Contents

Chapter 1

Genetic algorithms for numerical optimization

1.1 Introduction

Even since 1950s scientists became interested in simulating biological evolution. The techniques developed over the years, based on principles of natural selection and genetic inheritance, are now grouped under the name *evolutionary computing* or *evolutionary algorithms*.

Four main approaches of the basic idea have been developed. In Germany, Rechenberg and Schwefel invented *evolution strategies* in about the same period as Fogel, Owens and Walsh introduced *evolutionary programming* in the United States. In 1960s, John Holland from the University of Michigan developed *genetic algorithms*, and thirty years later, the principles of biological evolution were applied by John Koza to *genetic programming*, (Mitchell, 1998; Eiben and Smith, 2003).

Evolutionary algorithms have been successfully applied to a large variety of problems from optimization, engineering design, resource scheduling, data mining, systems identification, structural design etc.

Genetic algorithms (GAs) are some of the most widely known strategies of evolutionary computation. Among other applications, they have been successfully used for difficult optimization problems with objective functions that are multi-modal, discontinuous and non-differentiable, (Houck et al., 1995). Traditional optimization algorithms would fail in most of these cases, as it is shown in the following examples.

Example 1.1 *The highly multimodal* Rastrigin function*, (Marco-Blaszka and Désidéri, 1999), defined by:*

$$f(\mathbf{x}) = nA - \sum_{i=1}^{n} \left(x_i^2 - A\cos 2\pi x_i \right), \quad x_i \in [-5.12, \ \ 5.15] \tag{1.1}$$

is shown in Figure 1.1 for $n = 2$ and $A = 2$.

A plot of Schwefels function*:*

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|}), \quad x_i \in [-500, \ \ 500], \ \ i = \overline{1, n} \tag{1.2}$$

is shown in Figure 1.2, for $n = 2$.

Both functions have a global minimum, which can be hardly determined by means of gradient techniques or any of the methods described in previous chapters. Classical search techniques would return a local optimum, depending of the initial point selected.

Genetic algorithms will be introduced as a method for single-variable function optimization and then extended for the multivariable case.

The general optimization problem will be formulated as maximization over a given interval $[a, \ b]$:

$$\max_{x} f(x), \quad x \in [a, b] \tag{1.3}$$

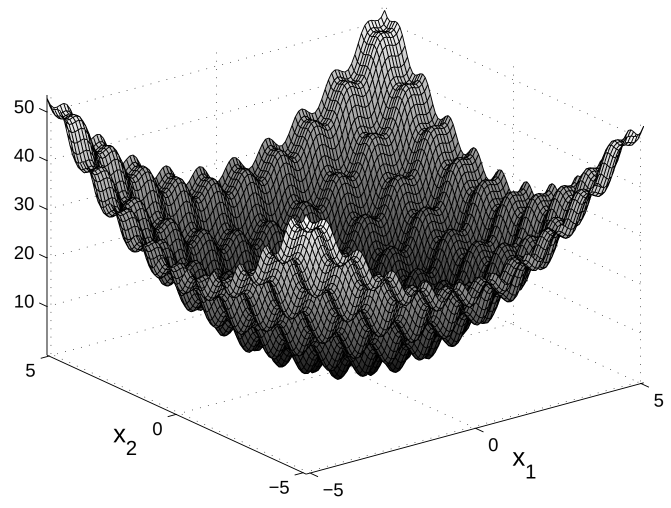

Figure 1.1: Rastrigin's function

1.2 Outline of a genetic algorithm

Genetic algorithms are probabilistic search techniques based on the ideas of natural evolution and the Darwinian principle of *survival of the fittest*. An extended terminology, borrowed from natural genetics, describes the main features of genetic algorithms. We shall introduce only those terms that apply directly to an optimization problem, (Michalewicz, 1996; Mitchell, 1998):

- An initial *population* of random *individuals* is created. Individuals, or *chromosomes* are made of *genes* and incorporate the variable information. They are candidate solutions of the problem.
- Individuals are assigned a *fitness* measure which is usually the objective function.
- The concept of survival of the fittest is implemented in a *selection* procedure. Individuals with higher fitness will have a better chance to survive.
- Selected individuals, (*the parents*), are chosen for reproduction (or *crossover*), to produce new individuals (*the offspring*), which will inherit some of the characteristics of the parents.
- Some of the new individuals are altered by *mutation* which is an arbitrary change in the genes of a selected chromosome.
- The newly produced individuals will be placed into the population of the next *generation* with a better average fitness. The process is repeated leading eventually to an optimum or to a best fit individual.

Algorithm 1 summarizes the general structure of a genetic algorithm. Each of the concepts introduced here will be detailed in the subsequent sections.

1.3 Fitness function

The fitness function evaluates the performance of an individual. For the basic optimization problem given by (1.3), the fitness function can be the same as the objective function $f(x)$.

