

## **BINARY-CODED GENETIC ALGORITHMS**

### **1. Introduction**

Genetic algorithms are methods that imitate genetic and selection mechanisms of nature. Genetic algorithms are simple, yet powerful optimization techniques. They have proved their success in different optimization problems in different fields and with different intractability conditions.

Idea of evolutionary computing was introduced in the 1960s by I. **Rechenberg** in his work "*Evolution strategies*" (*Evolutionstrategie* in original). His idea was then developed by other researchers. **Genetic Algorithms** (GAs) were invented by John **Holland** and developed by him and his students and colleagues. This lead to Holland's book "*Adaption in Natural and Artificial Systems*" published in 1975.

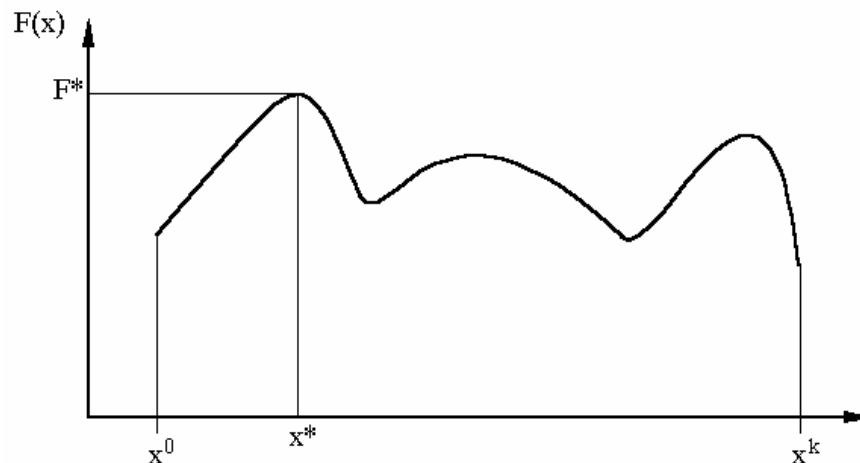
In 1992 John **Koza** has used genetic algorithm to evolve programs to perform certain tasks. He called his method "**genetic programming**" (GP). LISP programs were used, because programs in this language can expressed in the form of a "parse tree", which is the object the GA works on.

The main criteria in which genetic algorithms differ from the methods already described are as follows:

1. Parameters are not used as such, but only in their coded form.
2. Optima are obtained from population of points.
3. They are designed according to the rules of probability and not those of determination.

### **2. How do genetic algorithms work?**

To explain how a GA works, consider the optimization problem in which a function  $F(x)$  - shown in Figure 1- is to be maximized. It is required to search for the max value ( $F^*$ ) and its corresponding decision variable value ( $x^*$ ), given that the decision variable  $x$  is confined in the range ( $x^0 \leq x \leq x^k$ ).



