Multi-Objective Optimization (MOO)

Arna Fariza

Definition of Optimization

- The process of finding the best solution from a set of feasible solutions for a problem.
- **Single-Objective vs. Multi-Objective Optimization**
  - Single-Objective Optimization: Focuses on optimizing a single criterion, such as minimizing cost or maximizing profit.
  - Multi-Objective Optimization: Involves optimizing two or more conflicting objectives simultaneously.

MOO

- Multi-objective optimization or
- Multi-objective programming or
- Vector optimization or
- Multicriteria optimization or
- Multiattribute optimization or
- Pareto optimization

MOO

- is an area of multiple criteria decision making that is concerned with mathematical optimization problems involving more than one objective function to be optimized simultaneously.
- **Minimizing cost** while **maximizing comfort** while buying a car, and **maximizing** performance while **minimizing** fuel consumption and emission of pollutants of a vehicle are examples of multi-objective optimization problems involving two and three objectives, respectively.
- In practical problems, there can be more than three objectives.

Key Concepts in Multi-Objective Optimization

- **Objective Functions:** Functions that represent the criteria to be optimized.
- **Conflicting Objectives:** In real-world problems, objectives often conflict. For example, in engineering, you may want to minimize weight and maximize strength.
- **Pareto Optimality:**
  - **Pareto Front:** A set of non-dominated solutions where no objective can be improved without worsening another.
  - **Dominance:** A solution dominates another if it is better in at least one objective without being worse in others.
  - **Pareto Optimal Solutions:** Solutions that are not dominated by any other feasible solution.

Practical Example

- Example Problem: Consider optimizing a drone’s flight time (**objective 1**) and camera resolution (**objective 2**).
    - **Objective 1:** Maximize flight time by reducing battery usage.
    - **Objective 2:** Maximize camera resolution by using higher-powered components.
    - **Conflicting Nature:** A higher resolution camera consumes more power, reducing flight time.
    - **Solution:** Using a multi-objective algorithm (genetic algorithm) to find a balance between both objectives and present a set of Pareto-optimal designs.

Methods of Solving Multi-Objective Optimization Problems

- Scalarization Methods
- Pareto-Based Methods
- Evolutionary Algorithms
- Other Methods

Scalarization Methods

- **Weighted Sum Method:** Converts the multi-objective problem into a single-objective one by assigning weights to each objective.
- **Limitations:** Does not work well for non-convex Pareto fronts.

Example: Optimizing a Product Design for Cost and Quality

- Problem Statement:

  Suppose you're designing a product, and you want to optimize two conflicting objectives:

  - **Minimize cost** (Objective 1: $f_1(x)$).
  - **Maximize quality** (Objective 2: $f_2(x)$).

  These two objectives conflict because improving quality usually increases costs. You decide to use the Weighted Sum Method to handle this multi-objective problem.

Step 1: Define the Objective Functions

Let's assume the objectives are represented as functions:

$f_1(x)$ = Cost of the product.

$f_2(x)$ = Quality score of the product.

You want to **minimize** the total cost and **maximize** the quality, but to apply the Weighted Sum Method, both objectives must be in the same direction.

Since minimizing one and maximizing another is conflicting, we turn **maximization of quality** into **minimization** by taking the negative of the quality score:

The new objective function for quality becomes: $-f_2(x)$.

So, your two objectives now are:

Minimize $f_1(x)$ (Cost),

Minimize $-f_2(x)$ (Negative Quality).

Step 2: Combine the Objectives Using Weights

The Weighted Sum Method combines the objectives into a single scalar objective function by assigning a weight to each:

$$F(x) = w_1 \cdot f_1(x) + w_2 \cdot (-f_2(x))$$

Where:

w_1 and w_2 are weights that reflect the relative importance of each objective.

The weights must sum to 1, so $w_1 + w_2 = 1$

For example, if **cost** is more important than quality, you might assign:

$w_1 = 0.7$ (70% weight to cost),

$w_2 = 0.3$ (30% weight to quality).

