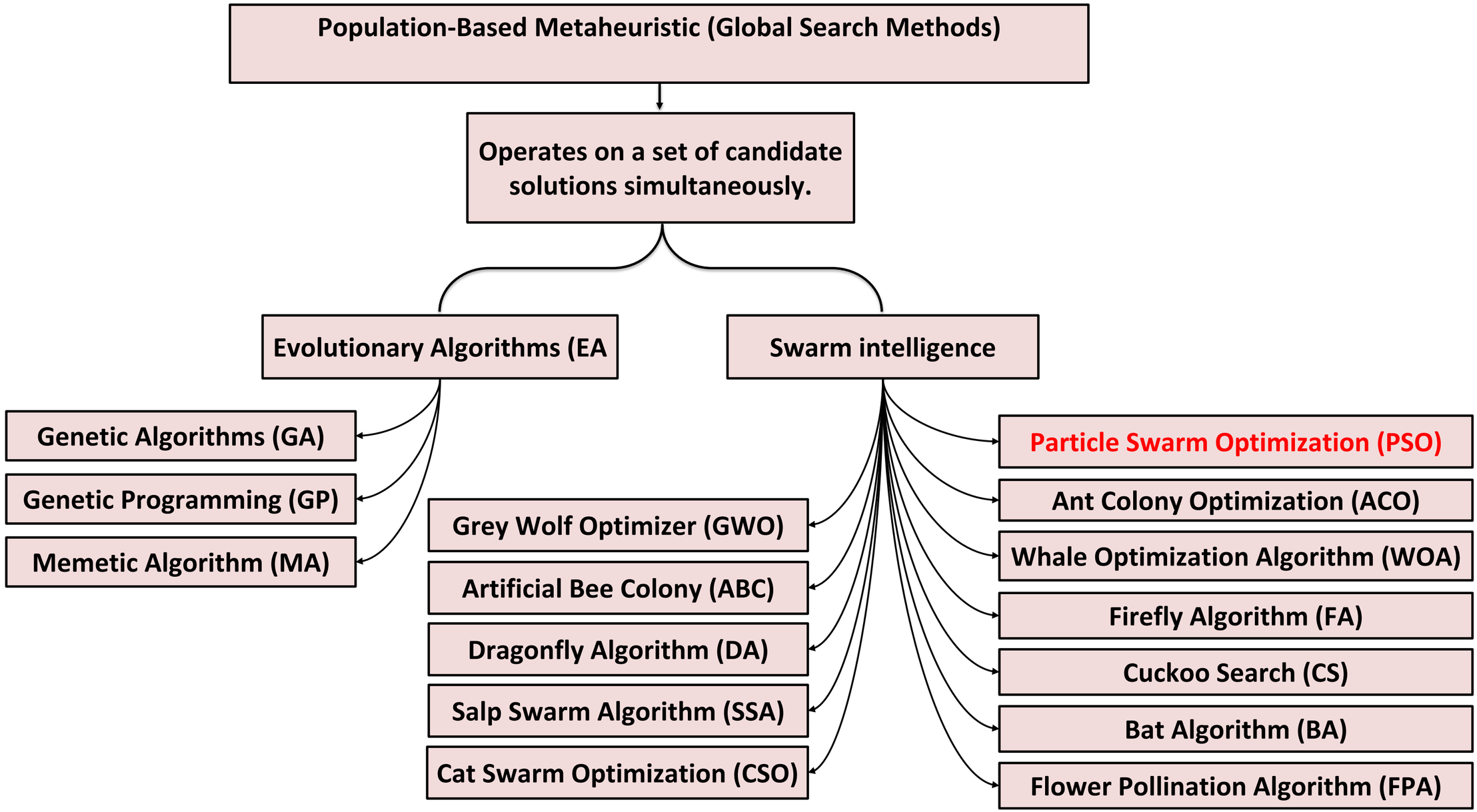
Nature Inspired Computation
DSAI 403

Assoc. Prof. **Mohamed Maher Ata**
Zewail city of science, technology, and innovation
momaher@zewailcity.edu.eg
Room: S028- Zone D

Population-Based Search (Global Search)



Population-Based Search (Global Search)

- Population-based search methods explore the search space using a **group of candidate solutions** (a population) simultaneously. Each member of the population represents a potential solution.

Characteristics:

- **Multiple solutions** in parallel are evolved or optimized together.
- Each solution can have different characteristics, and they interact or evolve to improve the population as a whole.
- Tends to explore **larger areas of the search space**.
- Helps **balance exploration** (searching across the space) with **exploitation** (refining the best solutions).

Advantages:

- Less likely to get stuck in local optima.
- Better for complex, high-dimensional, or noisy problems.

Disadvantages:

- More computationally expensive.
- Requires careful tuning of population size and other parameters.

Swarm intelligence

- Swarm Intelligence (SI) is a computational and behavioral concept inspired by the collective behavior of decentralized, self-organized systems found in nature such as ants, bees, birds, fish, and even human crowds.

