

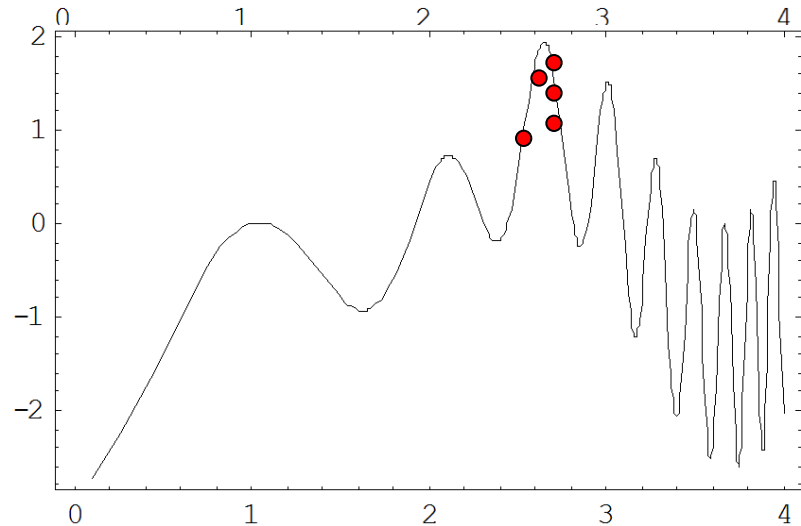
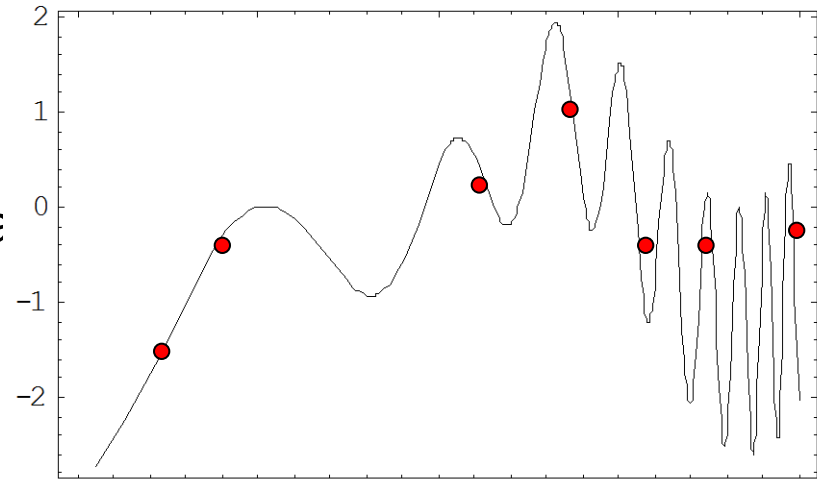
# Population based metaheuristics

- Particularities
- Classes of population based metaheuristics
- General structure
- Main components
- Evolutionary algorithms:
  - Encoding
  - Selection
  - Reproduction: crossover and mutation

# Population based metaheuristics

Problem solving =

- Search for the solution using a population of candidate solutions
- The search is guided by a function which „measures” the closeness to the solution

