Lecture 7

- **Differential Evolution (DE) Algorithm**
- **DE Java implementation**

CENG 632- Computational Intelligence, 2024-2025, Spring
Assist. Prof. Dr. Osman GÖKALP

Differential Evolution

- The Differential Evolution (DE) algorithm was first proposed by Storn and Price in 1995 as a technical report.
- It is among the most successful and important algorithms used for **numerical (continuous) optimization**.
- **Key features**
  - Population-based
  - Mutant and trial vectors generation
  - Simplicity

DE solution representation

Since **DE** is a **numerical optimization** algorithm, the solutions are in the form of a **vector of real numbers** (remember the Evolution Strategies (ES) algorithm).

$$\vec{X} = (x_1, x_2, \dots, x_D), \textit{where } x_i \in [a_i, b_i], \forall i = 1,2, \dots, D.$$

x_1	x_2	...	x_D

The **solution** $\vec{X}$ consists of D **real numbers**. In other words, the problem is D dimensional.

DE populations

Since DE is a population-based algorithm, it must have **multiple solution vectors**.

$$P = \{\vec{X}_1, \vec{X}_{2,\ldots,}\vec{X}_{NP}\}$$

$x_{1,1}$	$x_{1,2}$	$\cdots$	$x_{1,D}$
$x_{2,1}$	$x_{2,2}$	$\cdots$	$x_{2,D}$
		$\cdots$	
		$\cdots$	
		$\cdots$	
$x_{NP,1}$	$x_{NP,2}$	$\cdots$	$x_{NP,D}$

Population P consists of NP individuals, i.e. real number solution vectors.

Basic DE Algorithm

Initialize population $P = \{\vec{X}_1, \vec{X}_{2,\dots,}\vec{X}_{NP}\}$ randomly within the given bounds.

while TERM_COND is not met **do**

for i = 1 **to** NP **do**

1. Choose different 3 random population indices r1≠r2≠r3≠i in the range [1, NP].
2. Choose a random size index J in the range [1, D].
3. Create a **mutant vector** $\vec{V}_i$:
   $\vec{V}_i = \vec{X}_{r1} + F(\vec{X}_{r2} - \vec{X}_{r3})$
4. Ensure $\vec{V}_i$ is within the correct boundaries by truncating.
5. Create a **trial vector** $\vec{U}_i$: //[crossover current sol. and the mutant sol.]

   for j = 1 **to** D **do**

   $$\vec{U}_{i,j} = \begin{cases} \vec{V}_{i,j}, & if(rand \le CR) \textbf{ or } j = J \\ \vec{X}_{i,j}, & if(rand > CR) \text{ and } j \ne J \end{cases}$$

   end for
6. **if** $f(\vec{U}_i) \le f(\vec{X}_i)$ **then** $\vec{X}_i = \vec{U}_i$ **end if** //[selection step - minimization]

