. This is not the case with roulette-wheel selection and, in fact, a common problem is that an individual which happens to have above-average fitness values in the first, randomly generated generation comes

(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)

Figure 3.9: *Tournament selection, with tournament size equal to two, in a population consisting of five individuals, described in detail in Example 3.4.*

to dominate the population, which, in turn, sometimes leads to the GA getting stuck at a local optimum. This problem can, to some extent, be avoided by using **fitness ranking** which is a procedure for reassigning fitness values. In its simplest form, the best individual is given fitness N (i.e. equal to the population size), the second best individual is given fitness $N-1$, and so on, down to the worst individual that is given fitness 1. Letting $R(i)$ denote the ranking of individual i, defined such that the best individual i_{best} has ranking $R(i_{\text{best}}) = 1$, the fitness values obtained by ranking can thus be summarized as

$$F_i^{\text{rank}} = (N + 1 - R(i)). \tag{3.16}$$

The ranking procedure can be generalized as

$$F_i^{\text{rank}} = F_{\max} - (F_{\max} - F_{\min}) \left(\frac{R(i) - 1}{N - 1} \right), \tag{3.17}$$

which yields equidistant fitness values in the range $[F_{\min}, F_{\max}]$. The fitness values obtained before ranking, i.e. F_i, are referred to as **raw fitness values**, and we note that ranking removes the requirement that (raw) fitness values should be non-negative in connection with roulette-wheel selection.

The selection methods introduced thus far are the two most common ones. However, they suffer from one potential drawback: unless the selection parameters are gradually adjusted,[9] the extent to which high-fitness individuals are preferred over low-fitness ones (the **selection pressure**) is constant over time, something that might not always be favourable. In the early stages of a GA run, none of the individuals are likely to be very good compared to the global optimum, and the selection procedure would do well to include a natural, easily parameterized, mechanism for exploring the search space more or less freely in the early stages, without paying too much attention to fitness values, and then towards the end of a run resort to making small adjustments of the current best individuals. Thus, to summarize, the two selection methods just presented do not address the dilemma of exploration versus exploitation.

To mitigate this, one may use **Boltzmann selection**, a procedure that introduces concepts from physics into the mechanisms of GAs. In this selection scheme, the notion of a temperature T is introduced, and the basic idea is to use T as a tunable parameter for determining the selection pressure. The method derives its name from the fact that the corresponding equations (see below) are similar to the Boltzmann distribution which, among other things, can be used for determining the distribution of particle speeds in a gas. Boltzmann selection can be implemented in different ways that correspond either to roulette-wheel selection or tournament selection. An

[9] Specifically, the selection pressure exerted by tournament selection can be adjusted during a GA run by increasing either the tournament size or the tournament selection parameter p_{tour} (or both).

example of the former is to select individual i with probability p_i given by

$$p_i = \frac{e^{\frac{F_i}{T}}}{\sum_{j=1}^{N} e^{\frac{F_j}{T}}}, \tag{3.18}$$

where, as usual, F_i denotes the fitness of individual i. Eqn (3.18) shows that individuals are selected with approximately equal probabilities if T is large, whereas for small T, individuals with high fitness are more likely to be selected. A Boltzmann selection scheme that resembles tournament selection first picks two individuals j and k randomly from the population. Next, the function

$$b(F_j, F_k) = \frac{1}{1 + e^{\frac{1}{T}\left(\frac{1}{F_j} - \frac{1}{F_k}\right)}} \tag{3.19}$$

is considered, where F_j and F_k are the fitness values of the two individuals in the pair. During selection, a random number r is generated and the selected individual i is determined according to

$$i = \begin{cases} j & \text{if } b(F_j, F_k) > r, \\ k & \text{otherwise.} \end{cases} \tag{3.20}$$

If T is large, selection occurs with almost equal probability for both individuals regardless of their fitness values. However, if T is small, the better of the two individuals is selected with a probability that approaches one as T tends to zero.

