1

Evolutionary Algorithms

This chapter presents the basic principles of evolutionary algorithms as an introduction to the subsequent chapters. After a brief history of the domain in section 1.1, a generic evolutionary model is described in section 1.2. Sections 1.3 to 1.5 detail widespread variants of the operators composing the evolutionary algorithms, with a particular emphasis on binary representation. The chapter ends with a short presentation in section 1.6 of the famous *genetic algorithms* that have the originality to favor the binary representation associated with a transcription genotype–phenotype.

1.1. From natural evolution to engineering

According to Charles Darwin [DAR 59], the evolution of living beings rests on several facts:

– the variations of individual characteristics between parents and offspring;

– the heritability of much of these characteristics;

– a competition that selects the fittest individuals of a population in relation to their environment, in order to survive and reproduce.

From these facts, Darwin deduced that competition allows the transmission of hereditary beneficial variations among individuals that accumulate from generation to generation.

In the 1950s, the development of the electronic computer facilitated the simulation of this theory and some researchers desired to test it to solve engineering problems. But these works were not convincing because of the weak performances of the calculators available at that time. Furthermore, the extreme slowness of the evolution appeared prohibitive to usefully simulate this process.

During the 1960s and 1970s, as soon as calculators of more credible capacity became accessible, many attempts to model the process of evolution were undertaken. Among those, three approaches emerged and progressed independently until the beginning of the 1990s:

– the *evolution strategies (ESs)* of H. P. Schwefel and I. Rechenberg [REC 65, BEY 01], which are derived from an experimental optimization method to solve fluid dynamics problems;

– the *evolutionary programming* (EP) of L. J. Fogel *et al.* [FOG 66] which aimed, in the mid-1960s, to evolve the structure of finite-state automata with iterated selections and mutations; it was desired to be an alternative to artificial intelligence at the time;

– *Genetic algorithms* (GAs) were presented in 1975 by J.H. Holland [HOL 92], with the objective of understanding the underlying mechanisms of systems able to self-adapt to their environment.

Thereafter, these approaches underwent many modifications according to the variety of the problems faced by their founders and their pupils. Genetic algorithms became extremely popular after the publication of the book "*Genetic Algorithms in Search, Optimization and Machine Learning*" by D. E. Goldberg in 1989 [GOL 89]. This book, distributed worldwide, resulted in exponential growth in interest in this field. While there were about a few hundred publications in this area during the 20 year period before this book was published, there are several hundreds of thousands of references related to evolutionary computation available today, according to the website "google scholar"[1]. Researchers in this field have organized common international conferences for presenting and combining their different approaches.

The widespread term *Evolutionary Computation* appeared in 1993 as the title of a new journal published by the MIT Press. It was widely used to designate all methods based on the metaphor of the biological evolution

1 https://scholar.google.com/scholar?q="genetic+algorithms"

theory, as well as many others. For example, although it is inspired by a simplified model of social behavior, it is common to consider “Particle Swarm Optimization” as an evolutionary approach. “Particle Swarm Optimization” algorithms have notable common points with other more conventional evolutionary algorithms by assimilating “swarm” and “population”, “particles” and “individuals”. According to Michalewicz [MIC 12]: “*It seems that Evolutionary Algorithms, in the broad sense of this term, provide just a general framework on how to approach complex problems.*”

1.2. A generic evolutionary algorithm

Evolutionary algorithms are iterative algorithms that make the individuals of a population evolve over a number of *generations*. A generation is in fact an iteration of the main loop of these algorithms (Figure 1.1). Any individual in a population contains at least the information required to represent a more or less efficient solution to the target problem or a part of a solution, such as in “Learning Classifier Systems” [BOO 89] [URB 09].

An evolutionary algorithm works on a population of interacting *parent* individuals to produce a set of *offspring* individuals at every *generation*. At the end of each generation, selected offspring replace parents to build the parent population at the next generation, in such a way that its size is constant or, more rarely, controlled according to a given policy.

In accordance with the Darwinist guiding principle, the fitter an individual is, the more often it is selected to reproduce or survive. To make this selection possible, a *fitness* value is attached to each individual. It may be evaluated with a given *fitness function*, or possibly by other means such as simulations. For instance, in the context of optimization tasks, the fitness function is the objective function or else it strongly depends on the objective function. For each generation, the fitnesses of the offspring have to be evaluated, which can be computation intensive if the population is large.

The *variation operators* generate new offspring from one or several parents according to a given *representation*. For instance, to solve a given problem in a domain of $\mathbb{R}^n$, this representation could be an array of n floating point numbers. But it could also be an array of binary digits that represents a vector of these n real values, with an appropriate encoding. Thus, the choice of a representation for the individuals depends not only on the problem, but also on the properties of the available variation operators. These properties have to favor the generation of good offspring as much as possible. The choice of appropriate representation and variation operators has a critical influence on the performance of evolutionary algorithms.

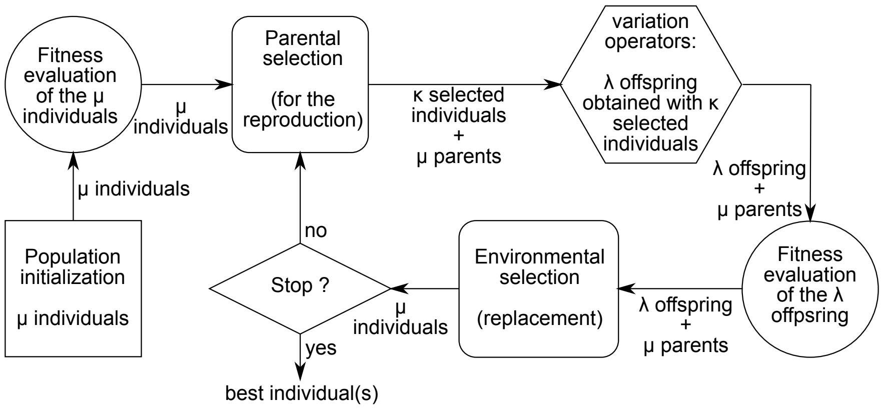
Fitness evaluation of the μ individuals
μ individuals
Parental selection (for the reproduction)
κ selected individuals + μ parents
variation operators: λ offspring obtained with κ selected individuals
λ offspring + μ parents
μ individuals
Population initialization μ individuals
no
Stop ?
yes
best individual(s)
μ individuals
Environmental selection (replacement)
λ offspring + μ parents
Fitness evaluation of the λ offpsring

Figure 1.1. *The generic evolutionary algorithm*

Interactions between individuals of a population characterize the evolutionary algorithms. Several parents can interact to generate new offspring. Offspring and parents interact to select the fittest of them for reproduction or survival through *election operators*. Obviously, when a population contains only one parent, for instance, there is no interaction between several parents. However, this particular case is still an evolutionary algorithm if the interactions are specified in the algorithm and occur when they can be applied.

Figure 1.1 depicts the generic evolutionary algorithm. The operators involved in this algorithm are generic because many variants of these operators have been proposed:

– first, the initialization step generates μ individuals to constitute a population, μ being a free parameter of the algorithm. The population initialization can be obtained from dedicated heuristics if some information about the problem to solve is known, or in the contrary case, by default, the initialization step generates the individuals randomly in the search domain according to a distribution depending on the evolutionary algorithm variant used;

– the fitness values of the μ individuals are then determined during the “fitness evaluation” step;

– the *parental selection* or *selection for the reproduction*, or simply the *selection* operator determines how many times any individual will be reproduced in a generation, depending on its fitness: the higher the fitness of an individual, the more it reproduces. The selection operator generates a total of κ copies of parents that will be used by the variation operators;

– assuming that each selected copy can be used only once by the variation operators, κ has to be large enough to allow the variation operators to generate λ offspring, where λ is another free parameter of the algorithm. For instance, if the variation operators produce two offspring from one pair of parents, then $\kappa = \lambda$. If the variation operators produce only one offspring from one pair of parents, then λ pairs of parents are required and $\kappa = 2\lambda$. The hexagonal shape related to “variation operators” shown in Figure 1.1 may represent a chain of several operators that are subsequently applied, such as mutation and crossover operators, typically;

– the fitnesses of the λ offspring are then evaluated;

– afterwards, the *environmental selection* or *selection for the replacement*, or simply the *replacement* operator decides which individuals will survive in the population in the next generation among the μ parents and λ offspring;

– when the current generation ends, a stopping test allows the algorithm to stop according to the criteria defined by the user. Otherwise, another generation begins with the new population built by the environmental selection operator.

