King's College London
University of London

Differential Evolution

Dr H.K. Lam

Department of Engineering
King's College London
Office S2.14, Strand Building, Strand Campus
Email: hak-keung.lam@kcl.ac.uk
https://nms.kcl.ac.uk/hk.lam

Nature-Inspired Learning Algorithms (7CCSMBIM)

Outline

1. Introduction
2. Basic Differential Evolution
   - DE$/x/y/z$
3. Variations to Basic Differential Evolution
   - Switching DE strategies
   - Hybrid DE strategies
     - Gradient-Based Hybrid Differential Evolution
     - Evolutionary Algorithm-Based Hybrids
     - Particle Swarm Optimization Hybrids
     - Self-Adaptive Differential Evolution

- To get the concept of Differential Evolution and know how it works.
- To get an idea of various versions of Differential Evolution.

Introduction

- Differential Evolution (DE) is a *stochastic*, *population-based* search strategy developed by Storn and Price in 1995.
- DE strategy is a branch of *Evolutionary Algorithms* (EA).
- The main difference is that distance and direction information (using *difference vectors*) of the population is used to guide the search process.
- It was originally developed for continuous-valued landscapes.

KING'S College LONDON
University of London

Notation

- $\mathbf{x}_i(t) = [x_{i1}, \cdots, x_{in_x}]$: the i^{th} individual in the population.
- n_x: number of elements in each individual.
- n_s: size of population (number of particles in the swarm).
- $C(t)$: population in the t^{th} generation/iteration.
- $\mathbf{u}_i(t)$: trial vector
- $\mathbf{x}_{i_1}$: target vector, $i \neq i_1$
- $\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)$: difference vector, $i \neq i_1 \neq i_2 \neq i_3; i_2.\ i_3 \sim U(1, n_s)$
- $\mathbf{x}_{i_2}(t)$, $\mathbf{x}_{i_3}(t)$: 2 randomly selected individuals
- $\beta \in (0, \infty)$: scale factor
- $U_I(1, n_s) \in \{1, 2, \ldots, n_s\}$: a random integer variable in the range of 1 and n_s
- $U(0,1) \in [0,1]$: a uniform random variable in the range of 0 and 1.
- $p_r \in [0,1]$: probability of crossover/recombination
- $\mathbf{x}_{\min} = \begin{bmatrix} x_{\min,1} & x_{\min,2} & \cdots & x_{\min,n_x} \end{bmatrix}$: lower search boundary
- $\mathbf{x}_{\max} = \begin{bmatrix} x_{\max,1} & x_{\max,2} & \cdots & x_{\max,n_x} \end{bmatrix}$: upper search boundary
- $\hat{\mathbf{x}}(t)$: the best individual from the population at generation t
- $\hat{\mathbf{y}}(t) = [\hat{y}_1(t), \cdots, \hat{y}_{n_x}(t)]$: the global best position since the first generation.
- $\mathbf{x}_{\min} = [x_{1_{\min}}, \cdots, x_{n_{x\min}}]$: a vector of constants denoting the lower bound of $\mathbf{x}_i(t)$.
- $\mathbf{x}_{\max} = [x_{1_{\max}}, \cdots, x_{n_{x\max}}]$: a vector of constants denoting the upper bound of $\mathbf{x}_i(t)$.
- $\mathbf{v}_i = [v_{i1}, \cdots, v_{in_x}]$: a velocity vector.
- $r_{1j}(t)$, $r_{2j}(t) \in [0,1]$: a random number.
- t: iteration/generation number

Basic Differential Evolution

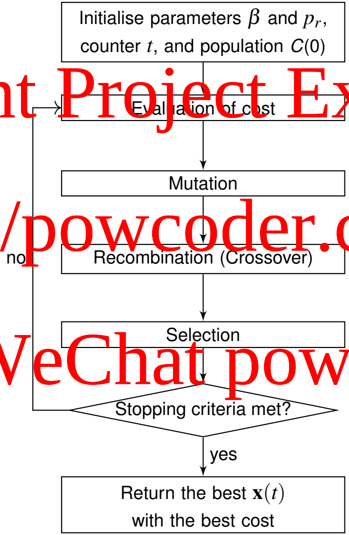

Figure 1: Flowchart of basic differential evolution.