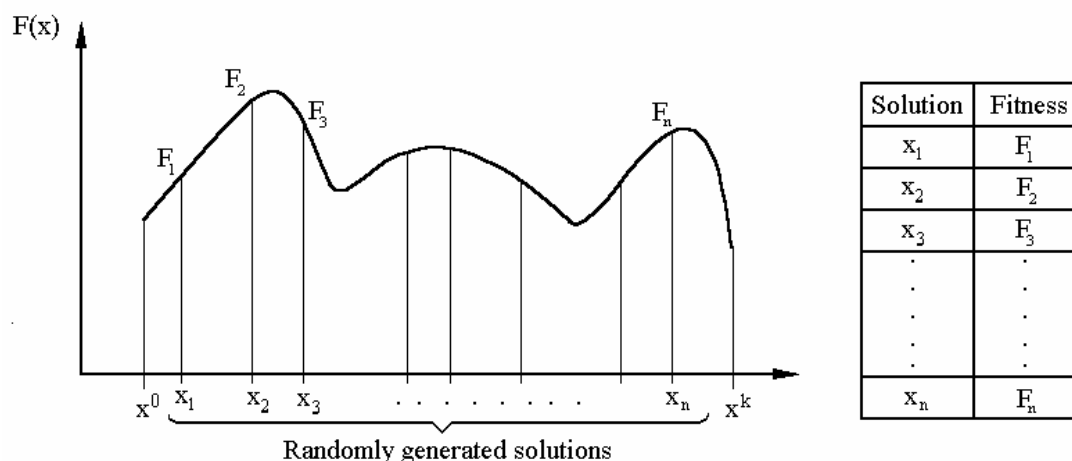
**Fig. 1 A sample function  $F(x)$**

First of all, to apply the GA to any optimization problem, the decision variables should be coded into a format called a *chromosome*. Chromosomes are strings of DNA and serves as a model for the whole organism. A chromosome consist of genes, blocks of DNA. Each gene encodes a particular protein. Basically can be said, that each gene encodes a trait, for example color of eyes. Possible settings for a trait (e.g. blue, brown) are called alleles. Each gene has its own position in the chromosome. This position is called locus. This chromosome, like natural chromosomes, is a string of small coding units called *genes*. Each gene is identified by its position and its value. For example the chromosome “8001AHK934” consists of 10 genes, the first gene has a value of “8”, the fifth has a value of “A” and so on. Every gene can accept any value called *allele* from a set of gene values called *allele set*. The  $\{0, 1\}$  allele set is the most widely used in Gas, especially for theoretical studies. A chromosome (*genotype*) can be interpreted to its corresponding real meaning (*phenotype*) in the optimization problem and vice versa by what is called *coding scheme*.

Actually the determination of the suitable coding scheme is not so easy, as it should consider the type of decision variables and may consider either some of the problem constraints while avoiding the use of excess data. In few words, GA coding should be minimal and completely expressive.

The first step in the GA is called the *initialization* process (Figure 2). In the initialization process a set of alternative solutions to the problem ( $x_0, x_1, x_2, \dots, x_n$ ) is randomly generated, while considering the satisfaction of the problem constraints. This set is called the *initial population*. The term *population* is used to refer to any set of solution alternatives, while each solution alternative in a population is called an *individual*. The number of solution alternatives in a population is called *population size*, which must be specified by the algorithm user-.

The objective function value  $F(x)$  is then computed for every individual, this value is called individual's *fitness*.



**Fig. 2 Initialization process**

Cairo University Faculty of Engineering	<b>Optimum Design</b> M.Sc. Qualification	Dr. Hesham A. Hegazi Binary Coded GAs
--	--	--

After initialization, GA applies recursively a set of operators to produce successive generations of populations. These operators (Figure 3) are applied a number of times till a new population has been formed, such that this new population has the same populations size as the initial one. The algorithm terminates after a specific number of generations or when certain criterion is satisfied.

The first operator is the *selection* or *reproduction* operator, in which two individuals are selected randomly from a population with a probability of selection proportional to the individual's fitness. That is the individual with better objective function has higher selection probability. The roulette wheel selector is one of the selection schemes that satisfy this condition.

During reproduction, first occurs **recombination** (or **crossover**). Genes from parents form in some way the whole new chromosome. The new created offspring can then be mutated. **Mutation** means, that the elements of DNA are a bit changed. These changes are mainly caused by errors in copying genes from parents. The **fitness** of an organism is measured by success of the organism in its life.

The second operator is the *crossover* operator, in which the two selected individuals are mated together and produce another two individuals (children). This operator is similar to the natural crossover process of chromosomes, by which the genes of the parent individuals are exchanged together and produce the children's chromosomes that will inherit some genes from the first parent and the other genes from the other parent. Crossover is not necessarily applied to all pairs of selected individuals. A choice is made, depending on a specific probability called *crossover probability*, which is typically between 0.5 and 1.0. If crossover is not applied, the children are simply a duplication of the selected individuals.

The third operator is the *mutation* operator, in which the produced children's genes may be altered randomly by replacing its value by another one from the allele set. It is done at small probability called *mutation probability*. This operator helps the algorithm to reach parts of the search space which perhaps cannot be reached by crossover alone.