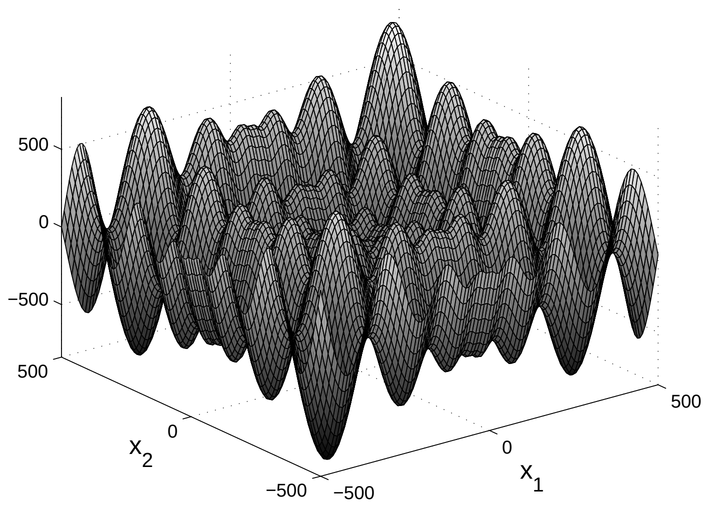

Figure 1.2: Schwefel's function

Algorithm 1 Genetic algorithm

```
Initial population: generate a population of $N$ random individuals
while stop criterion not fulfilled do
    Selection: select the fittest individuals for reproduction
    Reproduction: create new individuals by applying genetic operators (crossover, mutation)
    Recombination: creation of the new population
end while
```

GAs are generally developed to solve a maximization problem. A minimization problem can always be converted into a maximization one by changing the sign of the function:

$$\min_x f(x) = \max_x(-f(x)) \tag{1.4}$$

Some of the selection methods that will be introduced, require that the values of the fitness function are strictly positive for any value of $x \in [a, b]$. In this case, a common practice is to add a large positive value to $f(x)$ and solve the problem:

$$\max_x \left(f(x) + C\right), \quad C > 0 \tag{1.5}$$

1.4 Solution representation

The representation, or encoding, of one individual depends on the nature of the problem. It is generally a string of parameters, also known as genes, from a given alphabet. In Holland's original design, the genes were binary digits (0 and 1). They can also be real values, integers, symbols, etc.

The set of all feasible individuals is known as the *search space*.

1.4.1 Binary representation

Consider the problem of maximizing a single-variable function over an interval $[a,\ b]$ with an accuracy ε. A chromosome, in the binary representation, is a binary vector with the number of bits determined by the precision required.

The length of the interval is $b - a$ and the solution can be one of the $(b - a)/\varepsilon$ real numbers located between a and b and equally spaced by ε (Figure 1.3).

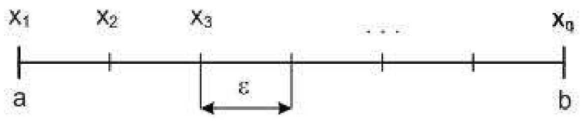

Figure 1.3: Potential solutions in $[a,\ b]$

All potential solutions will be mapped into binary numbers between $00\ldots0$ and $11\ldots1$. The lower bound of the interval (a) is mapped into a string of zeros and the upper bound (b) is mapped into a string of ones. The number of bits (n) of one chromosome is determined from the following relation:

$$2^n - 1 \geq \frac{b-a}{\varepsilon} \tag{1.6}$$

Note that $2^n - 1$ is an integer number whose base 2 representation is a string of ones of length n. Thus, n will be chosen so that $2^n - 1$ is the closest integer that is greater or equal than the number of potential solutions in the interval. The interval $[a,\ b]$ is mapped, actually, into the base 2 representation of the first $2^n - 1$ integer numbers.

For the reverse process, of mapping a binary string into a real number from $[a,\ b]$ two steps have to be taken:

- Convert the binary string to base 10 and obtain an integer number X.
- The corresponding real number, x, is calculated from:

$$x = a + \frac{b-a}{2^n - 1}X \tag{1.7}$$

Example 1.2 *Consider the interval $[a,\ b] = [-2,\ \ 2]$ and the desired accuracy of the solution $\varepsilon = 10^{-2}$. The length of the interval is 4 and the number of real values equally spaced by 10^{-2} is $4/10^{-2} = 400$. The chromosomes will be represented as binary strings of 9 bits between*

$$\underbrace{0\,0\,\ldots\,0}_{9}\ \ \textit{and}\ \ \underbrace{1\,1\,\ldots\,1}_{9}$$

because

$$2^9 - 1 = 511 > 400 > 2^8 - 1 = 255 \tag{1.8}$$

The precision will actually be better than 10^{-2} because there are 512 possible solutions in the search space, thus the distance between two consecutive values is $4/512 = 0.007$.