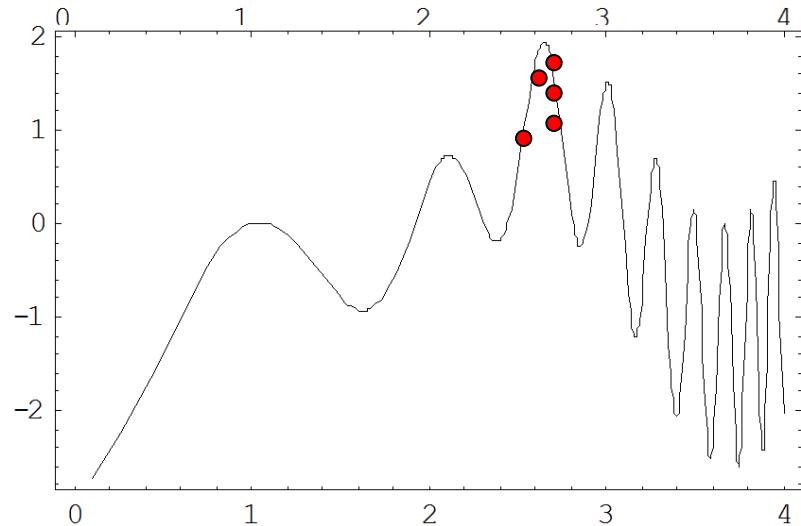
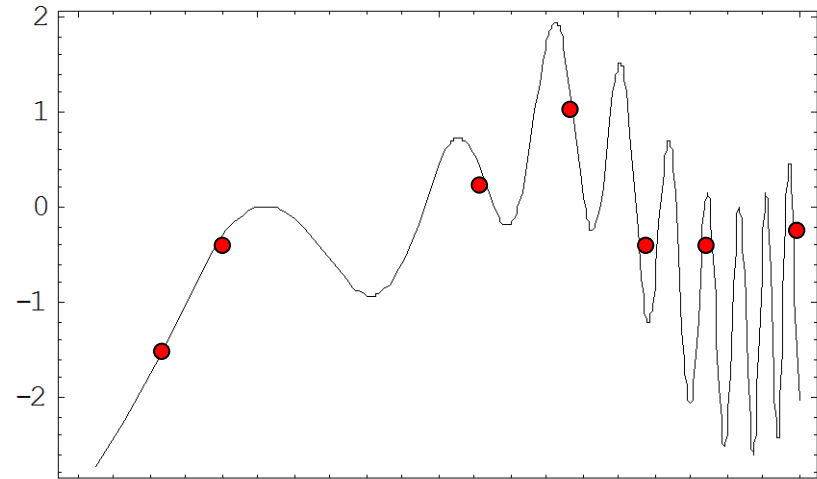


# Population based metaheuristics

There are two main search mechanisms:

- **exploration** = the search space is explored by the population elements which collect information about the problem (it involves **cooperation** between the population elements)
- **exploitation** = the information collected during exploration is exploited and the potential solution(s) is/are refined (it involves **competition** between the population elements)

**Remark:** There are many metaheuristics based on this paradigm



# Population based metaheuristics

- Evolutionary algorithms
  - Genetic Algorithms
  - Evolution Strategies
  - Evolutionary Programming
  - Genetic Programming
- Swarm intelligence algorithms
  - Particle Swarm Optimization
  - Ant Colony Optimization
  - Artificial Bee Colony
  - ... a large set of bio-inspired metaheuristics
- Other population based algorithms
  - Differential Evolution
  - Estimation of Distribution Algorithms

# General structure

Population initialization (random or based on some specific heuristics)

Population evaluation (computation of the objective function value(s))

REPEAT

    Construct a new population of candidate solutions

    Evaluate the candidate solutions

    Select the elements of a new population

UNTIL <stopping condition>

## Remark:

- The generation of new candidate solutions depend on the algorithm type
- At each cycle (generation) an entire new population is constructed and it competes with the previous population for „survival” (this is a so-called **generational** or **synchronous** updating strategy)

# General structure

**Variant:** the candidate solutions are analyzed and potentially assimilated into population just after their generation (**steady state** or **asynchronous** updating strategy)

**Population initialization:**  $(s_1, s_2, \dots, s_m)$

**Population evaluation** (computation of the objective function value(s))

REPEAT

  FOR  $i = 1:m$

    construct a new candidate solution  $(s'_i)$

    evaluate the new candidate solution

**decid** if the new candidate solution is accepted into the population

UNTIL <stopping condition>

**Remark:** a new candidate solution is usually accepted if it is better than the poorest element of the current population

