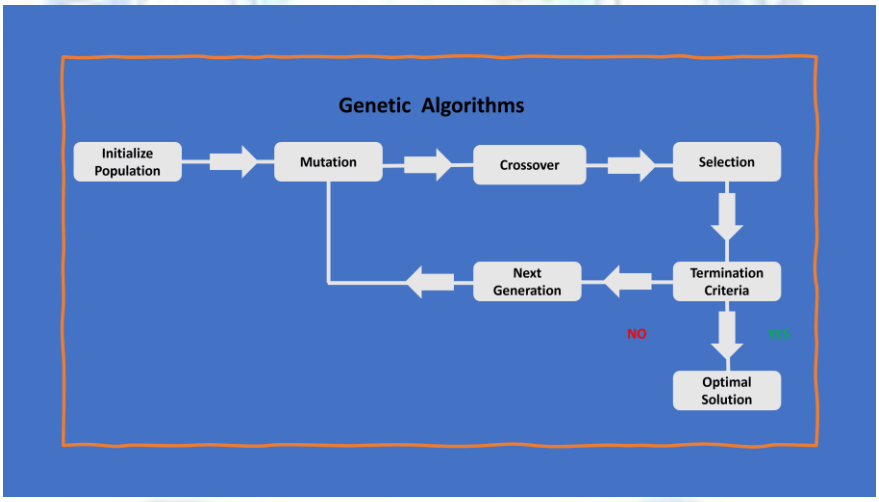
4.2 Differential Evolution

Differential Evolution is a smart search algorithm that improves potential solutions repeatedly to optimize problems, similar to how evolution operates.

Rainer Storn and Kenneth Price developed it back in the mid-1990s. Since then, the Differential Evolution has become popular for tackling tough optimization problems in different areas. People like it because it's strong, flexible, and works well in many situations.

Evolutionary Computation Background

Differential Evolution (DE) functions as a nature-inspired search strategy within the broader category of evolutionary computation, encompassing a variety of optimization algorithms that draw inspiration from natural selection and genetics. To understand DE better, let's look at the related concept of genetic algorithms (GA), a prominent member of the evolutionary computation family:



In genetic algorithms, the population treats individuals as candidate solutions, applies genetic operations (crossover and mutation) to them, and assesses their fitness. The principles guiding genetic algorithms lay the groundwork for comprehending the population-based optimization approach employed by DE.

Population-Based Optimization

Differential Evolution operates on a population of candidate solutions, setting it apart from traditional optimization methods that focus on a single solution. This population-based approach fosters diversity and facilitates thoroughly exploring the solution space. Leveraging three key operations: mutation, crossover, and selection, DE iteratively evolves the population towards optimal solutions.

While population-based operation is a fundamental characteristic of Differential Evolution, it's noteworthy that many other evolutionary algorithms share this trait.

In genetic algorithms, we treat possible solutions as individuals in a group. These individuals undergo genetic operations like crossover and mutation, and then we evaluate their performance. The ideas behind genetic algorithms help us grasp the population-based optimization method used by DE.

Genetic Algorithms vs. Differential Evolution

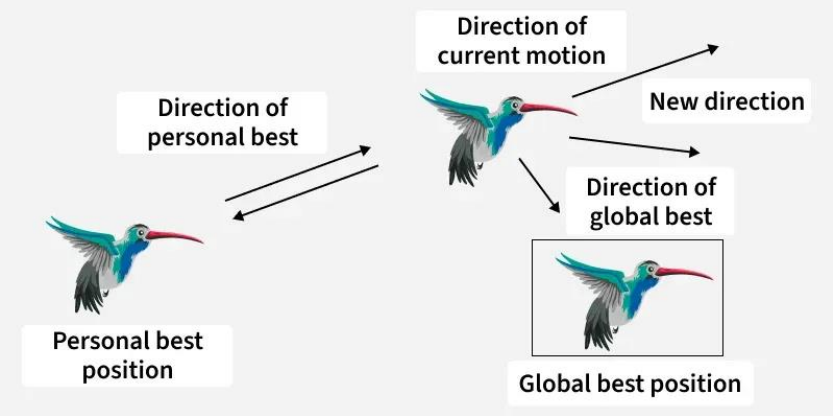
Genetic Algorithms and Differential Evolution are two distinct evolutionary optimization algorithms with differing approaches to solving optimization problems.