The mapping of the original interval into binary numbers is presented in Figure 1.4.

If we want to obtain the corresponding real value of the individual 101010101, it has to be converted first to base 10 and we obtain $X = (101010101)_2 = (341)_{10}$. The real value is then:

$$x = -2 + \frac{4}{2^9 - 1}341 = 0.6692 \tag{1.9}$$

1.4.2 Real-value representation

In the *real-value* (or *floating-point*) representation, an individual can take any continuous value in the interval $[a,\ b]$. The precision is only limited by the roundoff error of the computer.

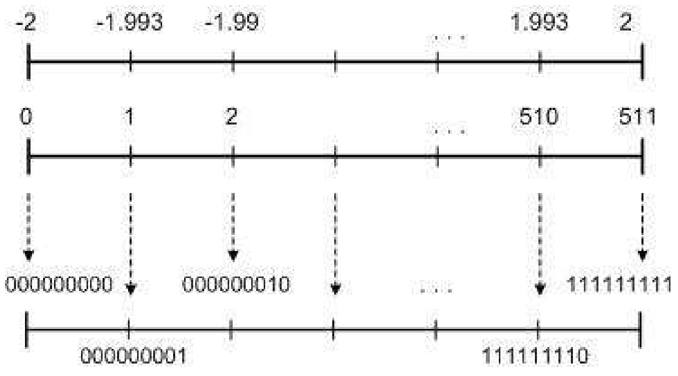

Figure 1.4: Potential solutions in $[-2,\ 2]$

1.5 Initial population

The initial population is a set of N randomly generated individuals from the search space. For a binary representation, N random binary strings of length n are generated. In case of real-value representation, N random numbers are generated in the interval $[a,\ b]$.

The population will evolve during the algorithm, but the size of the population (N) remains unchanged.

1.6 Selection

The selection process implements the "survival of the fittest" concept. Individuals undergo a probabilistic selection based on their fitness value which will decide wether or not they will survive in the next generation. Several copies of the same individual may be selected. Good individuals, with high fitness, have more chances to survive and reproduce, while the inferior ones are less likely to be selected.

The result of the selection process is an intermediate population known as *the mating pool*, of size N. Genetic operators (crossover and mutation) will be applied to a part of the individuals in the mating pool.

There are several schemes for selection which will be presented in the following sections: roulette wheel, stochastic universal sampling, ranking and tournament selection.

1.6.1 Roulette wheel selection

Roulette wheel is the original selection mechanism for GAs. Implementation of this approach requires that all the fitness values are positive numbers. Therefore, the objective function must be scaled as indicated by (1.5).

Consider a roulette wheel where all the chromosomes (x_i) in a population of size N, are located in sectors proportional to their fitness. Denote

$$F = \sum_{i=1}^{N} f(x_i) \tag{1.10}$$

the total fitness of the population. If this is equal to the circumference of the roulette wheel, each chromosome in the population will have a corresponding sector bounded by an arc equal to its fitness.

When spinning the wheel the probability that an individual will be selected is proportional to its fitness.

In order to obtain N individuals in the mating pool, the wheel is spun N times. The same individual can be chosen several times, which is very likely to happen when it has a large fitness value.

Example 1.3 *Figure 1.5 a) shows the roulette wheel selection procedure. The population is composed of 5 individuals placed on the roulette wheel in sectors proportional to their fitness. The largest interval is occupied by individual 4*

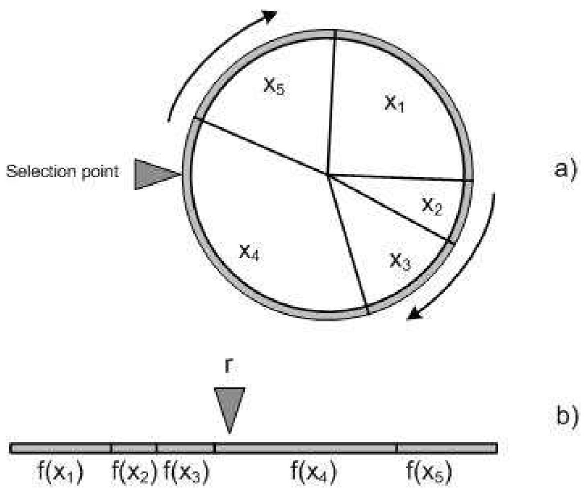

Figure 1.5: Roulette wheel selection

which has a large probability to be selected, even more than once. Individual 2 which is the least fit, corresponds to a smaller interval within the roulette wheel. It may be selected, by chance, but it is less probable that it will be selected for several times.

