Genetic Algorithm

Genetic Algorithm is a type of local search that mimics evolution by taking a population of strings which encode possible solutions and combines them based on a fitness function to produce individuals that are more fit.

Genetic Algorithms are computer programs that evolve in ways that resemble natural selection and can be applied to solve complex problems. Genetic Algorithms are inspired by **Darwin's theory** about evolution.

Genetic algorithms are a type of optimization algorithm, meaning they are used to find the optimal solution(s) to a given computational problem that maximizes or minimizes a particular function.

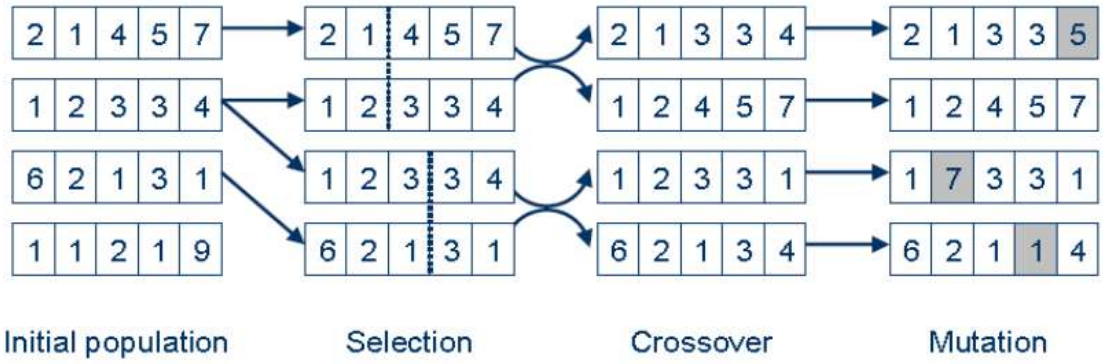
A genetic algorithm (or GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems. GAs are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called **recombination**).

The evolution usually starts from a **population of randomly generated individuals and happens in generations.**

In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness) and modified to form a new population. The new population is used in the next iteration of the algorithm

The **algorithm terminates** when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Example:



Individual - Any possible solution

Population - Group of all individuals

Fitness – Target function that we are optimizing (each individual has a fitness)

Selection

Selection is the component which guides the algorithm to the solution by preferring individuals with high fitness over low-fitted ones. It can be a deterministic operation, but in most implementations, it has random components.

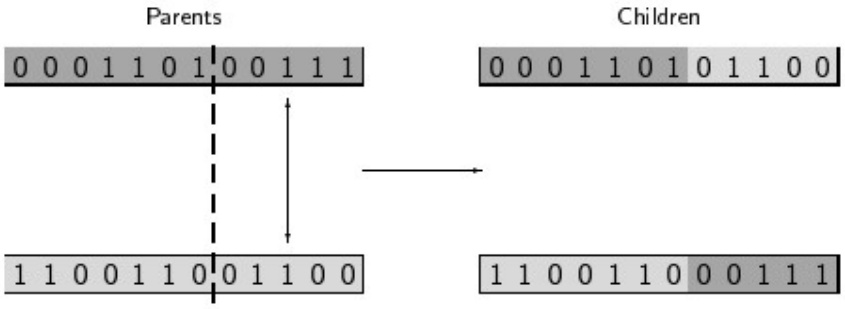
Crossover

In sexual reproduction, as it appears in the real world, the genetic material of the two parents is mixed when the gametes of the parents merge. Usually, chromosomes are randomly split and merged, with the consequence that some genes of a child come from one parent while others come from the other parents

This mechanism is called crossover. It is a very powerful tool for introducing new genetic material and maintaining genetic diversity, but with the outstanding property that good parents also produce well-performing children or even better ones.

Basically, crossover is the exchange of genes between the chromosomes of the two parents. In the simplest case, we can realize this process by cutting two strings at a randomly chosen position and swapping the two tails. This process,

which we will call one-point crossover in the following, is visualized in Figure below.