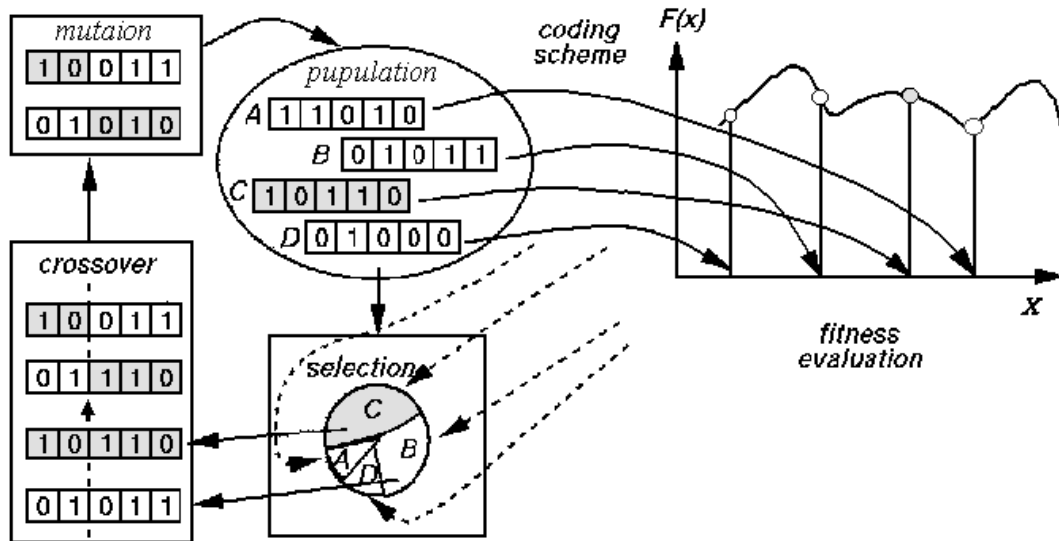


Fig. 3 GA Operators

### 3. GA Terminology

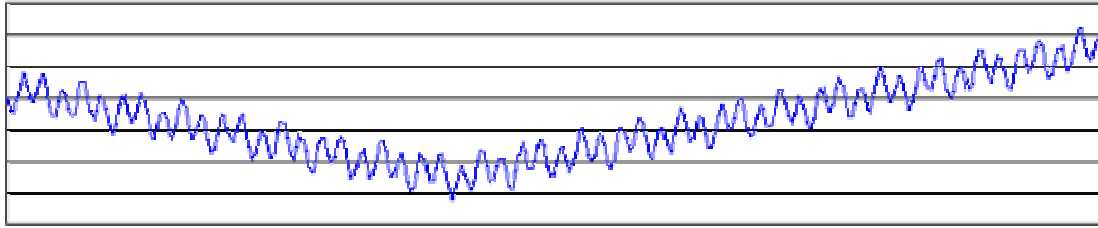
Many of the terms used in GAs stem from natural genetics. The following terms are extensively used in genetic algorithms.

- **Gene**: is the inheritance unit in natural genetic systems, and used in genetic algorithms to signify the coding unit.
- **Allele**: is the value assigned to a gene. Where the allele set is the set of possible values that can be assigned to a gene.
- **Chromosome**: is a string of genes, where a set of one or more chromosomes represents a solution alternative.
- **Gene locus**: is the position of a gene in a chromosome.
- **Individual**: is a solution alternative to the decision problem.
- **Population**: is a set of solution alternatives, characterized by its size.
- **Genotype**: is a word signifying the package of chromosomes used to encode a solution alternative.
- **Phenotype**: is the set of decoded decision variables, where their values represent a real meaning.

### 4. Search Space

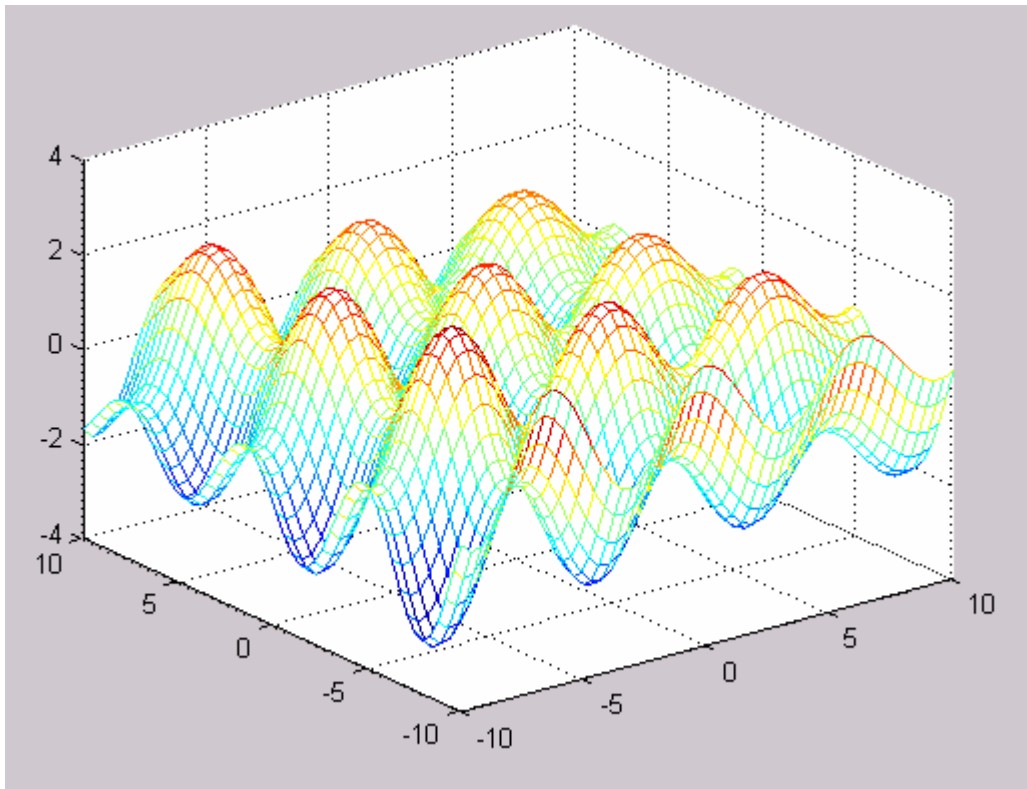
If we are solving some problem, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (it means objects among those the desired solution is) is called **search space** (also state space). Each point in the search space represents one feasible solution. Each feasible solution can be "marked" by its value or fitness for the problem. We are looking for our solution, which is one point (or more) among feasible solutions - that is one point in the search space.

