# Lecture 4
# Evolutionary Computation

Ning Xiong

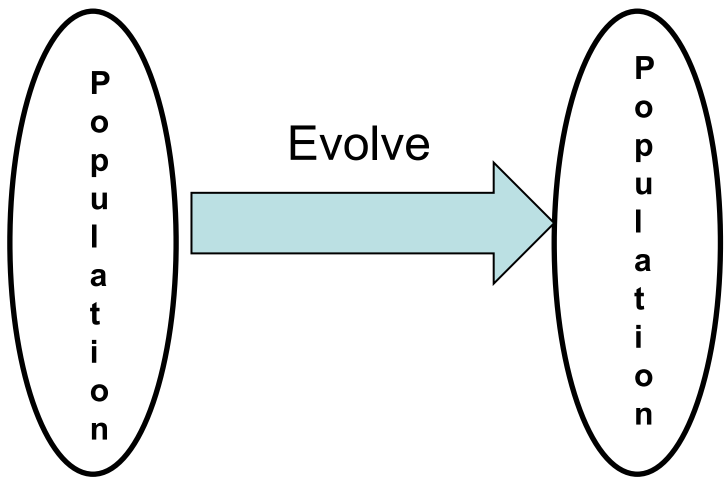
Mälardalen University

# Evolutionary Computation (EC)

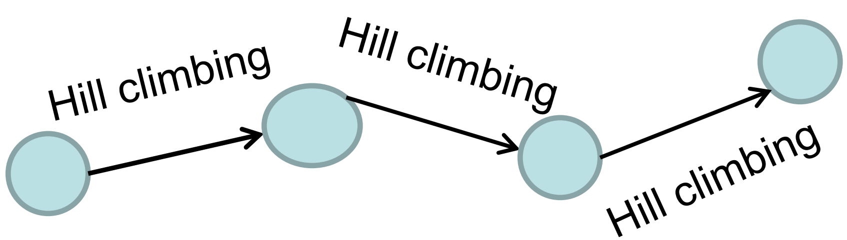
- EC: simulate **natural selection** in a computer program to improve system performance automatically
- Like natural selection, EC aims to facilitate stochastic search in problem space
- EC conducts randomized, parallel beam search. It becomes more popular with advancement of computer hardware.