Basic Components

- **Population:** it is a group of potential solution
- **Mutation:** it produces a *trial vector* for each selected individual by mutating a *target vector* with a *weighted differential vector*.
- **Crossover:** it produces offspring by applying crossover operation to the *trial vector* produced by the mutation operation.
- **Selection:** It determines if the parent or the offspring will survive to the next generation.

Differences between Differential Evolution and other evolutionary algorithms

- Mutation is applied first before crossover. Mutation generates a trial vector which is then used within the crossover operator to produce one offspring.
- Mutation "step sizes" are not sampled from a prior known probability distribution function (for example, normal distribution) but are influenced by differences between individuals of the current population (difference vectors).
- Crossover involves one single parent (individual) and its trial vector.
- Crossover generates one offspring only.
- Each parent (individual) will have its offspring.

Population

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{n_s} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n_x} \\ x_{21} & x_{22} & \cdots & x_{2n_x} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n_s1} & x_{n_s2} & \cdots & x_{n_sn_x} \end{bmatrix}$$

where n_s is the size of the population and n_x is number of decision variables in an individual.

$C(t)$ denotes the population at the t^{th} generation/iteration, e.g., $C(t) = \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \\ \vdots \\ \mathbf{x}_{n_s}(t) \end{bmatrix}$

Selection for mutation

1. Random selection for target/difference vectors
2. The best individual is selected as the target vector.

Selection for the next population: The offspring replaces the parent if the fitness of the offspring is better than its parent.

Mutation for each parent

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t))$$

where

- $\mathbf{u}_i(t)$: trial vector
- $\mathbf{x}_{i_1}$: target vector, $i \neq i_1$
- $\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)$: difference vector, $i \neq i_1 \neq i_2 \neq i_3$; $i_2, i_3 \sim U_I(1, n_s)$
- $\mathbf{x}_{i_2}(t)$, $\mathbf{x}_{i_3}(t)$: 2 randomly selected individuals
- $\beta \in (0, \infty)$: scale factor
- $U_I(1, n_s)$: a random integer variable in the range of 1 and n_s
- More than one difference vector can be used.

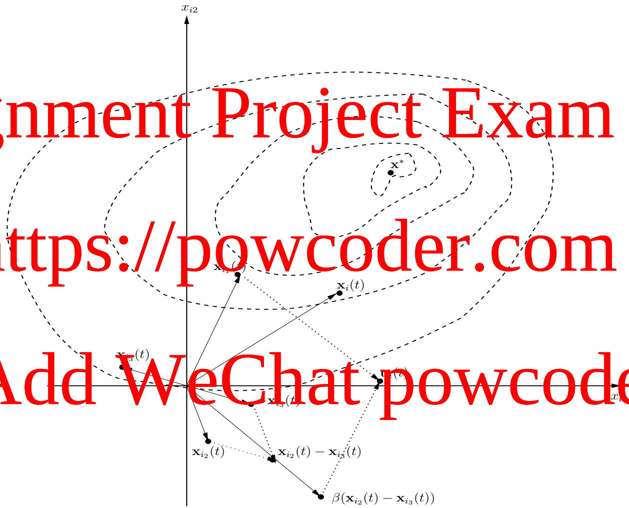

Figure 2: Mutation operation with $beta = 1.5$.

Difference vectors

- It provides information of the positions of individuals about the fitness of the landscape.
- Large distance between individuals: individuals should make large step sizes in order to explore as much of the search space as possible.
- Small distance between individuals: individuals should make step sizes small to exploit local areas.

KING'S College LONDON
University of London

Crossover: It produces an offspring $\mathbf{x}'(t)$ by implementing a discrete recombination of the trial vector $\mathbf{u}(t)$ and the parent vector $\mathbf{x}_i(t)$.

