CHAP I
Genetic algorithms

III. STATE OF THE ART

Genetic algorithms / milestones

1860: Charles Darwin: The Origin of Species by Means of Natural Selection, or the Struggle for Existence in Nature. (Genesis)

1966: Evolutionary programming by LJ Fogel. (1st inspiration)

1973: I. Rechenberg's evolution strategy. (1st simulation)

1975: John Holland, first formal model of genetic algorithms (the canonical genetic algorithm AGC) in his book: Adaptation in Natural and Artificial Systems. (1st model)

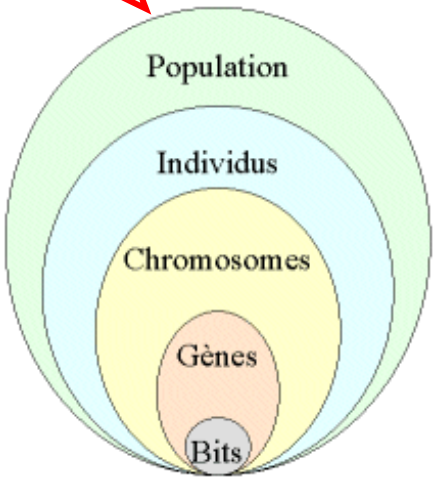
1989: David Goldberg publishes a popular book on genetic algorithms. (1st implementation)

1990s: Programming of a range of genetic algorithms transcribed in C++, called GAlib.

III. STATE OF THE ART

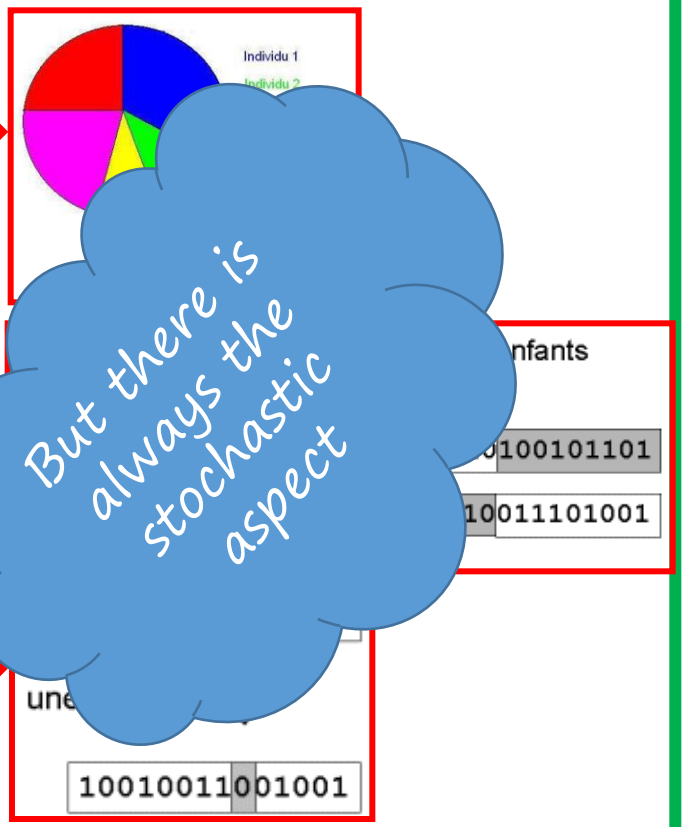
Genetic Algorithms / Terminology

AG = Evolution of Generations



Genetic operators

- Selection
- Crossing
- Mutation



But there is always the stochastic aspect

The "good" individuals survive, the less adapted ones disappear... Darwin

III. STATE OF THE ART

AG / Parameters to adjust:

Population size → depending on the problem and its size ;

Fitness function → correlated to the objective (conductor) ;

Crossing probability → from 0.6 to 0.8 ;