Thus, by starting from large values of T and then gradually lowering T towards zero, the GA will first carry out a more or less aimless exploration, and then as T is reduced, focus on exploitation of the results found. On the other hand, the introduction of Boltzmann selection raises the issues of how to set the initial value of T and how to reduce T during optimization, a question to which there is no definitive answer. Typically, the variation of T must be set through experimentation (see Exercise 3.4).

Selection normally occurs in a pairwise manner, i.e. the selection procedure is carried out twice, in order to obtain two individuals (or, rather, their chromosomes) that can then be subjected to crossover and mutation. As a final point, it should be noted that selection is done *with replacement*, i.e. the selected individual is *not* removed from the pool of available individuals. Thus, it is possible for a single individual to be the progenitor of several new individuals.

3.2.1.3 Crossover

Crossover is an essential component in GAs. It allows partial solutions from different regions of the search space to be assembled, thus allowing for a wide-ranging, non-local search. However, crossover may not always be beneficial: in GAs, typical population sizes (N) are around 30–1,000, much smaller than the thousands

or millions of individuals found in many (though not all) biological populations. Through crossover, a successful individual will spread very quickly in the population, hence reducing diversity, a process akin to inbreeding in natural populations. Thus, crossover is, in fact, a bit *too* efficient and may cause the population to become trapped at a local optimum. It is therefore common to carry out crossover only with a certain probability p_c, referred to as the **crossover probability**. In cases where crossover is *not* carried out, the two selected individuals are simply copied as they are.

The crossover procedure can be implemented in various ways. The simplest (and most common) version, illustrated in Fig. 3.5, is **single-point crossover** in which a single crossover point is randomly chosen among the $m-1$ possible points in a chromosome of length m. The procedure can be generalized to k-point crossover in which k crossover points are selected randomly and the chromosome parts are chosen from either parent[10] with equal probability. In **uniform crossover**, the number of crossover points is equal to $m-1$.

These crossover methods are applicable both to binary and real-valued chromosomes. An additional crossover method applicable to real-valued chromosomes is **averaging crossover** in which a gene g taking the values g_1 and g_2 in the two parents takes the values

$$g_1 \leftarrow \alpha g_1 + (1-\alpha) g_2 \tag{3.21}$$

and

$$g_2 \leftarrow (1-\alpha) g_1 + \alpha g_2 \tag{3.22}$$

in the offspring. α is a number in the range $[0, 1]$. All crossover procedures introduced thus far are instances of **length-preserving crossover**, which is applicable in cases such as function optimization when the structure of the system (e.g. the number of variables) being optimized is known and fixed. However, in many situations, this is not the case. An example that will be considered in Section 3.3 is linear genetic programming in which the crossover procedure is allowed to modify the size of the chromosomes.

3.2.1.4 Mutation

Mutations play the important role of providing new material for evolution to work with. While mutations rarely have a positive effect on fitness when they occur, they may bring advantages in the long run. In GAs, the value of the mutation probability p_{mut} is usually set by the user before the start of the GA run, and is thereafter left unchanged. Typical values for p_{mut} are c/m, where c is a constant of order 1 and m is the chromosome length, implying that on average around one mutation occurs per chromosome. A brief motivation for this choice can be found in Appendix B,

[10] In obvious analogy with biology, the selected individuals in an EA are sometimes referred to as the **parents**, and the new individuals that are generated are referred to as the **offspring**.

Section B.2.5. However, note that mutations are normally carried out on a gene-by-gene basis, i.e. the decision whether or not to mutate is considered independently for each gene. Thus, in principle, the number of mutated genes can range from 0 to m.