Among the evolutionary algorithm variants, parental selection or else environmental selection may be independent of fitnesses, provided that there is a bias in favor of the fittest individuals from generation to generation.

1.3. Selection operators

1.3.1. *Selection pressure*

The individuals which have the best fitnesses are reproduced more often than the others and replace the worst ones. Sometimes, a population contains a non-optimal super-individual with a much higher fitness than the others.

Depending on the nature or the parameters of the selection operators, it could potentially reproduce much more quickly than the others. Its copies could then invade the population before the variation operators find better solutions. The exploration of the search space becomes local, since it is limited to a random local search centered on the super-individual. In this way, there will be a high risk that the algorithm cannot find a global optimum because it remains trapped in any local optimum.

In another case, according to the nature of the parameters of the selection operators, low fitness individuals and high fitness individuals could have almost the same average number of offspring. In this case, the convergence could be strongly slowed down unnecessarily.

In the first case mentioned above, the super-individual creates a selection pressure that is too high, while in the second case, the selection pressure is too low.

If the variation operators are disabled, the best individual should reproduce more quickly than the others, until its copies completely take over the population. This observation leads to the first evaluation of the selection pressure for evolutionary algorithms, suggested by Goldberg and Deb in 1991 [GOL 91]. The *takeover time* τ^* is defined as the number of generations required to fill the population with copies of the best individual under the action of the selection operators only: the lower the value of τ^*, the higher the selection pressure.

The *selection intensity* S is another concept, borrowed from the "population genetics" theory [HAR 06], to evaluate the selection pressure. Let $\bar{f}$ be the average fitness of the population before the selection. Let $\bar{g}$ be the average fitness of the selected individuals. Then, S measures the increase in the average fitness of the individuals of a population determined before and after selection with the standard deviation σ_f of the individual fitnesses before selection taken as a unit of measure:

$$S = \frac{\bar{g} - \bar{f}}{\sigma_f}$$

If the selection intensity is computed for the reproduction, then $\bar{f} = \sum_{i=1}^{\mu} f_i/\mu$, where f_i is the fitness of an individual i, and $\bar{g} = \sum_{i=1}^{\kappa} g_i/\kappa$, where g_i is the fitness of the selected individual i and κ is the number of selected individuals. The definitions presented above are general and are applicable to any selection technique. It is possible to present other definitions, whose validity is possibly limited to certain techniques, as we will see later with regard to the *proportional selection*.

1.3.2. *Genetic drift*

Genetic drift is also a concept which derives from the *population genetics* theory [HAR 06]. It is related to random fluctuations of the frequency of alleles in a population of small size, where an *allele* is a variant of an element of a sequence of DNA having a specific function. For this reason, hereditary features can disappear or be fixed at random in a small population even without any selection pressure.

This phenomenon also occurs within the framework of the evolutionary algorithms if their selection operators are stochastic. At the limit, even for a population composed of different individuals subject to neutral selection[2], in the absence of the variations generated by mutation and crossover operators, the population converges towards a state where all the individuals are identical. Figure 1.2 illustrates the effect of genetic drift in a population of 100 individuals in a two-dimensional search space Ω, according to the generation number. Individuals are represented as points, possibly "stacked" as vertical lines when several individuals are at the same location in Ω. Individuals are subject to neutral selection, without mutation or crossover. The figure shows that at the 100th generation, all the individuals are identical.

The genetic drift can be estimated from the time required to obtain a homogeneous population using a Markovian analysis. But these results are approximations and are difficult to generalize outside the cases studied in the literature [ROG 99]. However, it is verified that the time of convergence towards an absorbing state becomes longer as the population size increases.

Another approach assesses the genetic drift rate caused by a selection operator [ROG 99]. Let r be the ratio of $E(V_f')$: the expectation of the fitness variance of the population after selection, to V_f: the fitness variance before

2 Neutral selection: the number of offspring of an individual is independent of its fitness.

selection. In the case of neutral selection, A. Rogers and A. Prügel-Bennett showed that r depends only on variance V_s of the number of offspring for each individual and on population size P, assumed constant:

$$r = \frac{E(V_f')}{V_f} = 1 - \frac{V_s}{P-1}$$

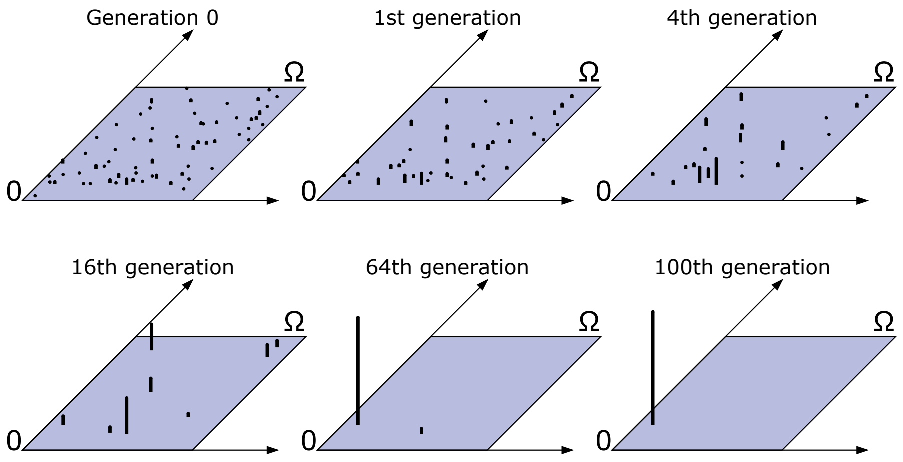

Figure 1.2. *Effect of genetic drift according to the generation number, for a population of 100 individuals in a two-dimensional search space* Ω

V_s is a characteristic of the selection operator. The expression clearly shows that increasing the population size or reducing the variance V_s of the selection operator decreases r, i.e. the genetic drift. Figure 1.3 shows the effect of genetic drift with the same conditions as mentioned previously (Figure 1.2), but for a population of 1,000 individuals. After 100 generations, there is still diversity in the population of 1,000 individuals, whereas the population of size 100 only contains identical individuals.

The effect of genetic drift is predominant when the selection pressure is low. This situation leads to a loss of diversity, which may involve a premature convergence *a priori* far away from the optimum, since it does not depend on the fitness of the individuals.

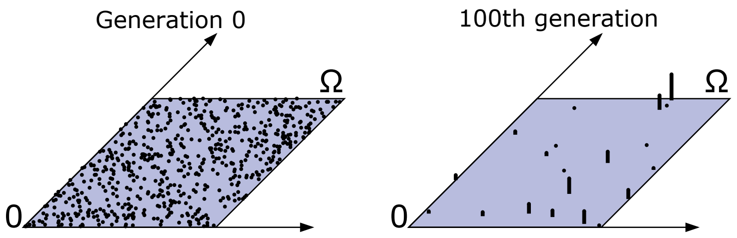

Figure 1.3. *Effect of genetic drift according to the generation number for a population of 1,000 individuals in a two-dimensional search space* Ω

In short, in order for an evolutionary algorithm to work efficiently, it is necessary that the selection pressure is neither too strong nor too weak. To fight the genetic drift, the population size must be large enough, unless the selection operator is characterized by a low variance.