For a computer implementation of this procedure, consider the unfolded circumference of the roulette wheel, (Figure 1.5 b). All chromosomes in the current population will line up, as segments having the length equal to the individual fitness. The total length of the segments is equal to F. Spinning the wheel is equivalent to generating a random number (r) from the interval $[0,\ F]$. The individual whose segment spans the random number will be selected, (Kwong et al., 1999).

The roulette wheel strategy is described by Algorithm 2, (Obitko, 1998).

Algorithm 2 Roulette wheel selection

```
Calculate the sum of all individual fitness
    $$F = \sum_{i=1}^{N} f(x_i)$$
for j=1:N do
  Generate a random number, $r \in [0,\ F]$
  Set $S = 0, i = 0$
  while $S \leq r$ do
    Set $i \leftarrow i + 1$
    Set $S \leftarrow S + f(x_i)$
  end while
  Return the individual $x_i$
end for
```

1.6.2 Stochastic universal sampling

As an alternative to the roulette wheel selection, Baker (1987) suggested the stochastic universal sampling.

To introduce this selection method, like in the case of roulette wheel, we shall consider the individuals in a population located in sectors proportional to the individual fitness. The wheel has N selection points equally spaced and it will be spun just one time to select all N individuals to be transferred in the mating pool. The number of times an individual is selected is equal to the number of pointers that fall into its slot.

Example 1.4 *Consider the same population of 5 chromosomes as in Example 1.3. The wheel, having 5 equally spaced pointers is spun once. As shown in Figure 1.6 a) the selected individuals are* x_1, x_3, x_4, x_4, x_5.

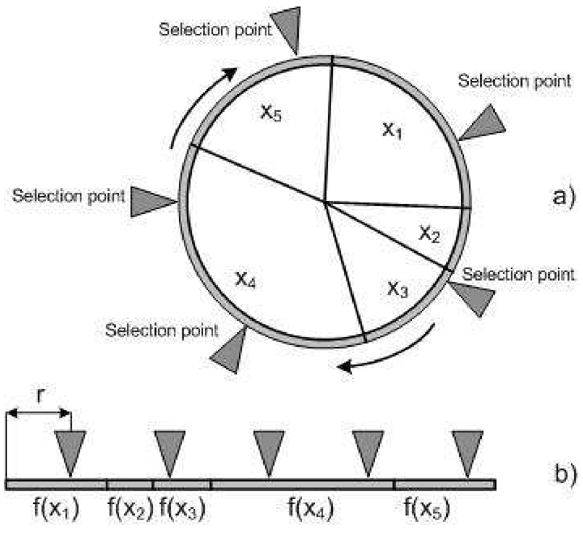

Figure 1.6: Stochastic universal sampling

The Figure 1.6 b) shows the unfolded circumference of the wheel, where the individuals are mapped to segments of a line, such that each individual's segment has the size equal to its fitness, as in roulette-wheel selection. The total length of the line is F as given by the relation (1.10).

If N individuals are to be selected, an array of N equally-spaced pointers is placed over the line. The distance between two of them is F/N. "Spinning the wheel", in this case, is the same as generating a random number $r \in [0, F/N]$ which will determine the position of the first pointer.

The algorithm 3 describes this procedure.

Algorithm 3 Stochastic universal sampling

```
Calculate the sum of all individual fitness
    $$F = \sum_{i=1}^{N} f(x_i)$$
Generate a random number $r \in [0,\ F/N]$
Compute the pointers $P_i = r + (i-1)F/N, \quad i = \overline{1:N}$
Set $i = 0$, $S = 0$
for j=1:N do
    Set $S \leftarrow S + f(x_j)$
    while $i \leq N$ and $P_i \leq S$ do
        Set $i \leftarrow i + 1$
        Return $x_j$
    end while
end for
```

1.6.3 Rank-based selection

Consider the case when the fitness of one individual in a population is very large. In the roulette wheel selection, the other individuals have a very small probability to be selected. This situation usually leads to a premature convergence, which means that the population is loosing diversity quickly and the solutions may converge to a local optimum.

In rank-based selection, each individual is assigned a rank based on its fitness. If the size of the population is N, the best individual receives the rank N, the second best, $N-1$, and the assignments continue until the worst will have the rank 1. The method is the same as roulette wheel selection when using the ranks instead of the fitness values.

1.6.4 Tournament selection

The tournament selection procedure follows two steps which are to be repeated N times:

- A group of $T \in [2,\ N]$ individuals are chosen randomly from the current population.
- The best individual from the group is selected for the mating pool.

The number T is known as *the tournament size*. If it has a large value, the least fit individuals have very few chances to be selected. In practice, a small tournament size is preferred, as this will maintain the diversity in the population and avoids the premature convergence.

1.7 Genetic operators

The selection procedure of a genetic algorithm produces a mating pool of size N. Genetic operators, *crossover* and *mutation*, are applied to individuals from this intermediate population with probabilities p_c and p_m, as detailed in the subsequent sections.

