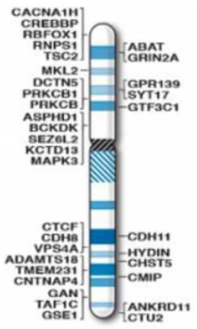
Genetic

Genetic algorithms are part of the evolutionist algorithms category. They make use of the evolution theory to solve problems.

These algorithms are "biosinspired" because they mimic living creatures' fitting in their environment for survival.

Genetic algorithms focus on the genetic material evolution inside a group of individuals. In each generation, individuals reproduce and share their genetic material. When applied to the population as a whole and followed on multiple generations, it is called genetic recombination.

Individuals, chromosoms and genes



A human has several chromosomes, each carrying thousands of genes. Likewise, in our model, each individual is to be represented by its genes, organised in chromosomes. For the sake of readability, we will use either "chromosome" or "individual" to refer to the geneti material of an individual.

In analogy with the evolution theory, these algorithms are based on the evolution of a population over time.

Overview

A genetic algorithm consist in the following steps:

- Creation of the base population
- Individual evaluation (is the individual fit to solve the problem ?)
- Individuals selection (the more fitting an individual, the more survival chances)
- Crossover / reproduction (Crossover of two individuals creates a new individual whose genes comes from both parents)
- Mutation (Some of the children's genes can mutate to create new genes)

The problem

Let's take a concrete example. This exercise consist in guessing a character string like `My password is hard !` or `IQlCqnWXVoVDDRFKFevaFzxmUxTxONwlLSwfkxmG`. Authorised characters are:

```
alphabet = string.ascii_letters + " !'."
```

There are 56 authorised characters. For a 100 characters long string, we would need $6.59.10^{+174}$ tries to test every possible solution!

Bruteforcing it would be inefficient. Instead, we will try to guess the solution from two pieces of information:

- The string length
- The fitting score of each solution (comprised between 0 and 1; The closer to 1, the better the solution is)

Creation of the base population

The first step is to create individuals for our base population.

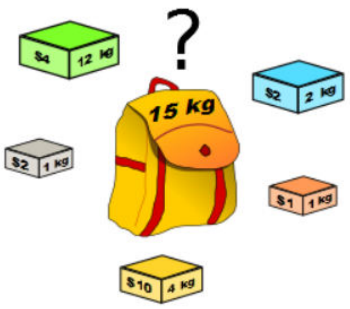
A chromosome is a set of genes. We could encode it in different ways:

- binary encoding: a binary string (set of 0s or 1s)
- multiple characters encoding: a string of characters

On the crossover step, we seek to have a wide genetic recombination. The binary encoding has a finer granularity for the crossing location but is not really natural and not fit for real data (for instance, modifying a certain bit in a floating number can result in invalid values).

In practice, we will use a different encoding regarding the problem to solve.

- The Knapsack problem:



> The Knapsack problem: What boxes do we choose in order to maximize the carried weight without exceeding 15kg?

`0110001`, each bit is refering to an object and indicates if it has been placed in the bag.

- Find a string of characters:

`"Aoljfon oaeznFjlf"`, each character in the string is to be found.

In the following exercise, you will code the function creating a new individual. The chromosome will be stored as a string.

In the following exercise, you will code the function creating a new individual. The chromosome will be stored as a string.

chromosome encoding

```
import random
from answer import alphabet

def get_letter():
    return random.choice(alphabet)

def create_chromosome(size):
    # TODO: Create a chromosome as a string of the right size
    string = ''.join([get_letter() for i in range(size)]) #my solution
    return string

```

Success!

Standard Error

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.012s

OK
```