Principles of Swarm Intelligence:

1	**Awareness**	Each member (agent) must know its surroundings and capabilities.	**Ex:** A bird senses nearby birds to adjust its flight path
2	**Resiliency**	The system should self-heal if some members fail or leave.	**Ex:** If a few ants die, others still find the food path.
3	**Solidarity**	Once a task is done, members autonomously look for new tasks.	**Ex:** After bees finish collecting from one flower, they move to another.
4	**Autonomy**	Each agent acts independently (not as a slave) while coordinating collectively.	**Ex:** Each fish in a school reacts locally, not by central command.
5	**Expandability**	The swarm should easily expand or shrink by adding or removing agents	**Ex:** Adding more drones in a swarm should improve coverage without redesign.

Particle Swarm Optimization

- Particle Swarm Optimization (PSO) is a powerful meta-heuristic optimization algorithm and inspired by **swarm behavior** observed in nature such as birds flocking or fish schooling.
- PSO is a Simulation of a simplified social system. The original intent of PSO algorithm was to graphically simulate the graceful but unpredictable choreography of a bird flock.
- In nature, any of the bird's observable vicinity is limited to some range. However, having more than one birds allows all the birds in a swarm to be aware of the larger surface of a fitness function.

Main idea

1. Each particle represents a potential solution in the search space.
2. Particles move around trying to find the best position (solution) by sharing information about what they've found so far.
3. Each particle remembers:
   Its personal best position (pbest), and
   The global best position found by the entire swarm (gbest).
4. Particles update their velocity and position according to their own experience and the swarm's experience. This balance between exploration (searching globally) and exploitation (refining good solutions) is key to PSO's power.

Inspiration for PSO

1	**Social Behavior**	PSO is inspired by swarm intelligence, which is the collective behavior of decentralized, self-organized systems, typically observed in nature, such as: ▪ **Bird Flocking:** Birds move in coordinated patterns, where each bird adjusts its position based on the position of its neighbors, striving for the best overall path. ▪ **Fish Schooling:** Fish exhibit collective movement in schools, seeking the best direction or path for the group while avoiding obstacles.
2	**Cooperation Over Competition**	In nature, individuals in a group (like birds or fish) are not competing but instead sharing information about the environment to collectively find the best solution or path. This behavior is modeled in PSO, where each particle learns from both its own experience and the experiences of its neighbors to move towards the optimal solution.
3	**Mathematical Model**	The process is inspired by the way individuals in a group constantly adjust their position and velocity based on their past experiences (personal best) and the best solution found by the group (global best).

Mathematical Model of PSO

- The movement of particles in PSO is guided by two fundamental equations: **the position update equation** and **the velocity update equation.**

1. The position update equation:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Where:

- $x_i(t)$ represents the current position of the particle (i) at iteration (t),
- $v_i(t+1)$ is the updated velocity, and $x_i(t+1)$ is the new position of the particle.

2. The velocity update equation:

$$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (pbest_i - x_i(t)) + c_2 \times r_2 \times (gbest_i - x_i(t))$$

