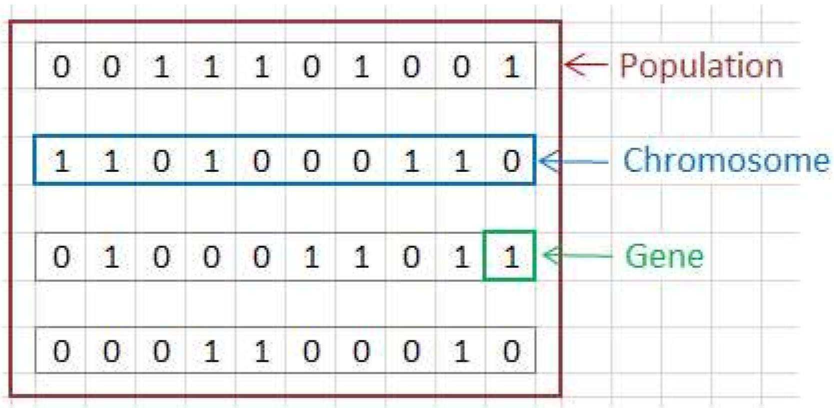
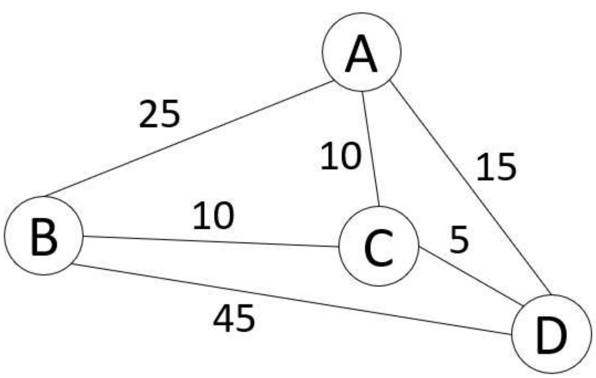
Genetic Algorithms (GA)

John Holland proposed genetic Algorithms in the 1970s. Initially, they were called "Reproductive Plans." These algorithms are maybe the most famous of the evolutive algorithms family.

The inspiration comes from the DNA structure, which is people's genetic code. All the information is stored in chromosomes that have a lot of genes. Holland's proposal consists of representing the solutions by binary arrays.



Example 1: Traveling salesman problem

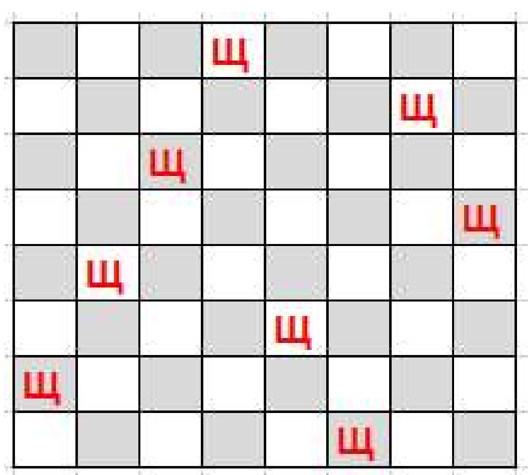


The problem consists of determining the path a salesperson must follow to minimize the distance. He/she starts and ends in the same node, and nodes cannot be repeated.

Example:

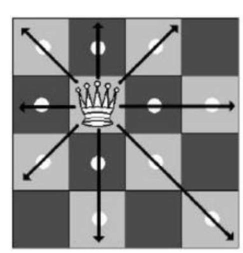
A – B – D – C – A = 25 + 45 + 5 + 10 = 85

Example 2: Traveling salesman problem



The problem consists of placing 8 queens on an 8x8 chessboard. The queens must not attack among them.

A queen attacks all the pieces that are in the same row, column, or diagonal.



wikipedia.org

Individuals' representation

The individuals' representation can be divided into genotype and phenotype:

- **Genotype**: it is the codified version of the solution
- **Phenotype**: it is the solution that represents an individual

We can use some of the following representations:

- **Binary representation**: It the Holland's original proposal. It consists of representing the solution using a binary array. It could be beneficial because a lot of problems can be represented using this technique.

- **Integer representation**: The representation of individuals as integer arrays is the best option for specific problems. For example: for evolving the trajectory of an agent using numbers for directions (left, right, up, and down).

