
PARALLEL CELLULAR ALGORITHM (PCA)

PSEUDOCODE

```
Initialize grid_rows, grid_cols, neighborhood, α, max_iter
Initialize population X[r][c] with random values
Compute fitness F[r][c] = f(X[r][c])

t = 0
while t < max_iter:
    for each cell (r, c):
        N = neighbors of (r, c)
        best = neighbor with best (lowest) fitness
        X_new[r][c] = X[r][c] + α * (X[best] - X[r][c])
        # (or use average of neighbors / direct best replacement)

    Replace X with X_new (synchronous update)
    Recompute all fitness values F[r][c]
    Track and store global best
    t = t + 1

Return global best solution found
```

ADVANTAGES

• Highly parallel — all cells update simultaneously.
• Very scalable for large optimization problems.
• Works naturally on grid-structured or spatial data.
• Good balance of local and global search via neighborhoods.
• Robust on noisy, multimodal, complex landscapes.
• Flexible update rules; easy to customize.
• Avoids premature convergence better than some algorithms.

DISADVANTAGES

• Convergence is slower (local neighborhood updates).
• Performance depends heavily on chosen neighborhood size.
• Not suitable for very high-dimensional vector problems.
• Requires more memory (entire grid stored).
• Can stagnate if neighborhood is too small.
• More complex tuning compared to PSO/CSA.

WHERE WE USE PCA

• Image processing (denoising, edge detection, segmentation)
• Grid-based optimization problems
• Large-scale spatial models
• Routing, scheduling, resource allocation
• Distributed computing environments
• Any scenario where parallel hardware (GPU/multicore) is available

• Problems benefiting from local-neighbor interactions

WHERE WE DO NOT USE PCA

• Simple convex or differentiable functions (GD is faster)
• Extremely high-dimensional optimization (1000+ vars)
• When very fast convergence is required
• Problems with no natural neighborhood or grid structure
• When memory is limited (large grids = heavy memory use)
• Real-time tasks needing instant optimization
• When each evaluation is expensive (many evaluations per iteration)

GREY WOLF OPTIMIZER (GWO)

PSEUDOCODE

```
Initialize number of wolves, search space bounds, max_iter
Randomly initialize wolf positions Xi (i = 1 to n)
Evaluate fitness f(Xi)

Identify alpha (best), beta (2nd best), delta (3rd best)

t = 0
while t < max_iter:

    Compute coefficient ’a’ decreasing from 2 → 0

    For each wolf i:
        Generate random vectors r1, r2 in [0,1]
        Compute A = 2*a*r1 - a
        Compute C = 2*r2

        Compute distances:
            D_alpha = |C * X_alpha - Xi|
            D_beta  = |C * X_beta  - Xi|
            D_delta = |C * X_delta - Xi|

        Compute candidate positions:
            X1 = X_alpha - A * D_alpha
            X2 = X_beta  - A * D_beta
            X3 = X_delta - A * D_delta

        Update wolf position:
            Xi_new = (X1 + X2 + X3) / 3

    Apply boundary limits
    Recompute all fitness values
    Update alpha, beta, delta
    t = t + 1

Return alpha wolf (best solution found)
```

ADVANTAGES

• Simple, easy-to-implement algorithm.
• Very few parameters → fast tuning.
• Strong balance between exploration ($A > 1$) and exploitation ($A < 1$).
• Good at escaping local minima.
• Works well on continuous, non-linear, multimodal problems.
• Computationally lightweight.
• Converges smoothly due to averaging of alpha/beta/delta guidance.

DISADVANTAGES

• May converge prematurely if diversity reduces too quickly.
• Not ideal for highly constrained optimization problems.
• Exploration decreases linearly → may get stuck late in search.
• Dependent on random coefficients (results vary run-to-run).
• Not suitable for extremely high-dimensional problems.

WHERE WE USE GWO

• Engineering optimization (mechanical, structural, electrical).
• ML model training (weight tuning, hyperparameter selection).
• Feature selection and dimensionality reduction.
• Image processing (clustering, segmentation, enhancement).
• Scheduling, routing, resource allocation.
• Non-linear, multi-modal optimization problems.
• Any scenario needing fast, derivative-free global optimization.

WHERE WE DO NOT USE GWO

• Simple convex optimization (gradient methods work better).
• High-dimensional optimization (>500–1000 variables).
• Problems requiring strict, fast convergence guarantees.
• Real-time systems (stochastic updates may be unpredictable).
• Discrete or combinational problems unless modified.
• Expensive function evaluations (many wolves × many iterations).