The combined objective function becomes:

$$F(x) = 0.7 \cdot f_1(x) + 0.3 \cdot (-f_2(x))$$

Step 3: Solve the Optimization Problem

Now, you solve the optimization problem by minimizing the single scalar objective $F(x)$.
The solution will give you a trade-off between cost and quality based on the specified weights.

Step 4: Interpret the Result

Let's assume specific cost and quality functions for the product:

$f_1(x) = 10x^2 + 5$ (cost function, where xxx represents a design parameter),

$f_2(x) = 20 - 2x$ (quality function, where higher values of xxx improve quality but also increase cost).

Using the weighted sum approach:

$$F(x) = 0.7 \cdot (10x^2 + 5) + 0.3 \cdot (-(20 - 2x))$$

$$F(x) = 7x^2 + 3.5 - 6 + 0.6x$$

$$F(x) = 7x^2 + 0.6x - 2.5$$

You would now minimize F(x) with respect to x to find the best trade-off between cost and quality.

Step 5: Varying the Weights

To explore different trade-offs between cost and quality, you can vary the weights:

$w_1 = 0.5$, $w_2 = 0.5$ (equal importance to both),

$w_1 = 0.9$, $w_2 = 0.1$ (cost is more important),

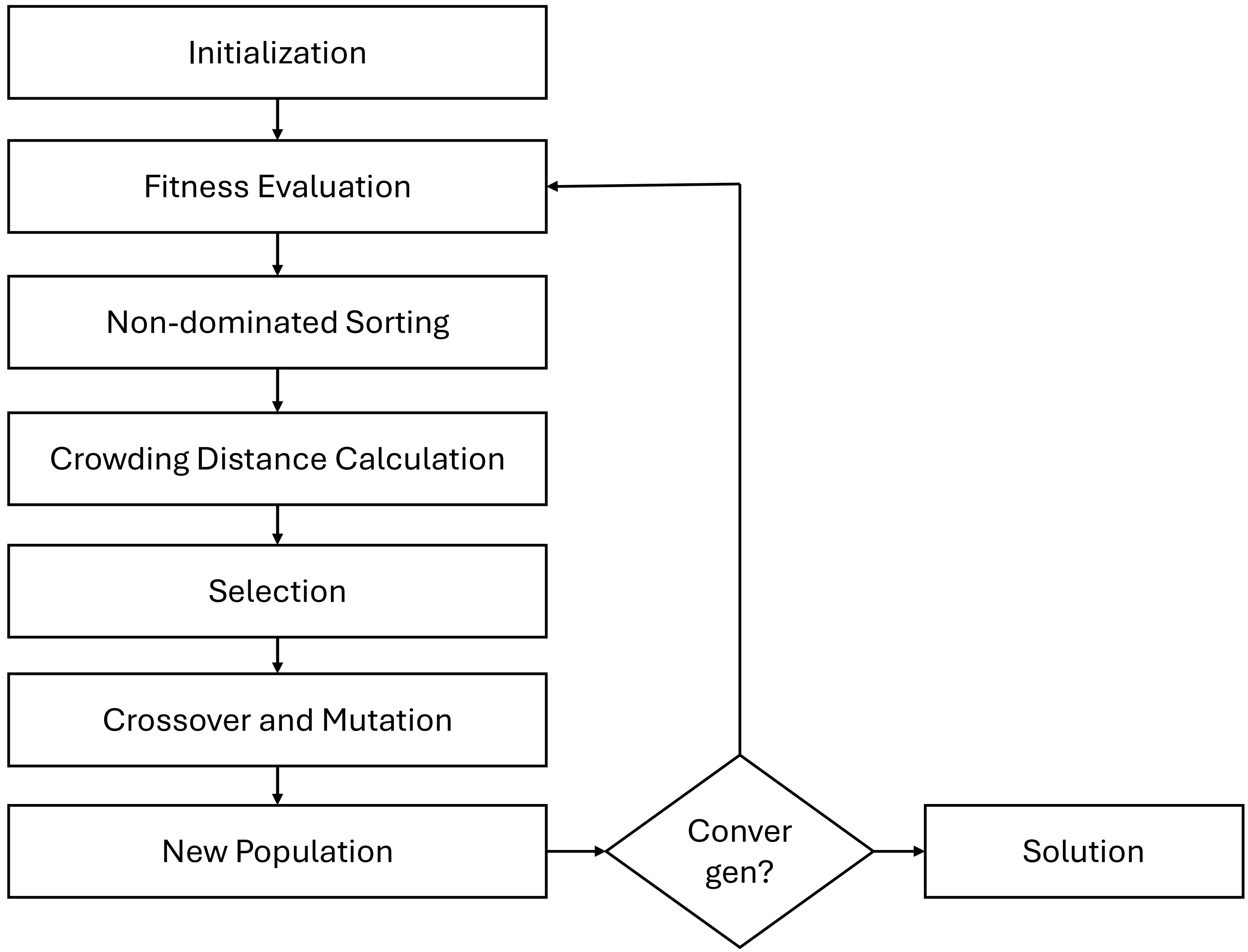
$w_1 = 0.3$, $w_2 = 0.7$ (quality is more important).