# Evolutionary computing

**Evolutionary computing** = design and application of techniques inspired by natural evolution

**Inspiration:** evolution of species =

- The species evolve by the development of new characteristics during reproduction caused by crossover and random mutations
- In the evolutionary process the fittest individuals survive (those which are well adapted to the environment)

# Analogy: evolution - optimization

## Evolutionary process

Natural environment

Individual (chromosome)

Population of individuals

Fitness (degree of adaptation to the environment)

Selection

Reproduction (crossover and mutation)

## Problem solving

Information about the problem

Configuration (candidate solution)

Population of candidates

Measure of the solution quality

Exploitation mechanism

Exploration mechanism



# Evolutionary computing: basic notions

**Chromosome** = set of genes  
corresponding to an individual  
(potential solution for the  
problem)  $(1,0,0,1)$

**Population** = finite set of individuals  
(chromosomes, candidate  
solutions)  $\{(0,0,0,0), (0,0,1,1),$   
 $(1,0,0,1), (1,0,1,0)\}$

**Genotype** = the pool of all genes of  
an individual or population

**Phenotype** = the set of all features  
represented by a genotype  $\{0,3,9,10\}$

# Evolutionary computing: basic notions

**Fitness** = measure of the quality of an individual (with respect to the problem to be solved)

**Generation** = stage in the evolutionary process of a population (iteration in the search process)

**Reproduction** = generation of new individuals (offsprings) starting from the current population (parents) by

- crossover
- mutation

Ex: ONEMAX problem

= find the binary string which maximizes the number of ones

$(1,0,0,1) \rightarrow 2$

Crossover:

$(1,0,0,1) \rightarrow (1,0,1,1)$

$(0,0,1,1) \rightarrow (0,0,0,1)$

Mutation:

$(1,0,1,1) \rightarrow (1,1,1,1)$

# Evolutionary computing: applications

**Scheduling:** vehicle routing problems, timetabling, routing in telecommunication networks

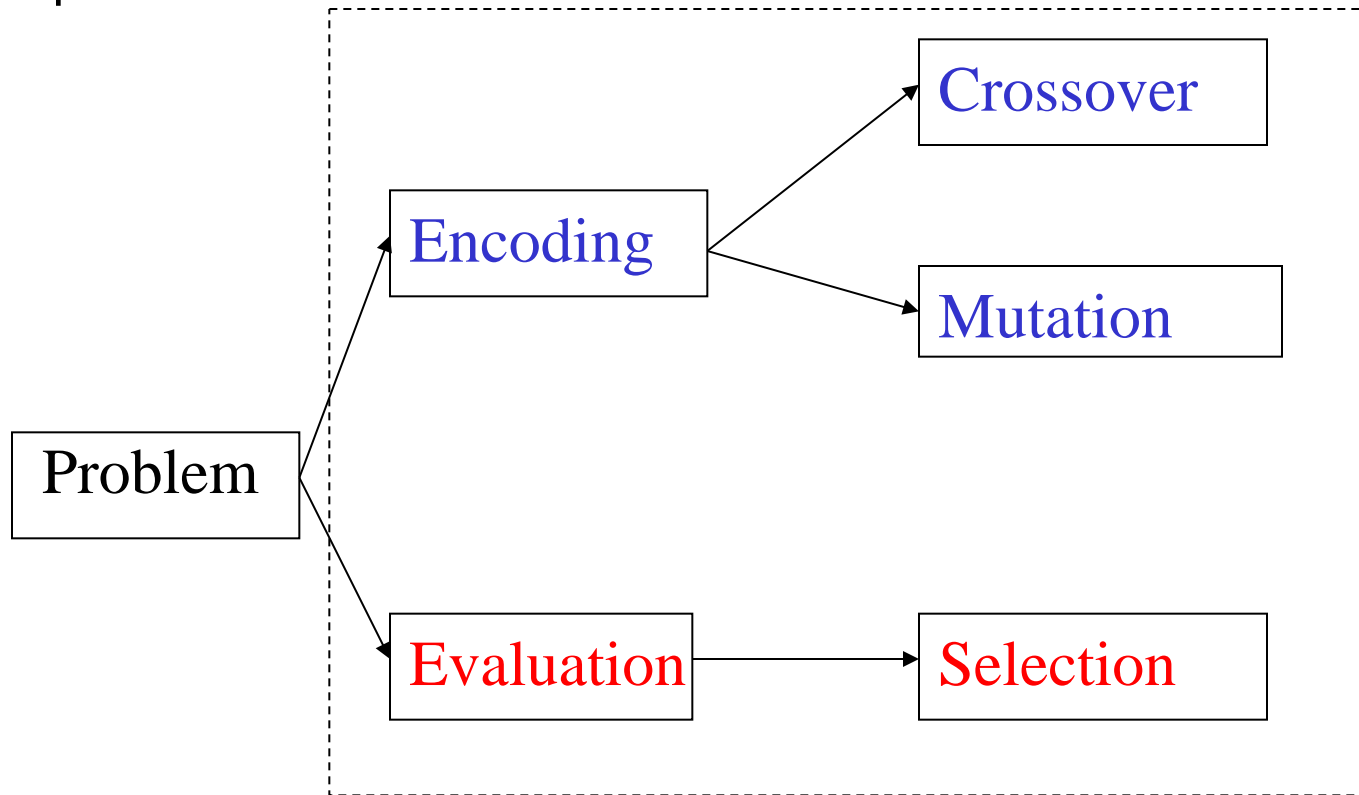
**Design:** digital circuits, filters, neural networks