# EC: Population-based search

- Population based search: exploiting more global information



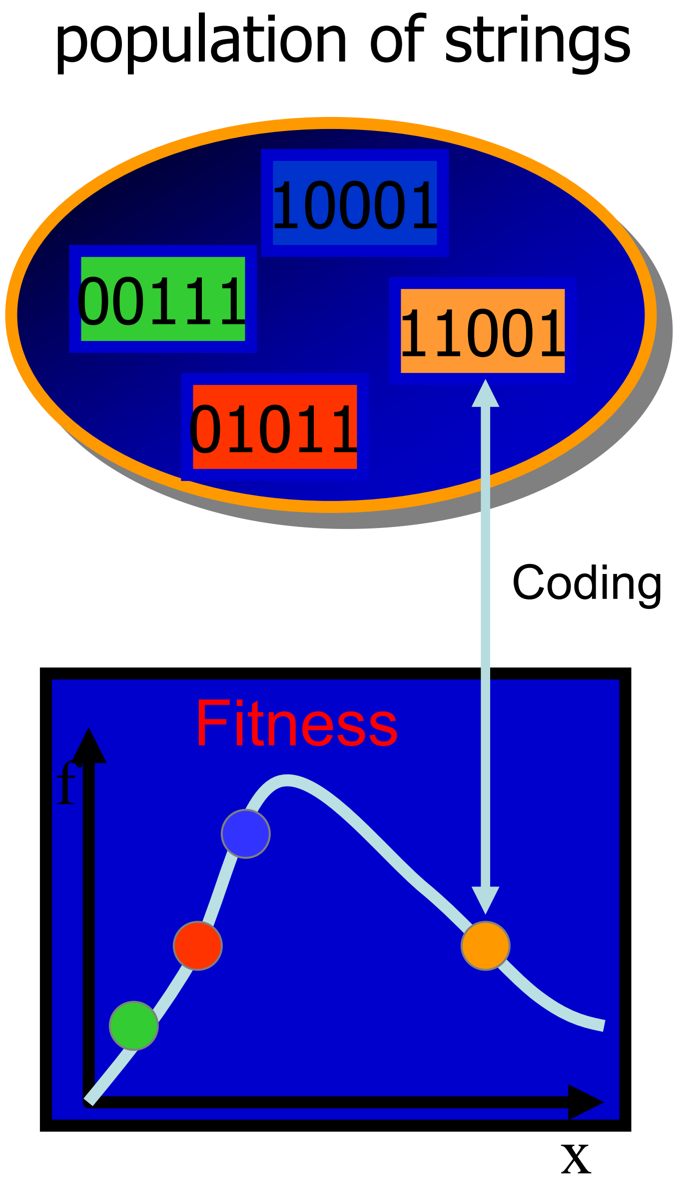

- Trajectory-based search: sequential search relying on local information



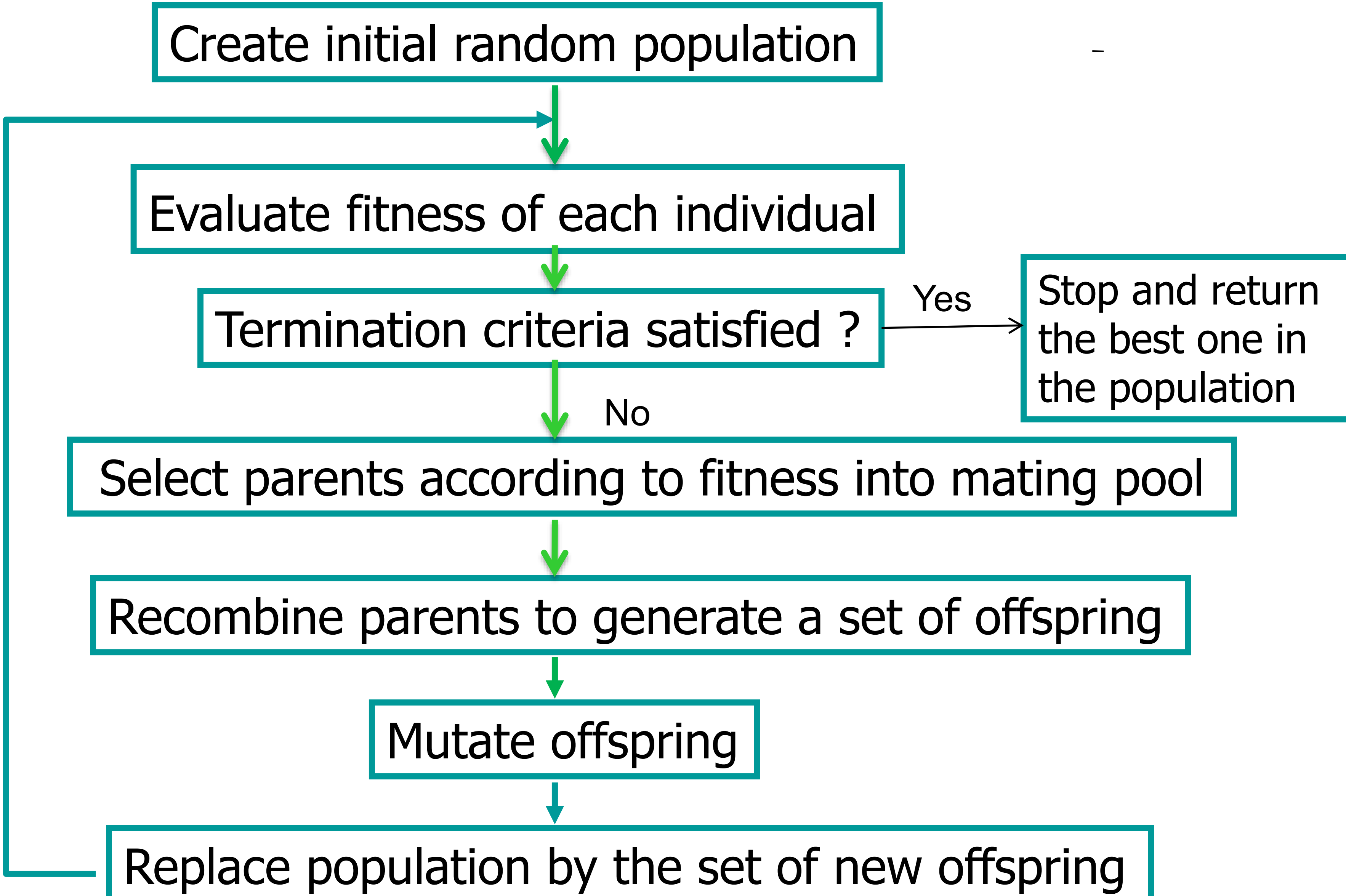
# Evolutionary algorithms

- Genetic algorithms
- Genetic programming
- Differential Evolution

# Population in a GA



# Flow Chart of Genetic Algorithm



# Key Issues of Designing GA

- How to represent a trial solution into a string (genetic code or chromosome)? → *Problem dependent*

- How to select individuals from the population for mating

- How to create new offspring from selected parents? What genetic operators (crossover, mutation) to use ?

- How to select offspring to form the next generation

- When to terminate the GA

# Coding scheme for GA

Various coding methods can be used, depending on the problem

Binary strings

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

Real-valued strings

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 3 | 6 | 8 | 2 | 4 | 1 | 5 |

# Selection in GA

- Darwinian idea: individuals with higher fitness have a better chance to be selected (survival of the fittest)
- **Fitness proportionate selection (roulette wheel)**:

$$P(S_i) = \frac{Fit(S_i)}{\sum_j Fit(S_j)}$$

**Tournament Selection (TS):** Choose the best from the k randomly selected individuals from the population. K is the size of the tournament.