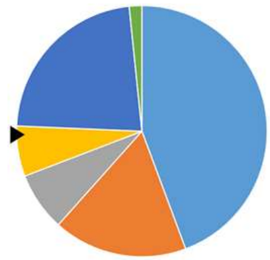
- **Real representation**: Many problems can be represented as real arrays, it is, $[x_1, x_2, \ldots, x_n]$ where $x_i \in \mathbb{R}$.

- **Permutation representation**: Some problems, such as sudoku or traveling agent, can be represented as a permutation of a set.

Selection of parents

There are some techniques for parents' selection:

Roulette selection, also called proportional selection, was the original proposal of Holland. The idea is simple. You can imagine a roulette where each section is assigned to an individual. If we have 10 individuals, the roulette is divided into 10 sections. The section size is proportioned to the individual's fitness.



The roulette's implementation can be as follows. Given an individual i, f_i represents its fitness, and, p_i is the individual's proportion, where it can be calculated as $p_i = \frac{f_i}{\sum_k f_k}$. The sum of all proportions must be 1. The segment between 0 and 1 is divided using those proportions for calculating ranges for each individual. Finally, to select an individual, a number is randomly calculated between 0 and 1, and the individual whose range corresponds to that number is selected as a parent.

Individual	Fitness f_i	Proportion p_i	Range
A	2	0.04	[0.00 – 0.04]
B	10	0.20	[0.04 – 0.24]
C	7	0.14	[0.24 – 0.38]
D	1	0.02	[0.38 – 0.40]
E	30	0.60	[0.40 – 1.00]

The roulette has some problems:

- If an individual has a fitness significantly bigger than others, the algorithm selects the same individual several times.
- If the individuals' fitness is similar among them, there is no pressure in the selection. In other words, the individuals will be selected as entirely random.

Tournament selection, may be the most known technique and easy to implement for parent selection. It consists of randomly choosing k individuals and selecting the fittest one. k represents the tournament size.

Advantages of the tournament:

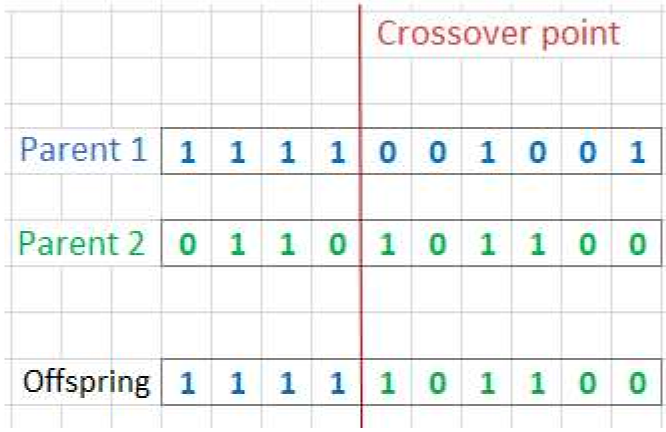
- It is too easy to implement.
- To change the selection's pressure, we can change the value of k. The bigger the value, the more the pressure.

Reproduction (crossover or recombination)

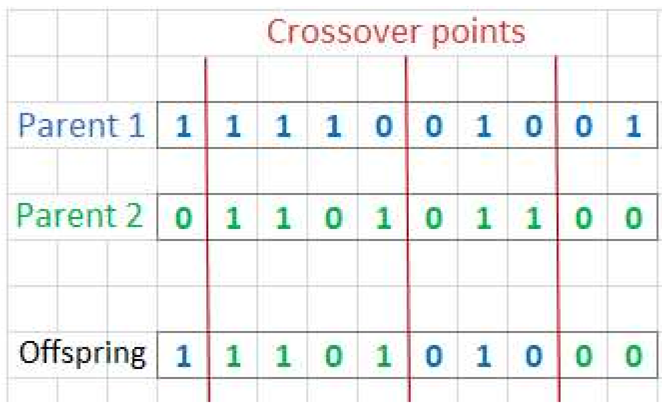
The goal of reproduction is to generate new individuals (offspring) combining the parents' properties. It can be implemented based on the individuals' representation.

Binary and integer representation

- **1 point crossover**: This technique divides the parents into two sections randomly choosing a crossover point, represented in the color red in the following figure. The offspring is created using the first section of the first parent and the second section of the second parent.