CUCKOO SEARCH ALGORITHM (CSA)

PSEUDOCODE

Initialize n (number of nests), Pa (discovery probability),
α (step size for Levy flight), and max_iter
Generate initial population Xi (i = 1 to n)
Evaluate fitness f(Xi)

```
t = 0
while t < max_iter:

    # Generate new cuckoo solution by Levy flight
    For each cuckoo i:
        Xi_new = Xi + α * Levy()

        Evaluate f(Xi_new)

        Randomly choose a nest j
        if f(Xi_new) < f(Xj):
            Replace Xj with Xi_new

    # Abandon worst nests (discovery probability Pa)
    For each nest:
        if rand() < Pa:
            Replace nest with a new random solution

    Keep the best nests (elitism)
    t = t + 1

Return the best solution found
```

ADVANTAGES

• Excellent global search ability due to Levy flights.
• Simple algorithm with very few parameters (n, Pa, α).
• Avoids premature convergence better than many optimizers.
• Efficient on multimodal and non-linear optimization problems.
• Strong exploration → capable of escaping local minima.
• Works well for continuous, real-valued optimization.

DISADVANTAGES

• Convergence may be slow in fine-tuning (weak exploitation).
• Results vary due to random Levy flights.
• Sensitive to parameter Pa and step size α.
• Not ideal for very high-dimensional problems.
• Requires many fitness evaluations → expensive for slow functions.
• Default algorithm handles only continuous search spaces.

WHERE WE USE CSA

• Optimization of continuous mathematical functions.
• Neural network training and hyperparameter tuning.
• Feature selection in machine learning.
• Scheduling & routing problems.
• Traveling Salesman Problem (with modifications).
• Engineering optimization (mechanical, electrical, structural).
• Solving knapsack and other metaheuristic problems.
• Situations where global search is more important than speed.

WHERE WE DO NOT USE CSA

- Simple convex problems (gradient descent is faster).
- Very high-dimensional optimization (>1000 variables).
- When quick convergence is required.
- When function evaluation is expensive (requires many iterations).
- Exact / deterministic solutions needed (CSA is stochastic).
- Discrete or combinational problems unless specially adapted.

ANT COLONY OPTIMIZATION (ACO)

PSEUDOCODE (FOR TSP)

```
Initialize number of ants, α (pheromone influence),
β (heuristic influence), ρ (evaporation rate),
max_iter, and initial pheromone τ(i,j) on all edges.

t = 0
while t < max_iter:

   For each ant k:
      Start at a random city
      Build a complete tour:
         At each step choose next city j from i with probability:
            P(i,j) = [τ(i,j)]^α * [η(i,j)]^β / Σ([τ(i,u)]^α * [η(i,u)]^β)
         where η(i,j) = 1 / distance(i,j)

      Compute tour length L_k

   Update pheromones:
      For all edges (i,j):
         τ(i,j) = (1 - ρ) * τ(i,j)       # evaporation

      For each ant k:
         For edges in ant k's tour:
            τ(i,j) += Q / L_k          # deposition (better tours deposit more)

   Keep track of the best tour found
   t = t + 1

Return the best tour and its length
```

ADVANTAGES

- Intuitive, nature-inspired algorithm.
- Excellent for combinatorial optimization problems (like TSP).
- Performs distributed parallel search.
- Balances exploration (pheromone evaporation) and exploitation (pheromone reinforcement).
- Adapts well to dynamic environments (changing distances / constraints).

• Can discover very high-quality solutions for large discrete problems.
• Works even when the search space is huge and complex.

DISADVANTAGES

• Can be slow for very large problem sizes.
• May suffer from premature convergence (pheromone stagnation).
• Requires careful parameter tuning (α, β, ρ, number of ants).
• Computationally expensive if many ants or iterations are used.
• Sensitive to initial pheromone settings.
• Not ideal for continuous optimization problems without modifications.

WHERE WE USE ACO

• Traveling Salesman Problem (TSP)
• Vehicle routing and delivery planning
• Network routing in telecommunications & internet systems
• Scheduling tasks (manufacturing, cloud computing, CPU jobs)
• Robotics path planning
• Resource allocation and logistics optimization
• Assignment problems (matching, routing, load balancing)
• Any complex combinatorial optimization with discrete search space

WHERE WE DO NOT USE ACO

• Simple continuous optimization problems
• Small or trivial problems (ACO is overkill)
• Real-time applications needing very fast responses
• Extremely large-scale problems with limited computation power
• Problems where gradient-based optimization works easily
• Situations with extremely expensive objective evaluations
• Tasks where premature convergence must be strictly avoided