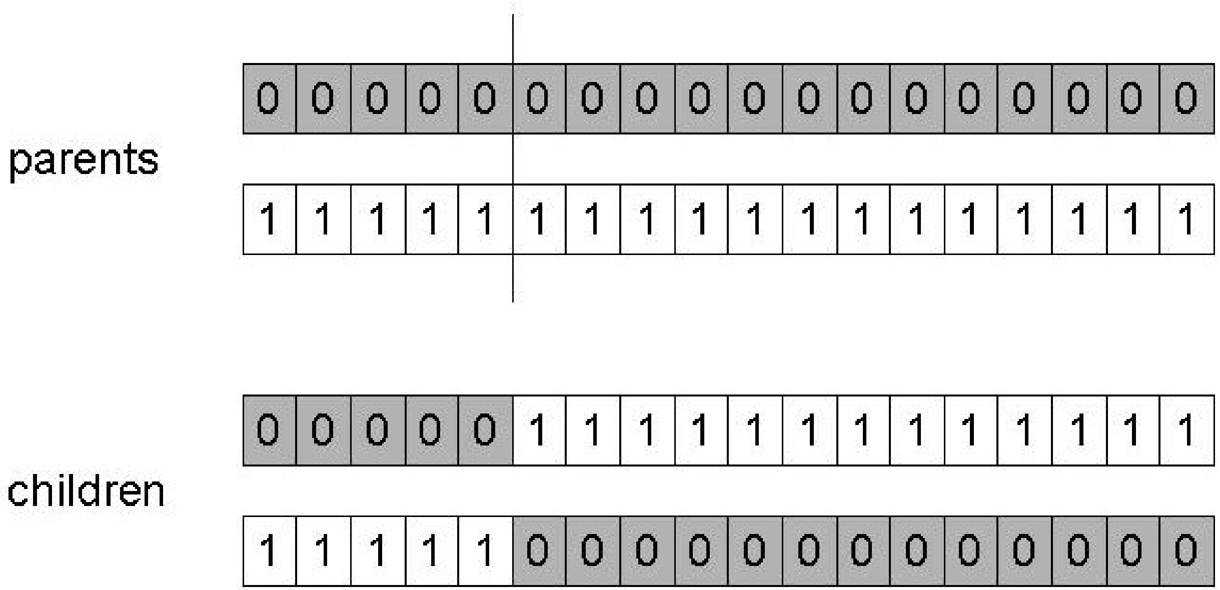
**Linear Order(LS):** The population is ordered according the fitness value and then a probability is given according to the order.

# Crossover

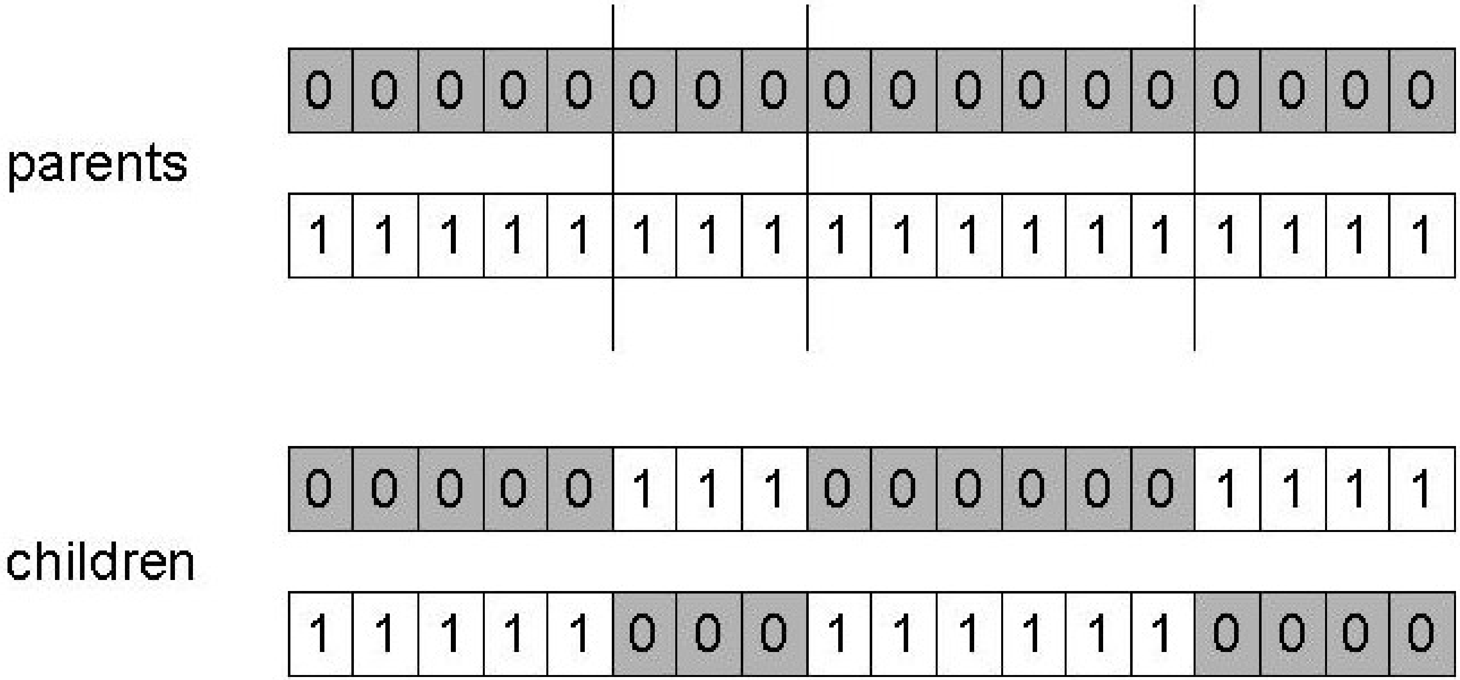
• crossover applied to a pair of parent strings with probability $p_c \in [0.6, 1.0]$