$$x'_{ij}(t) = \begin{cases} u_{ij}(t) & \text{if } j \in J \\ x_{ij}(t) & \text{otherwise} \end{cases}$$

where

- $x_{ij}(t)$: the j^{th} element of the parent vector $\mathbf{x}_i(t)$, $i = 1, 2, \ldots, n_s$, $j = 1, 2, \ldots, n_x$
- $x'_{ij}(t)$: the j^{th} element of the offspring vector $\mathbf{x}'_i(t)$, $i = 1, 2, \ldots, n_s$, $j = 1, 2, \ldots, n_x$
- $u_{ij}(t)$: the j^{th} element of the trial vector $\mathbf{x}'_i(t)$, $i = 1, 2, \ldots, n_s$, $j = 1, 2, \ldots, n_x$
- J: the set of element indices that will undergo crossover (the set of crossover points)

Crossover: It produces an offspring $\mathbf{x}'(t)$ by implementing a discrete recombination of the trial vector $\mathbf{u}(t)$ and the parent vector $\mathbf{x}_i(t)$.

$$x'_{ij}(t) = \begin{cases} u_{ij}(t) & \text{if } j \in J \\ x_{ij}(t) & \text{otherwise} \end{cases}$$

where

- $x_{ij}(t)$: the j^{th} element of the parent vector $\mathbf{x}_i(t)$, i = 1, 2, ..., n_s, j = 1, 2, ..., n_x
- $x'_{ij}(t)$: the j^{th} element of the offspring vector $\mathbf{x}'_i(t)$, i = 1, 2, ..., n_s, j = 1, 2, ..., n_x
- $u_{ij}(t)$: the j^{th} element of the trial vector $\mathbf{x}'_i(t)$, i = 1, 2, ..., n_s, j = 1, 2, ..., n_x
- J: the set of element indices that will undergo crossover (the set of crossover points)

Example: $n_x = 3$, $J = [1, 3]$

$\mathbf{u}_i(t) = \left[u_{i1}(t), u_{i2}(t), u_{i3}(t)\right]$

$\mathbf{x}_i(t) = \left[x_{i1}(t), x_{i2}(t), x_{i3}(t)\right]$

$\mathbf{x}'_i(t) = \left[u_{i1}(t), x_{i2}(t), u_{i3}(t)\right]$

Binomial crossover: The crossover points are randomly selected from the set of $\{1, 2, \ldots, n_x\}$

Algorithm 1: Binomial Crossover for Selecting Crossover Points

```
J ← { };
j* ~ U_I(1, n_x);
J ← J ∪ {j*};
for each j ∈ {1, 2, ..., n_x} \ j* do
    if U(0,1) < p_r then
        J ← J ∪ { j };
    end
end
```


- $U_I(1, n_x) \in \{1, 2, \ldots, n_x\}$: a random integer variable in the range of 1 and n_x
- $U(0,1) \in [0,1]$: a uniform random variable in the range of 0 and 1
- $p_r \in [0,1]$: probability of crossover/recombination

Remark: $j^* \sim U(1, n_x)$ is to make sure that at least one crossover point is selected.

KING'S College LONDON
University of London

Exponential crossover: It selects a sequence of adjacent crossover points from a randomly selected index, treating the list of potential crossover points as a circular array.

Algorithm 2: Exponential Crossover for Selecting Crossover Points

$J \leftarrow \{\}$;
$j \sim U_I(0, n_x - 1)$;
repeat
  $J \leftarrow J \cup \{j+1\}$;
  $j = (j+1)$ mod n_x;
until $U(0,1) \geq p_r$ *or* $|J| = n_x$;

- $U_I(0, n_x - 1) \in \{0, 1, \ldots, n_x - 1\}$: a random integer variable in the range of 0 and $n_x - 1$
- $p_r \in [0,1]$: probability of crossover
- $|J|$ is the number of elements in the set J.
- mod: Modulus (modulo) operator, e.g., 12 mod 5 = 2 (remainder of 12/5)

Remark: The list of potential crossover points is treated as a circular array in $j = (j+1)$ mod n_x.