PARTICLE SWARM OPTIMIZATION (PSO)

PSEUDOCODE

```
Initialize number of particles, W (inertia), C1, C2,
max_iter, and random initial positions Xi and velocities Vi.
For each particle:
    Evaluate fitness f(Xi)
    Set personal best pbest_i = Xi

Set global best gbest = best pbest

t = 0
while t < max_iter:

    For each particle i:
```

```
        Generate random r1, r2 in [0, 1]

        # Velocity update
        Vi = W*Vi + C1*r1*(pbest_i - Xi) + C2*r2*(gbest - Xi)

        # Position update
        Xi = Xi + Vi

        Evaluate fitness f(Xi)

        # Update personal best
        if f(Xi) < f(pbest_i):
            pbest_i = Xi

    # Update global best
    gbest = best pbest among all particles

    t = t + 1

Return gbest as best solution
------------------------------------------------------------
```

ADVANTAGES

• Very simple and easy to implement.
• Few parameters to tune (W, C1, C2).
• Works well for continuous optimization problems.
• Does not require gradient information (derivative-free).
• Fast global search ability in early iterations.
• Computationally efficient; parallelizable.
• Good at exploring complex multimodal landscapes.

DISADVANTAGES

• Slow convergence in fine-tuning stage (weak local search).
• May get stuck in local optima if diversity decreases.
• Sensitive to parameter settings (W, C1, C2 must be balanced).
• Performance degrades in very high-dimensional search spaces.
• No guarantee of exact global optimum (stochastic algorithm).
• Velocity explosion may occur if not clamped or controlled.

WHERE WE USE PSO

• Continuous mathematical optimization problems.
• Machine learning (weight tuning, hyperparameter optimization).
• Neural network training.
• Engineering design problems (mechanical, structural, electrical).
• Robotics path planning.
• Clustering, feature selection.
• Benchmark functions (Rastrigin, Rosenbrock, Sphere, etc.).
• Any black-box problem with unknown or non-differentiable objective.

WHERE WE DO NOT USE PSO

• Simple convex or smooth problems (gradient descent is faster).
• Extremely high-dimensional problems (>1000 variables).
• Problems requiring highly accurate local convergence.
• Discrete or combinatorial optimization (unless modified PSO is used).
• Real-time systems needing strict deterministic behavior.
• Expensive objective functions (requires many evaluations).

GENETIC ALGORITHM (GA)

PSEUDOCODE

```
Initialize population size, encoding scheme, crossover rate,
mutation rate, and max_generations.
Generate initial population of chromosomes.

For each chromosome:
    Evaluate fitness using fitness function.

t = 0
while t < max_generations:

    # Selection
    Select parent chromosomes using
    (Tournament Selection / Roulette Wheel).

    # Crossover
    With probability Pc:
        Apply crossover (Single-point / Two-point / Uniform)
        to generate offspring.

    # Mutation
    With probability Pm:
        Apply mutation (Bit Flip / Swap / Gaussian)
        to maintain diversity.

    # Form new population
    Evaluate fitness of new offspring.
    Replace old population with new one (elitism optional).

    Update best chromosome found.
    t = t + 1

Return best chromosome and its fitness.
```

ADVANTAGES

• Excellent global search capability.
• Works well for complex, non-linear & multi-objective problems.

• Does not require gradient or derivative information.
• Handles discrete, continuous, and mixed decision variables.
• Very flexible (supports many crossover/mutation types).
• Good at avoiding local minima due to mutation + crossover.
• Naturally parallel — population-based.

DISADVANTAGES

• Computationally expensive (many evaluations needed).
• Convergence speed can be slow.
• Sensitive to parameter tuning (Pc, Pm, population size).
• May produce infeasible solutions in constrained problems.
• No guaranteed optimal solution — stochastic behavior.
• Poor fine-tuning ability compared to gradient techniques.

WHERE WE USE GA

• Traveling Salesman Problem (TSP)
• Scheduling & resource allocation
• Feature selection & dimensionality reduction
• Neural network training & hyperparameter tuning
• Engineering design optimization (mechanical, structural, electrical)
• Circuit design & component placement
• Manufacturing optimization
• Multi-objective optimization problems
• Search problems where solution space is huge or discontinuous

WHERE WE DO NOT USE GA

• Simple convex problems (gradient descent is faster).
• Problems requiring guaranteed exact optimum.
• Very high-dimensional continuous problems (slow convergence).
• Real-time or low-latency applications.
• Optimization tasks where function evaluation is expensive.
• Situations with strict constraints (requires specialized GA variants).