• crossover site chosen randomly with uniform probability

- one-point crossover



# Three-Point Crossover



# Arithmetic Crossover

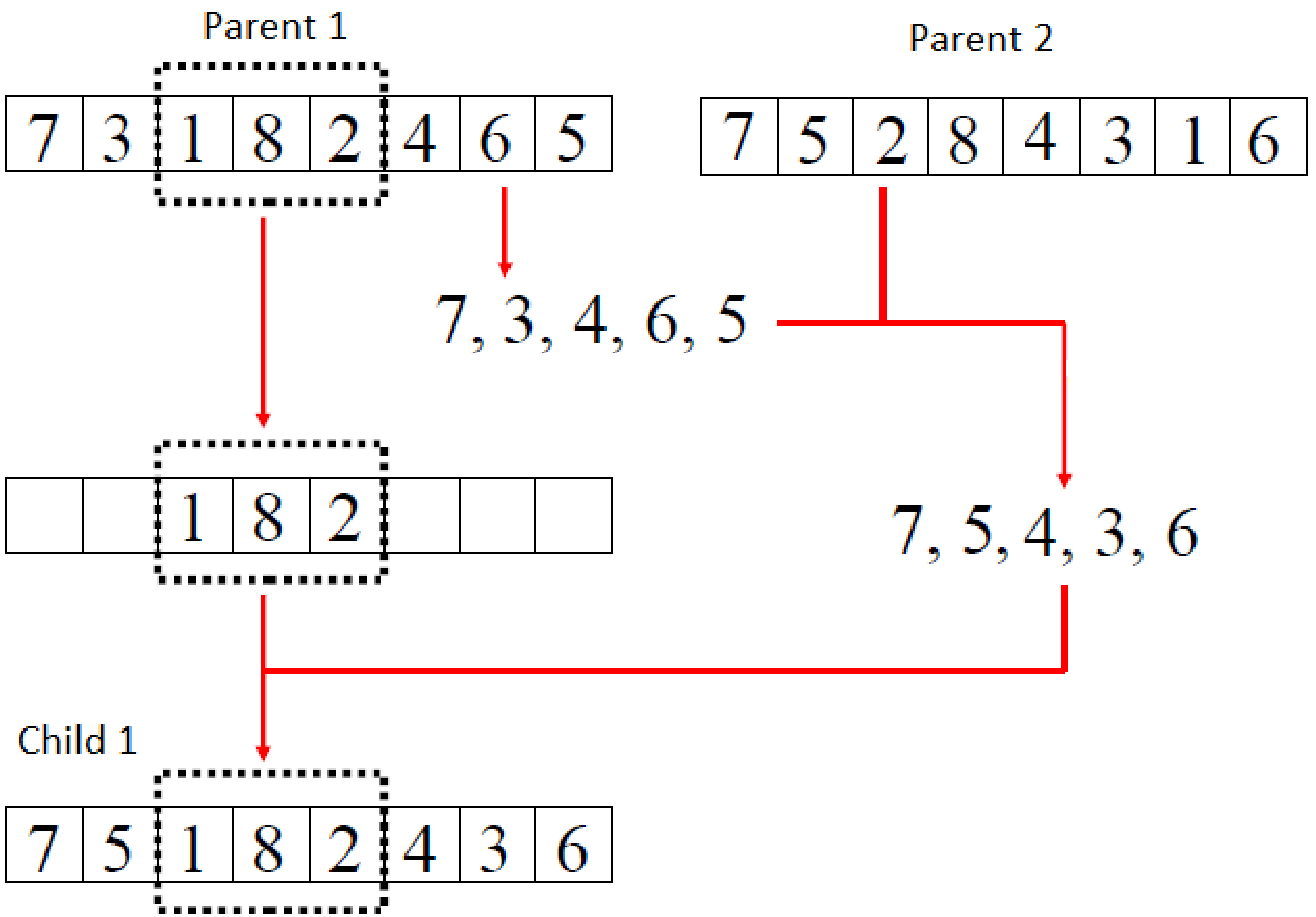
- Given two parent strings of real numbers as
  $X=[x_1, x_2, \ldots., x_n]$, $Y=[y_1, y_2, \ldots., y_n)$,

- Generate two offspring strings by means of linear combination as

$$X'=\alpha_1 X+(1- \alpha_1)Y$$
$$Y'=\alpha_2 X+(1- \alpha_2)Y$$

where $\alpha_1$, $\alpha_2$ are two uniform random numbers from [0, 1]

# Crossover with order representation



# Mutation

- mutation applied to individual bits with probability $p_m \in [0.001..0.1]$
- role of mutation is to maintain genetic diversity

