Multi-objective optimization

Most real-world optimization problems involve multiple conflicting objectives (*Konak, Coit & Smith, 2006*). Hence, multi-objective optimization problems have various practical applications. The use of evolutionary algorithms has been motivating for solving such problems. Because of the population-based nature of these algorithms, we obtain a set of solutions on every run. In a multi-objective optimization problem, the definition of optimality is not as simple as in single-objective optimization. When the optimal solution of an objective function conflicts with an optimal solution of another objective function, the problem becomes challenging. Therefore, to solve such problems, it is necessary to find a trade-off between objective functions. The obtained solutions of multi-objective algorithms are called nondominated solutions or Pareto-optimal solutions. Theoretically, if a multi-objective optimization problem is a minimization problem, it is formulated as follows (*Mirjalili et al., 2016*).

$$\begin{aligned} &\text{Min} \quad F(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x})] \\ &\text{s.t.} \quad L_i \leq \boldsymbol{x_i} \leq U_i, \quad i = 1, 2, \ldots, d \end{aligned} \tag{4}$$

Subject to the following equality and inequality constraints:

$$\begin{aligned} &g_i(\boldsymbol{x}) \leq 0 \quad j = 1, 2, \ldots, J \\ &h_k(\boldsymbol{x}) = 0 \quad k = 1, 2, \ldots, K \end{aligned} \tag{5}$$

where M is the number of objectives, and d is the number of decision variables (dimension) of solution $\boldsymbol{x}$, so that $\boldsymbol{x}_i$ should be in interval $[L_i, U_i]$ (i.e., box-constraint). Finally, f_i is the objective function that should be minimized. To compare two candidate solutions in multi-objective problems, we can use the concept of Pareto dominance. Mathematically, the Pareto dominance is defined as follows. If $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ and $\acute{\boldsymbol{x}} = (\acute{x}_1, \acute{x}_2, \ldots, \acute{x}_d)$ are two vectors in the search space, $\boldsymbol{x}$ dominates $\acute{x}(\boldsymbol{x} \succ \acute{\boldsymbol{x}})$ if and only if

$$\begin{aligned} &\forall i \in \{1, 2, \ldots, M\}, f_i(\boldsymbol{x}) \leq f_i(\acute{\boldsymbol{x}}) \wedge \\ &\exists j \in \{1, 2, \ldots, M\} : f_j(\boldsymbol{x}) < f_j(\acute{\boldsymbol{x}}) \end{aligned} \tag{6}$$

This means that solution $\boldsymbol{x}$ dominates solution $\acute{\boldsymbol{x}}$ (is better) if and only if the objective values of $\boldsymbol{x}$ are better than or equal to all objective values of $\acute{\boldsymbol{x}}$ (is not worse than $\acute{\boldsymbol{x}}$ in any of the values of the objective functions) and it has a better value than $\boldsymbol{x}$ in at least one of the objective functions. If the solution $\boldsymbol{x}$ is better than $\acute{\boldsymbol{x}}$ in all objectives, we call strong dominance but in the case that they have at least one equal objective, the weak dominance happens. All nondominated solutions construct a Pareto front.

Non-dominated Sorting Genetic Algorithm II (NSGAII)

NSGA-II

Step 1 Combine parent and offspring populations and create $R_t = P_t \cup Q_t$

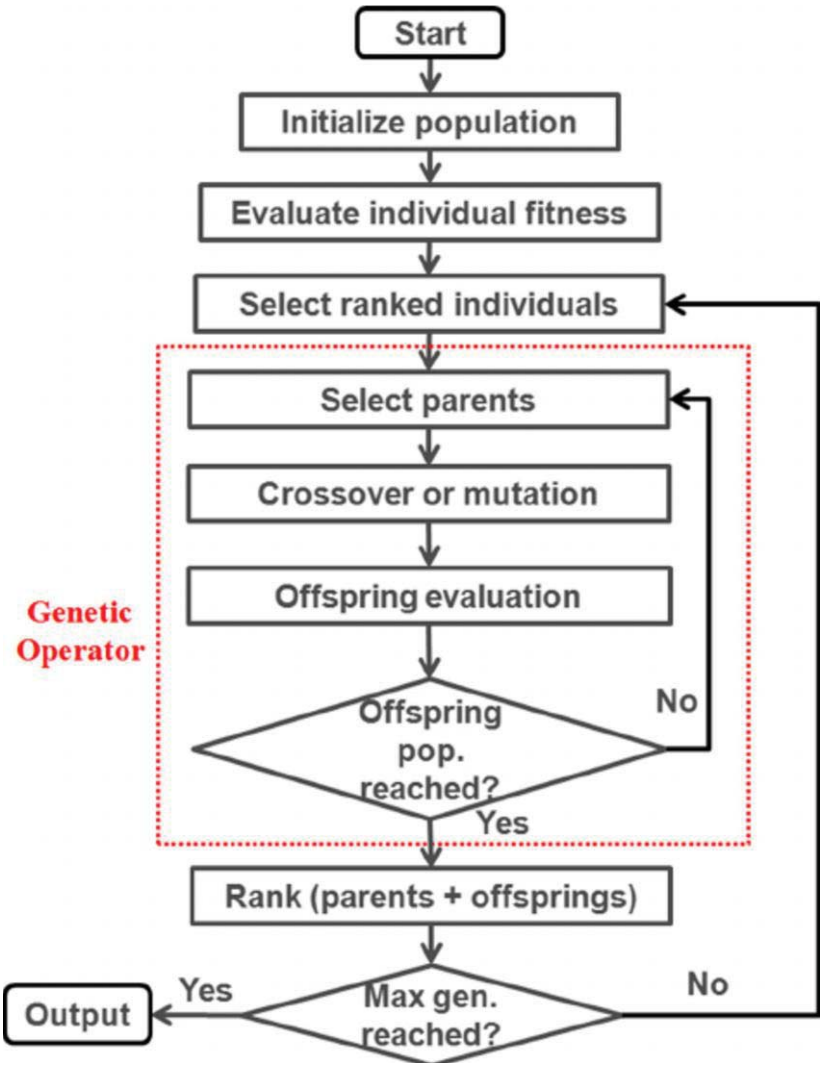
Perform a non-dominated sorting to R_t and identify different fronts: F_i, i = 1, 2,…

Step 2 Set new population P_{t+1} = null. Set a counter i = 1.