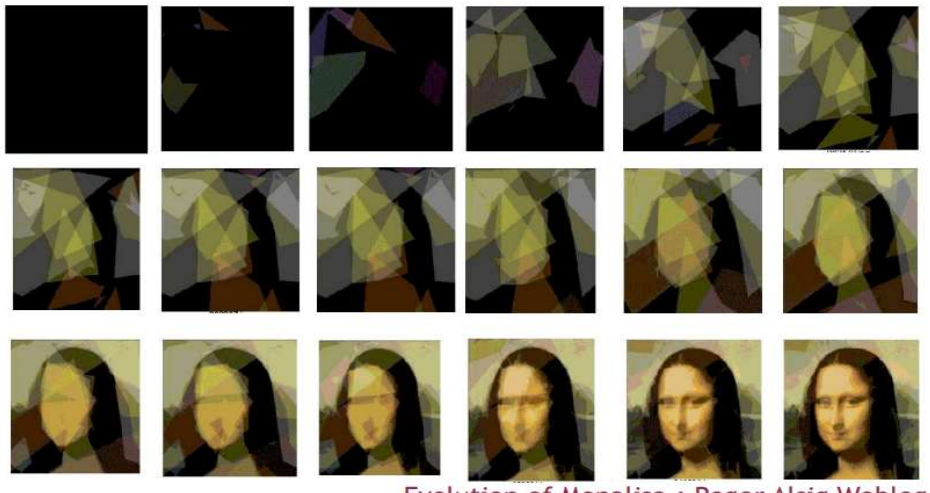


Mutation

The last ingredient of our simple genetic algorithm is mutation—the random deformation of the genetic information of an individual by means of radioactive radiation or other environmental influences. In real reproduction, the probability that a certain gene is mutated is almost equal for all genes

GA applications in real world:

- Artificial Intelligence
- Automotive Design
- Computer Gaming
- Predicting Protein Structure
- Optimization Problems
- Music
- Business



Evolution of Monalisa : Roger Alsig Weblog

The basic components common to almost all genetic algorithms are:

- a fitness function for optimization
- a population of chromosomes
- selection of which chromosomes will reproduce
- crossover to produce next generation of chromosomes
- random mutation of chromosomes in new generation

The **fitness function** is the function that the algorithm is trying to optimize. The term **chromosome** refers to a numerical value or values that represent a candidate solution to the problem that the genetic algorithm is trying to solve.

A genetic algorithm begins with a randomly chosen assortment of chromosomes, which serves as the first generation (initial population). Then each chromosome in the population is evaluated by the fitness function to test how well it solves the problem at hand.

Now the selection operator chooses some of the chromosomes for reproduction based on a probability distribution defined by the user. The fitter a chromosome is, the more likely it is to be selected. For example, if f is a non-negative fitness function, then the probability that chromosome $f(x)$ is chosen to reproduce might be

$$P = \frac{f(x)}{\sum f(x)}$$

The crossover operator resembles the biological crossing over and recombination of chromosomes in cell meiosis. This operator swaps a subsequence of two of the chosen chromosomes to create two offspring

The mutation operator randomly flips individual bits in the new chromosomes (turning a 0 into a 1 and vice versa).

Example: Maximizing a Function of One Variable

Consider the problem of maximizing the function

$$f(x) = \frac{-x^2}{10} + 3x$$

where ***x*** is allowed to vary between 0 and 31. To solve this using a genetic algorithm, we must encode the possible values of x as chromosomes.

For this example, we will encode ***x*** as a binary integer of length 5. Thus, the chromosomes for our genetic algorithm will be sequences of 0's and 1's with a length of 5 bits and have a range from 0 (00000) to 31 (11111).

To begin the algorithm, we select **an initial population of 10 chromosomes** at random. The resulting initial population of chromosomes is shown in Table 1. Next, we take the x-value that each chromosome represents and test its **fitness** with the fitness function. The resulting fitness values are recorded in the third column of Table 1.