1.3.3. *Proportional selection*

Proportional selection was originally proposed by J. Holland for genetic algorithms. It is used only for the reproduction. The expected number of selections n_i of an individual i is proportional to its fitness f_i. This implies that the fitness function is positive in the search domain and that it has to be maximized. Let μ be the population size and let κ be the total number of individuals generated by the selection operator for one generation, n_i can be expressed as:

$$n_i = \frac{\kappa}{\sum_{j=1}^{\mu} f_j} f_i$$

However, the effective number of offspring can be only integers. They can be obtained by successive random draws of offspring according to distributions chosen such that the expected number of offspring for parent i is equal to n_i. Two techniques are widespread and are described below: the *roulette wheel selection method, RWS* because it is the operator originally proposed for the genetic algorithms, but it suffers from high variance, and the *stochastic universal sampling method, SUS* because it guarantees a minimal variance of the sampling process [BAK 87].

1.3.3.1. *Roulette Wheel Selection (RWS)*

The RWS method exploits the metaphor of a biased roulette game, for which the wheel comprises as many pockets as individuals in the population. Each individual is associated to a pocket with a size proportional to its fitness. Once the game is started, the selection of an individual is indicated by the fall of the ball into its pocket. For instance, Figure 1.4 depicts a roulette wheel where the ball has fallen into pocket 7. So, individual 7 with fitness 156 is selected for the reproduction. The ball is spun κ times to select κ individuals.

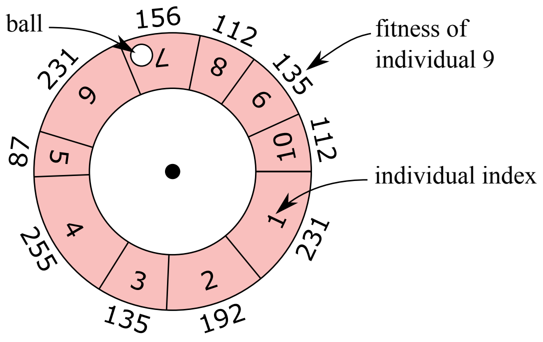

Figure 1.4. *Biased roulette wheel metaphor for the RWS method: individual 7 with fitness 156 is selected after drawing a random number represented as the ball position on a roulette wheel*

The RWS algorithm selects individual i such that:

$$\sum_{j=0}^{i-1} f_j \leq \mathcal{U}[0,1] \sum_{j=1}^{\mu} f_j < \sum_{j=1}^{i} f_j \quad \text{with} \quad f_0 = 0$$

where $\mathcal{U}[0,1]$ is a random number uniformly drawn from [0, 1]. f_j is the fitness of individual j.

1.3.3.1.1. Genetic drift

The effective number of selections of individual i follows a binomial distribution $\mathcal{B}(\kappa, p_i)$ with $p_i = f_i / \sum_{j=1}^{\mu} f_j$. Its variance $V_{\text{RWS}}(i)$ is $\kappa p_i(1-p_i) = n_i(1-p_i)$, where n_i is the expected number of individual i selections. For a given selection pressure (section 1.3.3.3), and if μ is large enough, $V_{\text{RWS}}(i) \approx n_i$. The variance of this process is high: it is possible that an individual with a good fitness value is never selected. By misfortune, it is also possible that bad quality individuals are selected for all the offspring. Thus, the high variance creates a high genetic drift level, allowing some poor individuals to have offspring to the detriment of better individuals. To reduce this risk, the population size must be large enough.

1.3.3.1.2. Algorithmic complexity

The selection of an individual requires $\mathcal{O}\log(n)$ comparisons by using a dichotomy algorithm, where $n = \mu + \kappa$. In this way, there are $\mathcal{O}(n\log(n))$ comparisons to select κ individuals.

1.3.3.2. *Stochastic Universal Sampling selection (SUS)*

The SUS method [BAK 87] still uses the metaphor of the wheel of a biased roulette game. However, the ball is spun only once to obtain the set of the κ selected individuals. When the ball falls in a pocket, it defines a random offset position. Offspring are then determined by a set of κ equidistant pointers around the wheel such that one of these pointers is set at the offset position, as shown in Figure 1.5. According to the figure, individuals 5 and 9 are not selected, and the others are selected once.

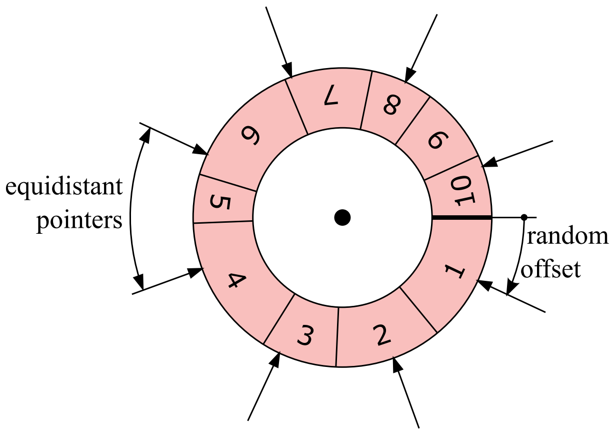

Figure 1.5. *SUS method: the selected individuals are determined by $\kappa = 8$ equidistant pointers. Thus, individuals 5 and 9 are not selected, and the others are selected once. The fitnesses corresponding to the individuals are given in Figure 1.4*

Let δ be the distance between the pointers: $\delta = \sum_{j=1}^{\mu} f_i/\kappa$. Let ω be the random offset: $\omega = \mathcal{U}[0, \delta]$. The SUS operator selects κ individuals i such that:

$$\forall k \in \{0, ..., \kappa - 1\}, \quad \sum_{j=0}^{i-1} f_j \leq \omega + k\delta < \sum_{j=1}^{i} f_j \quad \text{with} \quad f_0 = 0$$

1.3.3.2.1. Genetic drift

For an expected number of selections n_i of individual i, the effective number of selections is either its lower integer part $\lfloor n_i \rfloor$, or its upper integer part $\lceil n_i \rceil$ according to a Bernoulli distribution of parameter $q_i = n_i - \lfloor n_i \rfloor$, with variance $V_{\text{SUS}}(i) = q_i(1 - q_i)$. The maximum of V_{SUS} is 1/4 for $q_i = 1/2$. From expressions of $V_{\text{RWS}}(i)$ and $V_{\text{SUS}}(i)$, it can easily be shown that $V_{\text{SUS}}(i) < V_{\text{RWS}}(i)$ if $\kappa > 1$.

If $\kappa \geq \mu$, the best individual is certain to have at least one offspring. This is the case in Figure 1.6 where $\mu = \kappa = \lambda = 100$. When the variation operators are disabled, the population is kept unchanged indefinitely under SUS selection for a constant fitness function (neutral selection). In contrast, Figure 1.2 (p. 8) shows what happens during an evolution by replacing SUS selection with the RWS operator: its high variance results in a complete loss of diversity in the population after 100 generations.

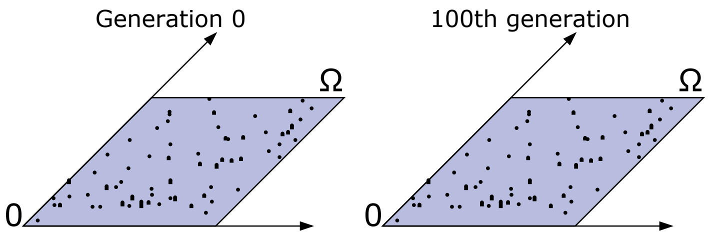

Figure 1.6. *Genetic drift does not occur with the SUS selection operator for a population of $\mu = \kappa = 100$ individuals in a two-dimensional search space Ω*

1.3.3.2.2. Algorithmic complexity

The number of comparisons to select the individuals is of the order of $\mathcal{O}(n)$, where $n = \mu + \kappa$.