Until $|P_{t+1}| + |F_i| < N$, perform $P_{t+1} = P_{t+1} \cup F_i$ and $i = i + 1$.

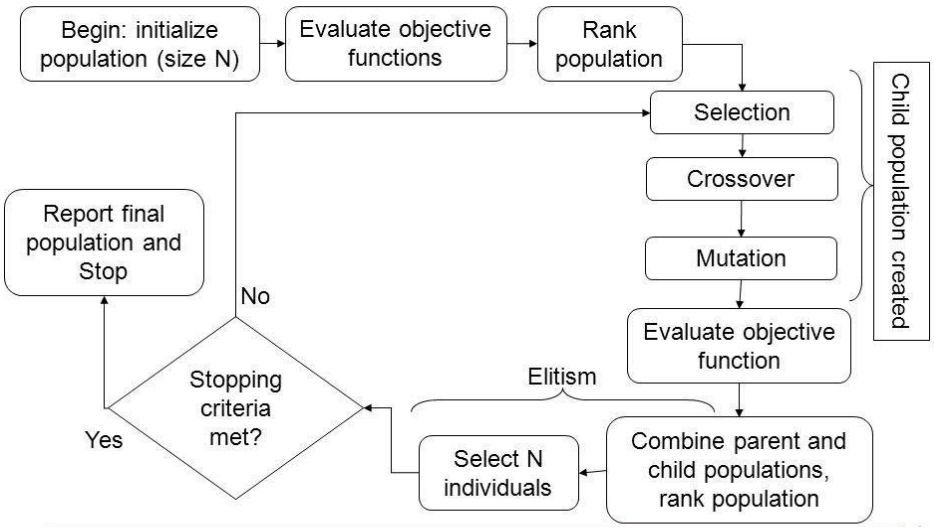
Step 3 Perform the Crowding-sort(F_i,<c) procedure given below and include the most widely spread ($N - |P_{t+1}|$) solutions by using the crowding distance values in the sorted F_i to P_{t+1}.

Step 4 Create offspring population Q_{t+1} from P_{t+1} by using the crowded tournament selection, crossover and mutation operators.



Flowchart NSGAII

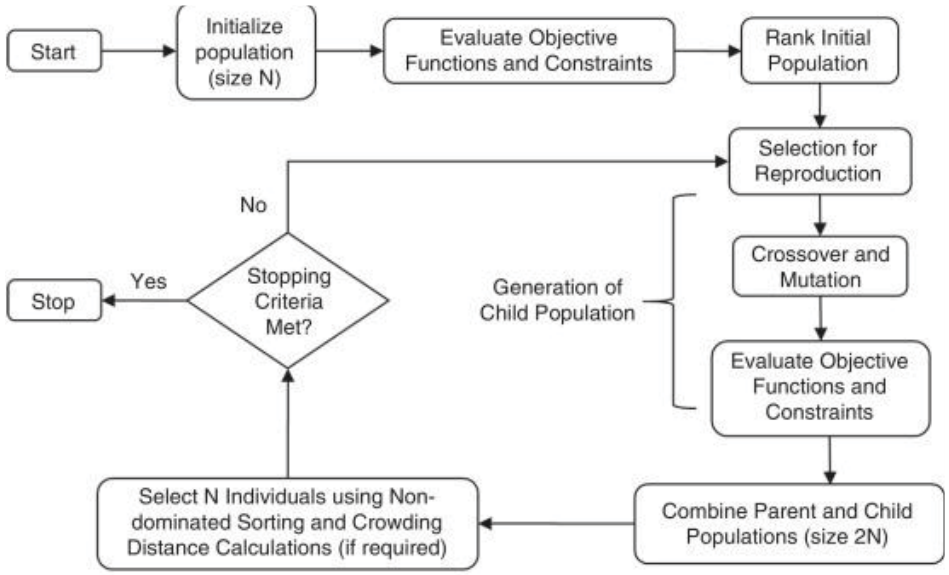
Flowchart NSGAII



Pseudocode NSGAII

```
Initialize Population
Generate N random solutions and insert into Population
for (i = 1 to MaxGenerations) do
    Generate ChildPopulation of size N
    Select Parents from Population
    Create Children from Parents
    Mutate Children
    Combine Population and ChildPopulations into CurrentPopulation with size
     2N
    for each individual in CurrentPopulation do
        Assign rank based on Pareto – Fast non-dominated sort
    end for
    Generate sets of non-dominated vectors along PF known
    Loop (inside) by adding solutions to next generation of Population starting
     from the best front
        until N solutions found and determine crowding distance between
          points on each front
end for
Present results
```