Table 1: Initial Population

Chromosome Number	Initial Population	x Value	Fitness Value $f(x)$	Selection Probability
1	01011	11	20.9	0.1416
2	11010	26	10.4	0.0705
3	00010	2	5.6	0.0379
4	01110	14	22.4	0.1518
5	01100	12	21.6	0.1463
6	11110	30	0	0
7	10110	22	17.6	0.1192
8	01001	9	18.9	0.1280
9	00011	3	8.1	0.0549
10	10001	17	22.1	0.1497
		Sum	147.6	
		Average	14.76	
		Max	22.4	

Since our population has 10 chromosomes and each '**mating**' produces 2 offspring, we need 5 matings to produce a new generation of 10 chromosomes. The selected chromosomes are displayed in Table 2. To create their **offspring**, a

crossover point is chosen at random, which is shown in the table as a vertical line.

Table 2: Reproduction & Second Generation

Chromosome Number	Mating Pairs	New Population	x Value	Fitness Value $f(x)$
5	01|100	01010	10	20
2	11|010	11100	28	5.6
4	0111|0	01111	15	22.5
8	0100|1	01000	8	17.6
9	0001|1	01010	10	20
2	1101|0	11011	27	8.1
7	10110	10110	22	17.6
4	01110	01110	14	22.4
10	100|01	10001	17	22.1
8	010|01	01001	9	18.9
			Sum	174.8
			Average	17.48
			Max	22.5

Example:

Consider the problem of finding the global maximum of the following function:

$$f_1: \quad \{0,\ldots,31\} \longrightarrow \mathbb{R}, \quad x \longmapsto x^2$$

Of course, the solution is obvious, but the simplicity of this problem allows us to compute some steps by hand in order to gain some insight into the principles behind genetic algorithms.

let us assume that a population size of m = 4, we obtain the following in the first step:

Individual No.	String (genotype)	x value (phenotype)	$f(x)$ x^2	pselect$_i$ $\frac{f_i}{\sum f_j}$
1	0 1 1 0 1	13	169	0.14
2	1 1 0 0 0	24	576	0.49
3	0 1 0 0 0	8	64	0.06
4	1 0 0 1 1	19	361	0.31

One can compute easily that the sum of fitness values is **1170**, where the average is **293** and the maximum is **576**. We see from the last column in which way proportional selection favours high-fitted individuals (such as no. 2) over low-fitted ones (such as no. 3)

From the results, that individuals **no. 1 and no. 4** are selected for the new generation, while no. 3 dies and **no. 2 is selected twice**, and we obtain the second generation as follows:

Set of selected individuals	Crossover site (random)	New population	x value	$f(x)$ x^2
0 1 1 0|1	4	0 1 1 0 0	12	144
1 1 0 0|0	4	1 1 0 0 1	25	625
1 1|0 0 0	2	1 1 0 1 1	27	729
1 0|0 1 1	2	1 0 0 0 0	16	256

So, we obtain a new generation with a sum of fitness values of **1754**, an average of **439**, and a maximum of **729**. We can see from this very basic example in which way selection favours high-fitted individuals and how crossover of two parents

can produce an **offspring** which is **even better than both of its parents**. **It is left to the reader as an exercise to continue this example.**

Example: the MAX - ONE problem

Suppose we want to **maximize the number of ones** in a string of l binary digits start with a population of ***n*** random strings. Suppose that l = 10 and n = 6

initial population:

$$s_1 = 1111010101 \quad f(s_1) = 7$$
$$s_2 = 0111000101 \quad f(s_2) = 5$$
$$s_3 = 1110110101 \quad f(s_3) = 7$$
$$s_4 = 0100010011 \quad f(s_4) = 4$$
$$s_5 = 1110111101 \quad f(s_5) = 8$$
$$s_6 = 0100110000 \quad f(s_6) = 3$$