1.3.3.3. *Proportional selection and selection pressure*

In the case of proportional selection, the expected number of selections of the best individual with fitness $\hat{f}$ among μ selections for a population of μ parents is appropriate to define selection pressure p_s:

$$p_s = \frac{\mu}{\sum_{j=1}^{\mu} f_j}\hat{f} = \frac{\hat{f}}{\bar{f}}$$

where $\bar{f}$ is the average of the fitnesses of the population. If $p_s = 1$, then all the individuals have equal chances to be selected, indicating an absence of selection pressure.

Let us consider the search for the maximum of a continuous function, e.g. $f(x) = \exp(-x^2)$. The individuals of the initial population are assumed to be uniformly distributed in the domain $[-2, +2]$. Some of them have a value close to 0, which is also the position of the optimum, and thus their fitness $\hat{f}$ will be close to 1. The average fitness of the population $\bar{f}$ will be:

$$\bar{f} \approx \int_{-\infty}^{+\infty} f(x)p(x)dx$$

where $p(x)$ is the presence probability density of an individual at x. A uniform density is chosen in the interval $[-2, +2]$, thus $p(x) = 1/4$ in this interval, and is 0 elsewhere. Thus,

$$\bar{f} \approx \frac{1}{4}\int_{-2}^{+2} e^{-x^2} dx$$

that is $\bar{f} \approx 0.441$, which gives a selection pressure of the order of $p_s = \hat{f}/\bar{f} \approx 2.27$. The best individual will thus have an expected number of offspring close to two at generation 0.

Figure 1.7 shows the repartition of a population of $\mu = \lambda = \kappa = 100$ individuals according to the number of generations to find the maximum of $f(x) = \exp(-x^2)$ in interval $[-2, 2]$. Individuals are represented as dots on the curve of $f(x)$. The figure also indicates the value of the fitness average for each graph by using a dashed line and the number of offspring n_{best} for the best individual taken as a measure of the selection pressure. Individuals are subject to a proportional selection (RWS) and uniform mutations in interval [–0.05, 0.05], i.e. $x' = x + \mathcal{U}[-0.05, 0.05]$, where x' is the offspring obtained from the mutation of x and $\mathcal{U}[-0.05, 0.05]$ is a uniform random number in $[-0.05, 0.05]$. The environmental selection replaces all the individuals in the population at the next generation with their offspring (generational replacement).

Figure 1.7 shows that when individuals concentrate towards the optimum over generations due to the selection pressure, the fitness average increases, approaching the maximum value and consequently, the selection pressure decreases, tending to one. This means that:

– every individual has an expected number of offspring close to one, even the best one;

– the stochastic fluctuations of the selection operator become predominant to determine the offspring number (genetic drift);

– the evolution is no longer guided by the fitness and therefore the algorithm fails to find the optimum with a high precision.

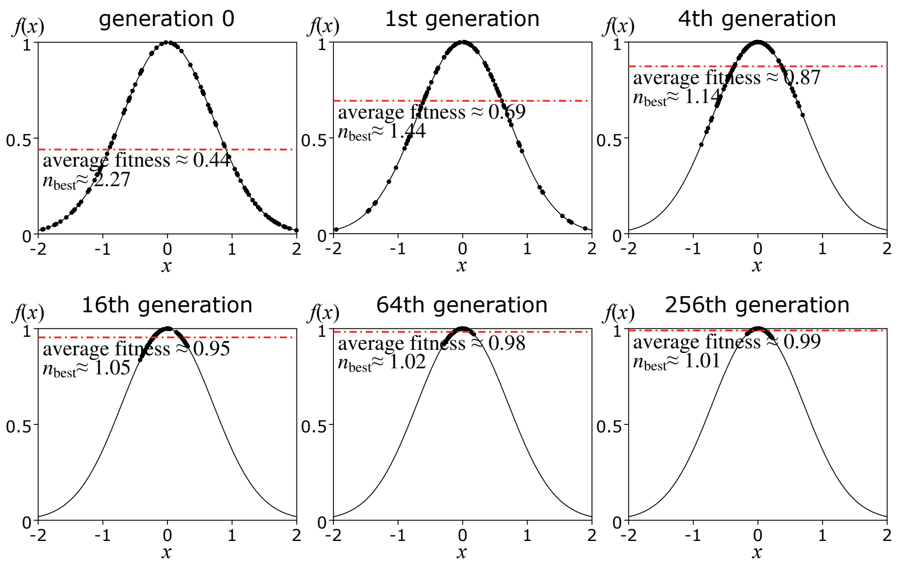

Figure 1.7. *Selection pressure n_{best} decreases when the population concentrates in the neighborhood of the optimum*

This undesirable behavior of proportional selection, where the selection pressure strongly decreases when the population approaches the optimum in the case of continuous functions, can be overcome by scaling the fitness function [GOL 89]. It is interesting to note the Boltzmann selection (De La Maza and Tidor 1993 [DE 93b]), because it makes a link with simulated annealing. The method uses a scaled fitness expressed as:

$$f_i' = \exp(f_i/T)$$

The value of parameter T, known as the “temperature”, determines the selection pressure. T is usually a decreasing function of the number of generations. Thus, the selection pressure increases with time.

1.3.3.4. *Rank-based selection*

Another way to control the selection pressure combines a proportional selection and a ranking of the individuals of the population. Individuals x are ranked from the best one (first) to the worst one (last), according to raw fitnesses $f(x)$. The actual fitness value f_r of each individual depends only on its rank r by decreasing value (see Figure 1.8) according to, for instance, the expression given below, which is usual:

$$f_r = \left(1 - \frac{r}{\mu}\right)^p$$

where μ is the number of parents, r is the rank of the individual considered in the population of the parents after ranking and p is an exponent which depends on the desired selection pressure. After ranking, a proportional selection is applied according to f_r. With our definition of the selection pressure $p_s = n_{\text{best}} = 1 + p$. Thus, p must be greater than 0 to obtain a selection pressure greater than 1. This ranking-based selection is interesting because:

– it is not affected by a constraint of sign: $f(x)$ can either be positive or negative;

– it is appropriate for a maximization problem as well as for a minimization problem, without any extra transformation;

– it does not consider the importance of the differences between the fitnesses of the individuals. Thus, a ranking-based selection method does not require exact knowledge of the objective function. It only needs to rank the individuals by comparing each one with the others.

These good properties mean that ranking-based selections are often preferred by the users of evolutionary algorithms when compared to fitness scaling methods. Linear ranking, for which $p = 1$, is quite a good choice by default.

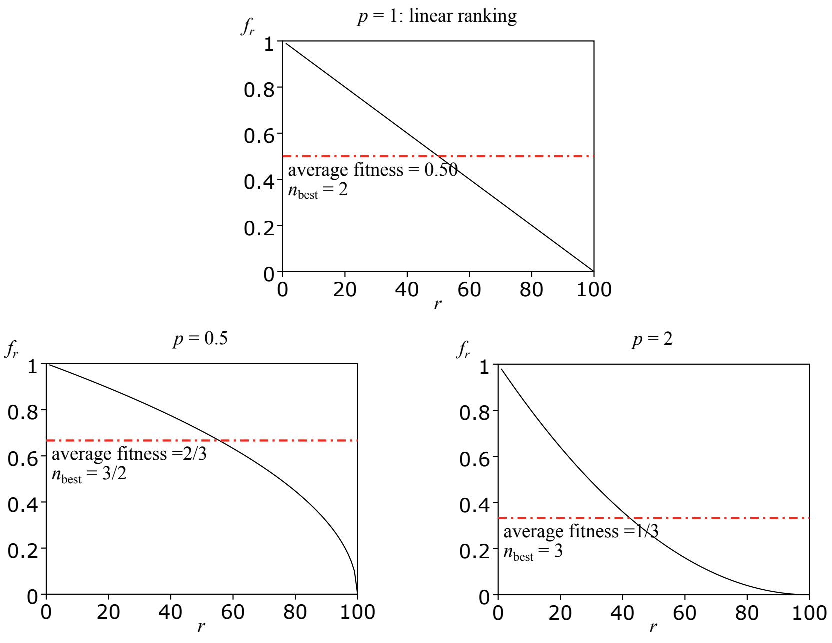

Figure 1.8. *Fitness $f_r = (1 - r/\mu)^p$ obtained after ranking, where r is the rank of an individual. μ is chosen equal to 100. The selection pressure is $n_{\text{best}} = 1 + p$*

1.3.4. *Tournament selection*

Tournament selection is an alternative to the proportional selection techniques which, as seen before, present difficulties in controlling the selection pressure during the evolution, while being relatively expensive in computational costs.