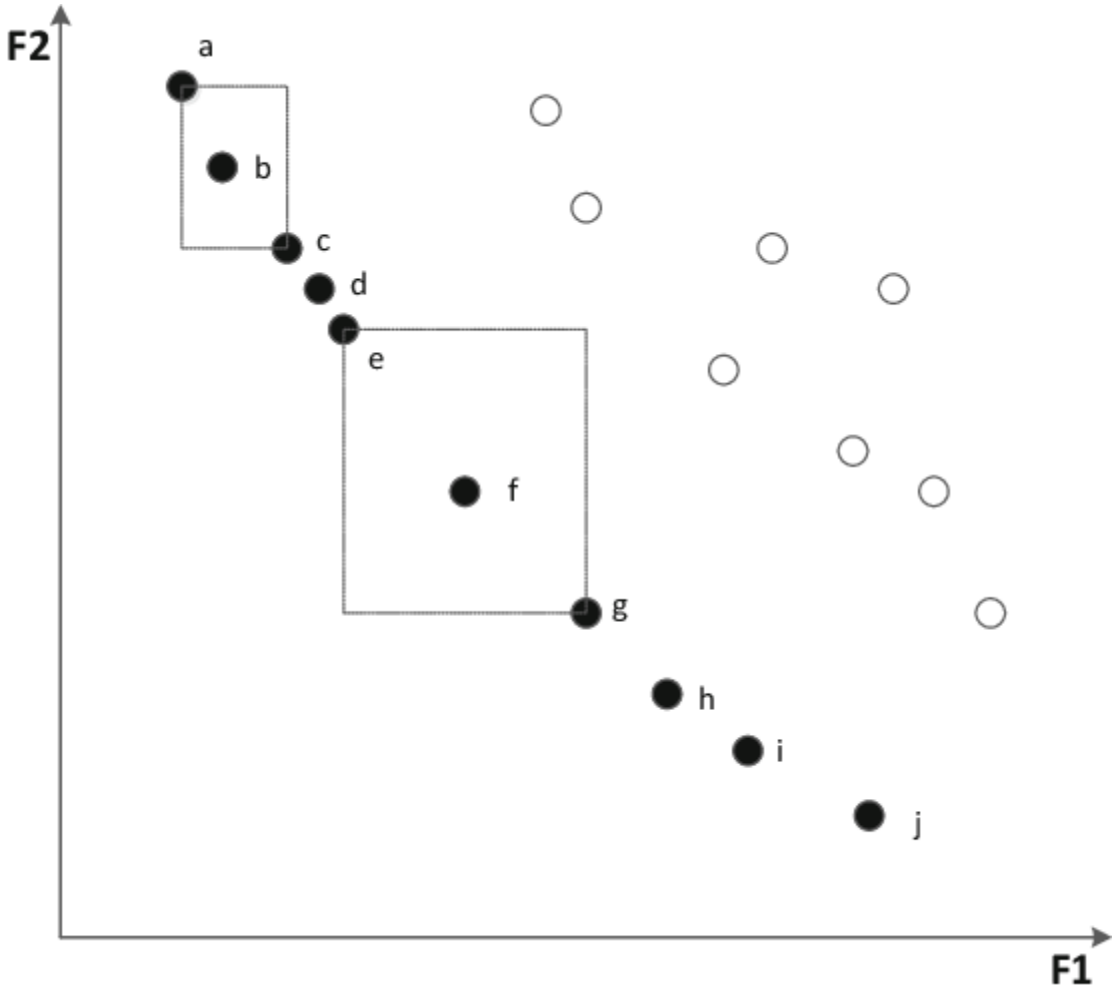
Flowchart NSGAII



Pseudocode NSGAII

```
procedure NSGA-II
   Generate a random initial population (N-size)
   Evaluate objectives
   Rank initial population based on Pareto dominance
   Compute the crowding distance for each solution in the initial population
   for i = 1 to Number of Epochs
      Generate offspring (through binary tournament, crossover and mutation)
      Mix current population and offspring
      Rank mixed population based on Pareto dominance
      Compute the crowding distance for each solution in the mixed population
      Select the N best individuals from the mixed population to form the new one
   end for
end procedure
```

Crowding distance



Crowding-sort(F_i<c)

Step 1 Call the number of solutions in F as $l = |F|$. For each i in the set, first assign crowding distance, $d_i = 0$.

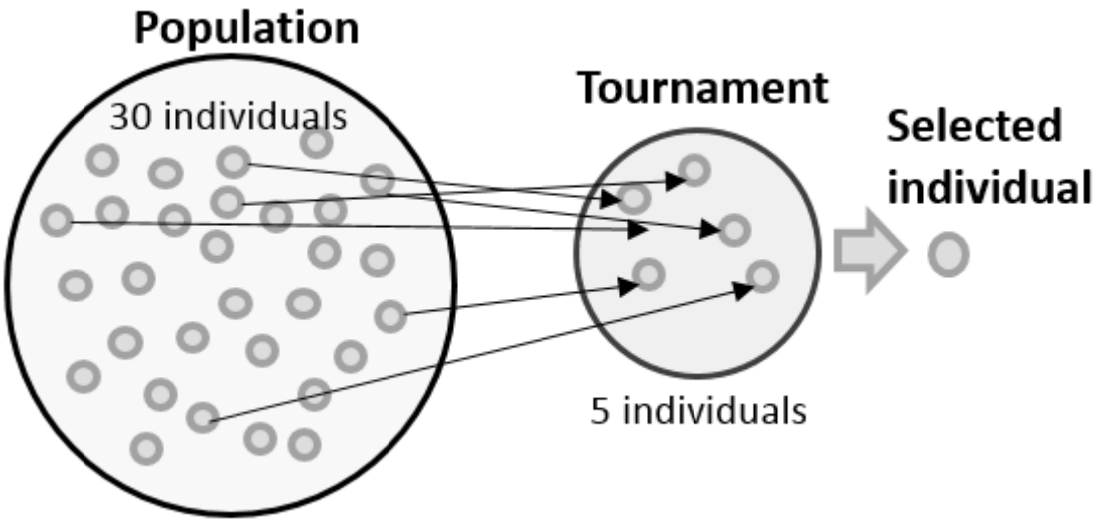
Step 2 For each objective function $m = 1,2,...,M$, sort the set in worse order of f_m or, find the sorted indices vector:

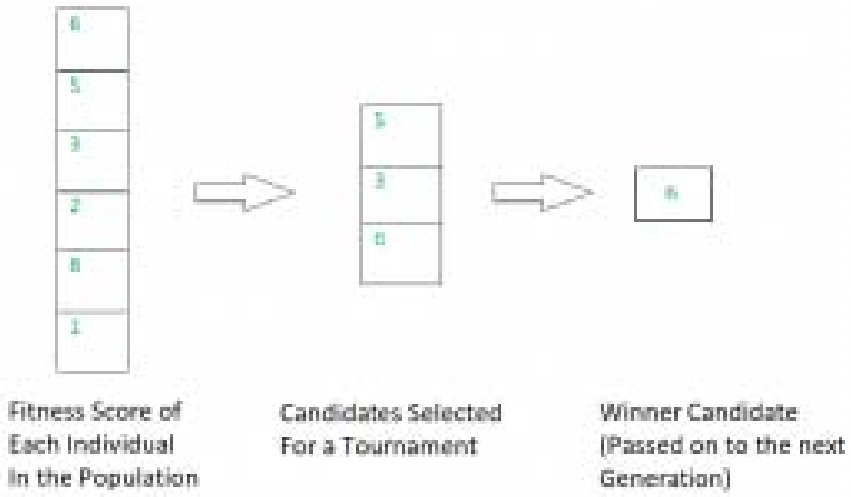
$I^m = \text{sort}(f_m, >)$

Step 3 For $m = 1,2,...,M$, assign a large distance to the boundary solutions, or $d_{I_1^m} = d_{I_l^m} = \infty$, and for all other solutions j = 2 to (l - 1), assign:

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{max} - f_m^{min}}.$$

Tournament selection





Simulated Binary Crossover (SBX)

Simulated Binary Crossover

Step 1 Choose a random number $u \in [0,1)$.