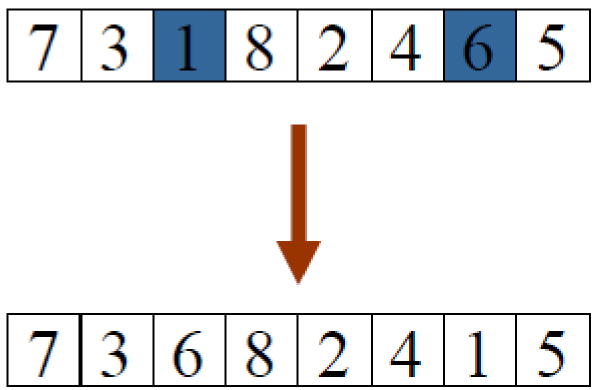
offspring: 1 1 0 0 0

Mutate 4th gene (bit flip)

mutated offspring: 1 1 0 1 0

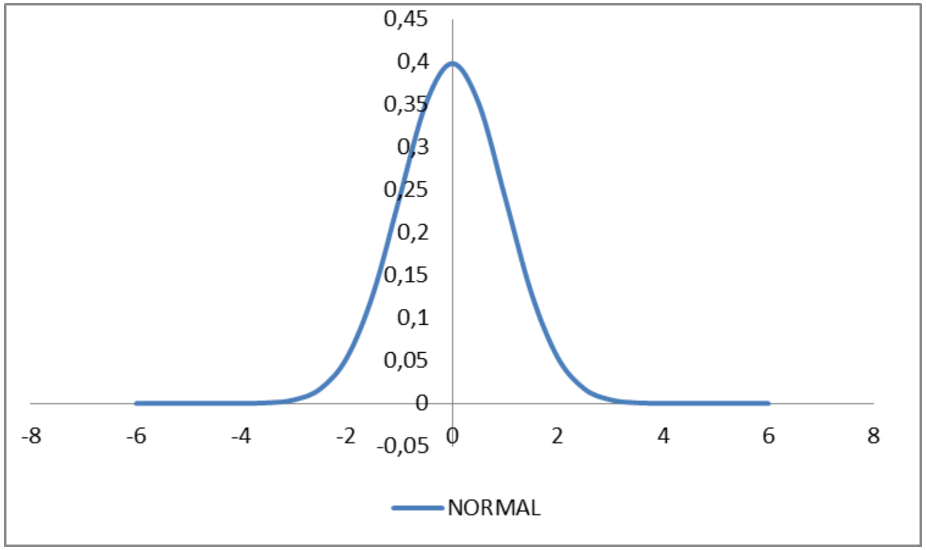
# Mutation (order representation)

- Select two random positions and change their elements.



# Mutation on Real Numbers

- Given an offspring string $X=[x_1, x_2, \ldots, x_n]$, mutation can be done by adding to each number in the string a random disturbance $\boldsymbol{u_i}$, which is subject to a normal density function $N(0, \delta)$.




- Consequently, we will have mutated offspring as:

$$X'=[x'_1, x_2, \ldots, x'_n]$$

with $$x'_i=x_i+u_i$$

- For example, in Matlab we can use the function sqrt(δ) * randn(1).

# Replacement

- The offspring replace the entire population.

- The best individual from the old generation enters the next generation (elitism)

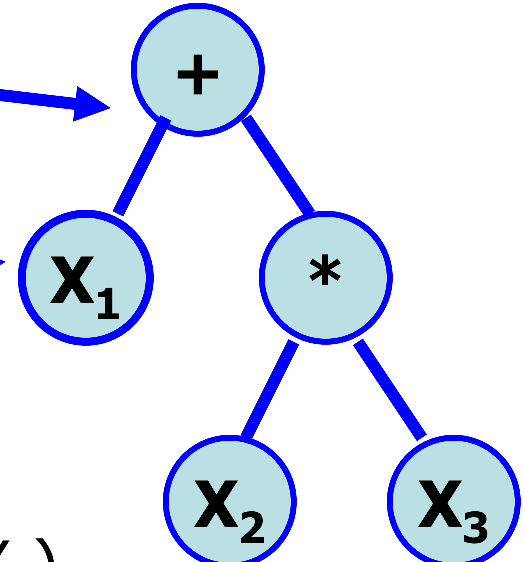
- The best N individuals from both the old generation and offspring enter the next generation. N is the population size

# Stop criteria

- We find the optimum or good enough solution
- Maximum number of fitness evaluations is reached
- Maximum number of iterations (generations) is reached