Varying mutation rates can, of course, also be applied. However, if a varying mutation rate is used, one immediately encounters the problem of specifying exactly *how* it should vary. Thus, constant mutation rates are more commonly used. In case of binary encoding schemes, mutation normally consists of bit-flipping, i.e. a mutated 0 becomes a 1, and vice versa. By contrast, in real-number encodings, the modifications obtained by randomly selecting a new value in the allowed range (e.g. $[0, 1]$) are typically much larger than in the binary case, because each real-valued gene encodes more information than a binary-valued one. Thus, an alternative approach, known as **real-number creep**, is frequently used instead. In real-number creep, the mutated value (allele) of a gene is centered on the previous value, and the **creep rate** determines how far the mutation may take the new value, as illustrated in Fig. 3.10. Mathematically, in creep mutation, the value g of the gene changes according to

$$g' \leftarrow \psi(g), \tag{3.23}$$

where ψ is a suitable distribution, for instance a uniform distribution as in the left panel of Fig. 3.10, or a normal distribution as in the right panel of the same figure. In the case of a uniform distribution, the mutated value can thus be obtained as

$$g \leftarrow g - \frac{C_\mathrm{r}}{2} + C_\mathrm{r} r, \tag{3.24}$$

where r is a uniform random number in $[0, 1]$ and C_r (the creep rate) is the width of the distribution.

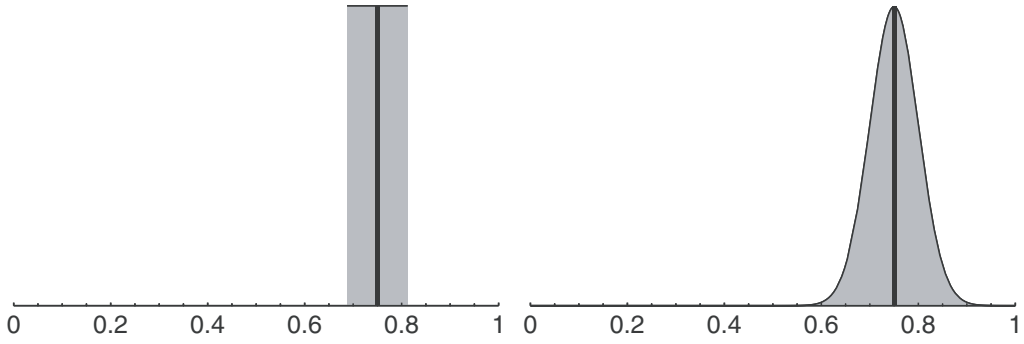

Figure 3.10: *Creep mutation using a uniform distribution (left panel) or a normal distribution (right panel). The initial value (equal to 0.75) of the gene* g*, before mutation, is illustrated as a thick black line. The mutated value is selected from the distributions shown in grey. For both distributions, the area of the shaded region is equal to 1. Random numbers following a normal distribution can be obtained using the Box–Müller transform described in Appendix C.*

In case a value outside the allowed range (e.g. [0, 1]) is obtained, which may happen if, say, g is close to one of the limits (0 or 1) and the distribution $\psi(g)$ is sufficiently wide, the mutated value is instead set to the limiting value.

3.2.1.5 Replacement

In Algorithm 3.1, generational replacement was used, meaning that all individuals in the evaluated generation were replaced by an equal number of offspring generated by applying selection, crossover and mutation. While common in GA applications, generational replacement is not very realistic from a biological point of view. In nature, different generations coexist, and individuals are born and die continuously, not only at specific intervals of time. By contrast, in generational replacement, there is no direct competition between individuals from different generations. Note, however, that there may still be some *de facto* overlap between generations, if the crossover and mutation rates are sufficiently low, because it is likely that some of the offspring will then be identical to the parents. However, this is a situation that one obviously wishes to avoid because re-evaluation of identical individuals does not improve the results obtained by the GA.

In general, replacement methods can be characterized by their **generational gap**, which simply measures the fraction of the population that is replaced in each selection cycle. Thus, for generational replacement, $G = 1$. In **steady-state replacement**, G equals $2/N$, i.e. two individuals are replaced in each selection cycle. In order to keep the population size constant, NG individuals must be deleted. In steady-state replacement, the two new individuals are typically evaluated immediately after being generated, and the two worst individuals, either among the original N individuals or among the $N + 2$ individuals (including the offspring), are deleted.