Step 2 Calculate

$$\beta_q = \begin{cases} (u\alpha)^{\frac{1}{\eta_c+1}}, & \text{if } u \le \frac{1}{\alpha}; \\ \left(\frac{1}{2-u\alpha}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise,} \end{cases} \tag{9}$$

where

$$\alpha = 2 - \beta^{-(\eta_c+1)},$$

$$\beta = 1 + \frac{2}{y_2 - y_1} min[(y_1 - y_l),(y_u - y_2)].$$

y_l and y_u: lower and upper limits of y

η_c : distribution index for crossover

Step 3 Compute children solutions:

$$\begin{aligned} c_1 &= 0.5[(y_1 + y_2) - \beta_q |y_2 - y_1|], \\ c_2 &= 0.5[(y_1 + y_2) + \beta_q |y_2 - y_1|]. \end{aligned} \tag{10}$$

Polynomial Mutation (PLM) operator

Step 1 Choose a random number $u \in [0,1)$.

Step 2 Calculate

$$\delta_q = \begin{cases} \left[2u + (1-2u)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} - 1, & if\ u \le 0.5, \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}}, & otherwise \end{cases}$$

where

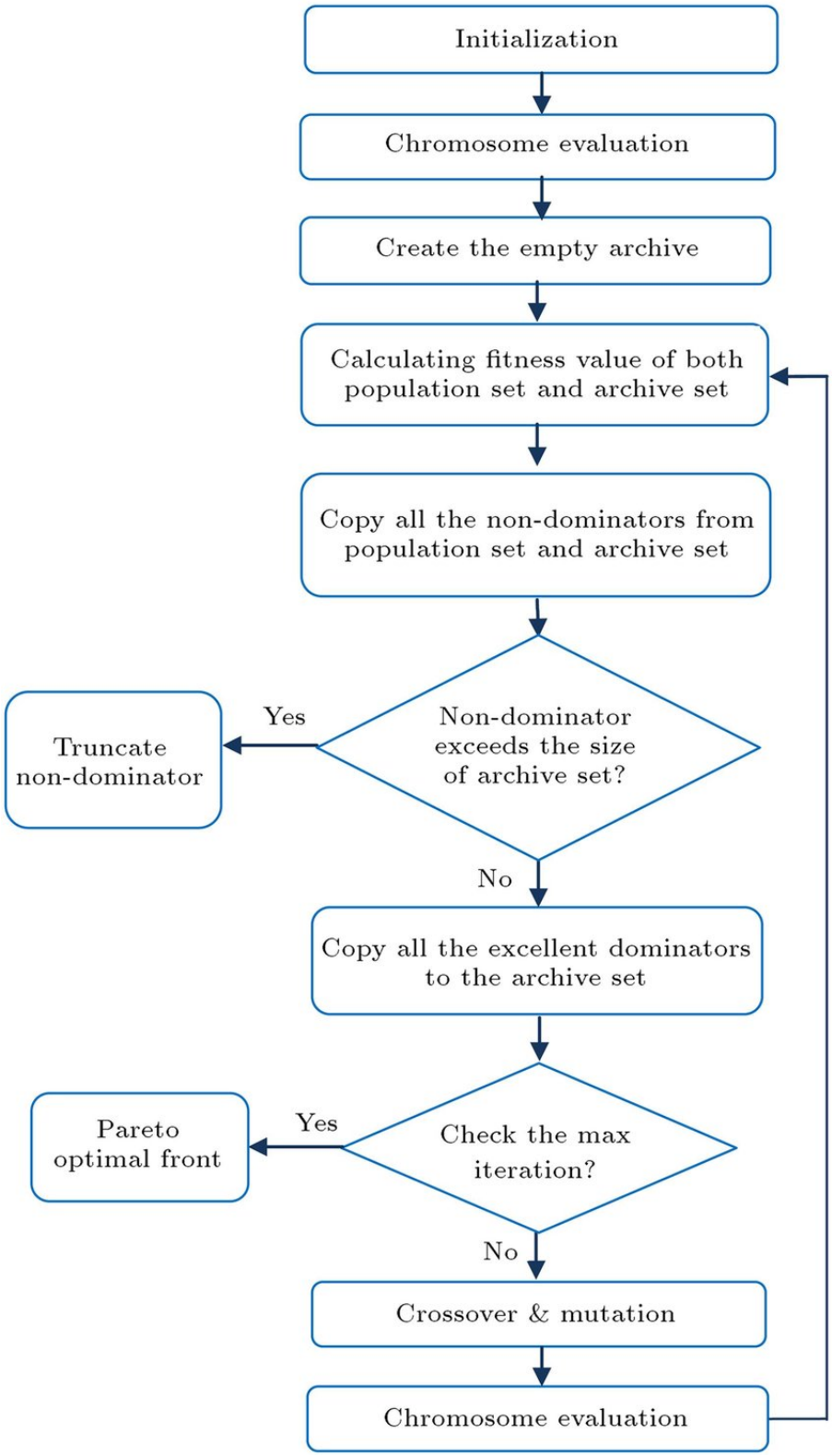
$$\delta = min[(y - y_l), (y_u - y)]/(y_u - y_l)$$

η_m : distribution index for mutation

Step 3 Calculate the mutated child:

$$c = y + \delta_q (y_u - y_l).$$

Strength Pareto Evolutionary Algorithm 2 (SPEA2)



SPEA2 Algorithm

Input: N (population size)

$\overline{N}$ (archive size)

T (maximum number of generations)

Output: **A** (nondominated set)

Step 1 **Initialization**: Generate an initial population P_0 and create the empty archive (external set) $P_0 = \phi$. Set $t = 0$.

Step 2 **Fitness assignment**: Calculate fitness values of individuals in P_t and $\overline{P}_t$.

Step 3 **Environmental selection**: Copy all nondominated individuals in P_t and $\overline{P}_t$ to $\overline{P}_{t+1}$. If size of $\overline{P}_{t+1}$ exceeds $\overline{N}$ then reduce $\overline{P}_{t+1}$ by means of the truncation operator, otherwise if size of $\overline{P}_{t+1}$ is less than $\overline{N}$ then fill $\overline{P}_{t+1}$ with dominated individuals in P_t and $\overline{P}_t$.