# Genetic Programming

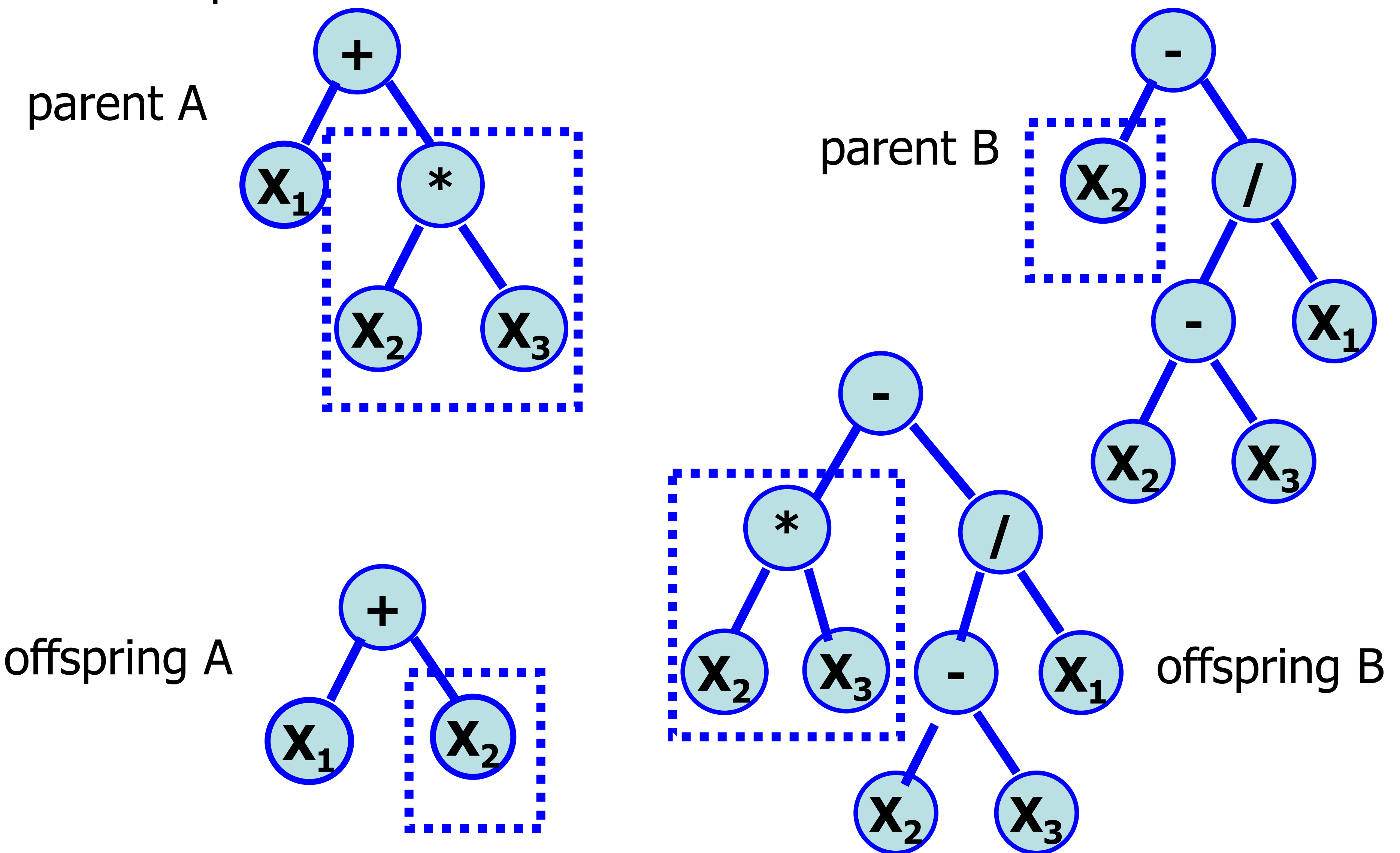
- Automatic generation of computer programs by means of natural evolution
- Individuals in population are programs represented as trees
- Tree nodes correspond to functions :
  - arithmetic functions {+,-,*,/}
  - logarithmic functions {sin,exp}
- Leaf nodes correspond to terminals :
  - input variables {$X_1$, $X_2$, $X_3$}
  - constants {0.1, 0.2, 0.5 }