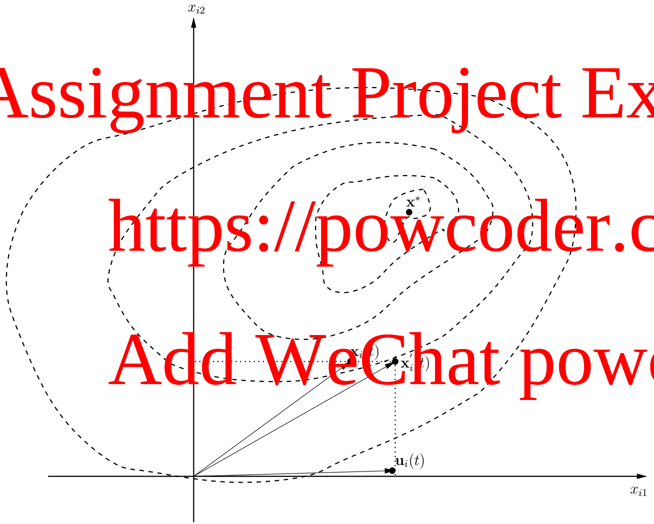

Figure 3: Crossover operation with $\beta = 1.5$.

Algorithm 3: General Differential Evolution Algorithm

Set the generation counter, $t = 0$;
Initialize the control parameters, β and p_r;
Create and initialise the population†, $C(0)$, of n_s individuals;
while *stopping condition(s) not true* **do**
  for *each individual,* $\mathbf{x}_i \in C(t)$ **do**
    Evaluate the fitness, $f(\mathbf{x}_i(t))$;
    Create the trial vector, $\mathbf{u}_i(t)$ by applying the `mutation` operator;
    Create an offspring, $\mathbf{x}_i'(t)$, by applying the `crossover` operator;
    if $f(\mathbf{x}_i'(t))$ *is better than* $f(\mathbf{x}_i(t))$ **then**
      Add $\mathbf{x}_i'(t)$ to $C(t+1)$;
    else
      Add $\mathbf{x}_i(t)$ to $C(t+1)$;
    end
  end
  $t \leftarrow t+1$
end
Return the individual with the best fitness as the solution;

†Initial population: $\mathbf{x}_i(0) \sim U(x_{\min,j}, x_{\max,j})$ where $\mathbf{x}_{\min} = \begin{bmatrix} x_{\min,1} & x_{\min,2} & \cdots & x_{\min,n_x} \end{bmatrix}$ and $\mathbf{x}_{\max} = \begin{bmatrix} x_{\max,1} & x_{\max,2} & \cdots & x_{\max,n_x} \end{bmatrix}$ define the search boundaries.

Assignment Project Exam Help

DE/x/y/z

https://powcoder.com

Add WeChat powcoder

KING'S College LONDON
University of London

Variations to the basic Differential Evolution strategies:

- Different ways to choose target vectors
- Different number of difference vectors to be used
- Different ways to choose crossover points

Notation: DE$/x/y/z$

- x: the method of selecting the target vector
- y: the number of difference vectors used
- z: the crossover method used
  - `bin`: Binomial crossover
  - `exp`: Exponential crossover

Some common DE strategies

- DE/rand/1/z
- DE/best/1/z
- DE$/x/n_v/z$
- DE/rand-to-best$/n_v/z$
- DE/current-to-best$/1+n_v/z$

DE/rand/1/z: This is the basic DE introduced before.

- A random individual from the current population is selected as the target vector denoted as $\mathbf{x}_{i_1}(t)$, $i \neq i_1$.
- Number of difference vectors is 1.
- Any crossover methods can be used.

Trail vector:

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta\left(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)\right),\ i = 1, 2, \ldots, n_s$$

DE/best/1/z:

- The best individual $\hat{\mathbf{x}}(t)$ from the current population is selected as the target vector.
- Number of difference vectors is 1.
- Any crossover methods can be used.

Trail vector:

$$\mathbf{u}_i(t) = \hat{\mathbf{x}}(t) + \beta\left(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)\right), i = 1, 2, \ldots, n_s$$

DE$/x/n_v/z$:

- Any methods selecting target vectors can be used.
- Number of difference vectors is n_v.
- Any crossover methods can be used.

Trail vector:

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta \sum_{k=1}^{n_v} \left(\mathbf{x}_{i_2,k}(t) - \mathbf{x}_{i_3,k}(t) \right), i = 1, 2, \ldots, n_s$$

DE/rand-to-best$/n_v/z$:

- Combines the *rand* and *best* strategies.
- Number of difference vectors is n_v.
- Any crossover methods can be used.