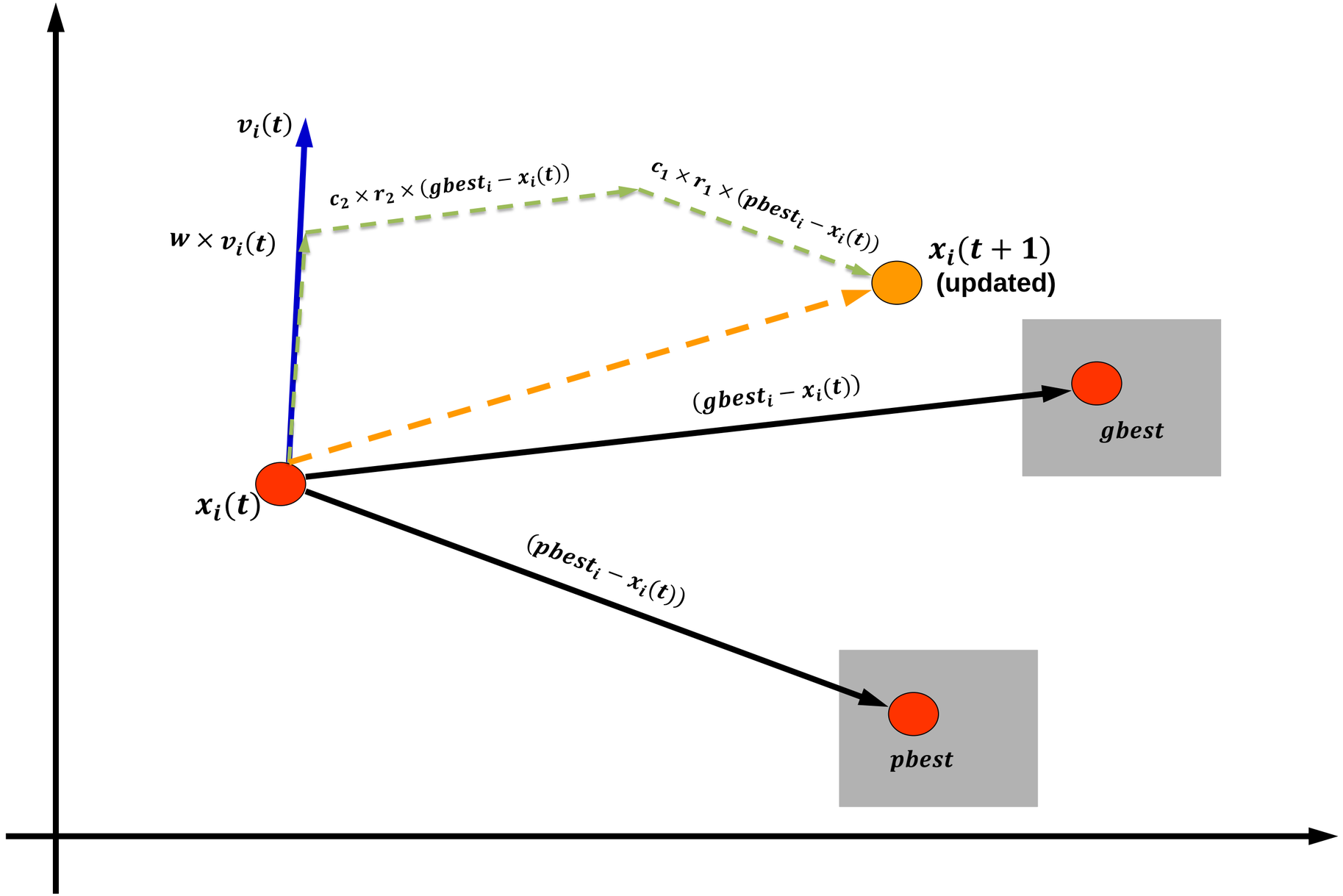
Where,

- $v_i(t)$ represents the velocity of particle (i)
- w is the inertia weight (controls exploration/exploitation)→it helps particles continue moving in the same direction (momentum).
- c_1 and c_2 are acceleration coefficients (cognitive and social learning coefficients) → c_1 encourages each particle to move toward its personal best however c_2 encourages movement toward the global best.
- r_1 and r_2 are random numbers between 0 and 1 (simulate the natural, irregular flight of birds in a flock)
- $pbest$ is the best position found by particle i (personal best)
- $gbest$ is the best position found by the swarm (global best).

Main components:

i. The inertia component ($w \times v(t)$): This term represents the particle's tendency to continue moving in its previous direction.
ii. The cognitive component ($c_1 \times r_1 \times (pbest - x(t))$): This term pulls the particle towards its personal best position.
iii. The social component ($c_2 \times r_2 \times (gbest - x(t))$): This term attracts the particle towards the global best position found by the swarm.

$$x_i(t+1) = x_i(t) + v_i(t+1)$$
$$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (pbest_i - x_i(t)) + c_2 \times r_2 \times (gbest_i - x_i(t))$$
$$x_i(t+1) = x_i(t) + w \times v_i(t) + c_1 \times r_1 \times (pbest_i - x_i(t)) + c_2 \times r_2 \times (gbest_i - x_i(t))$$



Example: (finding a new lcation)

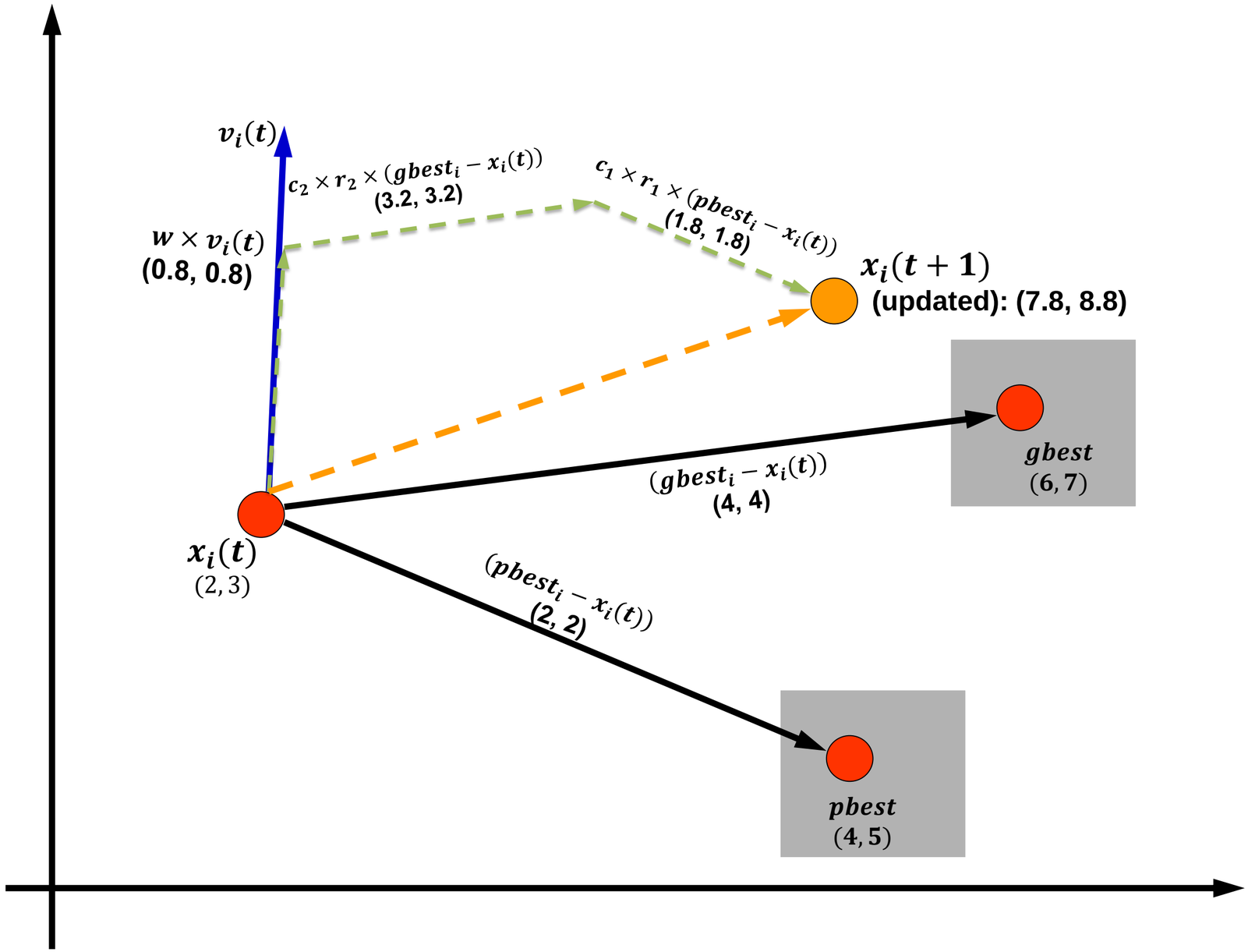
Suppose a particle's current position is $x(t) = (2, 3)$, its current velocity is $v(t) = (1, 1)$, its personal best position is $pbest = (4, 5)$, and the global best position is $gbest = (6, 7)$. If we set $w = 0.8$, $c_1 = 1.5$, $c_2 = 2$, and $r_1 = 0.6$, $r_2 = 0.4$, then the updated velocity would be:

$$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (pbest_i - x_i(t)) + c_2 \times r_2 \times (gbest_i - x_i(t))$$

$$v_i(t+1) = 0.8 * (1,1) + 1.5 * 0.6 * ((4,5) - (2,3)) + 2 * 0.4 * ((6,7) - (2,3))$$
$$= (0.8, 0.8) + (1.8, 1.8) + (3.2, 3.2) = (5.8, 5.8)$$

And the new position would be:

$$\boldsymbol{x_i(t+1)} = (2,3) + (5.8, 5.8) = (7.8, 8.8)$$



The parameter w is the inertia weight and it is a positive constant. This parameter is important for balancing the global search (exploration) and local search (exploitation)

1. High w → "keep flying fast" → more exploration
2. Low w → "slow down near the target" → more exploitation

Ex:

- At early iterations: w≈0.9 → more exploration
- At later iterations: w≈0.4 → more exploitation

This part is responsible for searching new solutions, finding the regions with potentially the best solutions

- This part is responsible for exploring the previous solutions, finding the best solution of a given region
- A high r_1 and low r_2 make the particle rely more on its own experience (exploration)
- A low r_1 and high r_2 make the particle follow the swarm's best position more closely (exploitation)

Diversification

intensification

$$v_i(t+1) = \boldsymbol{w} \times v_i(t) + c_1 \times r_1 \times (pbest_i - x_i(t)) + c_2 \times r_2 \times (gbest_i - x_i(t))$$

Inertia component

makes the particle move in the same direction with the same velocity:

Cognitive component (personal influence)

Improves the individual particles and makes the particle return to a previous position better than the current

Social component (social influence)

Makes the particle follow the best neighbors direction

Important note:

The constants c_1, and c_2 control the *strength* of self and social learning, while r_1 and r_2 randomly decide *when and how strongly* that influence appears. Their product defines the actual movement intensity in each iteration.

Assume that: $(pbest_i - x_i(t)) = 5$

Case	c_1	r_1	$c_1 \times r_1$	$5 \times c_1 \times r_1$	effect
High c_1, High r_1	2	0.9	1.8	9	Very strong pull to personal best → particle moves fast toward its own experience
High c_1, Low r_1	2	0.2	0.4	2	Strong confidence but weak random activation → mild movement this iteration
Low c_1, High r_1	0.5	0.9	0.45	2.25	Weak self-confidence though randomness is high → small attraction to own best
Low c_1, Low r_1	0.5	0.2	0.1	0.5	Both weak → almost no movement toward personal best

Assume that: $(gbest_i - x_i(t)) = 4$

Case	c_2	r_2	$c_2 \times r_2$	$4 \times c_2 \times r_2$	effect
High c_2, High r_2	2	0.8	1.6	6.4	Very strong attraction toward global best → swarm converges quickly
High c_2, Low r_2	2	0.3	0.6	2.4	Strong group trust, but weak random activation → limited social move
Low c_2, High r_2	0.5	0.8	0.4	1.6	Weak global influence despite random encouragement
Low c_2, Low r_2	0.5	0.2	0.1	0.4	Both low → minimal group effect, particle relies on own experience

Summary:

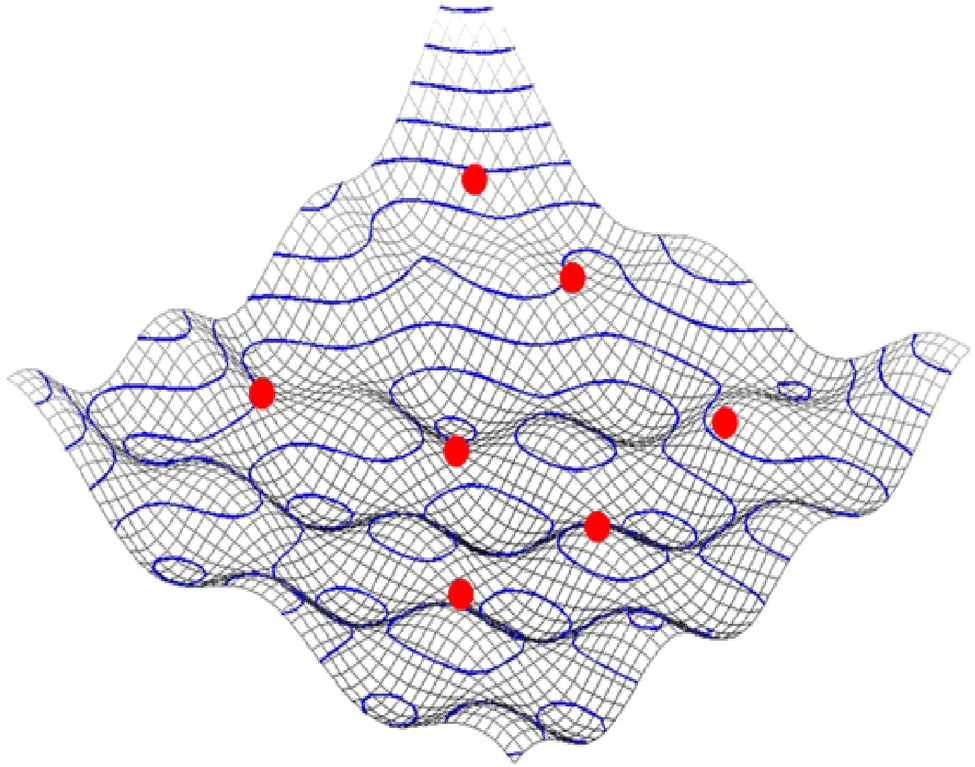
- The idea is similar to bird flocks searching for food. PSO uses a number of agents, i.e., particles, that constitute a swarm flying in the search space looking for the best solution.
- Each particle is treated as a point (candidate solution) in a N-dimensional space which adjusts its “flying” according to its own flying experience as well as the flying experience of other particles.

  Bird = a particle

  Food = a solution

  pbest = the best solution (fitness) a particle has achieved so far

  gbest = the global best solution of all particles within the swarm
- The basic concept of PSO lies in accelerating each particle toward its pbest and the gbest locations, with a random weighted acceleration at each time



Swarm Principles in PSO :

1. **Awareness**: Each particle knows its position, velocity, and fitness.
2. **Resiliency:** Even if some particles get stuck in local minima, others can guide the swarm.
3. **Solidarity:** When one particle reaches its best, others adapt to new goals.
4. **Autonomy**: Each particle updates itself based on its own and neighbors’ information.
5. **Expandability:** More particles can be added to improve exploration dynamically.

Example: (finding minimum value)

it is required to minimize $f(x) = x^2$. Assume that:

Number of particles	2
w	0.6
$\boldsymbol{c_1}$	1.2
$\boldsymbol{c_2}$	1.2
$\boldsymbol{r_1}$	0.5
$\boldsymbol{r_2}$	0.5

Solution:

Initial state:

Particle	$x_i(0)$	$v_i(0)$	$p_i(0)$	$f(p_i(0))$
1	3	0	3	9
2	1	0	1	1

Global best: $g(0) = 1$

Assume 3 iterations

Iteration 1:

Particle: 1	Particle: 2
$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (p_i(t) - x_i(t)) + c_2 \times r_2 \times (g_i(t) - x_i(t))$ $v_1(1) = w \times v_1(0) + c_1 \times r_1 \times (p_1(0) - x_1(0)) + c_2 \times r_2 \times (g_1(0) - x_1(0))$ $v_1(1) = 0.6 \times 0 + 1.2 \times 0.5 \times (3-3) + 1.2 \times 0.5 \times (1-3) = -1.2$ $x_1(1) = x_1(0) + v_1(1) = 3 - 1.2 = 1.8$ $f((x_1(1)) = 1.8^2 = 3.24$ update $p_1(1) = 1.8$ (as it is improved)	$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (p_i - x_i(t)) + c_2 \times r_2 \times (g_i - x_i(t))$ $v_2(1) = w \times v_2(0) + c_1 \times r_1 \times (p_2(0) - x_2(0)) + c_2 \times r_2 \times (g_2(0) - x_2(0))$ $v_2(1) = 0.6 \times 0 + 1.2 \times 0.5 \times (1-1) + 1.2 \times 0.5 \times (1-1) = 0$ $x_2(1) = x_2(0) + v_2(1) = 1 + 0 = 1$ $f((x_2(1)) = 1^2 = 1$ update $p_2(1) = 1$ (no change)
From the two particles, compare personal bests to update the global best: $g(1) = 1$ as $f(p_1) = 3.24$ *and* $f(p_2) = 1$	

Iteration 2:

Particle	$x_i(1)$	$v_i(1)$	$p_i(1)$	$f(p_i(1))$
1	1.8	-1.2	1.8	3.24
2	1	0	1	1

Global best: $g(1) = 1$

Particle: 1	Particle: 2
$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (p_i(t) - x_i(t)) + c_2 \times r_2 \times (g_i(t) - x_i(t))$ $v_1(2) = w \times v_1(1) + c_1 \times r_1 \times (p_1(1) - x_1(1)) + c_2 \times r_2 \times (g_1(1) - x_1(1))$ $v_1(2) = 0.6 \times -1.2 + 1.2 \times 0.5 \times (1.8 - 1.8) + 1.2 \times 0.5 \times (1 - 1.8) = -1.2$ $x_1(2) = x_1(1) + v_1(2) = 1.8 - 1.2 = 0.6$ $f((x_1(2)) = 0.6^2 = 0.36$ and it was 3.24 update $p_1(2) = 0.6$ (as it is improved)	$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (p_i - x_i(t)) + c_2 \times r_2 \times (g_i - x_i(t))$ $v_2(2) = w \times v_2(1) + c_1 \times r_1 \times (p_2(1) - x_2(1)) + c_2 \times r_2 \times (g_2(1) - x_2(1))$ $v_2(2) = 0.6 \times 0 + 1.2 \times 0.5 \times (1 - 1) + 1.2 \times 0.5 \times (1 - 1) = 0$ $x_2(2) = x_2(1) + v_2(2) = 1 + 0 = 1$ $f((x_2(2)) = 1^2 = 1$ update $p_2(2) = 1$ (no change)
From the two particles, compare personal bests to update the global best: $g(2) = 0.6$ as $f(p_1) = 0.36$ *and* $f(p_2) = 1$	

Iteration 3:

Particle	$x_i(2)$	$v_i(2)$	$p_i(2)$	$f(p_i(2))$
1	0.6	−1.2	0.6	0.36
2	1	0	1	1

→ Global best: $g(2) = 0.6$

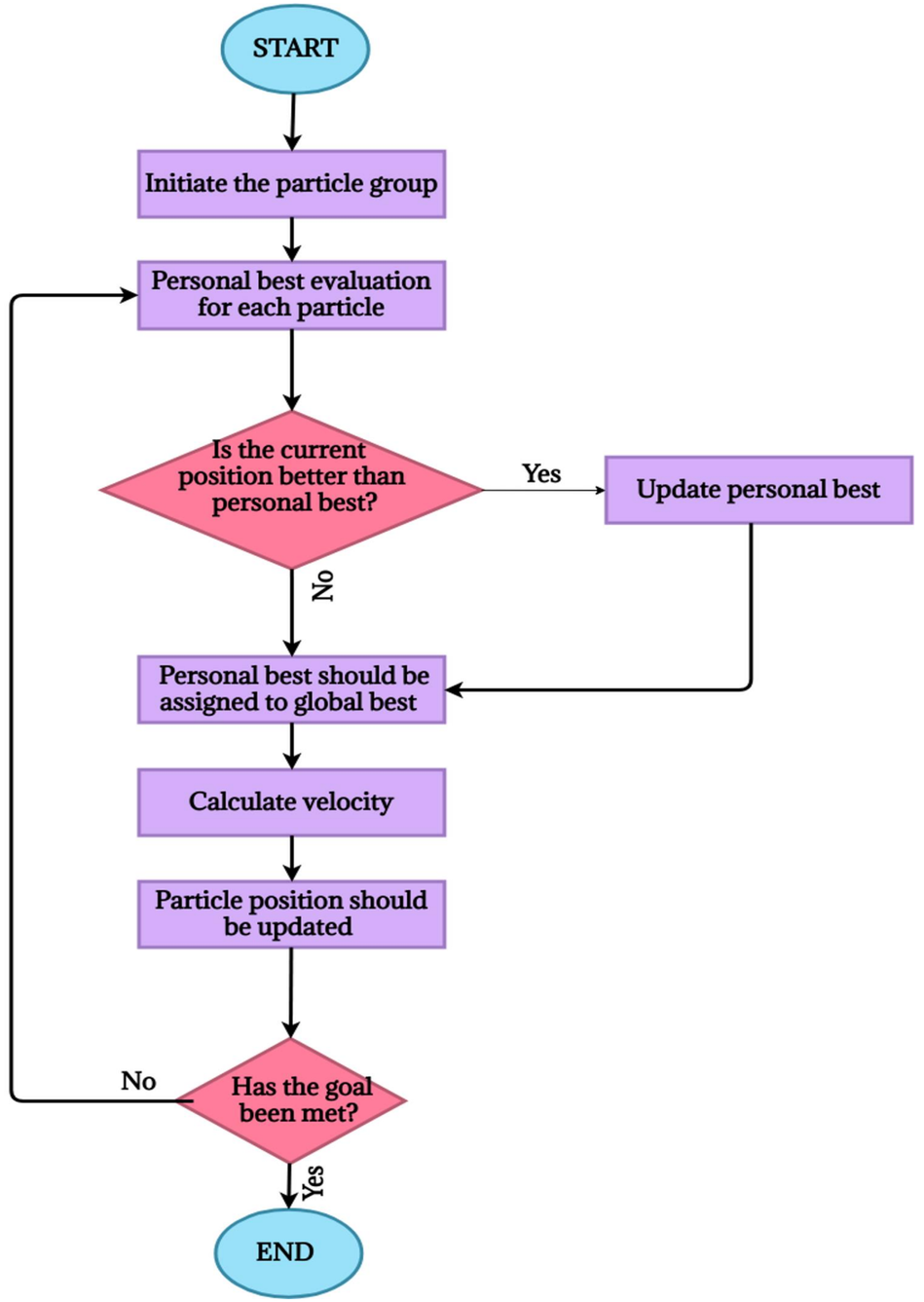
Particle: 1	Particle: 2
$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (p_i(t) - x_i(t)) + c_2 \times r_2 \times (g_i(t) - x_i(t))$ $v_1(3) = w \times v_1(2) + c_1 \times r_1 \times (p_1(2) - x_1(2)) + c_2 \times r_2 \times (g_1(2) - x_1(2))$ $v_1(3) = 0.6 \times -1.2 + 1.2 \times 0.5 \times (0.6 - 0.6) + 1.2 \times 0.5 \times (0.6 - 0.6) = -0.72$ $x_1(3) = x_1(2) + v_1(3) = 0.6 - 0.72 = -0.12$ $f((x_1(3)) = -0.12^2 = 0.0144$ and it was 0.36 update $p_1(3) = -0.12$ (as it is improved)	$v_i(t+1) = w \times v_i(t) + c_1 \times r_1 \times (p_i - x_i(t)) + c_2 \times r_2 \times (g_i - x_i(t))$ $v_2(3) = w \times v_2(2) + c_1 \times r_1 \times (p_2(2) - x_2(2)) + c_2 \times r_2 \times (g_2(2) - x_2(2))$ $v_2(3) = 0.6 \times 0 + 1.2 \times 0.5 \times (1 - 1) + 1.2 \times 0.5 \times (0.6 - 1) = -0.24$ $x_2(3) = x_2(2) + v_2(3) = 1 - 0.24 = 0.76$ $f((x_2(3)) = 0.76^2 = 0.5776$ and it was 1 update $p_2(3) = 0.76$ (as it is improved)
From the two particles, compare personal bests to update the global best: $g(3) = -0.12$ as $f(p_1) = 0.0144$ *and* $f(p_2) = 0.5776$	