In GAs, individuals in the population are typically represented as binary or integer strings, and the algorithm employs crossover and mutation operators to create offspring for subsequent generations.

DE, on the other hand, operates on vectors of real numbers, using a unique differential mutation strategy that combines vectors to generate trial vectors. DE's population update is more gradual, replacing individuals based on their performance compared to trial vectors. While GAs often use traditional selection mechanisms, DE introduces a different selection strategy. DE is known for its faster convergence, particularly in high-dimensional optimization problems, making it effective in scenarios where GAs may face challenges.

Particle Swarm Optimization (PSO)

Optimisation aims to find the best solution from a set of feasible options under given constraints. In many machine learning and engineering problems, the search space is complex, non-linear and multimodal, where traditional gradient-based methods may be ineffective. This makes population-based optimization techniques more suitable.



Particle Swarm Optimization (PSO) is a stochastic population based optimization technique inspired by swarm intelligence in nature. It is designed to solve complex optimization problems where the search space is large, non-linear or unknown, where traditional deterministic methods are ineffective.

- Each particle represents a potential solution and moves through the search space
- Movement is guided by both individual experience and collective swarm knowledge

- The algorithm iteratively improves solutions using a fitness function
- Simple to implement with fast convergence and few control parameters

How Particle Swarm Optimization (PSO) Works

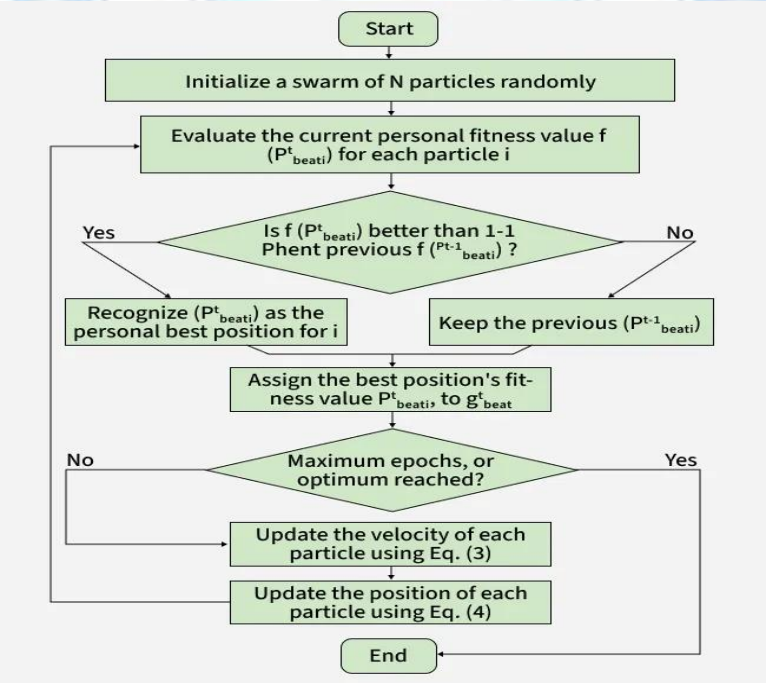
Particle Swarm Optimization (PSO) is an iterative, population based optimization algorithm. It works by moving a group of particles (candidate solutions) through the search space using simple mathematical rules based on personal and collective experience.