Each weight combination will yield a different solution, and the set of all solutions forms a **Pareto front** that helps in visualizing the trade-offs between the objectives.

Pareto-Based Methods

- **Genetic Algorithms :** Evolutionary algorithms that search for a set of Pareto-optimal solutions. NSGA-II (Non-dominated Sorting Genetic Algorithm) is one of the most popular.
- **Strengths:** Good for complex, non-linear, and non-convex problems.

NSGA-II



Initialization
Fitness Evaluation
Non-dominated Sorting
Crowding Distance Calculation
Selection
Crossover and Mutation
New Population
Conver gen?
Solution

Example: Optimizing a Car Design for Fuel Efficiency and Safety

Problem Statement:

You want to optimize the design of a car by balancing two conflicting objectives:

1. **Maximize fuel efficiency** (Objective 1: $f_1(x)$),
2. **Maximize safety** (Objective 2: $f_2(x)$).

Both objectives cannot be maximized simultaneously because increasing the safety features (e.g., adding weight for reinforced structures) often decreases fuel efficiency.

Step 1: Define the Objective Functions

Let's assume two simplified objective functions:

- $f_1(x) = 50 - 0.1 \times x_1$, where x_1 is the car's weight (lower weight means better fuel efficiency),
- $f_2(x) = 0.5 \times x_1 + 10$, where x_1 also affects safety (higher weight results in better safety).

Here, the objective is to **maximize** both fuel efficiency and safety. In optimization terms, maximizing these objectives is equivalent to **minimizing** their negative values:

- Minimize $-f_1(x) = -(50 - 0.1x_1)$,
- Minimize $-f_2(x) = -(0.5x_1 + 10)$.

Step 2: Initialize a Population of Solutions

In NSGA-II, we start with a random population of possible car designs. Each design is represented by a decision variable x_1 (the car's weight).

Let's assume we randomly generate a population of 5 initial car designs with different weights: $[1000, 1200, 1500, 1800, 2000]$ kg.

Step 3: Evaluate Fitness (Objective Functions)

For each design, we calculate the two objective functions $f_1(x)$ and $f_2(x)$:

- Car 1 ($x_1 = 1000$): $f_1(1000) = 40$, $f_2(1000) = 510$,
- Car 2 ($x_1 = 1200$): $f_1(1200) = 38$, $f_2(1200) = 610$,
- Car 3 ($x_1 = 1500$): $f_1(1500) = 35$, $f_2(1500) = 760$,
- Car 4 ($x_1 = 1800$): $f_1(1800) = 32$, $f_2(1800) = 910$,
- Car 5 ($x_1 = 2000$): $f_1(2000) = 30$, $f_2(2000) = 1010$.

Step 4: Non-dominated Sorting

The algorithm performs **non-dominated sorting** to classify solutions into different **Pareto fronts**. A solution is **non-dominated** if no other solution is better in both objectives.