$X_1+(X_2*X_3)$

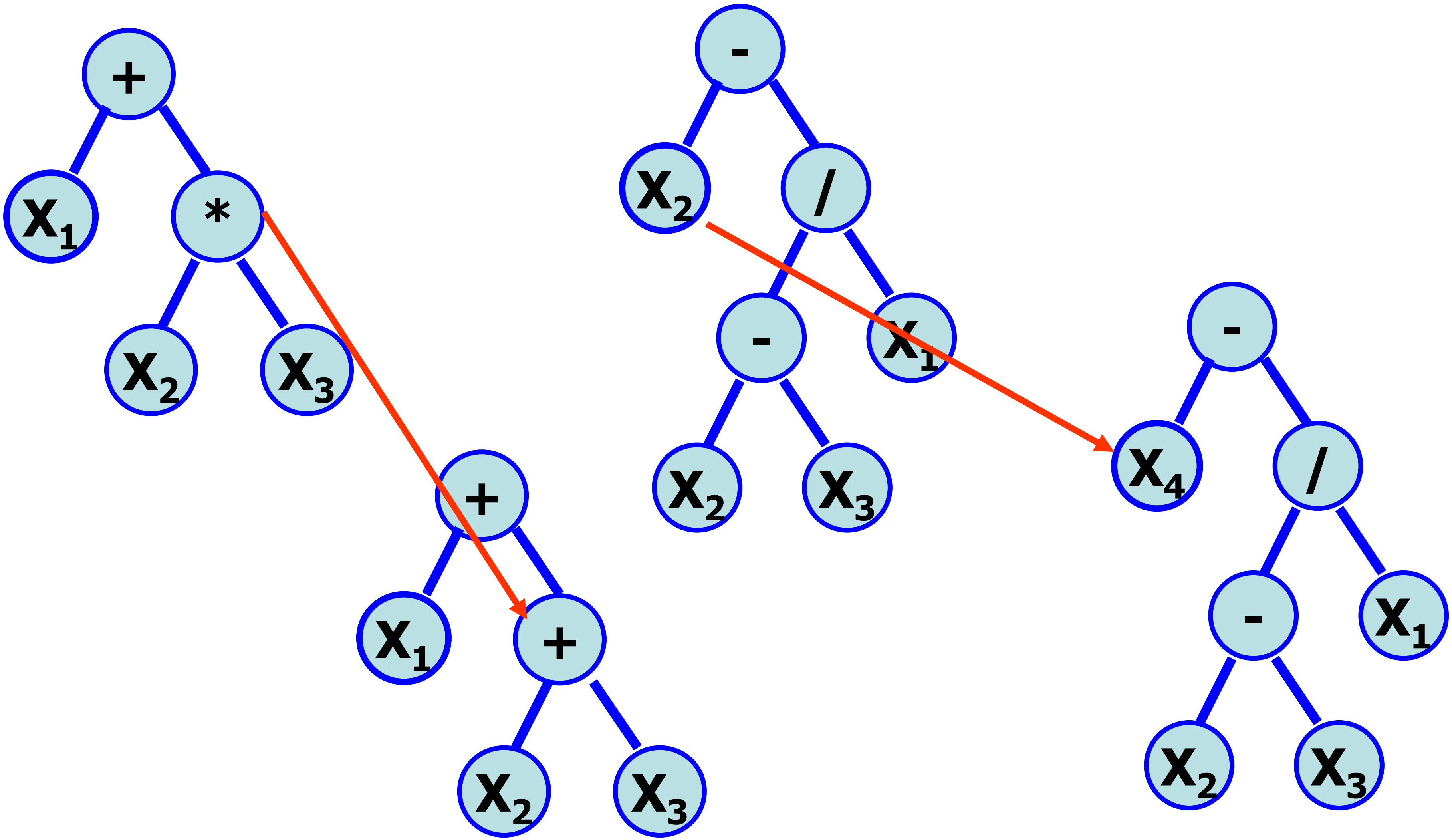
# Genetic Programming : Crossover

Crossover is performed by exchanging randomly chosen parts between parents



# Genetic Programming: Mutation

- A mutation operator can randomly change any function or any terminal in the tree.

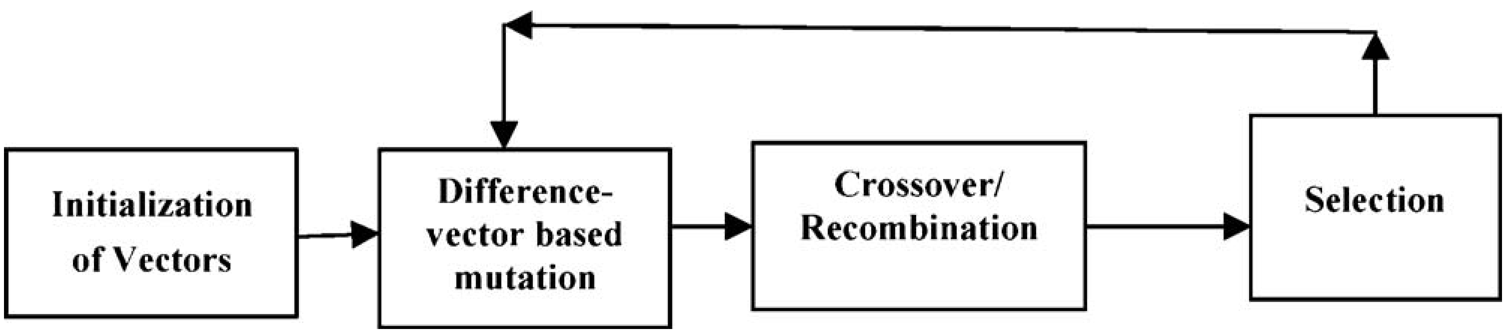


# Differential Evolution (DE)

- One of the most powerful evolutionary algorithms for optimization problems with real-valued parameters

- Population-based search method

- Population consists of target vectors of real numbers as possible solutions



# Working Steps of DE

For every target vector $\mathbf{X_i}$ in the population, do the following:

1. Randomly select three distinct vectors $\mathbf{X_{r1}}$, $\mathbf{X_{r2}}$, $\mathbf{X_{r3}}$ from the population

2. Calculate donor vector by adding the scaled difference between two of them to the third one

$$V_i = X_{r1} + F(X_{r2} - X_{r3})$$

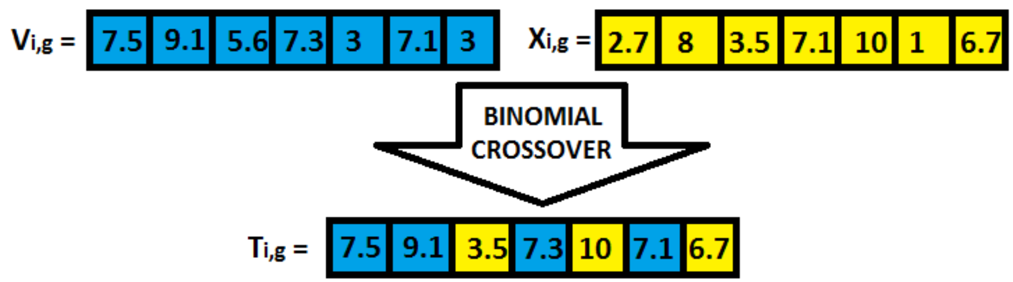
3. Do crossover to combine original vector $\mathbf{X_i}$ and donor vector $\mathbf{V_i}$ to acquire trial vector $\mathbf{T_i}$

4. Assess the fitness of the trial vector $\mathbf{T_i}$. If $\mathbf{T_i}$ is better than $\mathbf{X_i}$, it replaces $\mathbf{X_i}$ in the population