**Modelling:** predictive models in economy, finances, medicine etc.

**Data mining:** design of classification systems in engineering, biology, medicine etc.

# Designing an Evolutionary Algorithm

Components:



# Structure of an EA

Population initialization

Population evaluation

REPEAT

    Parents selection

    Crossover

    Mutation

    Offspring evaluation

    Survivors selection

UNTIL <stopping condition>

# Directions in Evolutionary Computing

**Genetic Algorithms** (Holland, 1962-1967):

**Encoding:** binary

**Crossover:** main operator

**Mutation:** secondary operator

**Applications:** combinatorial optimization

**Genetic Programming** (Koza, 1990):

**Encoding:** tree-like structures

**Crossover:** main operator

**Mutation:** secondary operator

**Applications:** programs evolution

**Evolution Strategies**

(Rechenberg, Schwefel 1965):

**Encoding:** real

**Mutation:** main operator

**Recombination:** secondary operator

**Applications:** continuous

optimization

**Evolutionary Programming** (L. Fogel, D. Fogel, 1960-1970):

**Encoding:** real / state digrams

**Mutation:** the only operator

**Applications:** continuous optimization

# Encoding

Is a key element when a genetic algorithm is designed

The encoding method is related to the problem to be solved

## Variants:

- Binary encoding (the classical variant for GA)
- Real encoding (appropriate for continuous optimization)
- Specific encoding (e.g. permutation, tree, graph etc)

# Binary encoding

Chromosome = binary sequence

Search space:  $\{0,1\}^n$ ,  $n$  is given by the problem size

Examples:

1. **ONEMAX:** find the binary sequence  $(x_1, \dots, x_n)$  which maximizes the function  $f(x_1, \dots, x_n) = x_1 + \dots + x_n$
2. **Knapsack:** there is a set of  $n$  objects of weights  $(w_1, \dots, w_n)$  and values  $(v_1, \dots, v_n)$  and a knapsack of capacity  $C$ ; find a subset of objects which can be included in the knapsack without overpassing its capacity and such that the total value of selected objects is maximal