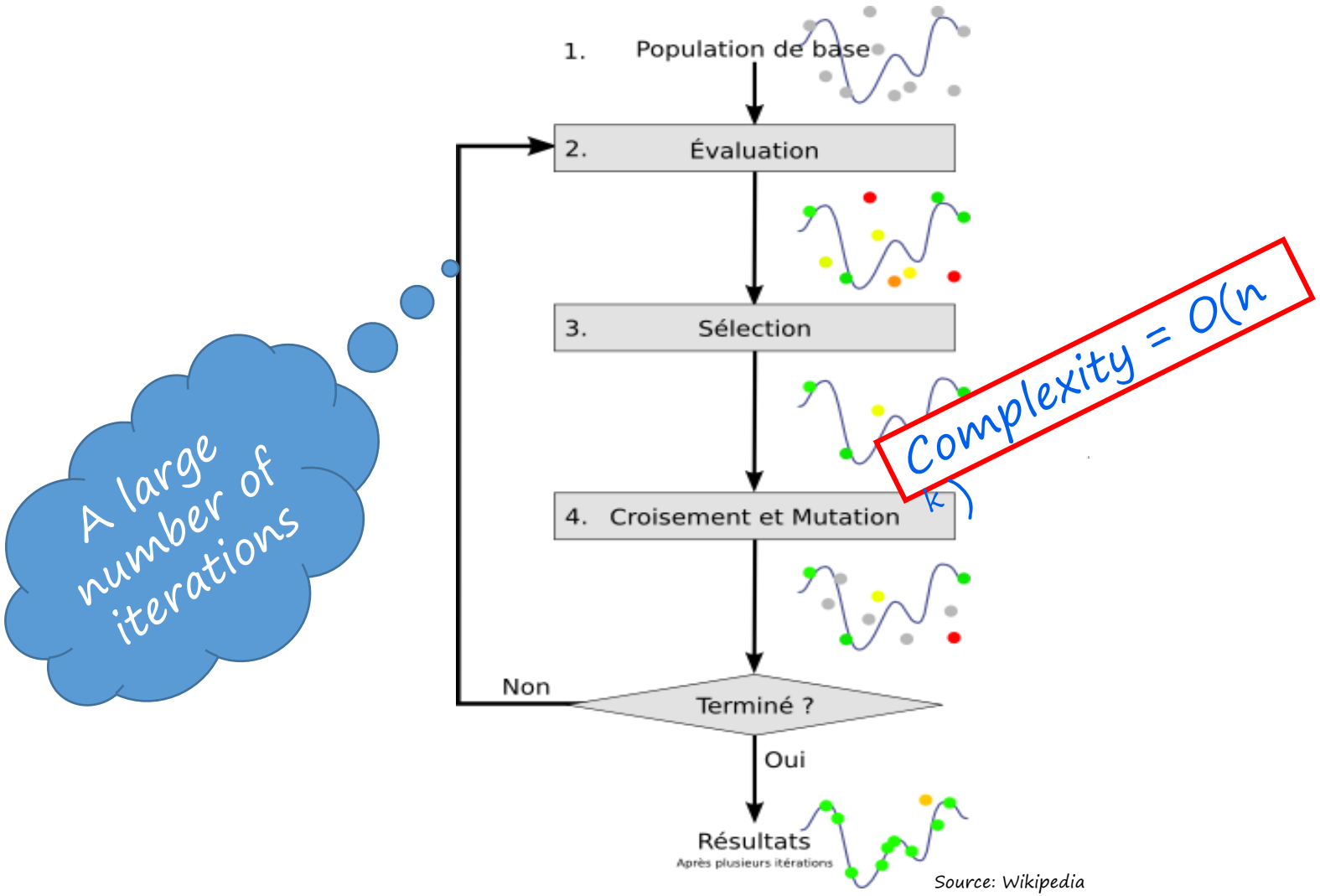
Prob. of mutation → from 0.01 to 0.05 ;

Stopping criterion → depending on the problem and its size ;

Elitism → minimal proportion of the population (double-edged) ;