- In this example, none of the designs are strictly better than the others in both fuel efficiency and safety, so they all belong to the **first Pareto front**.

Step 5: Crowding Distance Calculation

NSGA-II calculates a **crowding distance** to maintain diversity among solutions on the Pareto front. The crowding distance is a measure of how far apart a solution is from its neighbors in objective space.

- For example, Car 1 might have a high fuel efficiency (40) but relatively lower safety (510), while Car 5 has low fuel efficiency (30) but the best safety (1010). NSGA-II will keep these diverse solutions on the Pareto front.

Step 6: Selection, Crossover, and Mutation

The algorithm selects parent solutions for the next generation based on their rank (Pareto front) and crowding distance. It uses **crossover** and **mutation** operators to generate new car designs for the next population.

- **Crossover**: Combines two parent solutions to create new designs. For instance, combining Car 1 (weight = 1000) and Car 3 (weight = 1500) could produce a child design with a weight of 1250 kg.
- **Mutation**: Introduces random changes in the decision variables to explore new areas of the solution space. For example, a car's weight might mutate from 1250 to 1300 kg.

Step 7: Evolution of Solutions

NSGA-II evolves the population over multiple generations, with each new population undergoing the same process:

- Evaluate fitness (objectives),
- Perform non-dominated sorting to classify solutions,
- Calculate crowding distances to maintain diversity,
- Select, crossover, and mutate to generate a new population.

After several generations, the algorithm converges to a **Pareto front** of optimal solutions.

Step 8: Pareto Front and Trade-offs

At the end of the optimization process, you get a **Pareto front** of non-dominated solutions. These solutions represent the best trade-offs between fuel efficiency and safety.

For example:

- Car A: $x_1 = 1200$, $f_1 = 38$ (good fuel efficiency), $f_2 = 610$ (moderate safety),
- Car B: $x_1 = 1500$, $f_1 = 35$ (moderate fuel efficiency), $f_2 = 760$ (good safety),
- Car C: $x_1 = 1800$, $f_1 = 32$ (lower fuel efficiency), $f_2 = 910$ (best safety).

These solutions are **Pareto optimal** because no other design can improve one objective without worsening the other. The designer can now choose the solution that best balances fuel efficiency and safety based on their priorities.

Visualization

A plot of the Pareto front might look like this:

$$\text{Y-axis: Safety (higher is better), X-axis: Fuel Efficiency (higher is better)}$$

- Points representing designs are distributed along the Pareto front, illustrating the trade-off between the two objectives.
- You can visualize that designs with higher safety (Y-axis) tend to have lower fuel efficiency (X-axis), showing the inherent trade-off between these two conflicting objectives.

Evolutionary Algorithms

- **Differential Evolution (DE):** Used for optimizing real-valued problems by evolving a population of candidate solutions.
- **Particle Swarm Optimization (PSO):** Swarm-based algorithm inspired by social behavior.

Other Methods

- **Multi-Objective Simulated Annealing (MOSA):** A probabilistic technique that mimics the process of annealing in metals.
- **Game-Theoretic Approaches:** Model the optimization as a game between multiple players, each representing an objective.

Applications of Multi-Objective Optimization

- **Engineering Design:** Balancing performance, cost, and durability in the design of vehicles, aircraft, or machinery.
- **Finance:** Optimizing portfolios for maximum return and minimum risk.
- **Supply Chain Management:** Minimizing costs and maximizing customer satisfaction.
- **Energy Systems:** Trade-offs between efficiency, cost, and environmental impact in energy production.
- **Healthcare:** Balancing treatment effectiveness with minimizing side effects or costs.

Challenges in Multi-Objective Optimization

- **Curse of Dimensionality:** As the number of objectives increases, finding the Pareto front becomes computationally expensive.
- **Solution Diversity:** Ensuring that the Pareto front represents a wide range of trade-offs between objectives.
- **Computational Complexity:** Many real-world problems require high computational resources to find solutions.