- **N point crossover**: It is a generalization of 1-point crossover. In this case, the parents are divided into several sections. The offspring are created using different sections of parents, as is shown in the following figure.



- **Uniform crossover**: This technique chooses one by one the elements of the new individual. For each gene, it copies the gene of the first or the second parent randomly.

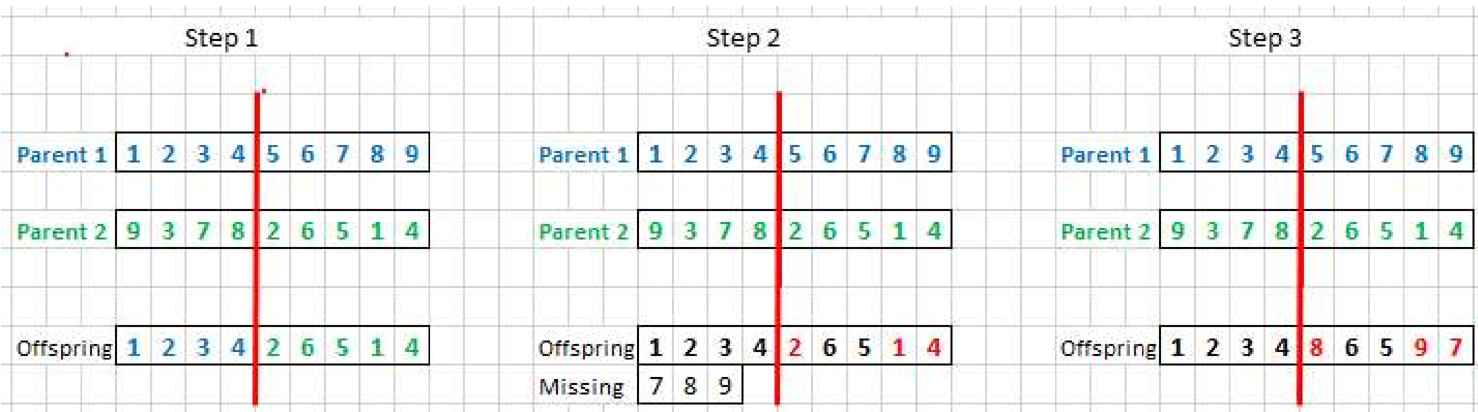
Real-valued representation

- **Discrete reproduction:** It is the same idea of a uniform crossover. For each element of the children, we randomly copy the value of parent 1 or parent 2.

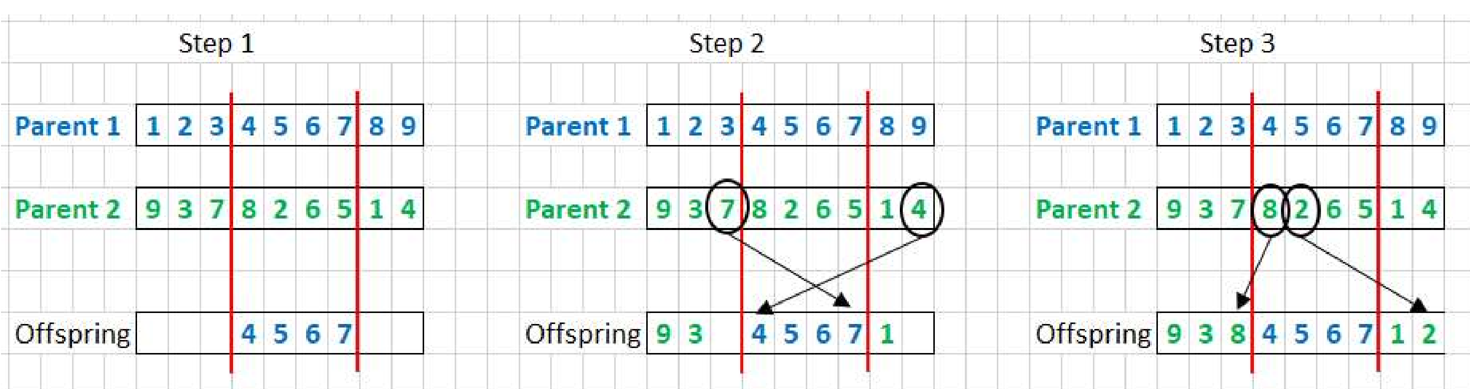
- **Asymmetric reproduction:** This technique consists of calculating the offspring o as the average of their parents p_1 and p_2. Formally, $o = \alpha p_1 + (1 - \alpha p_2)$, where α is a number between 0 and 1, generally, its value is 0.5.