Step 4 **Termination**: If $t \geq T$ or another stopping criterion is satisfied then set **A** to the set of decision vectors represented by the nondominated individuals in $\overline{P}_{t+1}$. Stop.

Step 5 **Mating selection**: Perform binary tournament selection with replacement on $\overline{P}_{t+1}$ in order to fill the mating pool.

Step 6 **Variation**: Apply recombination and mutation operators to the mating pool and set $\overline{P}_{t+1}$ to the resulting population. Increment generation counter ($t = t + 1$) and go to Step 2.

Multi-Objective Evolutionary Algorithms based on Decomposition (MOEA/D)

Algorithm 2. The MOEA/D general framework

Input:

- MOP
- the number of the sub-problems considered in MOEA/D, N
- a uniform spread of N weight vectors: $\lambda^1, \ldots, \lambda^N$
- the number of the weight vectors in the neighborhood of each weight vector, T
- the maximum number of generations, gen_{max}

Output:

- EP

Step 0 - Setup:

- Set EP = Ø
- $gen = 0$

Step 1 – Initialization

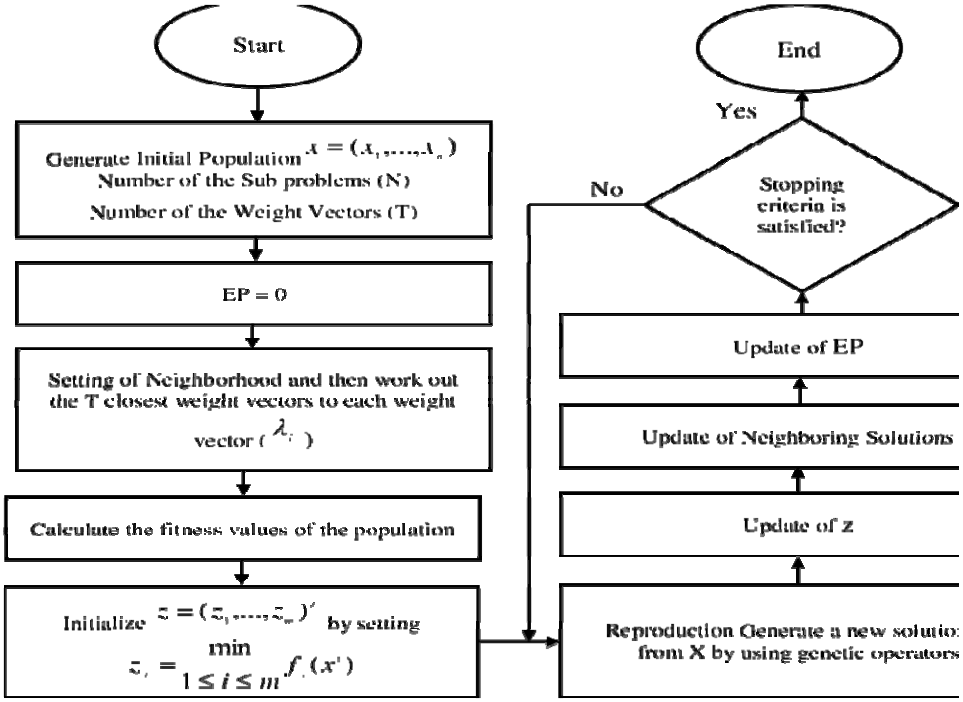
- Uniformly randomly generate an initial internal population, $IP_0 = \{x^1, \ldots, x^N\}$ and set $FV^i = F(x^i)$.
- Initialize $z = (z_1, \ldots, z_n)^T$ by a problem-specific method.
- Compute the Euclidean distances between any two weight vectors and then work out the T closest weight vectors to each weight vector. $\forall i = 1, \ldots, N$, set $B(i) = \{i_1, \ldots, i_T\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.

Step 2 – Update: For $i = 1, \ldots, N$

- Genetic operators: Randomly select two indexes k, l from $B(i)$, and then generate a new solution y from x^k and x^l by using genetic operators.
- Update of z , $\forall j = 1, \ldots, n$, if $z_j < f_j(y)$, then set $z_j = f_j(y)$.
- Update of Neighboring Solutions: For each index $j \in B(i)$, if $g^{te}(y|\lambda^j, z) \leq g^{te}(x^j|\lambda^j, z^*)$, then set $x^j = y$ and $FV^j = F(y^j)$.
- Update of EP: Remove from EP all the vectors dominated by $F(y)$.
  Add $F(y)$ to EP if no vector in EP dominate $F(y)$.

Step 3 – Stopping criteria

- If $gen = gen_{max}$, then stop and output EP,
  otherwise $gen = gen + 1$, go to Step 2.


Start
Generate Initial Population $x = (x_1,\ldots,x_n)$
Number of the Sub problems (N)
Number of the Weight Vectors (T)
EP = 0
Setting of Neighborhood and then work out the T closest weight vectors to each weight vector (λ_i)
Calculate the fitness values of the population
Initialize $z = (z_1,\ldots,z_m)'$ by setting $z_i = \min_{1 \le i \le m} f_i(x')$
Reproduction Generate a new solutio from X by using genetic operators
Update of z
Update of Neighboring Solutions
Update of EP
Stopping criteria is satisfied?
No
Yes
End

DECOMPOSITION OF MULTIOBJECTIVE OPTIMIZATION

- *A. Weighted Sum Approach*

$\lambda = (\lambda_1, \ldots, \lambda_m)^T$: *be a weight vector*

$\sum_{i=1}^{m} \lambda_i = 1$

$F(x) = (f_1(x), \ldots, f_m(x))^T$: *Object solution*

$$Maximize \quad g^{ws}(x \mid \lambda) = \sum_{i=1}^{m} \lambda_i f_i(x)$$

MOEA/D algorithm

Inputs:

- Multi-Objective Optimization (MOO) problem.
- N : The number of sub problems (the population size).
- W: A set of N evenly sampled weight vectors $\{\lambda_1, \ldots., \lambda_N\}$.
- T : The neighborhood size
- A stopping criterion

Steps:

1. Initialization:
   - Generate the initial population $x^1 \rightarrow x^N$ at random such that, N is the population size where, x^{i} is the current solution to the ith sub problem.
   - Initialize the reference point z as mentioned in Eq.(2).
   - Calculate the Euclidean distances for each couple of weight vectors to determine the neighbors for each vector (the set of its T closest weight vectors). For each $i = 1 \rightarrow N$, set $B(i) = \{i_1, \cdots, i_T\}$, such that $\lambda^{i_1}, \cdots, \lambda^{i_T}$ are the T closest weight vector to λ^{i}.
   - For each $i = 1 \rightarrow N$:Evaluate the fitness value $F(x^{\mathrm{i}})$.
2. Update:

   For $i = 1 \rightarrow N, do$

   - Reproduction: Select at random two indexes k, l from $B(i)$ then, by using the genetic operators (i.e crossover and mutation) generate a new solution from x^k and x^i.
   - Repair: Apply a problem specific improvement heuristic on y to generate y^i.
   - Update z: For each $j = 1 \rightarrow m$, If $z_j < f_j(y')$ then set $z_j = f_j(y^i)$.
   - Neighboring solutions update: For each index $j \in B(i)$, $if\ g^{te}(y'|\lambda^j, z) \leq g^{te}(x^j|\lambda^j, z),\ then\ set\ x^j = y'$ and $FV^j = F(y')$.
3. If stopping criteria is met, then stop. Else go back to step 2.

NSGAIII

Algorithm 1 Generation t of NSGA-III procedure

Input: H structured reference points Z^s or supplied aspiration points Z^a, parent population P_t

Output: P_{t+1}

$S_t = \emptyset$, $i = 1$
Q_t = Recombination+Mutation(P_t)
$R_t = P_t \cup Q_t$
$(F_1, F_2, \ldots)$ = Non-dominated-sort(R_t)
repeat
  $S_t = S_t \cup F_i$ and $i = i + 1$
until $|S_t| \geq N$
Last front to be included: $F_l = F_i$
if $|S_t| = N$ **then**
  $P_{t+1} = S_t$, break
else
  $P_{t+1} = \cup_{j=1}^{l-1} F_j$
  Points to be chosen from F_l: $K = N - |P_{t+1}|$
  Normalize objectives and create reference set Z^r: `Normalize`($\mathbf{f}^n, S_t, Z^r, Z^s, Z^a$)
  Associate each member $\mathbf{s}$ of S_t with a reference point: $[\pi(\mathbf{s}), d(\mathbf{s})]$ =`Associate`(S_t, Z^r) % $\pi(\mathbf{s})$: closest reference point, d: distance between $\mathbf{s}$ and $\pi(\mathbf{s})$
  Compute niche count of reference point $j \in Z^r$: $\rho_j = \sum_{\mathbf{s} \in S_t/F_l} ((\pi(\mathbf{s}) = j)\ ?\ 1\ :\ 0)$
  Choose K members one at a time from F_l to construct P_{t+1}: `Niching`($K, \rho_j, \pi, d, Z^r, F_l, P_{t+1}$)
end if

4 Scalarization techniques

Classically, multiobjective optimization problems are often solved using scalarization techniques (see, for instance, Miettinen 2012). Also in the theory and practice of evolutionary multiobjective optimization scalarization plays an important role, especially in the so-called decomposition based approaches.

In brief, scalarization means that the objective functions are aggregated (or reformulated as constraints), and then a constrained single-objective problem is solved. By using different parameters of the constraints and aggregation function, it is possible to obtain different points on the Pareto front. However, when using such techniques, certain caveats have to be considered. In fact, one should always ask the following two questions:

1. Does the optimization of scalarized problems result in efficient points?
2. Can we obtain all efficient points or vectors on the Pareto front by changing the parameters of the scalarization function or constraints?

4.1 Linear weighting

A simple means to scalarize a problem is to attach non-negative weights (at least one of them positive) to each objective function and then to minimize the weighted sum of objective functions. Hence, the multiobjective optimization problem is reformulated to:

Definition 14 Linear Scalarization Problem. The linear scalarization problem (LSP) of an MOP using a weight vector $w \in \mathbb{R}^m_{\succ \mathbf{0}}$, is given by

$$\text{minimize} \sum_{i=1}^{m} w_i f_i(x), x \in \mathcal{X}.$$

4.1.2 ϵ-constraint method

A rather straightforward approach to turn a multiobjective optimization problem into a constraint single-objective optimization problem is the ϵ-constraint method.

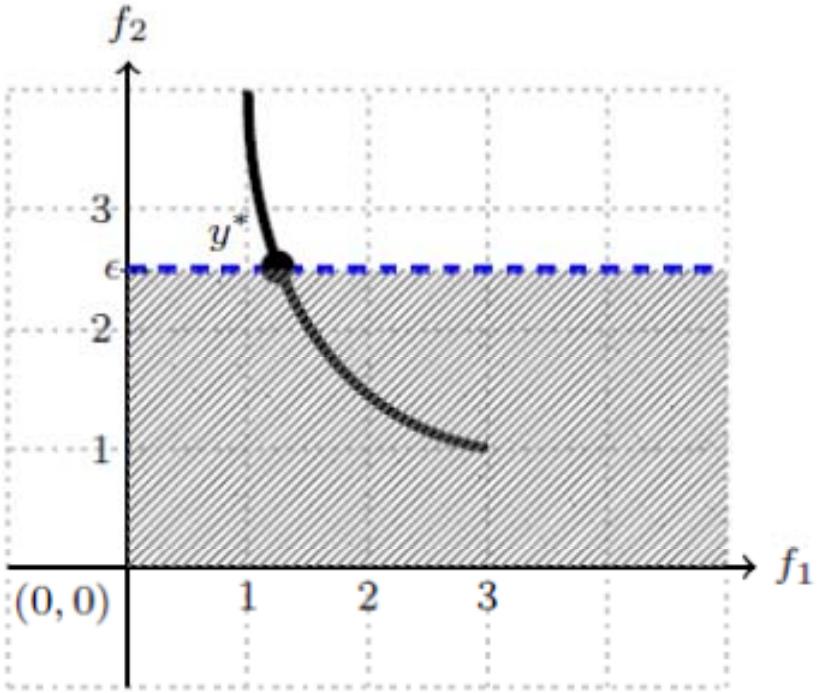
Definition 18 ϵ–constraint Scalarization. Given a MOP, the ϵ–constraint scalarization is defined as follows. Given $m-1$ constants $\epsilon_1 \in \mathbb{R}, \ldots, \epsilon_{m-1} \in \mathbb{R}$,

$$\text{minimize } f_1(\mathbf{x}), \text{ subject to } g_1(\mathbf{x}) \leq \epsilon_1, \ldots, g_{m-1}(\mathbf{x}) \leq \epsilon_{m-1},$$

where $f_1, g_1, \ldots, g_{m-1}$ constitute the m components of vector function $\mathbf{f}$ of the multiobjective optimization problem (see Definition 1).