1.3.4.1. *Deterministic tournament*

The simplest tournament consists of choosing at random p individuals in the population and selecting the one that has the best fitness for reproduction.

During a selection step, there are as many tournaments as selected individuals. The individuals that take part in a tournament are replaced in the population, or withdrawn from it, according to the choice of the user. A drawing without replacement makes it possible to carry out $\lfloor N/p \rfloor$ tournaments for a population of N individuals. A copy of the population is re-generated when it is exhausted, as many times as required, until the desired number of selections is reached. The variance of the tournament process is high, which favors genetic drift. It is however lower in the case of drawing without replacement. This method of selection is very much used, because it is much simpler to implement than a proportional reproduction with properties similar to the ranking selection.

The selection pressure is adjusted by the number of contestants p in a tournament. Indeed, let us consider the case where the contestants in a tournament are replaced in the population. Then, the probability that the best individual of the population is not selected with p random drawings is $((N-1)/N)^p$. Assuming that N is large compared to p, this probability is approximately $1 - p/N$ by a Taylor expansion limited to the first order. Thus, the probability that the best individual is drawn at least once in a tournament is close to p/N. If there are κ tournaments in a generation, the best individual will have $n_{best} = p\kappa/N$ expected selections. Let us consider again the definition of selection pressure that was proposed for the proportional reproduction, with $\kappa = N$. Then, the selection pressure is equal to p, which is greater than or equal to 2.

1.3.4.2. *Stochastic binary tournament*

With the stochastic binary tournament, the best of the two contestants wins with a probability p, which is a parameter whose value is chosen between 0.5 and 1. It is still easy to calculate the selection pressure generated by this process. The best individual takes part in a tournament with a probability of $2/N$ (see the previous section 1.3.4.1). Furthermore, the winner of the tournament will be selected with a probability p. The two events being independent, the probability that the best individual of the population is selected after a tournament is then $2p/N$. If there are N tournaments, the best parent will thus have $2p$ expected offspring. The selection pressure thus will range between 1 and 2.

1.3.5. *Truncation selection*

This selection is very simple to implement, since it only chooses the n best individuals from a population, where n is a parameter chosen by the user. If the operator is used for the reproduction, $n = \kappa$ to select κ parents. If the operator is used for the replacement and thus generates the population of μ individuals for the next generation, then $n = \mu$.

1.3.6. *Environmental selection*

The *environmental selection* or *replacement selection* method determines which individuals in generation g, among offspring and parents, will constitute the population in generation $g + 1$.

1.3.6.1. *Generational replacement*

This kind of replacement is the simplest, since the population of the parents in generation $g + 1$ will be composed of all the offspring, and only them, generated at generation g, thus: $\mu = \lambda$ to keep the population size constant. The canonical genetic algorithm is proposed with a generational replacement.

1.3.6.2. *Replacement "(μ, λ) – ES"*

A truncation selection of the best μ individuals among the λ offspring constitutes the population for the next generation. This operator was introduced for the evolution strategies [REC 65, BEY 01]. In this case, λ is larger than μ.

1.3.6.3. *Replacement "$(\mu + \lambda)$ – ES"*

A truncation selection of the best μ individuals from the union of the set of the μ parents and the set of the λ offspring constitutes the population for the next generation. This operator was introduced for the evolution strategies [REC 65, BEY 01]. It is said to be elitist since it preserves the best individual found so far in the population.

1.3.6.4. *One-to-one selection*

This operator is a deterministic binary tournament typically between each parent of a population and its only offspring. The best of them is kept in the population for the next generation. Thus, the variance of the selection number for each parent is 0. In this context, the parental selection is disabled because each parent has only one offspring, whatever its fitness is. This very simple operator is especially used for the powerful *Differential Evolution* and *Particle Swarm Optimization* algorithms, described in Chapter 2.

1.3.6.5. *Steady state replacement*

In each generation, a few (often one or two) offspring are generated. They replace a lower or equal number of parents to produce the population at the next generation. This strategy is particularly useful when the representation of a solution is distributed on several individuals, possibly the entire population, as for *Learning Classifier Systems* [BOO 89] [URB 09]. In this way, the loss of a small number of individuals in each generation: those that are replaced by the offspring, does not excessively disturb the solutions, which evolve gradually.

The replaced parents can be chosen in various ways. With uniform replacement, the replaced parents are selected at random. The choice can also depend on the fitness: the worst parent is replaced, or else it is selected stochastically, according to a probability distribution that depends on the fitness or other criteria.

Steady state replacement generates a population where the individuals are subject to large variations of lifespan in terms of generation numbers and therefore offspring numbers. The high variance of these values involves high genetic drift, especially as the population is small [DE 93a].

1.3.6.6. *Elitism*

An elitist strategy consists of at least keeping the individual with the best fitness in the population, from generation to generation. The fitness of the best individual from the current population is thus monotonically non-decreasing from one generation to the next.

There are various elitist strategies such as the "$(\mu + \lambda)$ – ES" mentioned above. In another current alternative, the k best parents in the current generation are kept in the population for the next generation. The replacement operator then has to replace the $N - k$ remaining individuals, where N is the population size.

These strategies may considerably speed up the performance of evolutionary algorithms for some classes of fitness functions, such as convex functions. But, they may be disappointing for other classes, such as multimodal functions, because elitism increases the selection pressure around the best individuals of the population, even when they are trapped in local optima. However, this is an algorithm design issue. Thus, Chapter 2 presents elitist elaborated algorithms able to quite efficiently find near-optimal solutions for massively multimodal functions. Figure 1.9 shows a graphical representation in a 2D search space of such a multimodal function, which counts 15^d local minima in domain $[-40, 40]^d$, where dimension d is an integer usually taken between 2 and 100. Its analytic expression is given in section 2.10 (p. 89).

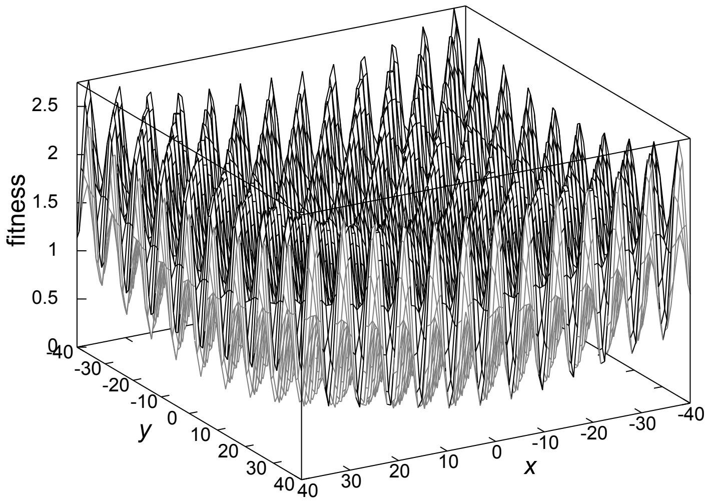

Figure 1.9. *The Griewank's multimodal function in domain* $[-40, 40]^2$

Choosing a non-elitist strategy can be advantageous, but there is no guarantee that the fitness function of the best individual in the current population is increasing during the evolution. This obviously implies keeping a copy of the best solution found so far by the algorithm, without this copy taking part in the evolutionary process. It should be noted, this is a requisite precaution for any stochastic optimization algorithm.