Encoding:  $(s_1, \dots, s_n)$

$s_i=0$  object  $i$  is not selected

$s_i=1$  object  $i$  is selected



# Binary encoding

## 3. Optimization of a function defined on a continuous domain.

$$f: [a_1, b_1] \times \dots \times [a_n, b_n] \rightarrow \mathbb{R}$$

$$X = (x_1, \dots, x_n) \rightarrow V = (v_1, \dots, v_n) \rightarrow U = (u_1, \dots, u_n) \\ \rightarrow Y = (y_1, \dots, y_r, y_{r+1}, \dots, y_{2r}, \dots, y_{nr})$$

$$v_i = (x_i - a_i) / (b_i - a_i) \quad (v_i \text{ belongs to } [0, 1])$$

$$u_i = [v_i * (2^r - 1)] \quad (u_i \text{ is a natural number from } \{0, \dots, 2^r - 1\} \Rightarrow \text{it can be represented in base 2 on } r \text{ positions})$$

$$(y_{r(i-1)+1}, \dots, y_{ri}) = \text{binary representation of } u_i$$

# Binary encoding

**Remark.** The binary encoding has the disadvantage that close values can correspond to binary sequences with a large Hamming distance (ex.  $7=(0111)_2$ ,  $8=(1000)_2$ )

**Solution:** use of the **Gray** code (successive values have binary sequences which are different in only one position)

$$(b_1, \dots, b_r) \rightarrow (g_1, \dots, g_r)$$

$$g_1 = b_1$$

$$g_i = (b_{i-1} + b_i) \bmod 2$$

# Binary encoding

Gray code:

$(b_1, \dots, b_r) \rightarrow (g_1, \dots, g_r)$

$g_1 = b_1$

$g_i = (b_{i-1} + b_i) \bmod 2$

Decoding:

$b_j = (g_1 + \dots + g_j) \bmod 2$

Nr.	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

# Particular encoding

It is specific to the problem to be solved

**Example:** permutation-like encoding

$$(s_1, s_2, \dots, s_n), \quad s_i \text{ in } \{1, \dots, n\}, s_i \neq s_j \text{ for all } i \neq j$$

**Problem:** TSP

$s_i$  = index of the town visited at step  $i$

**Remarks.** It ensures the fact that the constraints are satisfied.

# Evaluation of the population elements

## Fitness

- measures the quality of an individual
- as the value of the fitness is larger the probability of the element to survive is larger

## Problem: unconstrained optimization

The fitness function is proportional with the objective function (for a maximization problem) and inverse proportional with the objective function (for a minimization problem)

## Problem: constrained optimization

The fitness function depends both on the objective function and on the constraints

# Evaluation of the population elements

**Constraints:** included in the objective function by using the penalty method

$$\max_{x \in D} f(x)$$

$$g_i(x) = 0, \quad i = \overline{1, k_1}$$

$$h_i(x) \geq 0, \quad i = \overline{1, k_2}$$

$$F(x) = f(x) - a \sum_{i=1}^{k_1} g_i^2(x) - b \sum_{i=1}^{k_2} \varphi(h_i(x)), \quad a > 0, b > 0$$

$$\varphi(u) = \begin{cases} 0, & u \geq 0 \\ -u, & u < 0 \end{cases} \quad \begin{array}{l} \text{(no penalty if the constraint is satisfied)} \\ \text{(the penalty is larger if the constraint is} \\ \text{not satisfied)} \end{array}$$

# Evaluation of the population elements

Example: knapsack problem

$$\max_s \sum_{i=1}^n v_i s_i$$

$$C - \sum_{i=1}^n w_i s_i \geq 0$$

Fitness function

$$F(s) = \begin{cases} \sum_{i=1}^n v_i s_i, & \text{daca } \sum_{i=1}^n w_i s_i \leq C \\ a \sum_{i=1}^n v_i s_i - b \left( \sum_{i=1}^n w_i s_i - C \right), & \text{daca } \sum_{i=1}^n w_i s_i > C \end{cases}$$

$$a, b > 0, a + b = 1$$

**Rmk:** the weights  $a$  and  $b$  control the relative importance of the two components: objective function and constraints

# Selection

## Aim:

- decide which of the elements from the current populations will be used to construct offspring (parents selection)
- decide which of the elements from the offspring population will belong to the next generation (survivors selection)