Trail vector:

$$\mathbf{u}_i(t) = \gamma\hat{\mathbf{x}}(t) + (1-\gamma)\mathbf{x}_{i_1}(t) + \beta\sum_{k=1}^{n_v}(\mathbf{x}_{i_2,k}(t) - \mathbf{x}_{i_3,k}(t)), i = 1, 2, \ldots, n_s$$

where $\gamma \in [0, 1]$ is predefined scalar to control the greediness of the mutation operator.

- $\gamma \to 1$ favours exploitation.
- $\gamma \to 0$ favours exploration.
- Adaptive $\gamma(t)$ can be used: The value of $\gamma(t)$ increases from $\gamma(0) = 0$ with each new generation towards the value 1.

DE/current-to-best$/1+n_v/z$:

- Parent vector $\mathbf{x}_i(t)$ is selected as the target vector.
- Number of difference vectors is $1+n_v$. The first difference vector is formed by the parent vector and the best individual $\hat{\mathbf{x}}(t)$ from the current population.
- Any crossover methods can be used.

Trail vector:

$$\mathbf{u}_i(t) = \mathbf{x}_i(t) + \beta(\hat{\mathbf{x}}(t) - \mathbf{x}_i(t)) + \beta \sum_{k=1}^{n_v} (\mathbf{x}_{i_1,k}(t) - \mathbf{x}_{i_2,k}(t)), i = 1, 2, \ldots, n_s$$

Variations to Basic Differential Evolution

- Switching DE strategies
- Hybrid DE strategies
  - Gradient-based hybrid DE
  - Evolutionary algorithm-based hybrid DE
  - Particle swarm optimization hybrid DE
- Self-adaptive DE strategies
  - Dynamic parameters
  - Self-adaptive parameters

Switching DE strategies

Switching DE strategies

Empirical studies:

- DE/rand/1/bin maintains good diversity (exploration).
- DE/current-to-best/2/bin shows good convergence characteristics (exploitation).

Switching DE strategy switches dynamically between DE/rand/1/bin and DE/current-to-best/2/bin according to a probability assigned to each DE strategy.

- $p_{s,1}$: probability that DE/rand/1/bin will be applied.
- $p_{s,2} = 1 - p_{s,1}$: probability that DE/current-to-best/2/bin will be applied.
- $p_{s,1}$ and $p_{s,2}$ are needed to be computed in each generation.

KING'S College LONDON
University of London

$$p_{s,1} = \frac{n_{s,1}(n_{s,2} + n_{f,2})}{n_{s,2}(n_{s,1} + n_{f,1}) + n_{s,1}(n_{s,2} + n_{f,2})}$$

where

- $n_{s,1}$ and $n_{s,2}$ are the number of offspring that survive to the next generation for `DE/rand/1/bin` and `DE/current-to-best/2/bin` respectively
- $n_{f,1}$ and $n_{f,2}$ are the number of discarded offspring for each strategy.
- Initial probability: $p_{s,1} = p_{s,2} = 0.5$
- Learning period: For the first 50 iterations, for each individual, use $p_{s,1} = p_{s,2} = 0.5$ to choose which DE strategy is applied to the i^{th} individual and record (the average) $n_{s,1}$, $n_{s,2}$, $n_{f,1}$ and $n_{f,2}$.

Switching DE strategies

KING'S College LONDON
University of London

After the learning period, choose the DE strategy to be applied to the i^{th} individual according to the following algorithm.

Algorithm 4: Switching Differential Evolution Algorithm

Compute $p_{s,1}$ for the current generation;
$r \sim U(0,1)$;
if $r < p_{s,1}$ **then**
    `DE/rand/1/bin` is applied;
else
    `DE/current-to-best/2/bin` is applied;
end

Hybrid DE strategies

KING'S College LONDON
University of London

Gradient-based hybrid DE

- **Acceleration**:

  An *acceleration operator* using *gradient descent* to improve convergence speed without decreasing diversity (by adjusting the best individual toward obtaining a better position).

- **Migration**:

  A *migration operator* to provide the DE with the improved ability to escape local optima by increasing population diversity.