III. STATE OF THE ART

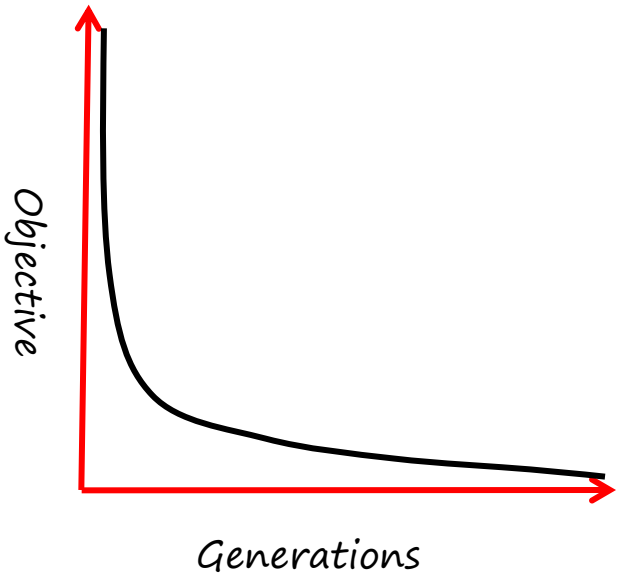
AG / General Algorithm



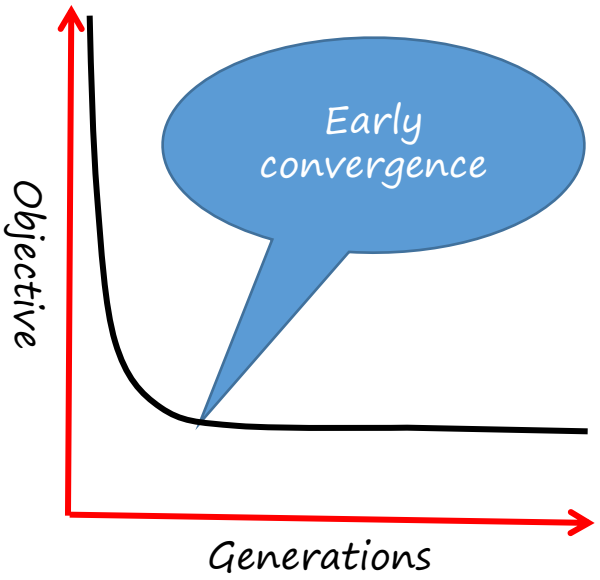
III. STATE OF THE ART

And if all goes well...

The ideal



Otherwis
e ...



Introduction

- Genetic algorithms (GA) are part of *evolutionary algorithms*.
- Suitable for finding a solution in a space characterized by *a large number of dimensions and local minima*

- Based on the simulation of the mechanisms of natural selection and genetics
  ↳ process of evolution and adaptation in the natural environment
- Stochastic Exploration Algorithms

Characteristics of AGs

- GAs are distinguished from classical (enumerative or gradient-based) methods of searching in a state space because they:
  - use a coding of the problem parameters
  - work on a population and not on a single situation
    - ↳ avoid the trap of a local minimum
  - use values of the function studied
    - ↳ not its derivative nor an auxiliary function
  - use probabilistic transition rules
    - ↳ not deterministic

Principles

- **Modeling** :
  - Search space states ⇒ symbol strings:

    Individuals or *chromosomes*
  - Set of individuals ⇒ The *population*
  - The population evolves during the resolution
- **Evolution** :
  - We measure the *adaptation* (" *fitness"*) of each individual to the search space

↳ adequacy as a solution

  - From one *generation* to the next, we seek to preserve the best adapted individuals in order to *reproduce them and apply genetic* operators to their descendants.

Assessment, calculation of adaptation (1)

- Depends on the problem.
- Possible examples:
  - image comparison number of similar pixels
  - control of a robot number of impacts against walls
  - classification number of well-classified examples
  - artificial life average amount of food ingested in a simulation
  - regression of sum function or variance of errors on a set of examples

Assessment, calculation of adaptation (2)

- Common adaptation functions:
  - sum of absolute errors between the calculated values and the expected output values, for each of the *fitness cases* :

$$\varphi = \sum_{i=1}^{n} \left| P(e_i) - s_i \right|$$

  - Sum of the squares of the deviations between the calculated value and the expected value ("squared error"):

$$\varphi = \sum_{i=1}^{n} \left(P(e_i) - s_i \right)^2$$

  - We also use variance, standard deviation, relative standard deviation
  - Often the adaptation function is normalized:

$$f_a = \frac{1}{1 + f_s}$$

f_s is zero in the ideal case, >0 otherwise

Assessment, calculation of adaptation (3)