Particle	$x_i(3)$	$v_i(3)$	$p_i(3)$	$f(p_i(3))$
1	−0.12	−0.72	−0.12	0.0144
2	0.76	−0.24	0.76	0.5776

→ Global best: $g(3) = -0.12$

Important note:

- The current position x_i shows **where the particle is now**, while the personal best p_i shows **the best place it has ever been**.
- They become equal **only if the particle improves** and finds a better solution than before.
- If the particle moves to a **worse position**, p_i stays the same and doesn't update.
- In that case, the term $(p_i - x_i)$ in the velocity equation pulls the particle **back toward its best position**.
- So, in our simple example $x_i = p_i$ happened every time because each move was an improvement, but in real problems this is **not always true** as particles often move to worse places before improving again.
- The following flow chart represents the core methodology of PSO



Example:

We want to **maximize validation accuracy** by tuning 3 hyperparameters for a CNN:

Learning rate→ lr ∈ [0.001, 0.050]

Dropout → do ∈ [0.00, 0.60]

Hidden units → hid ∈ [32, 256] (will round to nearest integer for training)

Assume a **deterministic mock fitness** to stand in for validation accuracy:

$$\text{fitness} = 1 - [0.3\left(\frac{lr - 0.01}{0.01}\right)^2 + 0.4\left(\frac{do - 0.3}{0.3}\right)^2 + 0.3\left(\frac{hid - 128}{128}\right)^2]$$

Assume that:

Number of particles	3
w	0.5
$\boldsymbol{c_1}$	1.5
$\boldsymbol{c_2}$	1.5
$\boldsymbol{r_1}$	0.4767
$\boldsymbol{r_2}$	0.9442
iterations	1

Initial state:

Particle	Position ((lr, do, hid))	velocities	Fitness
P1	$(0.030, 0.50, 80)$	$(0, 0, 0)$	−0.4218
P2	$(0.008, 0.10, 140)$	$(0, 0, 0)$	0.8058
P3	$(0.015, 0.35, 220)$	$(0, 0, 0)$	0.7584

- $pbest$ initially equals each particle's position.
- $gbest$ = P2 (fitness 0.807586).

Iteration 1:

Particle	positions(1) (lr, do, hid)	velocities(1) (v_lr, v_do, v_hid)	pbest(1) (lr, do, hid)	$f(p_i(1))$
P1	$(0.001, 0.000, 165)$	$(-0.03115860, -0.5665200, +84.97800)$	$(0.001, 0.000, 165)$	0.331932861328125
P2	$(0.00800000, 0.1000000, 140)$	$(0.00000000, 0.0000000, 0.00000)$	$(0.008, 0.100, 140)$	0.80758550347222222222 $(gBest)$
P3	$(0.00508590, 0.000, 107)$	$(-0.00991410, -0.3540750, -113.30400)$	$(0.015, 0.350, 220)$	0.519479912398125

Iteration 2:

Particle	positions(2) (lr, do, hid)	velocities(2) (v_lr, v_do, v_hid)	pbest(2) (lr, do, hid)	$f(p_i(\mathbf{2}))$
P1	$(0.00100000, 0.00000000, 172)$	$(-0.005665200, -0.14163000, +7.0815000)$	$(0.00100000, 0.00000000, 165)$	0.32155078125
P2	$(0.00800000, 0.10000000, 140)$	$(0.000000000, 0.00000000, 0.0000000)$	$(0.00800000, 0.10000000, 140)$	0.80758550347222222222
P3	$(0.01134517, 0.21486000, 178)$	$(0.0062592670350, 0.21486000, 70.886550)$	$(0.01134517, 0.21486000, 178)$	0.91657823375634792633 $(new\ gBest)$

Note that: dimensions here was 3, accordingly complexity has been increased

Step (1): load data, split data, and define fitness function

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# ==========================
# Load and split data
# ==========================
iris = load_iris()
X_train, X_val, y_train, y_val = train_test_split(
    iris.data, iris.target, test_size=0.3, random_state=42
)

# ==========================
#  Define objective function (fitness)
# ==========================
def fitness_function(params):
    C, max_iter = params
    try:
        model = LogisticRegression(C=C, max_iter=int(max_iter))
        model.fit(X_train, y_train)
        preds = model.predict(X_val)
        acc = accuracy_score(y_val, preds)
        return acc
    except:
        return 0
```

Step (2): Initialize PSO parameters

```python
num_particles = 5
num_dimensions = 2  # [C, max_iter]
max_iterations = 10
w = 0.7     # inertia (balance between exploration & exploitation)
c1 = 1.5    # cognitive (personal)
c2 = 1.5    # social (global)

# Parameter bounds: [min, max] for each dimension
bounds = np.array([[0.01, 10], [50, 300]])
```

Step (3): Initialize particles

```python
np.random.seed(42)
positions = np.random.uniform(bounds[:, 0], bounds[:, 1], (num_particles, num_dimensions))
velocities = np.zeros((num_particles, num_dimensions))
personal_best_positions = positions.copy()
personal_best_scores = np.zeros(num_particles)

# Evaluate initial scores
for i in range(num_particles):
    personal_best_scores[i] = fitness_function(positions[i])

# Identify global best
global_best_index = np.argmax(personal_best_scores)
global_best_position = personal_best_positions[global_best_index].copy()
global_best_score = personal_best_scores[global_best_index]
```

Step (4): Main PSO loop

```python
for iteration in range(max_iterations):
    for i in range(num_particles):
        # Update velocity
        r1, r2 = np.random.rand(), np.random.rand()
        velocities[i] = (
            w * velocities[i]
            + c1 * r1 * (personal_best_positions[i] - positions[i])
            + c2 * r2 * (global_best_position - positions[i])
        )
        # Update position
        positions[i] += velocities[i]
        # Clamp to bounds
        positions[i] = np.clip(positions[i], bounds[:, 0], bounds[:, 1])
        # Evaluate new fitness
        score = fitness_function(positions[i])
        # Update personal best
        if score > personal_best_scores[i]:
            personal_best_scores[i] = score
            personal_best_positions[i] = positions[i].copy()
            # Update global best
            if score > global_best_score:
                global_best_score = score
                global_best_position = positions[i].copy()
    # Print results
    print(f"Iteration {iteration+1}/{max_iterations}")
    print(f"Global Best Position: {global_best_position}")
    print(f"Global Best Accuracy: {global_best_score:.4f}\n")

# ==========================
#  results summury
# ==========================
print("✅ Optimization Finished")
print("Best Hyperparameters Found:")
print(f"C = {global_best_position[0]:.3f}, max_iter = {int(global_best_position[1])}")
print(f"Validation Accuracy = {global_best_score:.4f}")
```

```
Iteration 1/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 2/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 3/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 4/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 5/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 6/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 7/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 8/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 9/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000

Iteration 10/10
Global Best Position: [  3.75165579 287.6785766 ]
Global Best Accuracy: 1.0000
```

```
✅ Optimization Finished
Best Hyperparameters Found:
C = 3.752, max_iter = 287
Validation Accuracy = 1.0000
```

Limitations of PSO

1. May converge prematurely to local optimum
2. Sensitive to parameter settings (w, c1, c2)
3. No guarantee of finding global best
4. High computational cost for complex fitness functions
5. Weak performance on high-dimensional or discrete problems

Summary Comparison

Algorithm	Type	Main Idea	Exploration Ability	Performance in ML / DL	Typical Use
Hill Climbing (HC)	Single-solution, deterministic	Always moves to a better neighbor	Very low and gets stuck easily	Weak in deep or high-dimensional models	Simple parameter tuning, small ML problems
Simulated Annealing (SA)	Single-solution, probabilistic	Accepts worse moves with decreasing probability	Moderate and escapes local minima early	Works for moderate ML tasks; slow for deep models	Clustering, image registration, model tuning
Tabu Search (TS)	Single-solution with memory	Avoids revisiting recent (tabu) solutions	Good for discrete search	Performs well in combinatorial or feature selection tasks	Feature selection, pattern matching, scheduling
Particle Swarm Optimization (PSO)	Population-based (multi-solution)	Particles share info (pbest, gbest) to find best region	High and strong global search	Excellent for ML, Deep Learning, CV, and NLP	Hyperparameter tuning, CNN optimization, transformer tuning