1.7.1 Crossover

The probability of crossover p_c gives the expected number $p_c N$ of chromosomes that undergo the crossover operation, according to the following procedure, (Michalewicz, 1996)

For each chromosome:

- Generate a random number $r \in [0,\ 1]$
- If $r < p_c$ select the chromosome for crossover

Two parent chromosomes are then chosen randomly to mate and produce two new offspring.

1.7.1.1 Binary crossover

One point crossover

A random number between 1 and the length of the chromosome is generated (the crossover point). The offspring chromosomes will have the first part copied, and the last part switched between the parents (Figure 1.7).

Example 1.5 *The crossover operation applied to two binary chromosomes is given below. The crossover point is a random number between 1 and 10 (the length of a chromosome) and, in this case, is equal to 4.*

Parent 1:	*1101*|*001110*
Parent 2:	0100|101101
Offspring 1:	*1101*|101101
Offspring 2:	0100|*001110*

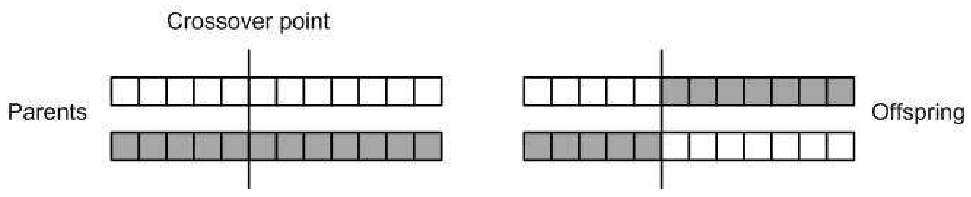

Figure 1.7: One point crossover

Multi-point crossover
Multiple crossover points can be randomly selected and the genes between them are interchanged to produce the offspring. Figures 1.8 and 1.9 show the two-point and the three-point crossover.

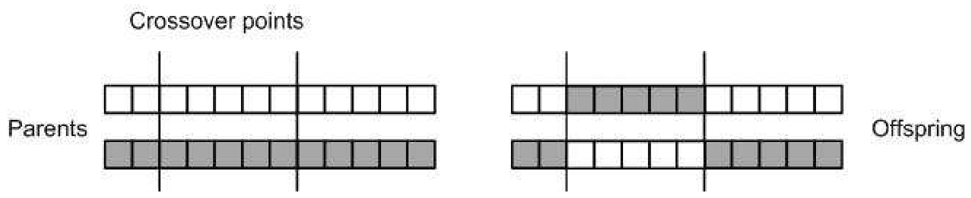

Figure 1.8: Two-point crossover

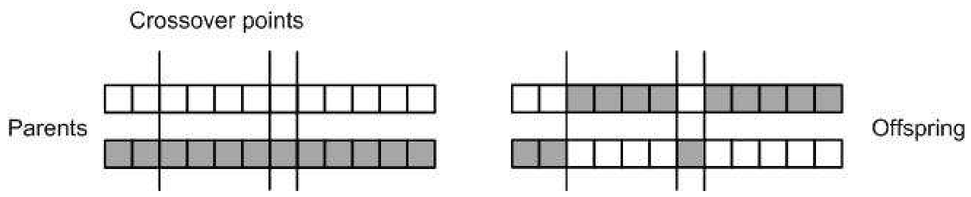

Figure 1.9: Three-point crossover

Uniform crossover
This strategy allows the offspring to have any combination of the parent chromosomes. A random binary mask of the same length as a parent chromosome is generated with a probability usually set to 0.5. The first offspring will inherit the genes of the first parent in the positions for which the mask is 0 and the genes of the second parent for the positions where the mask is 1. The other offspring is generated in the same manner: it will have the bits of the first parent where the mask is 1 and those of the second parent where the mask is 0 (Figure 1.10).

1.7.1.2 Real-valued crossover

Arithmetic crossover
If x and y are two selected parents, a random number $a \in [0,\ 1]$ is generated for each crossover operation and the offspring (x' and y') are calculated according to the relations:

$$\begin{aligned} x' &= a \cdot x + (1-a) \cdot y \\ y' &= (1-a) \cdot x + a \cdot y \end{aligned} \tag{1.11}$$

Example 1.6 *Let $x = 0.3$ and $y = 1.4$ be the parent individuals. If $a = 0.2$ was randomly generated in the range $[0,\ 1]$, the following two offspring are produced:*

$$\begin{aligned} x' &= 0.2 \cdot 0.3 + (1-0.2) \cdot 1.4 = 1.18 \\ y' &= (1-0.2) \cdot 0.3 + 0.2 \cdot 1.4 = 0.52 \end{aligned} \tag{1.12}$$

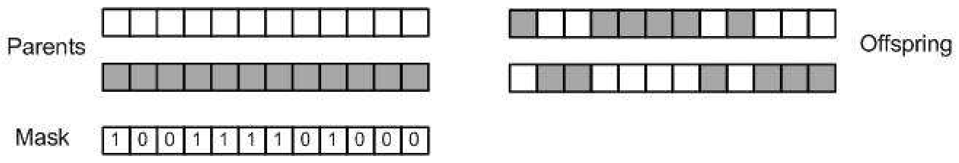

Figure 1.10: Uniform crossover

Heuristic crossover

The heuristic crossover is the only operator that uses fitness information and produces a linear extrapolation of the parents.