The looking for a solution is then equal to a looking for some extreme (minimum or maximum) in the search space. The search space can be whole known by the time of solving a problem, but usually we know only a few points from it and we are generating other points as the process of finding solution continues.



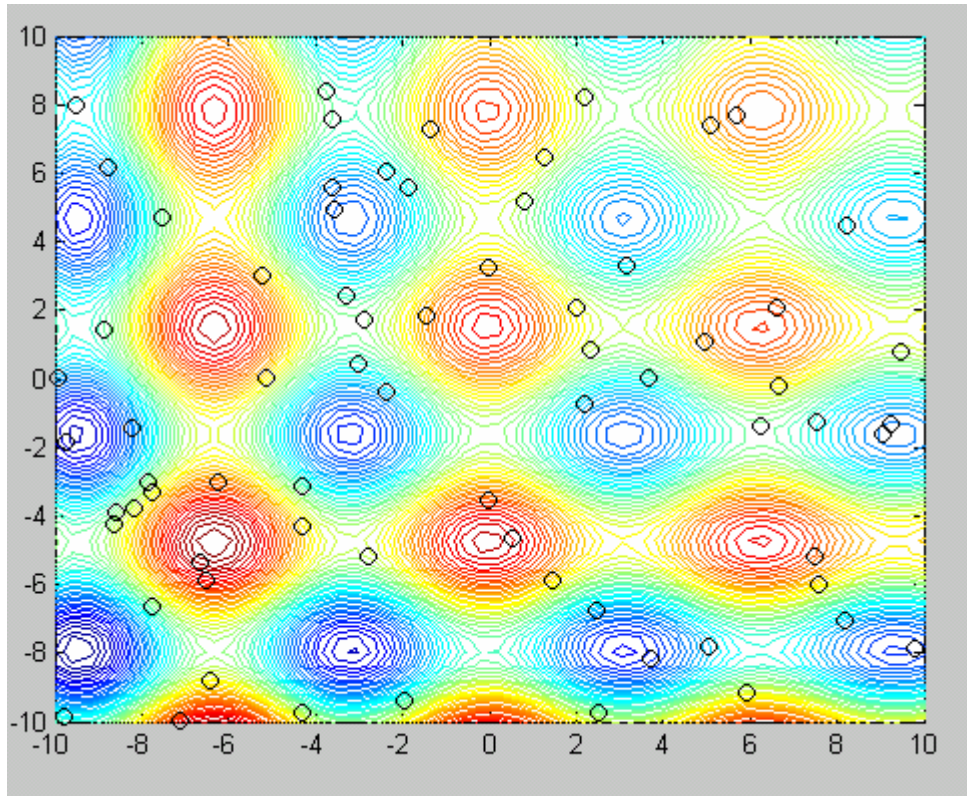
**Fig. 4 Example of a search space**

The problem is that the search can be very complicated. One does not know where to look for the solution and where to start. There are many methods, how to find some **suitable solution** (ie. not necessarily the **best solution**), for example **hill climbing**, **tabu search**, **simulated annealing** and **genetic algorithm**. The solution found by these methods is often considered as a good solution, because it is not often possible to prove what is the real optimum. Another example of a two dimensions function is shown in figure 5 (a,b).



**Fig. 5,a Example of an Objective Function of many local optimum**

$$F(x) = \cos(x)e^{-0.05x} + \sin(y)e^{-0.05y}$$



**Fig. 5,b Top View of Figure 4, with a population of points**

### **5. The general GA Algorithm**

Algorithm is started with a **set of solutions** (represented by **chromosomes**) called **population**. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (**offspring**) are selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

The outline of the Basic Genetic Algorithm is shown by the following pseudo-code as follows:

#### **[Step 0]**

- Generate random population of  $N$  chromosomes (suitable solutions for the problem)
- Set Generation counter  $G = 1$

#### **[Step 1]**

- 1.1 For each chromosome  $X_k$ ,  $k \in \{1, 2, \dots, N\}$ , evaluate  $F_k(X_k)$ , where  $F_k$  is the objective function value of the  $k^{th}$  chromosome.
- 1.2 Convert the objective function value into fitness value  $f_k$  such that optimization is converted into fitness maximization problem.