- Example: symbolic/function regression
  - We are looking for a function with one input and one output satisfying the following table:

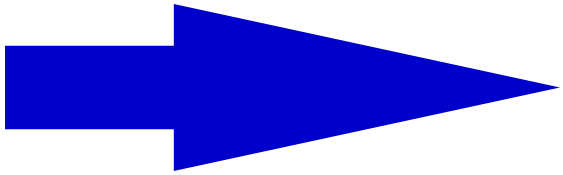
# cas	entrée: e_i	sortie: s_i
1	1	2
2	2	6
3	4	20
4	7	56
5	9	90

Each line represents a learning example or " *fitness case* ": for each input value, we know the output value that the program must approximate as best as possible; we will calculate the degree of adaptation of the latter as we go along.

Basic Algorithm

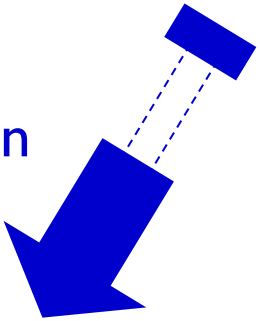
generation 0
(population
random)

genetic operations



generation 1

children kept in
function of their adaptation



generation *n* :
best individuals adapted =
best approximation to the solution of the problem

Algorithm

initialize the population (randomly generate a population of *N* *x* chromosomes)
calculate the degree of adaptation *f(x)* of each individual
As long as not finished or not convergent
 reproduction of parents
  select 2 individuals at a time
  apply genetic operators
 calculate the degree of adaptation *f(x)* of each child
 select survivors from parents and children
end As long as
conclude

can be subject to many variations

Genetic operators

The 3 most common:

- reproduction : the number of descendants of a chromosome follows its degree of adaptation
  - ↳ roulette , rank or tournament method
- crossover : 1-point or 2-point
- mutation

- The crossover and mutation rates applied during the genesis of a new population are parameters of the algorithm
- The mutation rate is generally low
- We generally work with a constant population.

Roulette Method

For each chromosome *i* we calculate its degree of adaptation f_i and we set:

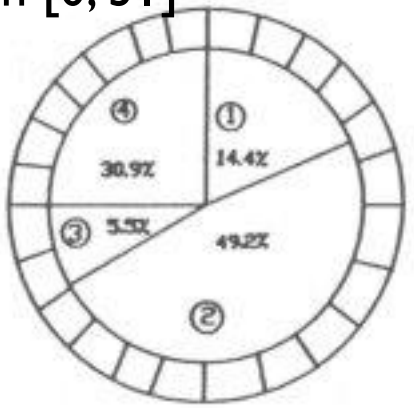
$$p_i = \frac{f_i}{\sum_i f_i} \times 100$$

We create a biased roulette where each *i* occupies a portion p_i.

To determine the descendants of a generation of size *n*, *n* draws are required

example : maximize $f(x) = x^2$ with n = 4 taken in [0, 31]

i	chain	f_i	% total
1	**01101**	169	14.4
2	**11000**	576	49.2
3	**01000**	64	5.5
4	**10011**	361	30.9
Total		1170	100

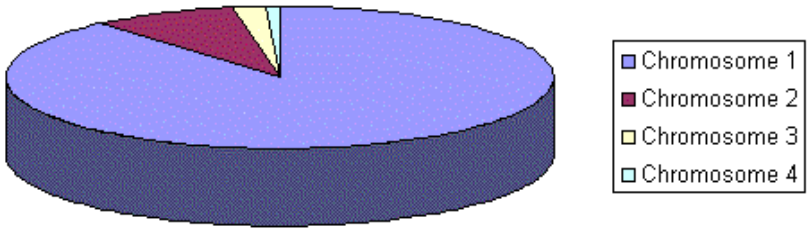


In a draw, chain 1 occupies 14.4% of the lottery wheel, there is a probability of 0.144 that we obtain a copy of this individual.