The offspring are created according to the following procedure:

1. Compute the fitness of the parents and denote x_{best} the best individual and x_{worst} the other one.
2. Generate a random number $r \in [0,\ 1]$ and compute the offspring:

$$\begin{aligned} x' &= x_{best} + r \cdot (x_{best} - x_{worst}) \\ y' &= x_{best} \end{aligned} \tag{1.13}$$

3. If x' is feasible (i.e. it is between the allowable upper and lower bounds of the variables) return the individuals calculated from (1.13)
4. If y' is not feasible, go to step 2 until the offspring is feasible or the failures exceed a number n set a priori. In this last case, return the offspring equal to the parents.

1.7.2 Mutation

Mutation is the genetic operation that changes randomly a part of the offspring resulted from crossover. It prevents the algorithm falling into a local optimum.

1.7.2.1 Binary mutation

A binary mutation (or a bit-flip mutation) is a bit-by-bit operation with probability p_m. Every bit in the current population (after crossover) has an equal chance to be mutated, i.e. changed from 0 to 1 or vice versa. The mutation can be implemented according to the following procedure, (Michalewicz, 1996):

For each chromosome in the current population and for each bit within the chromosome:

- Generate a random number $r \in [0,\ 1]$
- If $r < p_m$ mutate the bit.

The probability of mutation is usually chosen as a very small number, otherwise the algorithm may become a random search. Typically it is chosen between 1/population size and 1/chromosome length.

Example 1.7 *Consider the binary individuals given below. For all genes of all chromosomes in a population a random number r is generated between 0 and 1. If* $r < p_m = 0.15$ *the gene is switched from 0 to 1 and vice versa.*

	Individual 1					*Individual n*			
Before mutation	*1*	*1*	***0***	*1*	…	*1*	***1***	*0*	*0*
r	*0.41*	*0.22*	***0.09***	*0.9*	…	*0.25*	***0.13***	*0.34*	*0.95*
After mutation	*1*	*1*	***1***	*1*	…	*1*	***0***	*0*	*0*

1.7.2.2 Real-valued mutation

Uniform mutation

In a uniform mutation, an individual, chosen with the probability p_m, is replaced by another randomly chosen value from the search space.

Boundary mutation

Boundary mutation selects an individual (with probability p_m) and sets it equal to either the upper of lower bound of the search space $[a,\ b]$. For each individual selected for mutation, a random number $r \in [0,\ 1]$ is generated. The mutated individual is set equal to a if $r < 0.5$ and to b if $r \geq 0.5$, (Houck et al., 1995)

Non-uniform mutation

The non-uniform mutation operator increases the probability that the amount of the mutation will be close to 0 as the generation number increases.

If x_i is a selected individual, the result of the non-uniform mutation, denoted x_i' is calculated from, (Houck et al., 1995):

$$x_i' = \begin{cases} x_i + (b - x_i)F(G), & \text{if } r_1 < 0.5 \\ x_i - (x_i + a)F(G), & \text{if } r_1 \geq 0.5 \end{cases} \tag{1.14}$$

where:
a and b are the bounds of the search space,

$$F(G) = \left[r_2\left(1 - \frac{G}{G_{max}}\right)\right]^{r_2} \tag{1.15}$$

G is the current generation,
G_{max} is the maximum number of generations,
$r_1,\ r_2 \in [0,\ 1]$ are two random numbers,
b is a shape parameter.

1.8 Elitism

Elitism is the method that copies the best individual directly to the new population. It prevents loosing the best solution found to date.

Any genetic algorithm is said to be elitist if the following condition holds for all generations, (Reeves and Rowe, 2003):

Let x_k be the best individual at the generation t. Then at generation $t + 1$ either:

- x_k is in the population, or
- something better than x_k is in the population

A genetic algorithm can be made elitist by the following adaptation, (Reeves and Rowe, 2003):

- record the current best individual x_k in the population
- generate the next population as usual
- if there is nothing better than x_k, add x_k to the new population, replacing some other individual (for example the worst individual in the new population).

Elitism may be implemented with more than one individuals, according to the same procedure. A number N_e of best individuals can be selected to survive in the new population in exchange of the worst ones. In usual implementations, N_e is small (e.g. between 1 and 7) to prevent the premature convergence.