The method is illustrated in Fig. 4 (left) for $\epsilon_1 = 2.5$ for a biobjective problem. Again, by varying the constants $\epsilon_1 \in \mathbb{R}, \ldots, \epsilon_{m-1} \in \mathbb{R}$, one can obtain different points on the Pareto front. And again, among the solutions weakly dominated solutions may occur. It can, moreover, be

difficult to choose an appropriate range for the ϵ values, if there is no prior knowledge of the location of the Pareto front in $\mathbb{R}^m$.


f_2
3
y^*
ϵ
2
1
f_1
$(0,0)$
1
2
3

Tchebycheff method

As stated before, multi-objective optimization problems can be solved by different methods. Traditional multi-objective optimization methods seek a way to convert the multi-objective problem into a single-objective problem. One of these methods is the Tchebycheff method (*Jaszkiewicz, 2002*), which was used in this study to solve multi-objective subproblems. The Tchebycheff method looks for the optimal solutions that have the minimum distance from a reference point. The single-objective optimization problem is defined as Eq. (8).

$$\begin{aligned} &\text{Minimize} \quad g^{te}(x|\lambda_o, z^*) = \max\{\lambda_i|f_i(x) - z_i^*|\} \\ &\text{subject to} \quad x \in S, \end{aligned} \tag{8}$$

where $z^* = (z_{1^*},\ldots,z_{m^*})^T$ is a reference point used to evaluate the quality of the obtained solutions, m is the number of objective functions, and S is the search space. According to this equation, the distances between the objective function values of each solution x and reference point z^* are calculated. The single-objective optimization problem is regarded as minimizing the maximum of these distances. A uniform weight vector $\lambda = (\lambda_1, \lambda_1,\ldots, \lambda_m)$ is defined for each solution such that $\sum_i^m \lambda_i = 1$. Therefore, weight λ_i is assigned to the objective function f_i. To obtain each optimal solution of the minimization problem defined in Eq. (8), we need to find an appropriate weight vector. The obtained optimal solution would be one of the Pareto optimal solutions. As a result, the traditional methods are time-consuming because of continuous changes in the weights required to obtain the best solutions. Therefore, we consider a set of distributed weight vectors in the decomposition-based evolutionary methods for all the subproblems. Reference point selection is another issue that should be considered in the Tchebycheff method. For a minimization problem, the minimum value obtained for each objective function can be a reference point.

$$z_i^* = \min f_i(x)|x \in S \tag{9}$$

Therefore, the value of the reference point is also updated after each iteration. Figure 1 shows the Tchebycheff method for obtaining an optimal solution on the Pareto front. As an example, we show that the reference point has been placed at the center of the coordinates, where the values of both objective functions are minimal. We show a sample

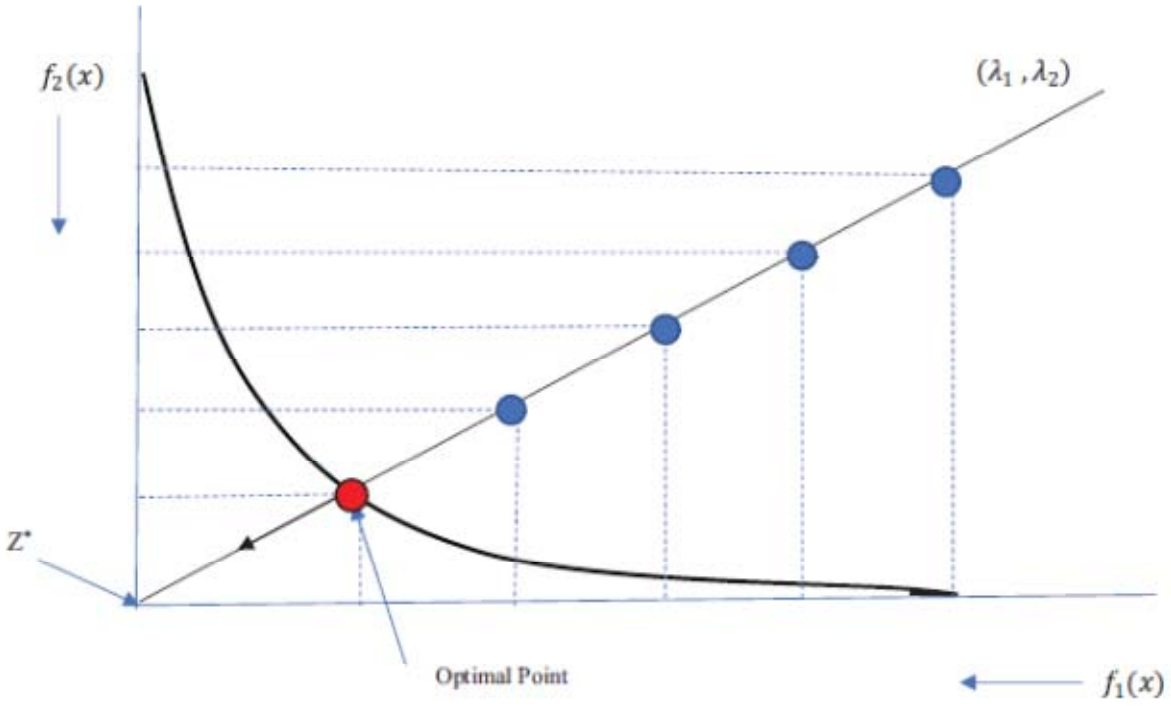


Figure 1 **Illustration of Tchebycheff method.**

weight vector (λ_1, λ_2), and the solutions from each iteration are shown in blue. The solutions converge toward the reference point in the direction of the weight vector until the optimal point on the Pareto front (in red) is obtained. At each iteration, the previous solution is replaced with a new solution if the new one outperforms the previous one.