Permutation representation

- **Simple permutation crossover:** the steps are described as follows:
  Step 1: Divide the individuals into two sections and copy the elements.
  Step 2: Calculate the repeated and the missing elements.
  Step 3: Randomly assign the missing values in the positions of the repeated elements.



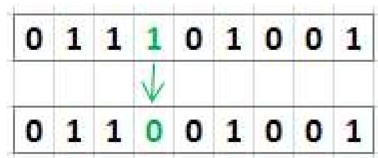
- **Partially mapped crossover:** the steps are described as follows:
  Step 1: Divide the individuals into 3 sections and copy the elements of the intermediate section of the first parent.
  Step 2: Copy the elements of the second parent (first and third sections) but ignore the elements that appear in the offspring.
  Step 3: Assign the missing values using the elements of the second part of the second parent that do not appear in the offspring.



Mutation

Mutation's goal is to modify individuals to explore the search space. Some of the techniques are the following:

- **Bitwise mutation** is used in binary representation and consists of randomly selecting one or several genes and changing their values.



- **Random resetting** is used in integer presentation and consists of randomly selecting one or several genes and resets its values.

- **Uniform mutation** is used in real-valued representation. It randomly selects one or several genes and chooses a random value between the minimum and maximum values.

- **Swap mutation** is used in permutation representation and consists of randomly selecting two elements and swapping their values.

Holland's original proposal of Genetic Algorithms was:

- Representation: binary
- Parents' selection: roulette
- Crossover: 1 point
- Mutation: bitwise
- Population model: generational

Implementation of the Genetic Algorithm phases

Parents' selection using roulette
Parameters: • Population of *N* individuals • Fitness Return: new population of *N* individuals Begin: Create the roulette, this is, calculate the range of each individual For each individual in the new population: Randomly calculate a number between 0 and 1 Copy in the new population the individual whose range corresponds to the random number Return the new population End

Crossover

Parameters:
• Population of *N* parents
• *Pr*: probability of reproduction
Return: new population of N individuals
 	
Begin:
For each individual in the new population:
If a random number between 0 and 1 is less than *Pr*:
Randomly select two parents and combine them
Add the offspring to the new population
Else
Randomly select one parent and clone it
Add the clone to the new population
End if
Return the new population
End

Mutation

Parameters:
• Population of *N* parents
• *Pm*: probability of mutation
Return: new population of N individuals
 	
Begin:
For each individual in the population:
If a random number between 0 and 1 is less than *Pm*:
Mutate the individual
Add the mutated individual to the new population
Else
Clone the individual and put it in the new population
End if
Return the new population
End

Elite individual

If in the new population, we find an individual better than the elite:
    Replace the elite with the fittest individual
Else:
    Select an individual from the population with negative tournament
    Replace the selected individual with the elite

Genetic Algorithm

Parameters:
    N, population size
    G, Maximum number of generations
    Pr, Reproduction's probability
    Pm, Mutation's probability
Return: the elite individual
 	
Begin
  Create the initial population
  Calculate the population fitness
  Get the elite
  While the number of generations is less than G or we haven't found a good solution
      Select the parents
      Apply crossover
      Apply mutation
      Calculate the population fitness
      Get the elite or include the elite in the population
  End while
End

Recommended values: N = 30, G = 100, Pr = 0.8, Pm = 0.3

Population models

We can distinguish between two evolution models:

- **Steady-state model**: In each iteration, a few number of individuals (mostly only one) are created using crossover and mutation, and those individuals replace other individuals from the population. For selecting the individual that will be replaced, it can be used a negative tournament, it is, randomly choosing several individuals and remove the one with worse fitness.

- **Generational model**: Mostly used in Genetic Algorithms. It randomly creates an initial population of N individuals. Each generation, the whole population is modified applying the following process. First, the parents are selected based on their fitness. Those parents are using for creating the offspring using the crossover and mutation algorithms. To warranty the algorithm convergence, we need to store the elite individual, the fittest individual. If in the new generation we found an individual better than the elite, we replace the elite. If not, we include the elite in the generation.