1.9 Termination criteria

The genetic algorithms compute new generations until a termination criterion is met. Most frequently, the following stop conditions are used:

Generation number. The evolution is stopped when a user-specified maximum number of generations is reached.

Evolution time. The algorithm stops after running for a time that exceeds a user-specified limit.

Fitness threshold. When the fitness of the best individual in the current population is greater than a specified threshold, the algorithm can be terminated.

Population convergence. If the difference between the average fitness and the best fitness of the current population, is less than a specified amount, the algorithm stops.

1.10 Parameters of a genetic algorithm

The results returned by a genetic algorithm are very much dependent of the parameters it uses. Many authors proposed sets of parameters like: the population size, maximum number of generations, crossover and mutation probabilities, type of crossover and mutation, but they are not guaranteed to give very good results for any problem.

For example, a well known set is, (DeJong and Spears, 1990; Grefenstette, 1986):

- Population size: 50
- Number of generations: 1000
- Probability of crossover: 0.6
- Probability of mutation: 0.001

When the objective function is computationally expensive, Grefenstette (1986) proposed:

- Population size: 30
- Probability of crossover: 0.9
- Probability of mutation: 0.01

Most studies in the field of GAs do not recommend a large population size. Generally, it may be chosen between 20 and 100 individuals. The probability of crossover is typically chosen between 0.6 and 0.8 by the most authors, and the probability of mutation is recommended to be set somewhere between 0.05 and 0.1.

1.11 Genetic algorithms for multivariable function optimization

The general problem of maximizing a function of n variables can be solved using the same general framework of a genetic algorithm as described in the previous sections. Only the specific issues related to the number of variables will be addressed in this section.

The problem to be solved is:

$$\max_{\mathbf{x}} f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n), \quad \mathbf{x} \in [\mathbf{a},\ \mathbf{b}] \tag{1.16}$$

where $\mathbf{a} = [a_1\ a_2\ \ldots\ a_n]^T$ and $\mathbf{b} = [b_1\ b_2\ \ldots\ b_n]^T$ are $n \times 1$ vectors containing the lower and upper bounds of the variables.

1.11.1 Solution representation and the initial population

1.11.1.1 Binary representation

Consider a function $f(x_1,\ x_2,\dots,x_n)$ where the variables are constrained to be within $[a_i,\ b_i]$, $i=\overline{1,n}$, and the required precision is ε.

The same procedure as the one presented in section 1.4.1 can be applied for each variable taken separately. If the ranges $[a_i,\ b_i]$ have various lengths, the number of bits required for the representation of each variable are different.

A chromosome will be a binary string of a length, m, equal to the sum of the number of bits, of all variables.

The initial population is generated as N random binary strings of length m.

Example 1.8 *Consider the problem of maximizing a function of two variables $f(x_1,x_2)$, where $x_1\in[-2,\ 2]$ and $x_2\in[-1,1]$. The required precision is $\varepsilon=10^{-2}$.*

The number of bits (m_1 and m_2) necessary for the representation of the variables are obtained from:

$$2^{m_1}-1\geq\frac{4}{10^{-2}}=400 \tag{1.17}$$

$$2^{m_2}-1\geq\frac{2}{10^{-2}}=200 \tag{1.18}$$

Because

$$2^9-1=511>400>2^8-1=255 \tag{1.19}$$

$$2^8-1=255>200>2^7-1=127 \tag{1.20}$$

we obtain: $m_1=9$ and $m_2=8$. A chromosome will have a total length of 17 bits where the first 9 represent the variable x_1 and the last 8, the variable x_2.

1.11.1.2 Floating-point representation

For the floating-point representation, the chromosomes are represented by real-valued vectors. Each element of the vectors are within given ranges $[a_i,\ b_i]$, $i=\overline{1,n}$.

An initial population is obtained by generating N random vectors of n variables in the given search space.

1.11.2 Selection

Since the selection uses only the fitness function to choose the best fit individuals, any of the previously described strategies can be applied unchanged for multivariable problems.

1.11.3 Genetic operators

The genetic operators for binary encoded solutions can be applied in the same manner as in the case of single variable functions. For the floating-point representation of solutions, the operations are presented in the followings.

1.11.3.1 Crossover

If the selected individuals for crossover are $\mathbf{x}=(x_1,x_2,\dots,x_n)$ and $\mathbf{y}=(y_1,y_2,\dots,y_n)$, the operation may be implemented in several variants, (Eiben and Smith, 2003):

Simple crossover. A crossover point k is chosen randomly between 1 and the length of the individual (i.e. the number of variables). The offspring $\mathbf{x}'$ and $\mathbf{y}'$ will inherit the first k elements of the parents and the rest are calculated as the arithmetic average of the parents:

$$\begin{aligned}\mathbf{x}' &= (x_1,\ x_2,\ \ldots x_k,\ ax_{k+1} + (1-a)y_{k+1}, \\ &\quad \ldots,\ ax_n + (1-a)y_n) \qquad (1.21)\\ \mathbf{y}' &= (y_1,\ y_2,\ \ldots y_k,\ ay_{k+1} + (1-a)x_{k+1}, \\ &\quad \ldots,\ ay_n + (1-a)x_n) \qquad (1.22)\end{aligned}$$

where $a \in [0,\ 1]$ is a random number.