1.3.7. *Selection operators: conclusion*

Table 1.1 summarizes the properties of basic selection operators used by evolutionary algorithms in terms of selection pressure, variance of the selected individual number and algorithmic complexity. Among these operators, the best one is “one-to-one selection”, but it is rather used by specific evolutionary algorithms (Chapter 2). The tournament selection is widespread because of its ease of implementation, despite a high variance. The population ranking associated with a selection process such as truncation selection or SUS is also often used, although its algorithm complexity is the worst.

	Selection pressure	Variance	Complexity
RWS	Non-controlled	High	$\mathcal{O}(n \log n)$
SUS	Non-controlled	Low	$\mathcal{O}(n)$
Tournament selection	Controlled	High	$\mathcal{O}(n)$
Ranking + SUS	Controlled	Low	$\mathcal{O}(n \log n)$
One-to-one selection	Controlled	0	$\mathcal{O}(n)$

Table 1.1. *Comparisons of basic selection operators according to their control of the selection pressure, variances and algorithmic complexities. n is the population size*

1.4. Variation operators and representation

1.4.1. *Generalities about the variation operators*

The variation operators are generally stochastic operators that transform and combine copies of one or several individuals in a population to create offspring, possibly fitter than their parents, which partly inherit the features of them. These operators are classified into two categories:

– the *mutation* operators, which alter an individual independently of the others;

– the *crossover* or *recombination* operators, which generate one or more offspring from the combinations of several parents, often two of them but also possibly the combination of the whole parent population.

The way of modifying an individual depends strongly on the structure of the solution that it represents. Thus, if we aim to solve an optimization problem in a continuous space, e.g. a domain of $\mathbb{R}^n$, *a priori*, it will be appropriate to choose a vector of $\mathbb{R}^n$ to represent a solution. In this case, the crossover operator combines at least two “parent” vectors of $\mathbb{R}^n$ to create one or several “offspring” vectors in the same space. On the other hand, if an evolutionary algorithm is used to solve instances of the *Traveling Salesman Problem*, an individual could represent a Hamiltonian path as a list of vertices. The variation operators should then generate only new Hamiltonian paths. These examples show that it is impossible to design universal variation operators, independently of the problem under consideration. They are inevitably related to the *representation* of the solutions in the search space.

Variation operators have to:

– *explore* the search space, in order to discover its promising areas, which are more likely to contain the global optima;

– *exploit* these promising areas, by focusing the search there to find the optima with the required accuracy.

For instance, a variation operator that draws offspring at random uniformly in the search space will have excellent qualities of exploration. But it will likely fail to discover an optimum in a reasonable time with the required accuracy, because there is no exploitation of the promising areas.

The search should be more efficient if the offspring are generated preferentially close to their parents, according to an appropriate distance, depending on the problem to solve. Such a policy is reasonable to the extent that parents are located in promising areas since they are selected according to their fitnesses. But if all the offspring generated by a variation operator are close to their parents, the search algorithm could be trapped in a local optimum: in this case, there is not enough exploration compared to exploitation.

A good balance should be found between exploitation and exploration abilities of variation operators. This requirement is not easy to ensure.

1.4.2. *Crossover*

The crossover operator uses at least two parents to generate one or more offspring. The *crossover rate*, which is *a priori* a parameter of the evolutionary algorithm, determines the proportion of the crossed individuals in the population. The operator is generally stochastic, and thus the repeated crossover with the same couple of distinct parents yields different offspring. It often respects the following properties:

– the crossover of two identical parents will generate offspring, identical to the parents;

– by extension, on the basis of an index of proximity depending on the chosen representation, defined in the search space, two parents which are close in the search space will generate offspring close to them.

These properties are satisfied by the "classical" crossover operators like most of those described in this book, but they are not absolute. In the current state of knowledge of evolutionary algorithms, the design of a crossover operator does not follow precise rules.

In the simplest version of an evolutionary algorithm, the selected individuals are mated at random. As a result, some selected parents could be located on several peaks of the fitness function, as depicted in Figure 1.10. The figure shows that the probability distribution of offspring is uniform between parent solutions x_1 and x_2. This widespread crossover is referred to as the flat crossover [RAD 90]. In this case, the conditional probability distribution of offspring given by the parents is inappropriate because offspring are likely to have poor fitness values. The crossover is said to be *lethal* if, from high fitness parents, it generates offspring with too low fitness to survive.

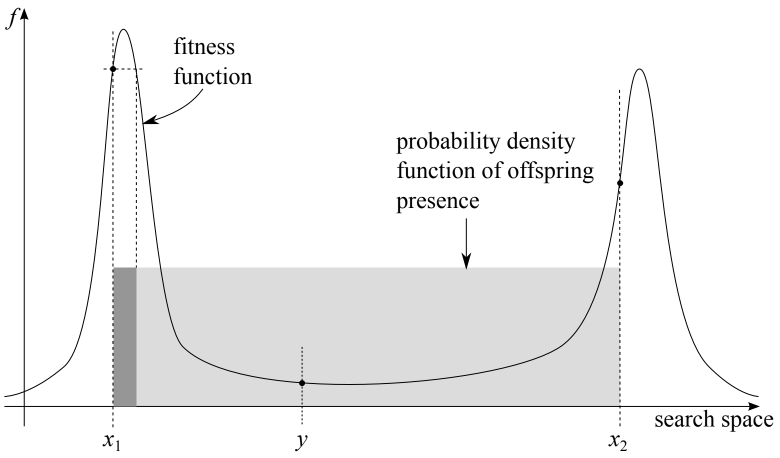

Figure 1.10. *Let x_1 and x_2 be parents placed on different peaks of fitness function f. Assuming that the crossover x_1 and x_2 yields an offspring y uniformly drawn from interval $[x_1, x_2]$, y is likely to have a poor fitness value. The probability that offspring are better than their parents is represented by the dark gray region*

If the fitness function is continuous, the following approaches can be used to avoid a too high proportion of lethal crossovers:

1) by restricting the mating to similar parents;

2) by giving more chance to yield offspring close to their parents, as with the Simulated Binary Crossover [DEB 95].

The first option is inspired by the natural genetics metaphor: individuals of different species cannot be crossed to yield viable offspring if they are too dissimilar. In the frame of evolutionary algorithms, the simplest way consists of crossing individuals if distance between them is less than a threshold r_c called the *restriction radius* [GOL 89]. However, a too small value for r_c could significantly lower the effective crossover rate as well as the exploration in the search space, which could slow down or even freeze the search of the optimum. r_c is difficult to estimate because it depends on distances between peaks, which are not known in general. It is possible to consider a decreasing radius r_c during the evolution to overcome this problem.