Each particle in PSO has

- Position (x): current solution
- Velocity (v): direction and speed of movement
- Fitness value: quality of the solution

Each particle stores:

- pBest: best position found by itself
- gBest: best position found by the entire swarm



Step 1: Initialization

- Randomly initialize N particles within the search space $[m, inxmaxx]$
- Assign a random velocity to each particle
- Evaluate the fitness value of each particle

pBest = current position
gBest = best pBest among all particles

Step 2: Velocity Update

At each iteration the velocity of a particle is updated using:

$$v_i^{t+1} = w \cdot v_i^t + c_1 \cdot r_1 \cdot (pBest_i - x_i^t) + c_2 \cdot r_2 \cdot (gBest - x_i^t)$$

where

- w: inertia weight (controls exploration)
- c_1: cognitive coefficient (self-learning)
- c_2: social coefficient (swarm learning)
- r_1, r_2: random values in [0,1]

Step 3: Position Update

After updating velocity the position is updated as:

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

If the new position goes outside [minx,maxx] clip it to the boundary.

Step 4: Update Best Positions

- If current fitness is better than pBest, update pBest
- If current fitness is better than gBest, update gBest

Step 5: Convergence

- Repeat Steps 2–4 for a fixed number of iterations or until convergence
- The swarm gradually moves toward the optimal solution

Application

1. Health Care Applications

- **Intelligent Diagnosis:** Optimizes models for faster and accurate disease detection.
- **Medical Robots:** Guides robots for precise surgical or therapeutic tasks.

2. Environmental Applications

- **Wild Vegetation Monitoring:** Tracks forest and vegetation growth efficiently.
- **Agriculture Monitoring:** Optimizes crop management and environmental monitoring.
- **Flood Control and Routing:** Predicts floods and finds optimal evacuation or routing paths.

3. Industrial Applications

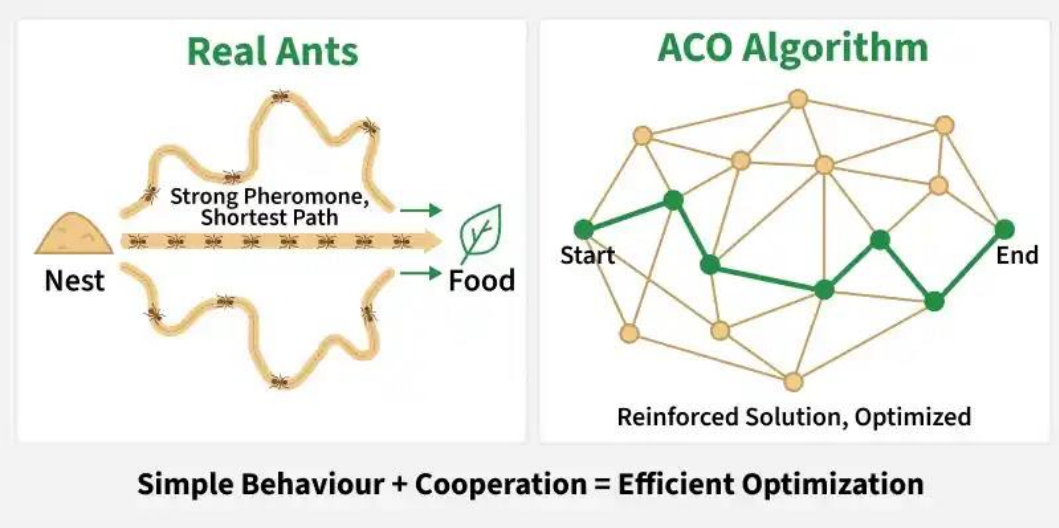
- **WSN Deployment:** Optimizes sensor placement in Wireless Sensor Networks.
- **Product Defection Prediction:** Predicts defects to improve product quality.
- **Microgrid Design and Operation:** Optimizes design and operation for stability.

4 Smart City Applications

- **Smart City Planning:** Optimizes urban energy, traffic and resources.
- **Smart Home Automation:** Efficiently controls and schedules home appliances.
- **Appliance Scheduling:** Reduces energy consumption via optimal use.

Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a nature-inspired algorithm that learns from how real ants collectively find the shortest path to food without any central control. Instead of fixed rules, it improves solutions over time through simple behaviour and cooperation making it effective for complex optimization problems. In nature, ants follow a simple process like:



Key Components

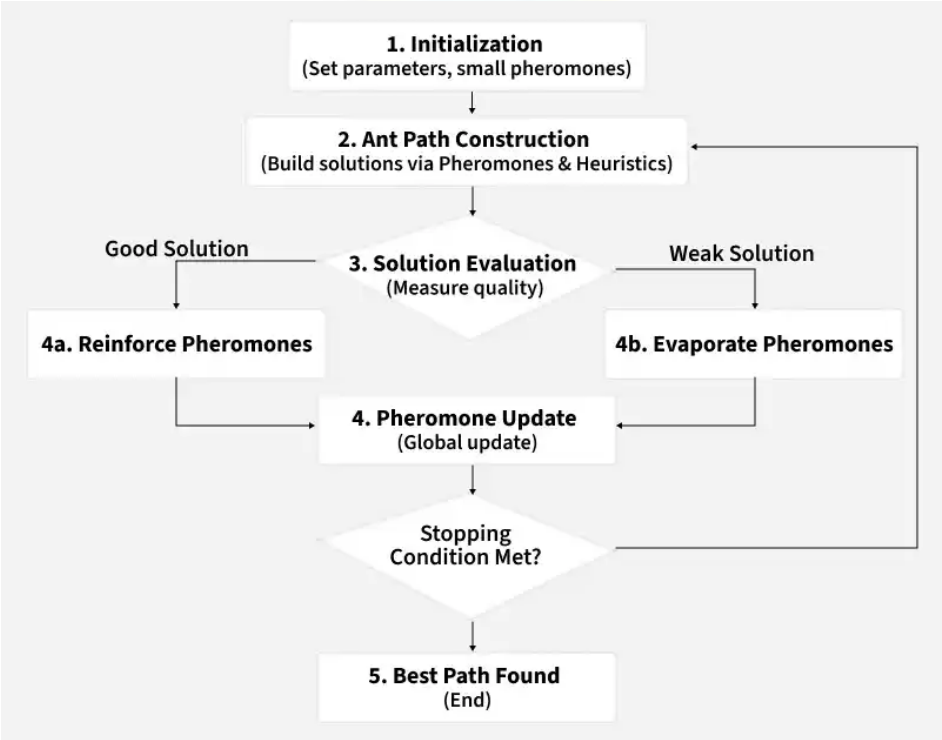
Ant Colony Optimization is built on a few core components that work together to guide the search toward optimal solutions.

1. **Ants:** Simple agents that explore the problem space and construct solutions step by step.
2. **Pheromone trails:** Numerical values that store past experience and indicate which paths are more promising.
3. **Heuristic information:** Problem specific knowledge such as distance, cost or time that helps ants make informed decisions.

4. **Probability based movement:** Ants choose paths using probabilities influenced by pheromones and heuristic values, rather than fixed rules.
5. **Pheromone evaporation:** Pheromone levels gradually decrease over time, preventing premature convergence and encouraging continued exploration.

How it Works

Ant Colony Optimization operates through an iterative loop that progressively refines solutions based on collective learning.