- 1.3 Apply the **Selection** operator, and copy (probabilistically) the high fitness chromosomes to a temporary population. (Survival of fitness)
- 1.4 Select (probabilistically) pairs of chromosomes to apply the **Crossover** operator.
- 1.5 Select (probabilistically) pairs of chromosomes to apply the **Mutation** operator.
- 1.6 Replace the population by the temporary population

**[Step 2]** (End of the generation loop)

If  $G < G_{max}$  (a pre-specified number of generations)

Let  $G = G + 1$ , and Goto Step 1

Else Deliver the chromosome with the highest fitness as the problem's solution.

## 5.1 **Coding**

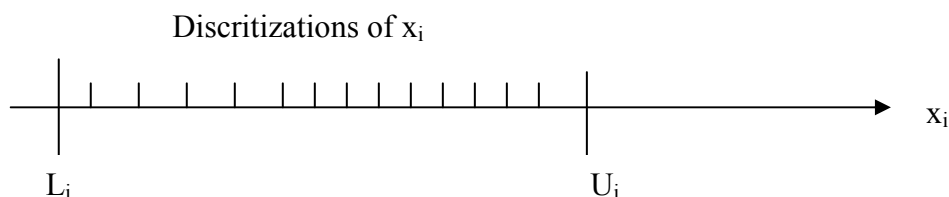
The chromosome should in some way contain information about solution which it represents. The most used way of encoding is a binary string. The chromosome then could look like this:

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution. Of course, there are many other ways of encoding. This depends mainly on the solved problem. For example, one can encode directly integer or real numbers, sometimes it is useful to encode some permutations and so on.

Other types of encoding are Permutation encoding, Value encoding, and Tree encoding. In **permutation encoding**, every chromosome is a string of numbers, which represents number in a **sequence**. In **value encoding**, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects. In **tree encoding** every chromosome is a tree of some objects, such as functions or commands in programming language.

Given a function  $F(x_1, x_2, x_3, \dots, x_n)$  to be optimized where each variable  $x_i$  ranges between two extremes  $L_i$  and  $U_i$ ,  $i \in \{1, 2, \dots, n\}$ .



If  $x_i$  is to discretized into a set of discrete values using binary coding then define an array of binary digits of length " $l$ "

$$\begin{array}{c}
 2^l \quad \dots\dots\dots 2^2 \quad 2^1 \quad 2^0 \\
 \boxed{v_l} \quad \boxed{\dots\dots\dots} \quad \boxed{v_2} \quad \boxed{v_1} \quad \boxed{v_0} \quad j \in \{1, 2, \dots, l\}
 \end{array}$$

The array is filled with values  $v_j$  where  $v_j = 0$  or  $1$

The real value of the array is equal to

$$R_i = L_i + \sum_{j=0}^l \lambda \cdot 2^j$$

Where  $\lambda$  is the discretization increment

If all elements of the array are equal to zero then  $R_i = L_i$

If all elements of the array are equal to one, the  $R_i$  should be equal to  $U_i$  i.e.

$$L_i + \sum_{j=0}^l \lambda \cdot 2^j = U_i$$

The above equation can be used to estimate the length " $l$ " of the array:

$$\sum_{j=0}^l 2^j = \frac{U_i - L_i}{\lambda}$$

Then  $2^{l+1} - 1 = \frac{U_i - L_i}{\lambda}$

Finally  $l = \log_2 \left[ \frac{U_i - L_i}{\lambda} + 1 \right] - 1$

Since each variable has its own array, the set of arrays for all variables forms another array known as Chromosome (also called agent, individual, or string). Each location in the chromosome array is known as "Gene", and the value it assumes (0 or 1) is known as "Allele".

### Example:

**Considering the function**  $F(x_1, x_2) = x_1^2 + x_2^2$

$$x_1 \in [0, 31]$$

$$x_2 \in [0, 31], \lambda = 1$$

$$\begin{array}{cccccccccc}
 2^4 & 2^3 & \mathbf{x_2} & 2^1 & 2^0 & & 2^4 & 2^3 & \mathbf{x_1} & 2^1 & 2^0 \\
 \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}}
 \end{array}$$



Let  $X_1 =$ 

1	0	0	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---

$= (17, 7), F(X_1) = 338$

Let  $X_2 =$ 

0	0	0	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---	---	---

$= (3, 24), F(X_1) = 585$

## 5.2 Initialization of a Population

**Population size** says how many chromosomes are in population (in one generation). If there are too few chromosomes, GA have a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.