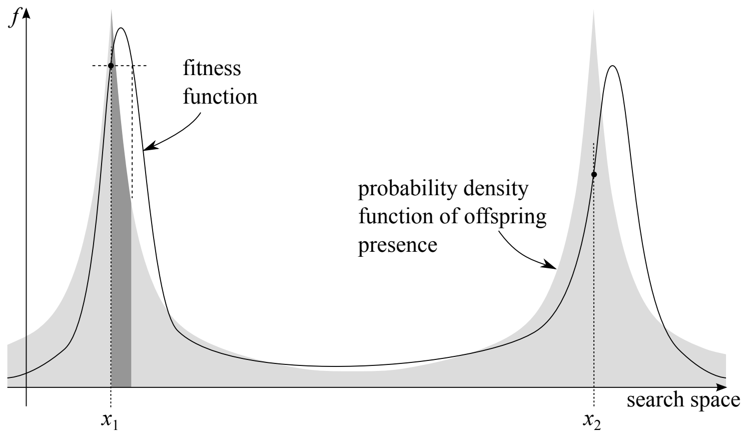

Figure 1.11. *Compared to the case of Figure 1.10, for the same fitness function, a multimodal probability density function of the offspring presence after crossover, for which the modes are parents x_1 and x_2, increases the probability that offspring are better than their parents. This probability is represented by the dark gray region*

Another option to avoid a high proportion of lethal crossover consists of using a multimodal probability distribution of offspring, such that its modes are the parents. An example is given in Figure 1.11. The areas of the dark gray regions in Figures 1.11 and 1.10 represent the probabilities that offspring have better fitness values than those of their parents. By comparing these figures, it appears that this probability is higher with the multimodal distribution.

1.4.3. *Mutation*

Most of the mutation operators alter an individual in such a way that the result of the transformation is often close to it, but not always, to preserve the exploration ability of the operator. They mainly perform a random local search around each individual to mutate. Thus, the mutation gives each individual a chance to approach the exact location of the maximum of a fitness function, as much as the characteristics or the parameters of the chosen operator allow it.

The proportion of the mutated individuals in the offspring population is equal to the *mutation rate*. In the frame of *genetic algorithms* (section 1.6), the mutation is considered as a minor operator, useful to preserve diversity in the population, which the crossover cannot ensure. The mutation rate is then typically low, about 0.01 to 0.1, whereas the crossover rate is high. Conversely, a 100% mutation rate was required for the first *Evolution Strategy* algorithms since they did not use crossover. A large enough mutation rate helps to preserve diversity in the population, which is useful for a good exploration of the search space. Thus, this operator can contribute to fight the reduction of the population variance due to a strong selection pressure or genetic drift.

Using mutation as a local search operator suggests to combine it with other more efficient, although more problem-dependent, local techniques such as a gradient method for example. This approach leads to the design of *hybrid* evolutionary algorithms.

1.5. Binary representation

The idea of evolving a population of binary vectors mainly comes from genetic algorithms, which implements a simple model of the transcription *genotype–phenotype* existing in the living world. Within the framework of genetic algorithms, the genotype of an individual is comprised of a string of binary symbols, or more generally, a string of symbols belonging to a low-cardinality alphabet. The phenotypes are solutions to a problem expressed in a “natural” representation, *a priori* easy to understand by people who want to solve it. They are used by the algorithm only for the fitness evaluation of an individual. Conversely, the binary genotypes are hard to interpret by a human expert. They undergo the action of the genetic operators: mutations and crossover.

For example, if a solution to a given problem is expressed naturally as a vector of real numbers, the phenotype could be this vector. Thus, the genotype is a binary string obtained from this vector. The simplest way consists of converting each of its components with a number of bits corresponding to the required precision. Then, these binary numbers can be concatenated to generate the genotype.

1.5.1. *Crossover*

For a binary representation, there exist three widespread variants of crossovers:

– the “single point” crossover;

– the “two point” crossover;

– the uniform crossover.

A pair of individuals being chosen randomly among the selected individuals, the “single point” crossover [HOL 92] is applied in two steps:

1) random choice of an identical cut point on the two bit strings (see Figure 1.12(a));

2) cut of the two strings (Figure 1.12(b)) and exchange of the two fragments located on the right (Figure 1.12(c)).

This operator generates two offspring from two parents. If only one offspring is needed by the evolutionary algorithm, it is chosen at random in the pair and the other one is discarded. The “single point” crossover is the simplest one for codings with a low cardinality alphabet, like binary coding.

A “two point” crossover can be implemented by randomly choosing two cut points c_1 and c_2, with $c_1 < c_2$, in parent individuals as shown in Figure 1.13. Symbols whose indices are in interval $[c_1, c_2]$ are exchanged between the two strings to obtain an offspring.

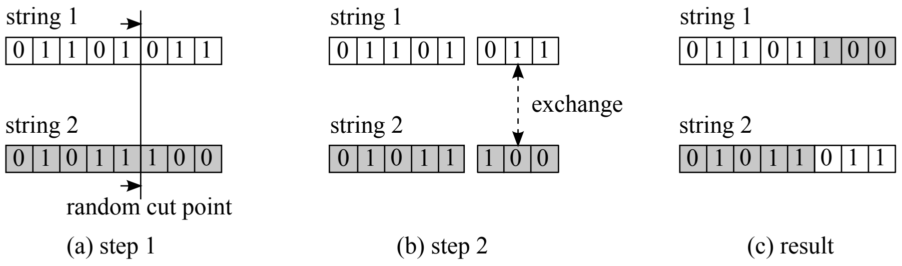

Figure 1.12. *“Single point” crossover of two 8 bit strings*

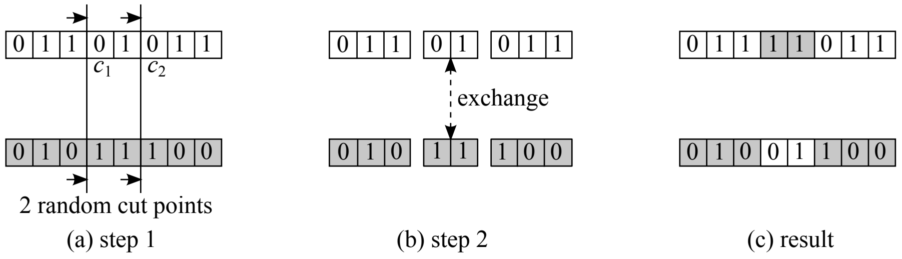

Figure 1.13. *“Two point” crossover of two 8 bit strings*

The “single point” and “two point” crossovers are usually employed in practice for their simplicity and efficiency. An immediate generalization of these operators consists of multiplying the cut points on each string. The uniform crossover [ACK 87] can be viewed as a multipoint crossover where the number of cuts is *a priori* unspecified. Practically, a “template string” is used. It is a binary string of the same length as the individuals. A “0” at the n^{ith} position of the template leaves the symbols in the n^{ith} position of the two strings unchanged. A “1” activates an exchange of the corresponding symbols (in Figure 1.14). The template is generated at random for each pair of individuals. The values “0” or “1” of the template elements are generally drawn with a probability of 0.5. Lower probabilities generate offspring closer to the parents in the sense of the Hamming distance[3].

3 Hamming distance: number of different symbols between two strings of the same length.

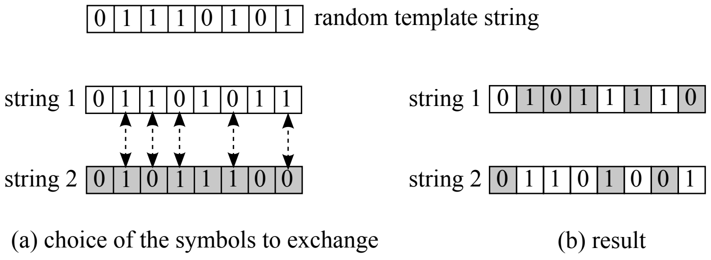

Figure 1.14. *Uniform crossover*

1.5.2. *Mutation*

The mutation operator on bit strings changes some of its symbols at random. The most common variants are the *deterministic mutation* and the *bit-flip mutation*. With the "deterministic" mutation, a fixed number of bits chosen at random are reversed for each mutated individual, i.e. a "1" becomes "0" and vice versa. With the "bit-flip" mutation, each bit can be reversed independently of the others according to a given probability. These operators have good exploitation abilities if the number of reversed bits is small enough in accordance with the Hamming distance (see section 1.4.1). However, this number should also be large enough to preserve the diversity in the population. This parameter is problem-dependent.