## Basic idea:

- the elements with a high fitness have a higher chance to be selected

## Selection mechanisms:

- proportional selection
- rank based selection
- tournament selection
- truncation selection



# Proportional selection

Current population:  $P=(x^1, \dots, x^m)$

Steps:

Fitness values:

$$(F_1, \dots, F_m)$$

Selection probabilities:

$$p_i = F_i / (F_1 + \dots + F_m)$$

- a) Compute the selection probabilities
- b) Generate random values according to the distribution

$$\begin{array}{cccc} 1 & 2 & \dots & m \\ p_1 & p_2 & \dots & p_m \end{array}$$

Rmk. If  $F_i$  are not strictly larger than 0 than they can be changed as follows:

$$F'_i = F_i - \min(F_i) + \text{eps}$$

Implementation:

- (i) "Roulette Wheel"
- (ii) "Stochastic Universal Sampling" (SUS)

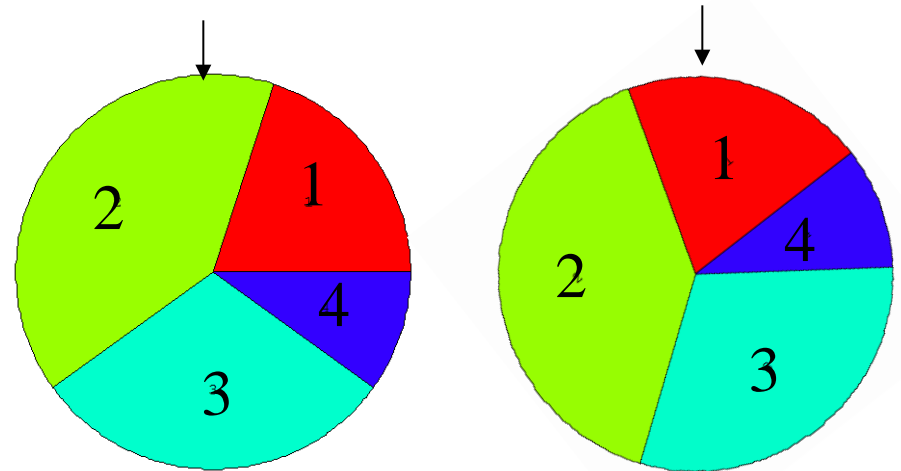
# Proportional Selection

## Roulette Wheel

- Let us consider a roulette divided in  $m$  sectors having areas proportional to the selection probabilities.
- Spin off the roulette and the index of the sector in front of the indicator gives the element to be selected from the population

Example:

1	2	3	4
0.2	0.4	0.3	0.1



# Proportional selection

## Implementation:

Ruleta ( $p[1..m]$ )

$i=1$

$s=p[1]$

$u=\text{random}(0,1)$

while  $s < u$  do

$i=i+1$

$s=s+p[i]$

endwhile

return  $i$

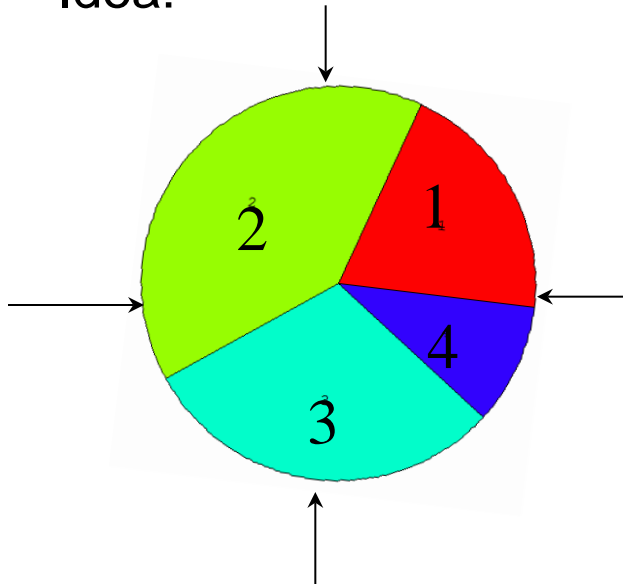
## Rmk.