Assessment metrics

The hypervolume indicator (*Auger et al., 2009*) is one of the well-known criteria for evaluating multi-objective optimization methods. This indicator evaluates multi-objective optimization algorithms according to both diversity and convergence to the optimal Pareto front. This indicator determines the volume of the n-dimensional space that is surrounded by a set of points. The number of dimensions would be equal to the number of objectives. Therefore, the volume of the two-dimensional space that is surrounded by the Pareto solutions is calculated for the feature selection problem. The larger this space, the wider the points (surrounding a larger space) and the closer the Pareto front to the optimal Pareto. A reference point is needed to acquire the intended volume. The selection of a reference point is one of the challenges for the calculation of the hypervolume indicator. For example, a point with the worst obtained values among the objective functions is an option for this purpose. As shown in Fig. 6, the volume of the gray regions between the solutions on the Pareto front and the reference point would be considered as the hypervolume indicator. The measure is defined in Eq. (14) (*Auger et al., 2009*):

$$HV(A) = \text{vol}\left(\bigcup_{a\in A}[f_1(a), r_1] \times [f_2(a), r_2] \times \ldots \times [f_M(a), r_M]\right), \tag{14}$$

where $a \in A$ is a point at which all candidate solutions are weakly dominated by it.

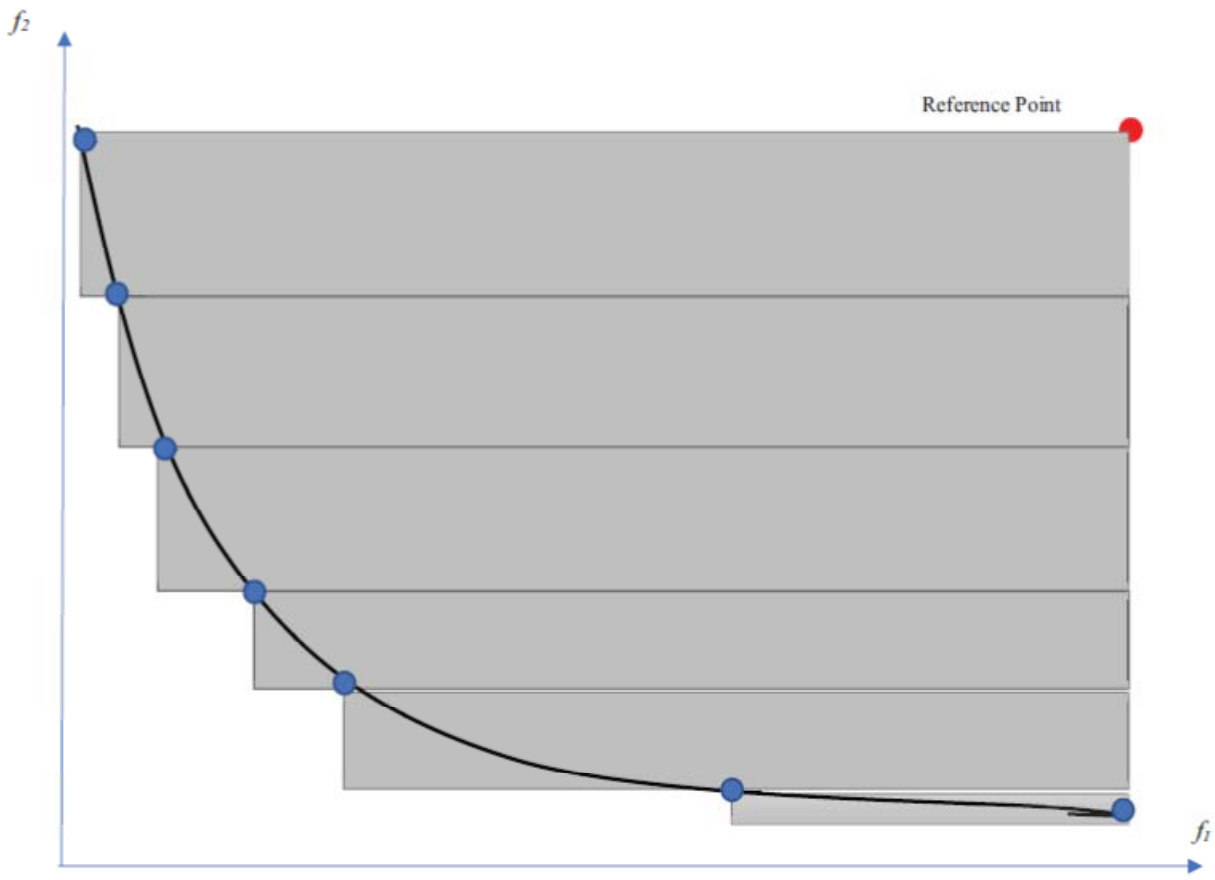


Figure 6 **Hypervolume indicator.**

Παράδειγμα Λειτουργίας ενός Γενετικού Αλγορίθμου

Maximizing the values of x^2 for integers in the range [0, 31]

For the representation we use a simple five-bit binary encoding mapping integers (phenotypes) to bit-strings (genotypes).

For parent selection we use a fitness proportional mechanism, where the probability p_i that an individual i in population P is chosen to be a parent is $p_i = f(i)/\sum_{j=1}^{P} f(j)$

Mutation is executed by generating a random number (from a uniform distribution over the range [0, 1]) in each bit position, and comparing it to a fixed threshold, usually called the **mutation rate**. If the random number is below that rate, the value of the gene in the corresponding position is flipped.

Recombination is implemented by the classic one-point crossover. This operator is applied to two parents and produces two children by choosing a random crossover-point along the strings and swapping the bits of the parents after this point.

Table 1. The x^2 example, 1: initialisation, evaluation, and parent selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
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
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Table 1 shows a random initial population of four genotypes, the corresponding phenotypes, and their fitness values. The cycle then starts with selecting the parents to seed the next generation. The fourth column of Table shows the expected number of copies of each individual after parent selection, being $\frac{f(i)}{Average\ fitness}$

As can be seen, these numbers are not integers; rather they represent a probability distribution, and the mating pool is created by making random choices to sample from this distribution.

The column “Actual count” stands for the number of copies in the mating pool.

Next the selected individuals are paired at random, and for each pair a random point along the string is chosen.

Table 2 shows the results of crossover on the given mating pool for crossover points after the fourth and second genes, respectively, together with the corresponding fitness values.

Table 2

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Table 3

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

This example shows a progress: the average fitness grows from 293 to 588.5, and the best fitness in the population from 576 to 729 after crossover and mutation.