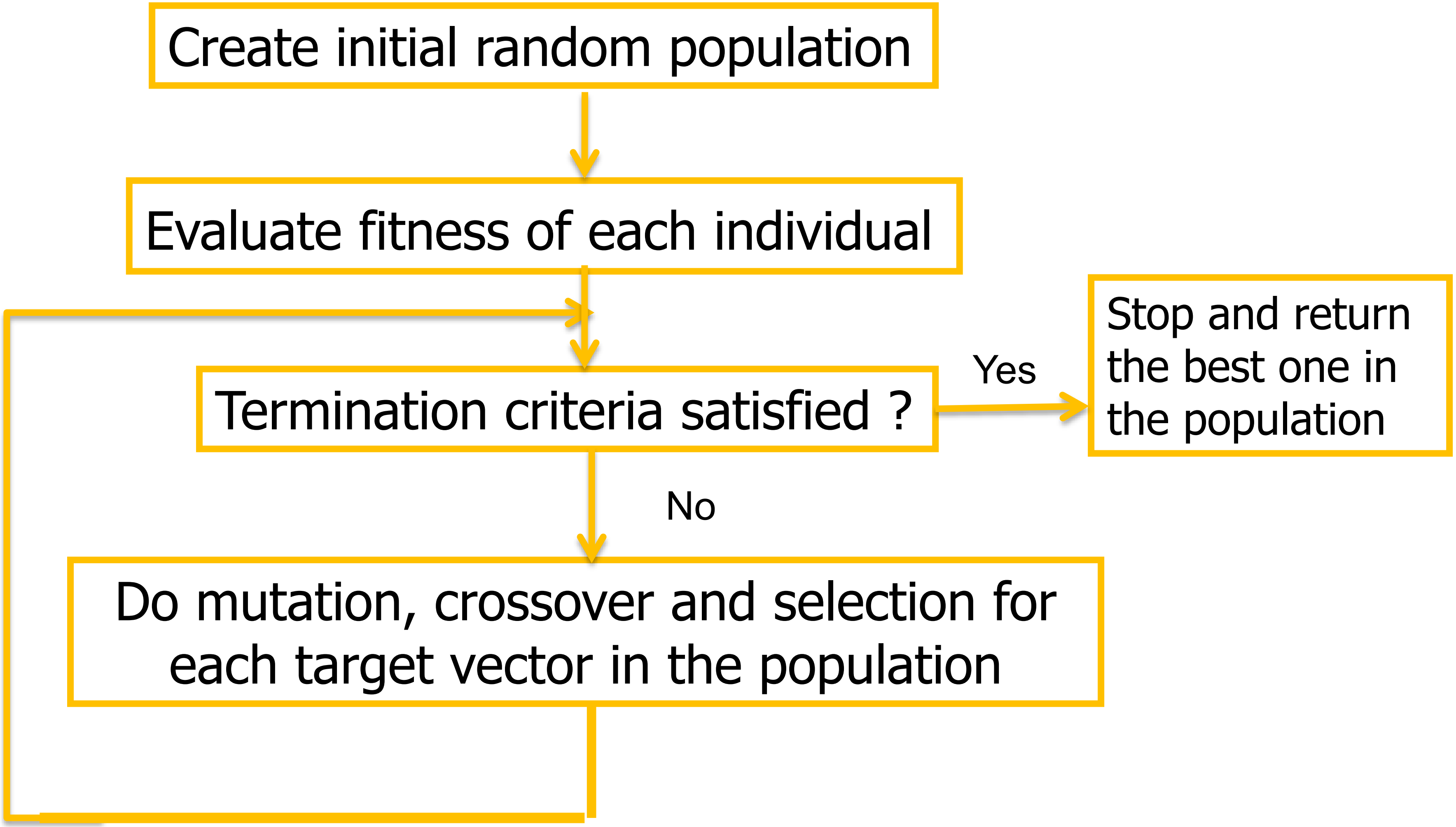
# Crossover in DE

$$T_i[j] = \begin{cases} V_{i,}[j] & if\ rand(0,1) < CR\ or\ j = j_{rand} \\ X_{i,}[j] & otherwise \end{cases}$$

- $T_{i,}$: Trial vector (one offspring)
- $V_{i,}$: donor vector
- $X_{i,}$: target i from the population
- $CR$: Crossover rate, $CR \epsilon [0,1]$
- $rand(0,1)$: random uniform number between 0 and 1.



# Differential Evolution – Flow chart



# Extra reading

- Genetic Algorithms
  - Norvig, Peter, and Russell, Stuart Jonathan, *Artificial intelligence: A modern approach*, Pearson Education, 2010 - ISBN: 9780132071482 (Pag. 125 – Pag 129)
  - Wikipedia page(s): https://en.wikipedia.org/wiki/Genetic_algorithm

- Differential Evolution
  - Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), 341-359.
  - Improving Differential Evolution with Adaptive and Local Search Methods, Miguel Leon (Link: http://www.diva-portal.org/smash/record.jsf?dswid=-9&pid=diva2%3A1366429&c=1&searchType=SIMPLE&language=en&query=Improving+Differential+Evolution+with+Adaptive+and+Local+Search+Methods&af=%5B%5D&aq=%5B%5B%5D%5D&aq2=%5B%5B%5D%5D&aqe=%5B%5D&noOfRows=50&sortOrder=author_sort_asc&sortOrder2=title_sort_asc&onlyFullText=false&sf=all)