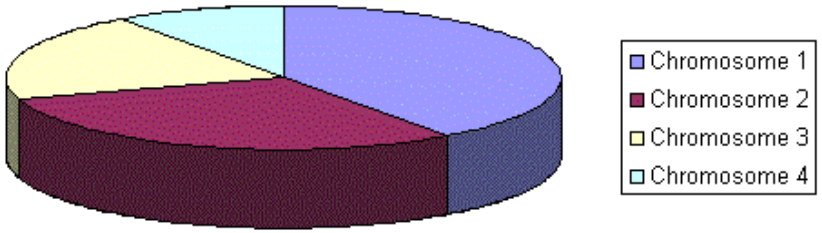
Rank method

- Parents are first sorted in order of fitness (*f value*), starting with the worst performers:

The lowest has a score of 1,
The second lowest has a score of 2
The third lowest has a score of 3
Etc.



- We divide each score by the sum of all scores:



- We then proceed as for roulette; the method gives more chance to the weakest chromosomes to be fished out

Tournament Method

- Starting from the population of *n* chromosomes, we form n pairs at random and determine the winner in each by its value of *f* . In the parameters of the GA, we determine a probability of victory of the strongest chromosome, representing its chance of being selected later. This probability must be large (between 70% and 100%). From the n pairs, we thus select n individuals for reproduction .

The crossing

Hope for improvement of new generations

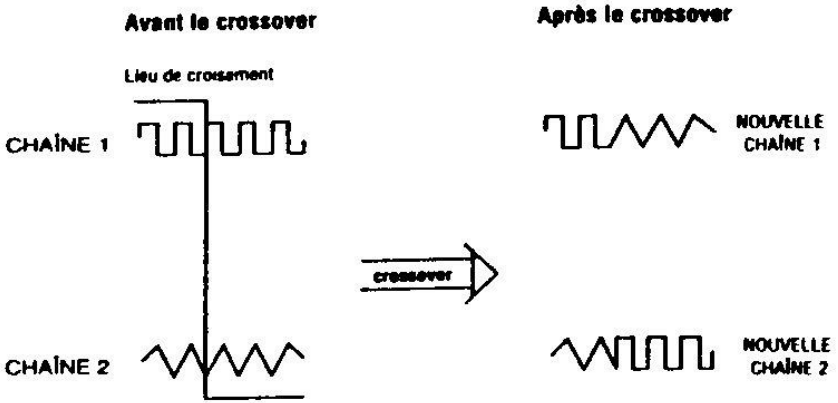
- 1 point intersection

  2 chromosomes of size *l* .

  We randomly choose an integer *k* between 1 and *l* -1.

  k represents the crossing point of the two chains

We create 2 new individuals by exchanging the characters of the initial strings between *k+* 1 and *l*



example : *l* = 5 and *k* = 3

C_1 = **0 1 1 | 0 1**
C_2 = **1 1 0 | 0 0**

$C_{1.2}$ = **0 1 1 | 0 0**
$C_{2.1}$ = **1 1 0 | 0 1**

The crossing

- 2-point intersection

  2 chromosomes of size l .

  We consider that they each form a closed ring and we randomly choose 2 integers k_1 *and* k_2 between 1 and l -1.

  k_1 *and* k_2 represent the crossing points of the two chains

  We create 2 new individuals by exchanging the characters of the initial strings between $k_1 + 1$ and k_2

example : $l = 10$, $k_1 = 1$ *and* $k_2 = 8$

C_1 = **0 | 1 1 1 0 0 1 1 | 1 1**
C_2 = **1 | 1 1 0 0 0 0 0 | 0 0**

$C_{1.2}$ = **1 | 1 1 1 0 0 1 1 | 0 0**
$C_{2.1}$ = **0 | 1 1 1 0 0 1 1 | 1 1**

The mutation

Allows you to exit local minima

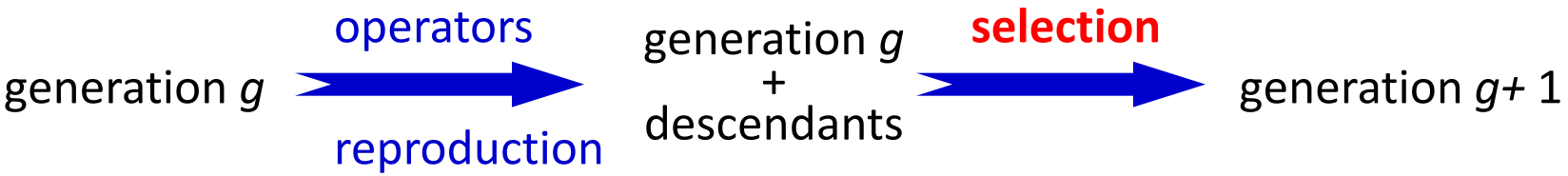
This is the random modification of the value of a character in the string.