Single arithmetic crossover. A crossover point k is generated between 1 and the length of an individual. At that position take the arithmetic average of the parents. The offspring will be calculated as:

$$\mathbf{x}' = (x_1, x_2, \ldots x_k, ax_{k+1} + (1-a)y_{k+1}, x_{k+1}, \ldots, x_n) \qquad (1.23)$$

$$\mathbf{y}' = (y_1, y_2, \ldots y_k, ay_{k+1} + (1-a)x_{k+1}, y_{k+1}, \ldots, y_n) \qquad (1.24)$$

with a being a random number in the range $[0,\ 1]$.

Whole arithmetic crossover. The offspring is calculated from:

$$\mathbf{x}' = a\mathbf{x} + (1-a)\mathbf{y} \qquad (1.25)$$

$$\mathbf{y}' = a\mathbf{y} + (1-a)\mathbf{x} \qquad (1.26)$$

where $a \in [0,\ 1]$ is a random number.

1.11.3.2 Mutation

Mutation for multivariable problems is a generalization of the single-variable case, (Houck et al., 1995):

Uniform mutation. If $k \in [1,\ n]$ is selected randomly for an individual $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the elements of the mutated vector are:

$$x_i' = \begin{cases} \text{random number in } [a_i, b_i], \text{ for } i = k \\ x_i, \text{ for } i \neq k \end{cases} \qquad (1.27)$$

Boundary mutation. One randomly selected variable x_k is replaced by the upper or lower bound:

$$x_i' = \begin{cases} a_i, \text{ for } i = k \text{ and } r < 0.5 \\ b_i, \text{ for } i = k \text{ and } r \geq 0.5 \\ x_i \text{ for } i \neq k \end{cases} \qquad (1.28)$$

Non-uniform mutation. One randomly selected variable x_k is replaced by an expression similar to (1.14). For the same notation as the one used in (1.14) and (1.15), the elements of the mutated vector are computed from:

$$x_i' = \begin{cases} x_i + (b - x_i)F(G), \text{ for } r_1 < 0.5 \text{ and } i = k \\ x_i - (x_i + a)F(G), \text{ if } r_1 \geq 0.5 \text{ and } i = k \\ x_i, \text{ for } i \neq k \end{cases} \qquad (1.29)$$

1.12 Conclusions

Although sometimes genetic algorithms are considered ”methods of last resort”, they have major advantages including:

- GAs are flexible and robust as global search methods
- They do not use gradient information
- GAs can deal with highly nonlinear or non-differentiable functions, as well as with multimodal functions
- GAs can solve problems with a large number of variables
- GAs are well suited for parallel implementation
- They make very few assumptions about the problem to be solved

Genetic algorithms are not the best approach to the solution of any problem. For example, when the problem is to find the optimum of a convex or a quadratic function, traditional methods behave much better and may compute the solution in only a few steps.

However, many problems are complicated and the usefulness of genetic algorithms has made them a good choice among the classical optimization methods.

Bibliography

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14 – 21. L. Erlbaum Associates Inc, Hillsdale, NJ, USA.

DeJong, K. and Spears, W. (1990). An analysis of the interacting roles of population size and crossover in genetic algorithms. In *Proceedings of the First Workshop Parallel Problem Solving from Nature*, pages 38–47. Springer-Verlag.

Eiben, D. A. E. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer.

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions Systems, Man and Cybernetics*, SMC-16(1):122–128.

Houck, C. R., Joines, J. A., and Kay, M. G. (1995). A Genetic Algorithm for Function Optimization: A Matlab Implementation. Technical Report NCSU-IE TR 95-09, North Carolina State University.

Kwong, S., Man, K. F., and Tang, K. S. (1999). *Genetic Algorithms: Concepts and Designs*. Springer.

Marco-Blaszka, N. and Désidéri, J.-A. (1999). Numerical solution of optimization test-cases by genetic algorithms. Technical Report 3622, Institut National de Recherche en informatique et en Automatique, Sophia Antipolis.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures: Evolution Programs*. Springer.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.

Obitko, M. (1998). Introduction to genetic algorithms. URL http://www.obitko.com/tutorials/genetic-algorithms/.

Reeves, C. R. and Rowe, J. E. (2003). *Genetic Algorithms: Principles and Perspectives : a Guide to GA Theory*. Springer.