KING'S College LONDON
University of London

Algorithm 5: Hybrid Differential Evolution with Acceleration and Migration

Set the generation counter, $t = 0$;
Initialize the control parameters, β and p_r;
Create and initialise the population†, $C(0)$, of n_s individuals;
while *stopping condition(s) not true* **do**
    Apply the `migration` operator if necessary;
    for *each individual,* $\mathbf{x}_i \in C(t)$ **do**
        Evaluate the fitness, $f(\mathbf{x}_i(t))$;
        Create the trial vector, $\mathbf{u}_i(t)$ by applying the `mutation` operator;
        Create an offspring, $\mathbf{x}_i'(t)$, by applying the `crossover` operator;
        if $f(\mathbf{x}_i'(t))$ *is better than* $f(\mathbf{x}_i(t))$ **then**
            Add $\mathbf{x}_i'(t)$ to $C(t+1)$;
        else
            Add $\mathbf{x}_i(t)$ to $C(t+1)$;
        end
    end
    Apply the `acceleration` operator if necessary;
    $t \leftarrow t+1$
end
Return the individual with the best fitness as the solution;

Acceleration operator:

$$\mathbf{x}(t) = \begin{cases} \hat{\mathbf{x}}(t+1) & \text{if } f(\hat{\mathbf{x}}(t+1)) < f(\hat{\mathbf{x}}(t)) \\ \hat{\mathbf{x}}(t+1) - \eta(t) \bigtriangledown f & \text{otherwise} \end{cases}$$

where

- $\hat{\mathbf{x}}(t)$ denotes the best individual of the current population, i.e., $C(t)$, before application of the mutation and crossover operators.
- $\hat{\mathbf{x}}(t+1)$ denotes the best individual for the next population, i.e., $C(t+1)$, after mutation and crossover have been applied to all individuals.
- $\eta(t) \in (0,1]$ is the learning rate (step size). If the gradient descent step failed to create a new vector, $\mathbf{x}(t)$, with better cost, the learning rate is reduced by a factor.
- $\bigtriangledown f$ is the gradient of the cost function.

- The new vector, $\mathbf{x}(t)$, replaces the worst individual in the new population, $C(t+1)$ (if $\mathbf{x}(t)$ is better than the worst individual).

Remark: When using gradient descent, it can speed up the search but the disadvantage is that the DE may get stuck in a local minimum, or prematurely converge. It can be alleviated by the migration operator which increases the population diversity.

KING'S College LONDON
University of London

Migration operator:

$$x'_{ij}(t) = \begin{cases} \hat{x}_j(t) + r_{ij}(x_{\min,j} - \hat{x}_j) & \text{if } U(0,1) < \dfrac{\hat{x}_j - x_{\min,j}}{x_{\max,j} - x_{\min,j}} \\ \hat{x}_j(t) + r_{ij}(x_{\max,j} - \hat{x}_j) & \text{otherwise} \end{cases}$$

where $r_{ij} \sim U(0,1)$.

$\boxed{\text{Spawned individual } \mathbf{x}'(t) \text{ becomes } \mathbf{x}_i(t+1).}$

The best individual: $\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \cdots & \hat{x}_j & \cdots & \hat{x}_{n_x} \end{bmatrix}$

The i^{th} individual: $\mathbf{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \cdots & x_{ij} & \cdots & x_{in_x} \end{bmatrix}$

The new i^{th} individual: $\mathbf{x}'_i = \begin{bmatrix} x'_{i1} & x'_{i2} & \cdots & x'_{ij} & \cdots & x'_{in_x} \end{bmatrix}$

Bounds: $x_{\min,j} \leq x_{ij} \leq x_{\max,j}$

The migration operator is applied only when:

$$\frac{\sum_{i=1,\mathbf{x}_i(t)\neq\hat{\mathbf{x}}(t)}^{n_s}\sum_{j=1}^{n_x} l_{ij}(t)}{n_x(n_s-1)} < \varepsilon_1 \text{ where } l_{ij}(t) = \begin{cases} 1 & \text{if } \left|\left(x_{ij}(t)-\hat{x}_j(t)\right)/\hat{x}_j(t)\right| > \varepsilon_2 \\ 0 & \text{otherwise} \end{cases}$$

where $\varepsilon_1 > 0$ and $\varepsilon_2 > 0$ are, respectively, the *tolerance for the population diversity* and *gene diversity* with respect to the best individual, $\hat{\mathbf{x}}(t)$.