3.2.1.6 Elitism

Even though the best individual in a given generation is very likely to be selected for reproduction, there is no guarantee that it will be selected. Furthermore, even if it is selected, it is probable that it will be destroyed during crossover. In order to make sure that the best individual is not lost, it is common to make one or a few exact copies of this individual and place them directly, without any modification, in the new generation being formed, a procedure known as **elitism**. If the population size N is even, and if a single copy of the best individual is retained as the first individual in the new population, $N - 2$ new individuals are formed via the usual sequence of selection, crossover and mutation. The final individual, completing the new population, is formed using asexual reproduction, i.e. only selection and mutation.

3.2.1.7 A standard genetic algorithm

By summarizing the discussion above one can define a standard GA, see Algorithm 3.2. Note that the standard GA is quite similar to the one introduced in Algorithm 3.1; the only difference is the introduction of a crossover probability p_c and elitism. The choice of selection operator is not specified, meaning that both roulette-wheel selection and tournament selection are appropriate selection

1. Initialize the population by randomly generating N binary strings (chromosomes) c_i, $i = 1, \ldots, N$ of length $m = kn$, where k denotes the number of bits per variable.
2. Evaluate the individuals:
   - 2.1. Decode chromosome c_i to form the corresponding variables x_{ij}, $j = 1, \ldots, n$.
   - 2.2. Evaluate the objective function f using the variable values obtained in the previous step, and assign a fitness value $F_i = f(\mathbf{x}_i)$.
   - 2.3. Repeat steps 2.1–2.2 until the entire population has been evaluated.
   - 2.4. Set $i_{\text{best}} \leftarrow 1, F_{\text{best}} \leftarrow F_1$. Then loop through all individuals; if $F_i > F_{\text{best}}$, then $F_{\text{best}} \leftarrow F_i$ and $i_{\text{best}} \leftarrow i$.
3. Form the next generation:
   - 3.1. Make an exact copy of the best chromosome $c_{i_{\text{best}}}$.
   - 3.2. Select two individuals i_1 and i_2 from the evaluated population, using a suitable selection operator.
   - 3.3. Generate two offspring chromosomes by crossing, with probability p_c, the two chromosomes c_{i_1} and c_{i_2} of the two parents. With probability $1 - p_c$, copy the parent chromosomes without modification.
   - 3.4. Mutate the two offspring chromosomes.
   - 3.5. Repeat steps 3.2–3.4 until $N - 1$ additional individuals have been generated. Then replace the N old individuals by the N newly generated individuals.
4. Return to step 2, unless the termination criterion has been reached.

Algorithm 3.2: *A standard genetic algorithm applied to the case of function maximization. See the main text for a detailed description.*

operators in the standard GA. Furthermore, the standard GA uses binary encoding, but it can easily be modified to operate with real-number encoding, in which case creep mutations should also be included in step 3.4. Note that the fitness of the best individual F_{best} in Algorithm 3.2 refers to the global best fitness value, i.e. the best value found so far, in any generation. This is so since a copy of the best individual found so far is made in each generation, see step 3.1.

Figure 3.11 shows the results of a typical GA run in which elitism was used. As can be seen in the figure, the maximum fitness values (F_{best}) rise rapidly at first, before entering a phase in which long periods of stasis are interrupted by sudden jumps.

3.2.1.8 Parameter selection

As is evident from the discussion above, applying a GA to an optimization problem involves a non-trivial selection of components and parameters, and even though