For binary encoding, it simply consists of changing a 0 into a 1
(and vice versa)

The mutation rate is usually chosen to be very low (≈ 0.001)
↳ for each character of the descendants, probability of 1/1000 that it mutates

If the mutation plays a secondary role (due to the low rate), it allows the exploration of dimensions (possibly useful), abandoned (wrongly) by the selection process or absent from the initial population.

The selection



We work with a constant population

Several strategies:

- generational : all descendants replace all parents
- introduction of the " *generation gap* " = percentage of parents replaced

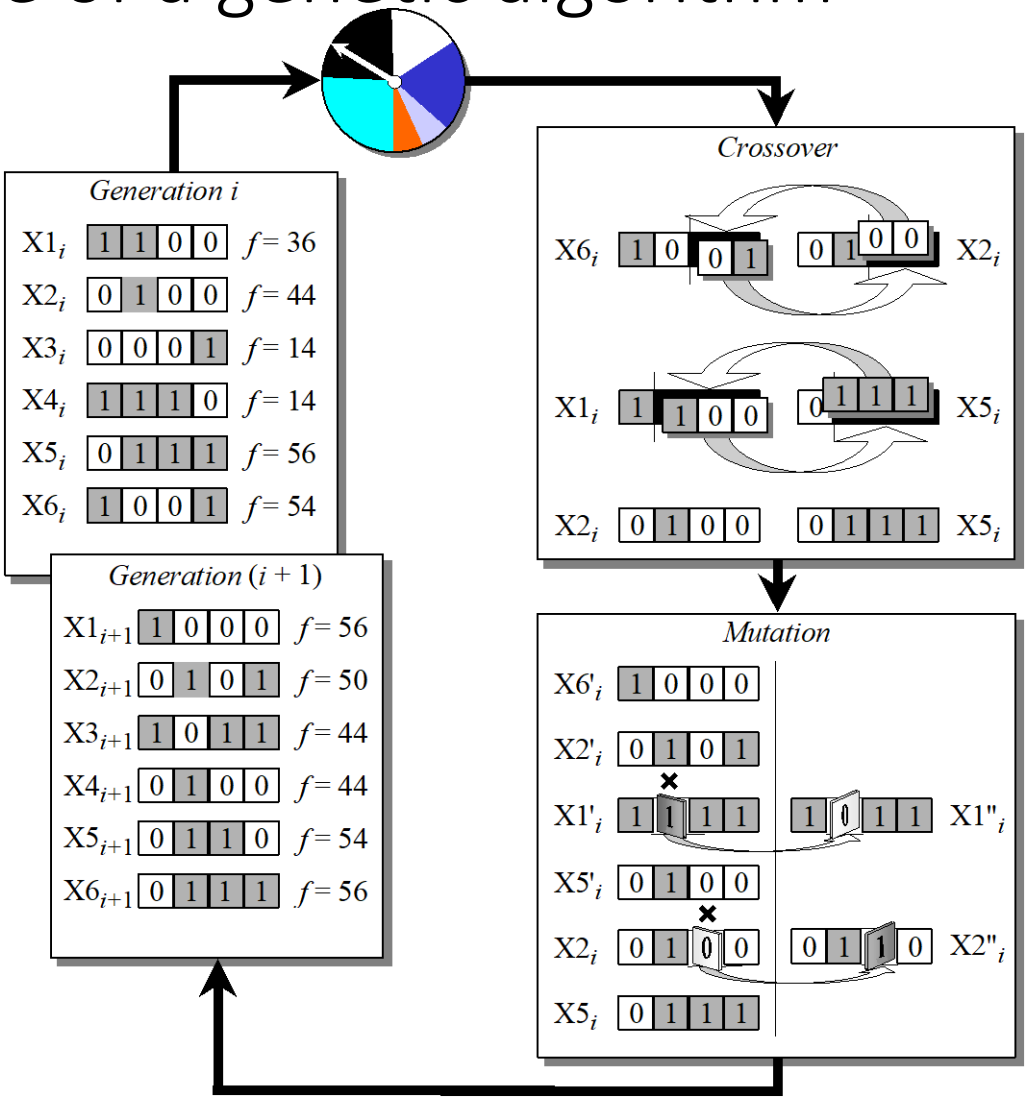
Which individuals should be kept?