$$\mathbf{x}_1 = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1j} & \cdots & x_{1n_x} \end{bmatrix} \qquad \hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \cdots & \hat{x}_j & \cdots & \hat{x}_{n_x} \end{bmatrix}$$

$$\mathbf{x}_2 = \begin{bmatrix} x_{21} & x_{22} & \cdots & x_{2j} & \cdots & x_{2n_x} \end{bmatrix}$$

$$\vdots$$

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \cdots & x_{ij} & \cdots & x_{in_x} \end{bmatrix}$$

$$\vdots$$

$$\mathbf{x}_{n_s} = \begin{bmatrix} x_{n_s1} & x_{n_s2} & \cdots & x_{n_sj} & \cdots & x_{n_sn_x} \end{bmatrix}$$

The migration operator is applied only when the diversity of the current population becomes too small, i.e.,

$$\frac{\overbrace{l_{11}(t)+l_{12}(t)+\cdots l_{ij}(t)\cdots+l_{n_sn_x}(t)}^{\text{exclude } \mathbf{x}_i(t)\neq\hat{\mathbf{x}}(t)}}{n_x(n_s-1)} < \varepsilon_1$$

Evolutionary Algorithm-Based Hybrids

Three variations:

1. Use **DE reproduction process** as a crossover operator in a simple **GA**.
2. Standard **Evolutionary Algorithm mutation operators** is used to increase DE population diversity by adding noise to the created trial vectors.

$$u_{ij}(t) = u_{ij}(t) + U(u_{\min,j}, u_{\max,j})$$

where $u_{\min,j}$ and $u_{\max,j}$ denotes the boundaries of the j^{th} element of the added noise.

3. **Rank-based** crossover and mutation operators
   - Rank-based selection is used to decide which individuals will take part to calculate difference vectors
   - At each generation, the cost of each individual in the population will be evaluated after crossover and mutation operations.
   - Individuals are arranged in ascending order: $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_s}$ where $f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots f(\mathbf{x}_{n_s})$ (assuming minimisation problem)
   - Note: crossover operation is performed before mutation operation.

KING'S College LONDON
University of London

Algorithm 6: Rank-Based Crossover Operator for Differential Evolution

Rank all individuals in ascending order of cost (assuming minimisation);
for $i = 1, 2, \ldots, n_s$ **do**
  $r \sim U(0,1)$;
  $\mathbf{x}_i'(t) = \mathbf{x}_i(t) + r\big(\mathbf{x}_{i+1}(t) - \mathbf{x}_i(t)\big)$;
  if $f(\mathbf{x}_i'(t)) < f(\mathbf{x}_{i+1}(t))$ **then**
    $\mathbf{x}_i(t) = \mathbf{x}_i'(t)$;
  end
end

When $i = n_s$, use $\mathbf{x}_1(t)$ as $\mathbf{x}_{i+1}(t)$.

KING'S College LONDON
University of London

Algorithm 7: Rank-Based Mutation Operator for Differential Evolution

Rank all individuals in ascending order of cost (assuming minimisation);
for $i = 1, 2, \ldots, n_s$ **do**
  $p_{m,i} = \frac{n_s - i + 1}{n_s}$;
  for $j = 1, 2, \ldots, n_x$ **do**
    $r_1 \sim U(0,1)$;
    if $r_1 > p_{m,i}$ **then**
      $r_2 \sim \{0,1\}$;
      $r_3 \sim U(0,1)$;
      if $r_2 = 0$ **then**
        $x'_{ij}(t) = x_{ij}(t) + (x_{\max,j} - x_{ij}) r_3 e^{-2t/n_t}$;
      else
        $x'_{ij}(t) = x_{ij}(t) - (x_{ij} - x_{\min,j}) r_3 e^{-2t/n_t}$;
      end
    end
  end
end

- $\mathbf{x}_i$ is the $\mathbf{x}_i$ after crossover
- t is the current generation number
- n_t is the maximum number of generations
- $r_2 \sim \{0,1\}$ means r_2 randomly takes either 0 or 1

Remark: Elitism is implemented, i,e., $\mathbf{x}_1$ does not mutate as $p_{m,1} = 1$ ($r_1 > p_{m,1}$ will never be satisfied in the above algorithm).