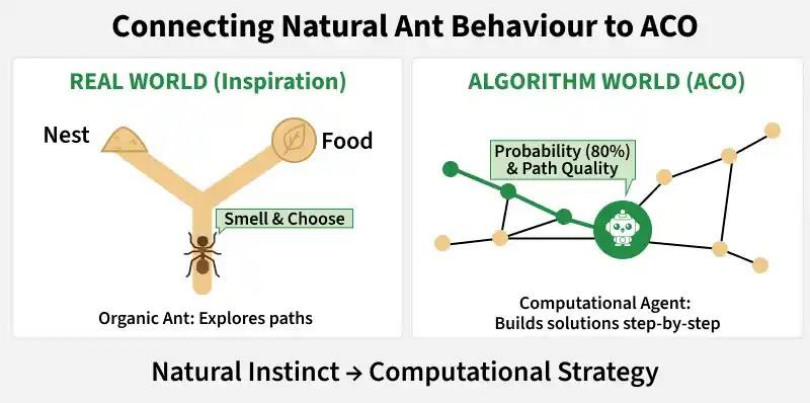


1. **Initialization:** Set initial parameters and assign small pheromone values to all paths.
2. **Ant path construction:** Each ant builds a solution by moving through the problem space based on pheromones and heuristic information.
3. **Solution evaluation:** The quality of each solution is measured using a fitness or cost function.
4. **Pheromone update:** Good solutions receive more pheromone reinforcement, while pheromones on other paths evaporate.
5. **Iteration until best path is found:** The process repeats until a stopping condition is met or the best path is discovered**.**

Turning Ant Behaviour into a Problem Solving Strategy

Ant Colony Optimization converts the natural behaviour of ants into a computational problem solving framework, where simple agents collectively build high quality solutions.

1. **Ants as agents**: In ACO, each ant is a simple computational agent that constructs a solution step by step, similar to how real ants explore routes.
2. **Pheromones as memory:** Pheromones are numerical values assigned to paths, representing how effective those paths were in previous solutions.
3. **Nodes and paths:** Nodes represent possible states or locations, while paths represent transitions between them, similar to routes between a nest and food.
4. **Decision making:** Ants select paths probabilistically, considering both pheromone strength and path quality instead of following fixed rules.



Parameters in Ant Colony Optimization

The performance and behaviour of Ant Colony Optimization depend on a few key parameters that balance exploration and exploitation.

1. **Number of ants:** Determines how many solutions are explored in each iteration. More ants improve exploration but increase computation.
2. **Pheromone importance:** Controls how strongly ants follow previously successful paths. Higher importance means more exploitation of known good solutions**.**
3. **Heuristic importance:** Defines how much ants rely on problem specific information such as distance or cost while choosing paths.
4. **Evaporation rate:** Decides how quickly pheromone values fade over time. A higher rate encourages exploration, while a lower rate favours stability.

How Ants Discover the Shortest Path

Ant Colony Optimization is inspired by how real ants search for food and collectively discover the shortest path without any central control. Their behaviour can be understood through a few simple ideas:

1. **Random exploration:** Ants initially move in different directions, exploring multiple possible paths.
2. **Pheromone trails:** When an ant finds food, it returns while leaving a chemical trail that guides other ants.
3. **Shorter paths strengthen faster:** Ants on shorter routes return more quickly and deposit pheromones more often, while longer paths fade due to evaporation.
4. **Indirect communication:** Ants coordinate through pheromone signals rather than a leader and is known as stigmergy.

Artificial Bee Colony (ABC)

The Artificial Bee Colony (ABC) algorithm is a swarm-based search algorithm inspired by the foraging behavior of honey bees. Developed by Prof. Dr. Derviş Karaboğa, this algorithm is used to solve optimization problems by simulating the intelligent foraging behavior of bees.

Key Concepts

1. **Initialization:** Generate initial random solutions (food sources).
2. **Employed Bees Phase:** Each employed bee searches for new solutions near their current food source.
3. **Onlooker Bees Phase**: Onlooker bees choose food sources based on the quality of the food source and search for new solutions**.**
4. **Scout Bees Phase:** If a food source is abandoned, a scout bee searches for a new random food source.
5. **Termination:** Repeat the employed, onlooker, and scout phases until a stopping criterion is met.

Artificial Bee Colony (ABC) Algorithm:

- **Swarm-based:** Simulates the foraging behavior of bees**.**
- **Employed Bees Phase:** Searches for new solutions around current solutions.
- **Onlooker Bees Phase:** Selects and improves food sources based on quality.
- **Scout Bees Phase:** Introduces new random solutions if necessary.