1. This algorithm corresponds to the simulation of a random variable starting from its distribution
2. One run gives the index of one element
3. To simultaneously generate the indices of several elements, the SUS (Stochastic Universal Sampling) variant can be used

# Proportional selection

## Stochastic Universal Sampling

Idea:



**Rmk:**  $k$  represents the number of elements which should be selected

$c[1..m]$  will contain the number of copies of each element

Algorithm:

```
SUS( $p[1..m], k$ )  
   $u = \text{random}(0, 1/k)$   
   $s = 0$   
  for  $i = 1, m$  do  
     $c[i] = 0$   
     $s = s + p[i]$   
    while  $u < s$  do  
       $c[i] = c[i] + 1$   
       $u = u + 1/k$   
    endwhile  
  endfor  
  Return  $c[1..m]$ 
```

# Proportional selection

## Disadvantages:

1. If the objective function does not have positive values the fitness values should be transformed
2. If the difference between the fitness value of the best element and that of other elements is large then it is possible to fill the populations with copies of the best element (this would stop the evolutionary process)

# Rank based selection

## Particularities:

the selection probabilities are computed based on the rank of elements in an increasingly sorted list by fitness (in the case of a maximization problem)

## Steps:

1. increasingly sort the fitness values
2. each distinct value in the list will have a rank (the smallest fitness value corresponds to rank 1)
3. divide the population in classes of elements having the same rank (e.g. k classes)
4. compute the selection probabilities:  $P_i = i / (1 + 2 + \dots + k)$
5. select classes using the roulette wheel or SUS methods; randomly select an element from each class.

# Tournament selection

## Idea:

an element is selected based on the result of a comparison with other elements in the population

The following steps should be followed to select an element:

1. Randomly select  $k$  elements from the population
2. From the  $k$  selected elements choose the best one

## Remark.

1. Typical case:  $k=2$

# Truncated selection

## Idea:

from the joined population of parents and offspring the  $k$  best elements are selected

## Remark.

1. This is the only completely deterministic selection
2. It is mainly used for evolution strategies



# Properties of selection

## Elitism:

- A selection method has the property of elitism if the best element in population is always selected as survivor (it always remains in the population for the next generations)
- Truncation selection is the only method satisfying always this property
- In order to ensure the elitism the best element of the current population can be explicitly inserted in the new population (e.g. by replacing the worst element)

# Properties of selection

## Selection pressure:

- It measures the likelihood that the best element in the population will take over the population
- A strong selection pressure generates a high exploitation and reduces the exploration of the search space (it could lead to local optima or stagnation)
- In the case of probabilistic selection (e.g. proportional selection, tournament selection) the selection pressure can be measured by using the “takeover time” = number of generations needed to fill the entire population with copies of the best element if only selection would be applied