KING'S College LONDON
University of London

Algorithm 8: Rank-Based Differential Evolution

Set the generation counter, $t = 0$;
Initialize the control parameters, β and p_r;
Create and initialise the population† $C(0)$, of n_s individuals
while *stopping condition(s) not true* **do**
    for *each individual,* $\mathbf{x}_i \in C(t)$ **do**
        Evaluate the fitness, $f(\mathbf{x}_i(t))$;
        Perform the rank-based `crossover` operation in Algorithm 6;
        Perform the rank-based `mutation` operation in Algorithm 7;
        if $f(\mathbf{x}_i'(t))$ *is better than* $f(\mathbf{x}_i(t))$ **then**
            Add $\mathbf{x}_i'(t)$ to $C(t+1)$;
        else
            Add $\mathbf{x}_i(t)$ to $C(t+1)$;
        end
    end
    $t \leftarrow t+1$
end
Return the individual with the best fitness as the solution;

KING'S College LONDON
University of London

Two variations:

1. **Switching** between *Particle Swarm Optimisation* and Differential Evolution strategies by sharing the same population.
2. **DE-based PSO:**

   Update the personal best using DE mutation operation:

$$y'_{ij}(t) = \begin{cases} \hat{y}_{ij}(t) + 0.5\left(z_{1j}(t) - z_{2j}(t)\right) & \text{if } U(0,1) \leq p_m \\ y_{ij}(t) & \text{otherwise} \end{cases}$$

where $i = 1, 2, \ldots, n_s$; $j = 1, 2, \ldots, n_x$; $\mathbf{y}_i(t) = \begin{bmatrix} y_{i1} & y_{i2} & \ldots & y_{in_x} \end{bmatrix}$ denotes the personal best; $\hat{\mathbf{y}}_i(t) = \begin{bmatrix} \hat{y}_{i1} & \hat{y}_{i2} & \ldots & \hat{y}_{in_x} \end{bmatrix}$ denotes the local or global best; $\mathbf{z}_1(t) = \begin{bmatrix} z_{11} & z_{12} & \ldots & z_{1n_x} \end{bmatrix}$ and $\mathbf{z}_2(t) = \begin{bmatrix} z_{21} & z_{22} & \ldots & z_{2n_x} \end{bmatrix}$ denotes the randomly selected personal best positions.

The offspring $\mathbf{y}'_i(t)$ replace the personal best $\mathbf{y}_i(t)$ if $f(\mathbf{y}'_i(t)) < f(\mathbf{y}_i(t)), i = 1, 2, \ldots, n_s$ (assuming minimisation).

Two approaches:

1. **Dynamic Parameters**

   Probability of recombination (in Binomial Crossover): $p_r(t) = p_r(t-1) - (p_r(0) - 0.7)/n_t$

   Scale factor (in Mutation): $\beta(t) = \beta(t-1) - (\beta(0) - 0.5)/n_t$

   where, e.g., $p_r(0) = 1.0$ and $\beta(0) = 0.3$.

2. **Self-Adaptive Parameters** (in Mutation)

$$\beta(t) = \begin{cases} \max\left\{\beta_{\min}, 1 - \left|\frac{f_{\max}(t)}{f_{\min}(t)}\right|\right\} & \text{if } \left|\frac{f_{\max}(t)}{f_{\min}(t)}\right| < 1 \\ \max\left\{\beta_{\min}, 1 - \left|\frac{f_{\min}(t)}{f_{\max}(t)}\right|\right\} & \text{otherwise} \end{cases}$$

- $\beta(t) \in [\beta_{\min}, 1]$; $\beta_{\min}$ is a lower bound on the scale factor;
- $\max(\cdot)$ is the maximum operator
- $f_{\min}(t)$ and $f_{\max}(t)$ are respectively the minimum and maximum cost values for the current population, $C(t)$
- As $f_{\min}(t)$ approaches $f_{\max}(t)$, the diversity of the population decreases, and the value of $\beta(t)$ approaches $\beta_{\min}$ which is to ensure smaller step sizes when the population starts to converge, otherwise larger step size to favour exploration