$X_1$	$X_2$	.....	$X_n$	
				<b>Chromosome 1 (<math>X_1</math>)</b>
				<b>Chromosome 2 (<math>X_2</math>)</b>
				<b>Chromosome N (<math>X_N</math>)</b>

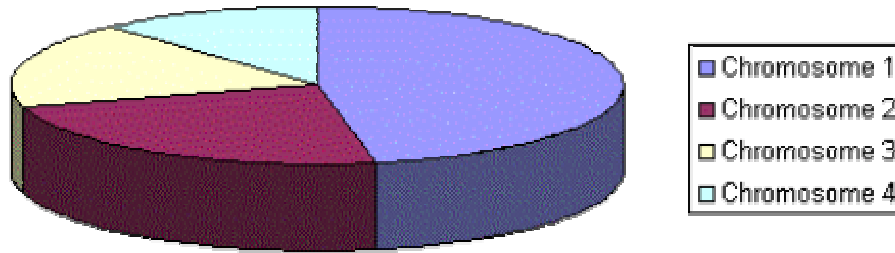
Each chromosome corresponds to a solution point in the space of the independent variables. The integer N is known as the population size (Typical value for N in GAs literature range between 50 – 200). Once, the population matrix is constructed, the population is filled at random with zeros and ones (i.e. each gene assumes a value 0 or 1 drawn randomly).

## 5.3 Selection

Chromosomes are selected from the population to be parents to crossover. The problem is how to select these chromosomes. According to Darwin's evolution theory the best ones should survive and create new offspring. There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others.

### Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a **roulette wheel** where are placed all chromosomes in the population, everyone has its place big accordingly to its fitness function, like on the following picture.



Then a marble is thrown there and selects the chromosome. Chromosome with bigger fitness will be selected more times.

The fitness of each chromosome is a measure of its importance relative to the objective function. An example of a fitness function is shown below.

Given a chromosome  $X_k$ ,  $k \in \{1, 2, \dots, N\}$

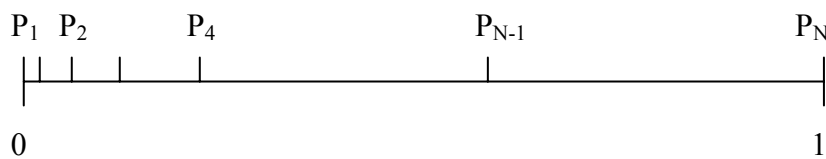
$$f_k(X_k) = F(X_k) \quad \text{case of maximization}$$

$$f_k(X_k) = \max (F(X_k)) - F(X_k) \quad \text{case of minimization}$$

As shown above, the minimization problem is turned into a maximization one. Use the fitness value to obtain a probability value  $P_k$  associated with each chromosome.

$$P_k = \frac{f_k}{\sum_{l=1}^N f_l}$$

Evaluate the cumulative probability  $P_k$  after sorting the population in ascending order according to the fitness value. Hence  $P_k$  is scaled from zero to one.



Repeat  $N$  times

- Generate a random number  $\alpha$  from a uniform distribution between zero and one.
- If  $\alpha$  falls between  $P_{k-1}$  and  $P_k$ , then copy chromosome  $X_k$  to the temporary population.

The selection operator is the responsible for the repetition of high fitness chromosomes.

## 5.4 Crossover

After we have decided what encoding we will use, we can make a step to crossover. Crossover selects genes from parent chromosomes and creates a new offspring. The

simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent.

Crossover can then look like this ( | is the crossover point):

Chromosome 1	11011   00100110110
Chromosome 2	11011   11000011110
Offspring 1	11011   11000011110
Offspring 2	11011   00100110110

There are other ways how to make crossover, for example we can choose more crossover points. Crossover can be rather complicated and very depends on encoding of chromosome. Specific crossover made for a specific problem can improve performance of the genetic algorithm.

**Crossover probability** says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is **100%**, then all offspring is made by crossover. If it is **0%**, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation.