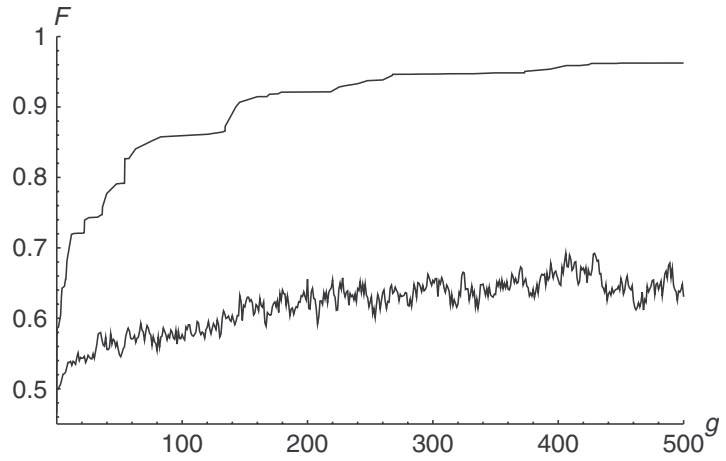

Figure 3.11: *A typical GA run for the maximization of the function* $\Psi_5^{[10]}$ *from Appendix D. The curves show the best (upper curve) and average fitness values obtained, as functions of the generation* g. *The fitness measure for this maximization task was simply taken as the function value. The population consisted of 100 individuals.*

there is no setup that is consistently superior over a wide range of problems, one can nevertheless exclude, for example, some particularly poor parameter choices. A few examples of such investigations will now be given.

Example 3.5

In this example, we shall compare roulette-wheel selection and tournament selection, keeping other operators and parameters fixed. A standard GA was implemented according to Algorithm 3.2 and was then applied to the problem of minimizing two of the benchmark functions introduced in Appendix D. Thirty bits per variable were used in the binary encoding scheme, giving a total chromosome length of $m = 30n$, where n is the number of variables, and m the chromosome length. The crossover rate was set to $p_c = 1.0$, and the mutation rate was set to $3/m$. For the two benchmark functions considered in this example, the (raw) fitness measures used were $F = 1/\Psi_1(x_1, x_2)$ and $F = 1/(1 + \Psi_2^{[3]}(x_1, x_2, x_3))$, respectively. Several different selection methods were tried. For each method, 3,000 runs were carried out on each benchmark problem, evaluating 10,000 individuals per run. For each run, the best fitness value obtained was stored.

Even though the distribution of the best result (from individual runs) is unknown, and probably quite different from a normal distribution, the distribution of the mean of the best result (over many runs) will approach a normal distribution, as described in Appendix C, Section C.1. The runs were therefore divided into 100 groups of 30 runs each. For each group, the average of the best result over the 30 runs was formed, giving one sample from this approximate normal distribution. The average and standard deviation of the 100 samples is given in Table 3.1. The estimates of the standard deviation s were obtained using eqn (C9).

From the table, it can be concluded that fitness ranking has a strong positive influence on the results obtained using roulette-wheel selection. For tournament selection, no particular conclusions can be drawn: a tournament size of five gives (slightly) better results than a tournament size of two for Ψ_1, but the opposite holds for Ψ_2. ■

Table 3.1: *Comparison of selection methods, see Example 3.5. The first column lists the selection methods, and the remaining columns show the averages and estimated standard deviations as described in the example. Note that the listed standard deviations measure the variation (over 100 samples) in the mean (over 30 runs) of the best result obtained.*

Selection method	$\Psi_1\ (x_1, x_2)$		$\Psi_2^{[3]}\ (x_1, x_2, x_3)$	
	Avg.	S.D.	Avg.	S.D.
RWS, no ranking	3.024	9.1×10^{-3}	1.2108	0.1811
RWS, with ranking	3.000	1.5×10^{-5}	0.3735	0.1069
TS, size $= 2$, $p_{\text{tour}} = 0.70$	3.003	1.4×10^{-3}	0.4253	0.1048
TS, size $= 2$, $p_{\text{tour}} = 0.90$	3.000	1.1×10^{-4}	0.3139	0.0870
TS, size $= 5$, $p_{\text{tour}} = 0.70$	3.000	0	0.6356	0.1822
TS, size $= 5$, $p_{\text{tour}} = 0.90$	3.000	0	0.7554	0.1872

RWS = roulette-wheel selection, TS = tournament selection.