It is natural to keep the best adapted ones:

- using biased roulette
- we strictly keep the best ones

- *k-elitist* strategy : we systematically keep the *k* best individuals from one generation to the next

Life cycle of a genetic algorithm



Initializing parameters

- Crossing
  - The probability varies from 0 to 100%:
    - 0% (no crossing) => perfect clones
    - 100% => no clones
- Mutation
  - Probability ranging from 0% to 100%
  - Normally low probability (~1/1000)
- Population size
  - Small size limits space exploration
  - Large size reduces convergence speed

Example of a variant of the algorithm

n = population size *c* = outcrossing rate
l = number of characters of each individual *m* = mutation rate

initialize a population (*n* individuals)
calculate the degree of adaptation of each individual
As long as not finished or not convergent
select the best *individuals , pair them and* perform a
cross to obtain the new individuals
each of the *l* characters of the new individuals mutates with a
probability *m*
keep only the *n* best individuals
calculate the degree of adaptation of each individual
end As long as
conclude

Detailed example

optimize $f(x) = x^2$ for x between 0 and 31 population of $n = 4$ individuals

Choice of parameter encoding: x coded in binary on $l = 5$ characters.

an individual $\in \{0,1\}^5$

degree of adaptation: we can use f directly in this case.

initial generation and pre-calculations

No.	chain	x	$f(x)$	p_i	expected $n \times p_i$	obtained
1	**0 1 1 0 1**	13	169	0.14	0.58	1
2	**1 1 0 0 0**	24	576	0.49	1.97	2
3	**0 1 0 0 0**	8	64	0.06	0.22	0
4	**1 0 0 1 1**	19	361	0.31	1.23	1
Total			1170	1	4	4
average			293			
max			576			

Detailed example

Crossing

individual	partner	position crossing	got	x	$f(x)$
0 1 1 0 | 1	2	4	**0 1 1 0 | 0**	12	144
1 1 0 0 | 0	1	4	**1 1 0 0 | 1**	25	625
1 1 | 0 0 0	4	2	**1 1 | 0 1 1**	27	729
1 0 | 0 1 1	3	2	**1 0 | 0 0 0**	16	256
Total					1754
Average					439
Max					729

Mutation

We try a mutation of 0.001 and nothing is changed.

Now we have to select the survivors and start again.

Why does it work?

a character of an individual = information

crossing = exchange of information between individuals = formation of new " points of view " (by destruction)

mutation = random dimension
- avoid forgetting features
- exit from a local minimum

reproduction And selection ⟹ promote information present in the best-adapted individuals

Stages of developing a genetic algorithm

1. Specification of the problem, definition of constraints and optimality criteria;
2. Encoding the problem domain in the form of a chromosome;
3. Definition of the adaptability function to evaluate the performance of the chromosome;
4. Definition of genetic operators;
5. Application of the algorithm and subsequent fine-tuning of the parameters.

Encoding

- Binary encoding (binary data)

  Chromosome A 101100101100101011100101

  Chromosome B 111111100000110000011111

  Backpack problem

- Permutation coding (integer data)

  Chromosome A 1 5 3 2 6 4 7 9 8

  Chromosome B 8 5 6 7 2 3 1 4 9

  Traveling Salesman Problem

- Value coding (any data type)

Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545

Chromosome B ABDJEIFJDHDIERJFDLDFLFEGT

Chromosome C (back),(back),(right),(forward),(left)

Calculating the weights of a network

- Coding by structure

Tree structure