### Crossover process

- 1- Select two chromosomes at random to apply crossover  $\{X_1, X_2\}$
- 2- Generate a random number  $\gamma \in [0,1]$   
If  $\gamma < \text{crossover probability}$  [Typical values of the crossover probability range between 0.5 to 0.9], then apply crossover, otherwise do not apply it, and just copy them to the new population.
- 3- Given  $X_1$  and  $X_2$  for example

$$\begin{array}{l}
 X_1 = [0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1, \ 1 \ 0 \ 1 \ 0 \ 1, \ \dots, \ 0 \ 0 \ 1 \ 0 \ 0 \ 1] \\
 X_2 = [1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0, \ 0 \ 0 \ 1 \ 1 \ 0, \ \dots, \ 1 \ 0 \ 0 \ 1 \ 1 \ 0] \\
 \begin{array}{ccc}
 x_1 & x_2 & x_n \\
 (l_1 = 7) & (l_2 = 5) & (l_n = 6)
 \end{array}
 \end{array}$$

For each variable  $x_i$ , find a random location  $Q$ ,  $Q \in [2, 3, 4, \dots, l_i]$ , then swap all bits after location  $Q$ .

### **Example:**

$$\begin{array}{l}
 \begin{array}{cc}
 x_1 & x_2 \\
 X_1 = [0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1, \ 1 \ 0 \ 1 \ 0 \ 1] \\
 X_2 = [1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0, \ 0 \ 0 \ 1 \ 1 \ 0]
 \end{array}
 \end{array}$$

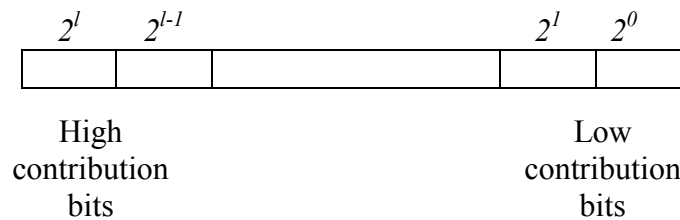
$$\text{Let } Q_1 = 3, Q_2 = 2$$

Then after Crossover

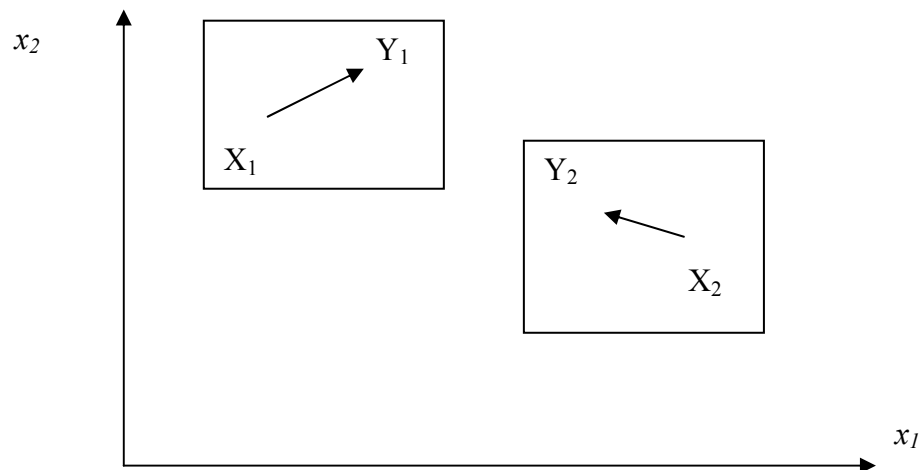
$$Y_1 = [0\ 1\ 1\ 1\ 1\ 0\ 0, 1\ 0\ 1\ 1\ 0]$$

$$Y_2 = [1\ 0\ 0\ 0\ 1\ 1\ 1, 0\ 0\ 1\ 0\ 1]$$

The effect of the crossover operator is the generation of a pair of new solution points in the vicinity of the parent solution points.



In continuous parameter optimization points, the swapping takes place within the low contribution bits leading to average small variations around the chromosomes, as shown in the following figure.



## 5.5 Mutation

After a crossover is performed, mutation takes place. This is to prevent falling all solutions in population into a local optimum of solved problem. Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation can then be following:

Original offspring 1	1101111000011110
Original offspring 2	1101100100110110
Mutated offspring 1	1100111000011110
Mutated offspring 2	1101101100110110

The mutation depends on the encoding as well as the crossover. For example when we are encoding permutations, mutation could be exchanging two genes.

**Mutation probability** says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is **100%**, whole chromosome is changed, if it is **0%**, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to **random search**.