end for

end while

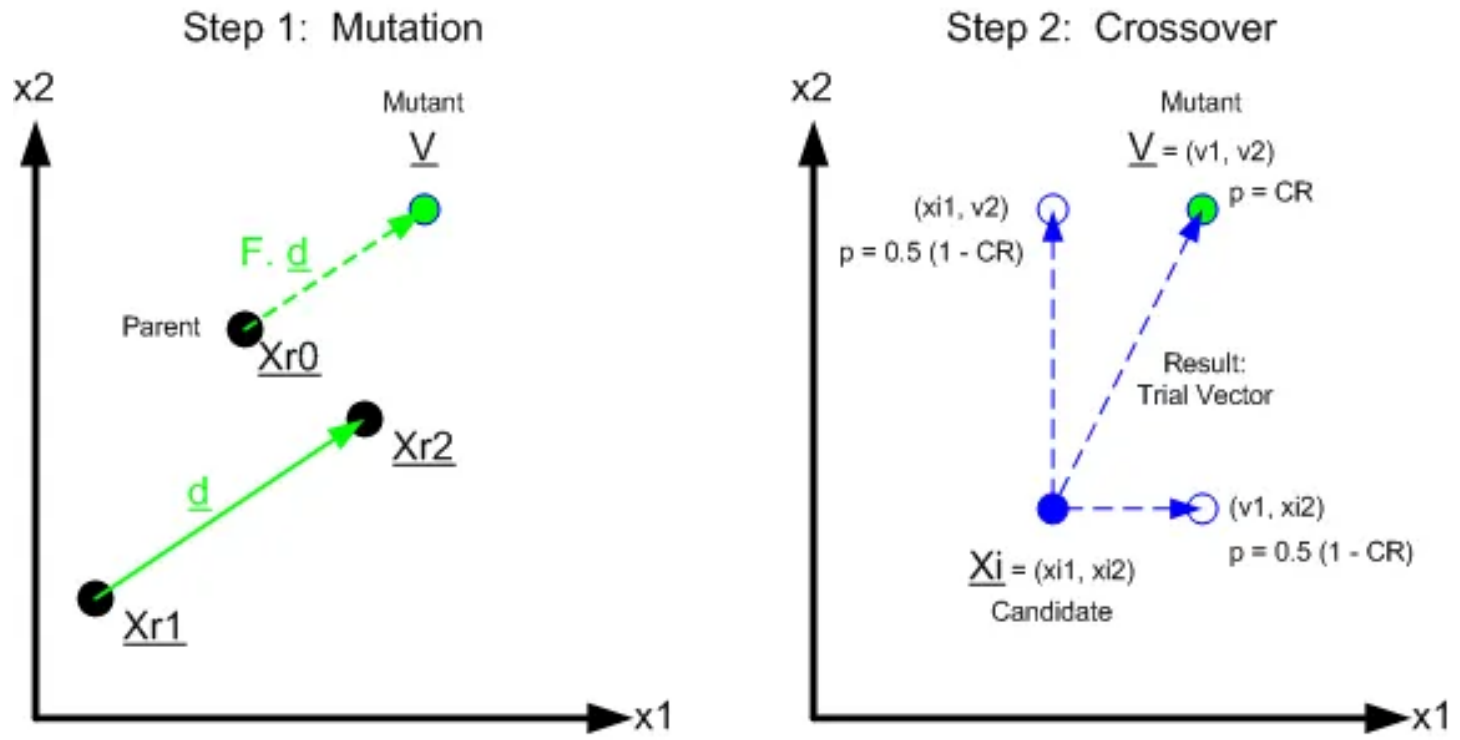
Basic DE Algorithm

- In the first step of the algorithm, the population is generated with random solution vectors.
- NP indicates the number of individuals in the population.
- When generating random solutions, pre-given search boundaries are taken into account:
  - **UB**: upper bound, **LB**: lower bound.
- Maximum runtime, iteration or number of fitness function evaluations can be used as the termination condition. In continuous optimization problems, the criterion of number of function evaluations is preferred.

Basic DE Algorithm

- Algorithm parameters:
  - NP: population size (# of solution vectors)
  - CR $\in [0, 1]$: crossover rate; the larger the **CR** value, the greater the effect of the mutant vector on the trial vector, as more components from the mutant vector are incorporated during crossover.
  - F $\in [0, 2]$: differential weight; the larger **F**, the larger the step size of the mutation, leading to greater exploration of the search space.

DE Visual Interpretation



Source: https://www.raynergobran.com/2011/04/differential-evolution-optimization/

DE mutation strategies

- The **mutant vector generation strategies** used in the DE algorithm are denoted by the notation **DE/a/b**. Commonly used mutation strategies are as follows:
  - DE/rand/1: $\vec{V}_i = \vec{X}_{r1} + F(\vec{X}_{r2} - \vec{X}_{r3})$
  - DE/rand/2: $\vec{V}_i = \vec{X}_{r1} + F(\vec{X}_{r2} - \vec{X}_{r3}) + F(\vec{X}_{r4} - \vec{X}_{r5})$
  - DE/best/1: $\vec{V}_i = \vec{X}_{best} + F(\vec{X}_{r1} - \vec{X}_{r2})$
  - DE/best/2: $\vec{V}_i = \vec{X}_{best} + F(\vec{X}_{r1} - \vec{X}_{r2}) + F(\vec{X}_{r3} - \vec{X}_{r4})$
  - DE/current-to-best/1: $\vec{V}_i = \vec{X}_i + F(\vec{X}_{best} - \vec{X}_i) + F(\vec{X}_{r1} - \vec{X}_{r2})$

DE JAVA Code Example

- In Lecture 7 DE_example.zip you will find DE implementation that solves some of the main benchmark functions*
- You can observe the results using different DE settings.

- Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. IEEE Transactions on Evolutionary computation, 3(2), 82-102.

(Link: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=771163)