When a bit string represents a vector of integer or real numbers, the search for the optimum of the fitness function might be countered by the difficulty of crossing the *Hamming cliffs*, due to the conversion of the bit strings towards real number vectors. For example, let us consider function $C(x)$ defined below:

$$C(x) = \begin{cases} x & \text{if } x \leq 16 \\ 0 & \text{otherwise} \end{cases}$$

Let $b(x) = \{b_1(x), \ldots, b_5(x)\}$ be a string of five bits to represent an integer individual x that ranges from 0 to 31. $b(x)$ can be simply defined as the standard representation of x in base 2. The maximum of $C(x)$ is obtained for $x = 16$, which thus corresponds to $b(0) = \{1, 0, 0, 0, 0\}$. The value $x = 15$, obtained from the string $\{0, 1, 1, 1, 1\}$, yields the highest fitness apart from the maximum: this value will thus be favored by the selection operators.

However, there is no common bit between $\{1, 0, 0, 0, 0\}$ and $\{0, 1, 1, 1, 1\}$. This means that there is no other individual with which $\{0, 1, 1, 1, 1\}$ can be crossed to give $\{1, 0, 0, 0, 0\}$. As for the mutation operator, it will have to change all the bits of string $\{0, 1, 1, 1, 1\}$ simultaneously to obtain the optimum. The Hamming distance between the optimum and the individual which has the nearest fitness is maximal, equal to the length of the strings. This is a Hamming cliff. It is unlikely to jump over it with a "bit-flip" mutation, and it is impossible with the "deterministic" mutation, unless it flips all the bits of the string, which is never used.

But the mutation will be able to easily find the optimum if there are individuals in the population that differ in only one bit of the optimal string. Here, these individuals are:

String $b(x)$	x	$C(x)$
$\langle 0, 0, 0, 0, 0\rangle$	0	0
$\langle 1, 1, 0, 0, 0\rangle$	24	0
$\langle 1, 0, 1, 0, 0\rangle$	20	0
$\langle 1, 0, 0, 1, 0\rangle$	18	0
$\langle 1, 0, 0, 0, 1\rangle$	17	0

Unfortunately, all these individuals have zero fitness and thus they are not likely to "survive" from one generation to the next one.

This annoying phenomenon, which hinders the progress towards the optimum, can be eliminated by choosing a *Gray code*, which ensures that two successive integers will have binary representations that differ only in one bit. Starting from strings $b(x)$ that represent integer numbers in base 2, it is easy to obtain a Gray code $g(x) = \{g_1(x), \ldots, g_l(x)\}$ by performing, for each bit i, the operation:

$$g_i(x) = b_i(x) \oplus b_{i-1}(x)$$

where the operator $\oplus$ implements the "exclusive or" operation and $b_0(x) = 0$. Conversely, the string of l bits $b(x) = \{b_1(x), \ldots, b_l(x)\}$ can be obtained from the string $g(x) = \{g_1(x), \ldots, g_l(x)\}$:

$$b_i(x) = \bigoplus_{j=1}^{i} g_j(x)$$

The Gray codes of $\{0, 1, 1, 1, 1\}$ and $\{1, 0, 0, 0, 0\}$ are respectively $\{0, 1, 0, 0, 0\}$ and $\{1, 1, 0, 0, 0\}$. The mutation of bit g_1 is therefore enough to reach the optimum. A Gray code is thus desirable from this point of view.

However, the Hamming cliffs are generally not the cause of significant decrease in the performance of a search algorithm. More details about Gray codes and binary representations are developed in [ROW 04]. Especially, it is noted that the use of several Gray codes modifies the landscape of the fitness function and can help to escape from local optima.

1.6. The simple genetic algorithm

The simple genetic algorithm is an evolutionary algorithm, similar to the one presented in Figure 1.1 (p. 4), with a notable particularity: it implements a transcription *genotype–phenotype* inspired by natural genetics. The *phenotype* is the expression of a solution to a problem in its usual formalism. A *genotype* is a string of symbols (often binary) from which the associated phenotype is built. The phenotype can then be evaluated to give a fitness value that can be used by the selection operators.

The flowchart of a simple genetic algorithm is presented in Figure 1.15. It implements a proportional selection operator (see section 1.3.3, p. 9) and a generational replacement, i.e. the population of the offspring replaces that of the parents. Another classical variant uses the steady state replacement (section 1.3.6.5, p. 19).

The variation operators alter the genotypes. As they are bit strings, crossover and mutation operators for binary strings (section 1.5) are naturally used. The crossover is considered as the main variation operator. The mutation is usually applied with a small rate to maintain diversity in the population [GOL 89]. An appropriate coding of the genotype should be designed, such that the variation operators produce viable offspring, satisfying the constraints of the problem as much as possible.

Holland, Goldberg and many other authors have worked on a mathematical formalization of the genetic algorithms based on a "Schema Theorem" [GOL 89]. It provides arguments for the choice of a binary representation. However, deceiving results obtained with this theorem have given rise to controversies and debates about its utility [VOS 98]. In particular, the suitability of binary encoding has been challenged.

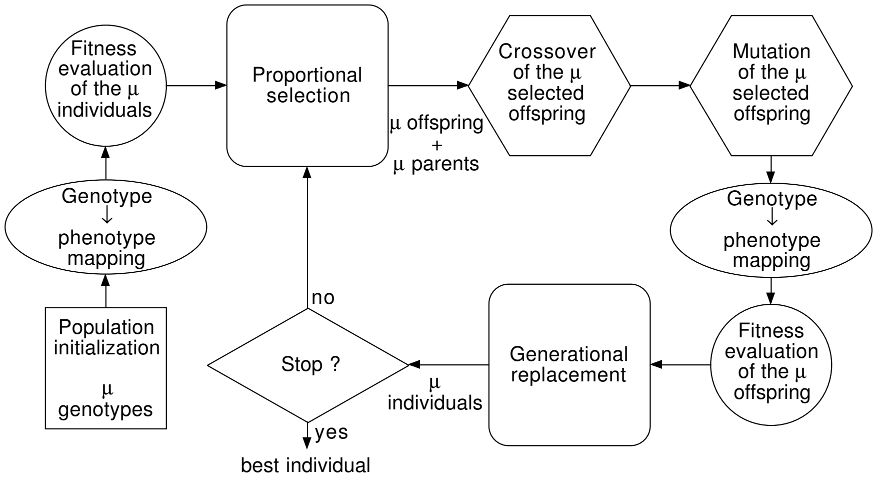
Fitness evaluation of the μ individuals
Proportional selection
μ offspring + μ parents
Crossover of the μ selected offspring
Mutation of the μ selected offspring
Genotype ↓ phenotype mapping
Population initialization μ genotypes
no
Stop ?
yes
best individual
μ individuals
Generational replacement
Genotype ↓ phenotype mapping
Fitness evaluation of the μ offspring

Figure 1.15. *A simple genetic algorithm*

Many variants of genetic algorithms have been proposed in order to improve their performances or to extend their application domains. Thus, the bit strings have been replaced by other data structures closer to the natural formalism of the problems to solve, provided that appropriate variation operators are available. This avoids the difficult and complex question of the design of an efficient coding. For example, the "Real Coded Genetic Algorithms" use genotypes that are real vectors, instead of binary strings, to solve problems defined in $\mathbb{R}^n$. In addition, proportional selection is often replaced by other kinds of selection. These modifications are significant enough that the specific features of the genetic algorithms blend in with the diversity of the other evolutionary approaches.

1.7. Conclusion

This introductory chapter has presented the basic principles of evolutionary algorithms and a collection of widespread selection and variation operators. Binary representation has been addressed to introduce the transcription genotype–phenotype implemented in the genetic algorithm. Other representations are used in the world of evolutionary algorithms. They will be described in further chapters with their associated variation operators.