**Example:**

$$\text{Let } X_1 = [1 \overset{x_1}{0} 1 1 0 0 1, 1 \overset{x_2}{1} 1 1 0 0 0]$$

$$\text{Let } Q_1 = 5, \text{ and } Q_2 = 2$$

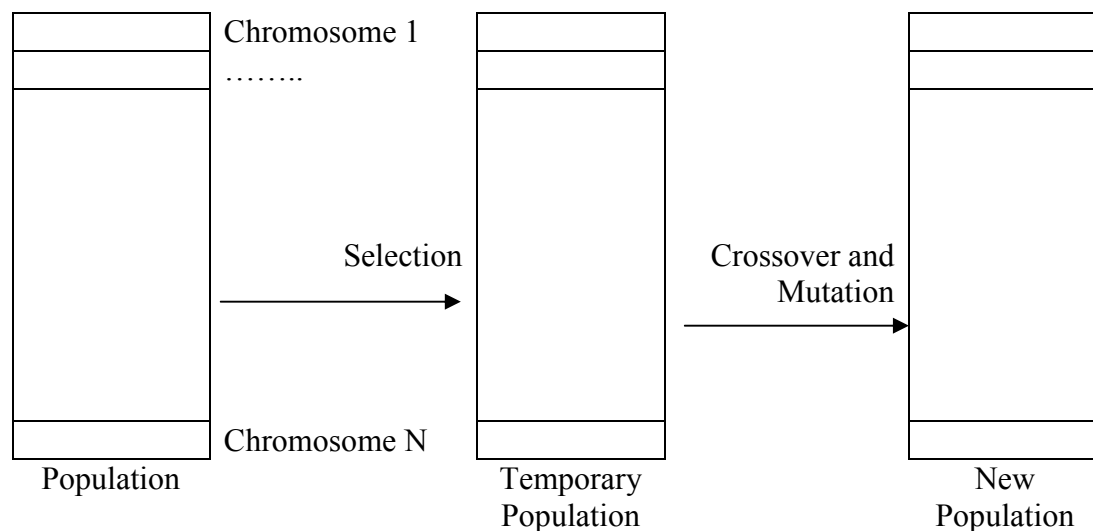
Then after Mutation

$$Y_1 = [1 0 1 1 1 0 1, 1 0 1 0 0 0]$$

The mutation operator is applied with a probability as follows:

- Select a chromosome X to apply the mutation
- Generate a random number  $\gamma \in [0,1]$   
If  $\gamma < \text{mutation probability}$  [Typical value of the mutation probability range between 0.001 and 0.03]  
The apply mutation, otherwise do not apply it, and just copy the chromosome to the new population.

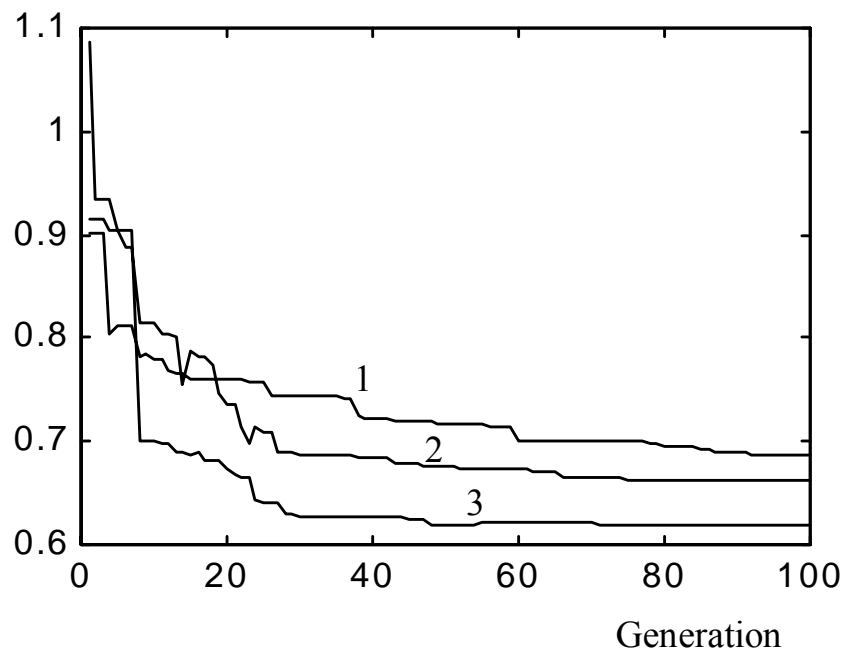
### **5.6 The overall action of the simple Genetic Algorithms**



**GENERATION**

The above shown generation is repeated for a pre-specified number of times (Maximum number of generation). The highest obtained fitness in each population is found to increase exponentially with generations. Typical curves of Objective Function vs generations for genetic algorithms are exemplified by curves below, the curves of fitness with the number of generation can be drawn the same way.

Objective  
Function



Convergence curves