Individual *i* will have a probability to be chosen $\frac{f(i)}{\sum_i f(i)}$

Suppose that, after performing selection, we get the following population:

$$s_1` = 1111010101 \ (s_1)$$
$$s_2` = 1110110101 \ (s_3)$$
$$s_3` = 1110111101 \ (s_5)$$
$$s_4` = 0111000101 \ (s_2)$$
$$s_5` = 0100010011 \ (s_4)$$
$$s_6` = 1110111101 \ (s_5)$$

Step 2: crossover

Before crossover:

s_1` = 1111010101 $s_2$` = 1110110101

After crossover:

s_1`` = 1110110101 $s_2$`` = 1111010101

Step 3: mutations

The final step is to apply random mutations: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)

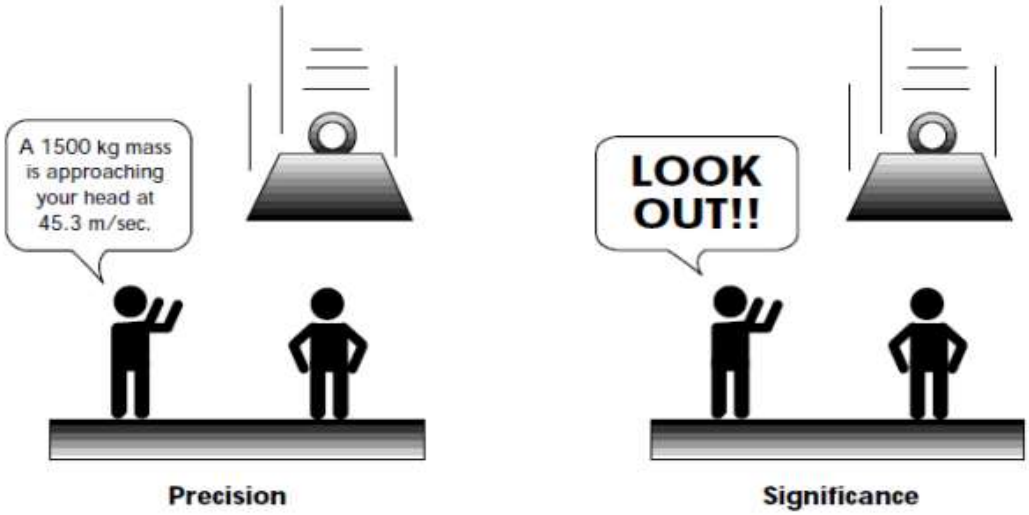
Initial strings	After mutating
s_1`` = 1110110101	s_1```` = 1110100101
s_2`` = 1111010101	s_2```` = 1111110100
s_3`` = 1110111101	s_3```` = 1110101111
s_4`` = 0111000101	s_4```` = 0111000101
s_5`` = 0100011101	s_5```` = 0100011101
s_6`` = 1110110011	s_6```` = 1110110001

And now, **iterate** … In one generation, the total population fitness changed from **34** to **37**, thus improved by roughly 9%. At this point, we go through the same process all over again, until a stopping criterion is met

GA Applications

Domain	Application Type
Control	Gas pipeline, missile evasion
Design	Aircraft design, keyboard configuration, communication networks
Game playing	Poker, checkers
Security	Encryption and Decryption
Robotics	Trajectory planning

Introduction to fuzzy logic



Fuzzy logic is an extension of Boolean logic by Lotfi Zadeh in 1965 based on the mathematical theory of fuzzy sets, which is a generalization of the classical set theory.

By introducing the notion of degree in the verification of a condition, thus enabling a condition to be in a state other than true or false, fuzzy logic provides a very valuable flexibility for reasoning, which makes it possible to take into account **inaccuracies** and **uncertainties**. One advantage of fuzzy logic in order to formalize human reasoning is that the rules are set in natural language. For example, here are some rules of conduct that a driver follows, assuming that he does not want to lose his driver's licence:

If the light is red...	if my speed is high...	and if the light is close...	then I brake hard.
If the light is red...	if my speed is low...	and if the light is far...	then I maintain my speed.
If the light is orange...	if my speed is average...	and if the light is far...	then I brake gently.
If the light is green...	if my speed is low...	and if the light is close...	then I accelerate.