# Crossover

**Aim:** combine two or several elements in the population in order to obtain one or several offsprings

**Remark:** in genetic algorithms there are usually two parents generating two children

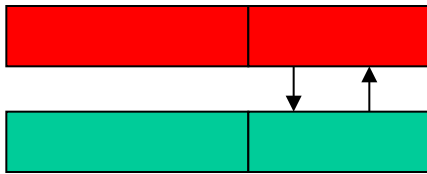
**Variants:**

- one cut-point
- uniform
- convex
- tailored for a given problem

# Cut-points crossover

One cut point

Parents

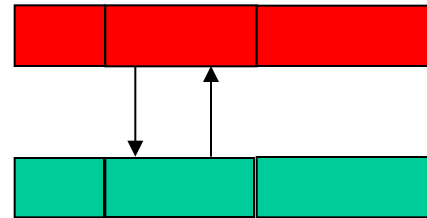


Children

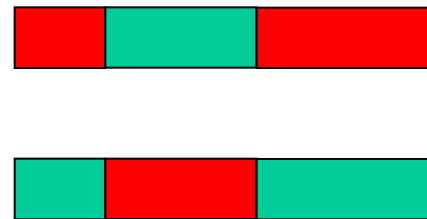


Two cut points

Parents



Children



# Cut-points crossover

## Remarks:

1. For each pair of selected parents the crossover is applied with a given probability ( $0.2 \leq P_c \leq 0.9$ )
2. The cut points are randomly selected
3. Numerical experiments suggest that two cut-points crossover leads to better results than one cut-point crossover

# Uniform crossover

Particularity : the genes of offspring are randomly selected from the genes of parents

Notations:  $x = (x_1, \dots, x_n)$ ,  $y = (y_1, \dots, y_n)$  – parents  
 $x' = (x'_1, \dots, x'_n)$ ,  $y' = (y'_1, \dots, y'_n)$  – offspring

$$x'_i = \begin{cases} x_i, & \text{with probability } p \\ y_i, & \text{with probability } 1 - p \end{cases}$$

$$y'_i = \begin{cases} y_i, & \text{if } x'_i = x_i \\ x_i, & \text{if } x'_i = y_i \end{cases}$$

# Convex crossover

**Specific:** is used for vectors of real values

**Notations:**  $x = (x_1, \dots, x_n)$ ,  $y = (y_1, \dots, y_n)$  – parents  
 $x' = (x'_1, \dots, x'_n)$ ,  $y' = (y'_1, \dots, y'_n)$  – offsprings

$$x'_i = ax_i + (1-a)y_i$$

$$y'_i = ay_i + (1-a)x_i$$

**Remark.**

- It can be extended in the case of an arbitrary number of parents - only one offspring is generated
- The coefficient  $a$  can be randomly selected from  $(0,1)$
- More general variants:
  - $A$  can be chosen from the interval  $(-p,p+1)$
  - A different random value can be chosen for each component

$$x'_i = a_i x_i + (1 - a_i) y_i$$

$$y'_i = b_i y_i + (1 - b_i) x_i$$

# Crossover for permutation – like elements

**Aim:** include heuristic schemes which are particular

**Example:** TSP (a tour is given by the order of visiting the towns)

Parents: A B C D E F G      Cutpoint: 3  
          A B E G D C F

Offspring: A B C E G D F  
           A B E C D F G



# Mutation

**Aim:** it allows to introduce new genes in the gene pool (which are not in the current genotype)

**Remark:** the mutation depends on the encoding variant

**Binary encoding:** the mutation consists of complementing some randomly selected genes

**Real encoding:** random perturbation

**Specific encoding:** perturbation based on particular heuristics

Variants:

1. Local (at chromosome level)
2. Global (at gene pool level)

# Mutation

## Chromosome level

### Steps:

1. Select chromosomes to be mutated (using a small mutation probability)
2. For each selected chromosome select a random gene which is mutated

### Remark:

The mutation probability is correlated with the population size (e.g.  $P_m=1/m$ )

# Mutation

## Pool gene level

**Assumptions:** all chromosomes are concatenated, thus they form a long binary sequence

**Mutation:** All genes are visited and for each one is decided (based on a mutation probability) if it is mutated or not

## Remark:

1. This variant allows to change several genes of the same chromosome

# Control parameters

The behavior of evolutionary algorithms is influenced by the parameters which control the structure of the population and the evolutionary mechanisms

## Control parameters:

- Population size
- Parameters involved in the stopping condition: maximal number of generations, maximal number of objective function evaluations, maximal number of stagnation steps etc
- Parameters involved in selection:
  - Sample size in the case of tournament selection
- Parameters involved in crossover:
  - Crossover probability
- Parameters involved in mutation:
  - Mutation probability
  - Parameters which correspond to the probability distributions used in random perturbation