Table 3.2: *Comparison of crossover probabilities. The first column lists the crossover probabilities, and the remaining columns show the averages and estimated standard deviations as described in Example 3.5.*

p_c	$\Psi_1\ (x_1, x_2)$		$\Psi_2^{[3]}\ (x_1, x_2, x_3)$	
	Avg.	S.D.	Avg.	S.D.
0.00	3.00002	1.3×10^{-5}	1.020	0.183
0.25	3.00004	2.8×10^{-5}	0.655	0.139
0.50	3.00007	4.7×10^{-5}	0.516	0.130
0.80	3.00012	6.1×10^{-5}	0.394	0.118
0.90	3.00016	7.7×10^{-5}	0.389	0.118
1.00	3.00019	9.3×10^{-5}	0.363	0.108

Example 3.6

Using the same two benchmark functions as in Example 3.5, the effects of using different crossover probabilities were investigated for the case of single-point crossover. The population size, the number of bits per variable and the mutation rate were the same as in Example 3.5. Tournament selection was used, with a tournament size of two, and $p_{\text{tour}} = 0.90$. The raw fitness measures were the same as in Example 3.5. Three thousand runs, each evaluating 10,000 individuals over 100 generations were carried out. As in Example 3.5, the runs were divided into 100 groups of 30 runs each, giving 100 samples from the distribution of the mean. The results are summarized in Table 3.2. In this case, no particular conclusions can be drawn. For Ψ_1, the crossover operator appears to have a negative effect, whereas for Ψ_2, the best results are obtained using large values of p_c. ■

Table 3.3: *Comparison of mutation probabilities, see Example 3.7. The first column lists the mutation probabilities, and the remaining columns show the averages and estimated standard deviations as described in Example 3.5.*

p_{mut}	$\Psi_1\ (x_1, x_2)$		$\Psi_2^{[3]}\ (x_1, x_2, x_3)$	
	Avg.	S.D.	Avg.	S.D.
0	4.48910	0.75150	2.22610	0.505339
$1/2m$	3.03806	0.17743	1.35071	0.268959
$1/m$	3.00000	0.00000	1.18274	0.204092
$3/m$	3.00069	0.00041	1.23678	0.192616
$10/m$	3.04104	0.00984	1.70607	0.229125
1	4.81652	0.72717	1.27124	0.274613

Example 3.7
Again using the same two benchmark functions as in Example 3.5, the effects of using different (but constant) mutation probabilities were studied. The population size and the number of bits per variable were the same as in Example 3.5. The mutation rate was set as c/m, where m is the chromosome length (60 for Ψ_1, and 90 for $\Psi_2^{[3]}$), for different values of c. Tournament selection was used, with a tournament size of two, and $p_{\text{tour}} = 0.80$. The crossover probability was set to 0.80. The raw fitness measures were the same as in Example 3.5. Three thousand runs were carried out, each evaluating 10,000 individuals over 100 generations. As in Example 3.5, the runs were divided into 100 groups of 30 runs each, giving 100 samples from the distribution of the mean. The results, shown in Table 3.3, support the simple estimate for the optimal mutation rate ($1/m$) derived in Appendix B, Section B.2.5. ■

3.2.2 Properties of genetic algorithms

The examples above, as well as Fig. 3.11, illustrate the convergence of the population in a GA towards highly fit individuals. However, the numerical examples say nothing specific about *how* GAs work. In this section, we shall address this issue, by briefly considering a few analytical models of GAs, starting with the **Schema theorem**.

3.2.2.1 The schema theorem
The schema theorem, derived by Holland [32], is one of the earliest results regarding the performance of GAs. Even though its importance has sometimes been exaggerated in the literature, it may be instructive to study it briefly. Consider a GA with a binary encoding scheme. As the name implies, the schema theorem relies on schemata (the plural form of the word *schema*) which are defined as strings consisting of 0s, 1s and wild-card symbols, here denoted x, which represent an arbitrary allele (either 0 or 1). Thus, for example, the schema 100xx1 represents