INDEX

NAME: Prakrithi Jain STD.: V SEC.: E ROLL NO.: _____ SUB.: Bio-Inspired systems LAB

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	21/08/25	Overview of Algorithms: 1) Genetic Algorithm of Optimization Problems 2) Particle Swarm Optimization for function optimization 3) Ant Colony Optimization for the Traveling Salesman Problem 4) Cuckoo Search 5) Grey wolf Optimizer 6) Parallel Cellular Algo & Programs 7) Optimization Gene Expression Algo		
2.	28/08/25	Genetic Algorithm Pseudocode Robotic process	10	
3	4/09/25	Gene Expression Programming Image Processing - GA, GS	10	
4	11/09/25	Particle Swarm Optimization	5	
5	9/10/25	Ant Colony Optimization Packets transmission via Router	7	
6	16/10/25	Cuckoo Search	10	
7	23/10/25	Grey wolf Optimizer	10	
8	30/10/25	Parallel Cellular Algorithm	10	

LAB-1 BIO INSPIRED SYSTEMS

OVERVIEW OF ALGORITHMS

1. Genetic Algorithms for Optimization Problems :

A genetic algorithm (GA) is a problem solving technique inspired by Charles Darwin's theory of natural evolution. It operates on the principle of natural selection, where the fittest individuals are chosen for reproduction to produce the next generation's offspring. Think of it as solving a puzzle with multiple potential solutions. By iteratively selecting, combining, and mutating these solutions, a genetic algorithm gets closer to the optimal solution with each step, like assembling a puzzle piece by piece.

Key Concept :

1) Population : is a group of potential solutions that evolve over time. It represents different guesses at the answer to the problem.

2) Chromosome : is a blueprint of the solution.

3) Gene : A gene is a part of chromosome that represents a part of solution where it holds a value that contributes to the overall solution.

4) Fitness function : Higher the fitness score, better the solution.

5) Selection : The fittest solutions have a higher chance of being selected.

6) Crossover: It mimics biological reproduction where 2 parents produce offspring that inherit traits from both.

7) Mutation: Ensures diversity within the population and prevents convergence to suboptimal solutions.

Advantages:

* Flexible: This algorithm can be applied to various domain it is not only limited to mathematics and computer science domain

* Global Search: This algorithm is good at finding a global optimum in a large search space.

* Simple and Parallelizable: This process is easy to understand and can be run on multiple processors at a single time.

Disadvantages:

* Expensive: requires a lot of resources and time for complex problems.

* No Optimal Solution: We cannot say that we have evaluated the best solution.

Applications:

* Used in engineering for optimizing structures, electronic circuits and systems.
* Used in AI domain to evolve neural networks, decision trees

2) Particle swarm Optimization for function Optimization

Particle swarm Optimization (PSO) is a population-based, stochastic optimization algorithm inspired by the social behaviour of bird flocks or fish schools. It is widely used for function optimization aiming to find the optimal solution (minimum or maximum) of a given objective function.

Key Concepts

- Particles : Each potential solution in the search space is represented as a particle.
- Swarm : A collection of these particles constitutes the swarm.
- Position and Velocity : Each particle has a position in search space and a velocity that dictates its movement.
- pBest : Each particle keeps track of the best position it has ever achieved.
- gBest : The entire swarm keeps tracks of the best position found by any particle in the swarm.

Advantages

* Simplicity : PSO is relatively easy to understand and implement.
* It is efficient as it generally requires fewer parameters to tune.

* It is effective as it has proven effective in solving a wide range of optimization problems including continuous, discrete and constrained problems.

Disadvantages:

* Premature Convergence to Local Optima:

In high dimensional search spaces, PSO can get trapped in local optima, failing to find the true global optimum. This is particularly true if the swarm converges too quickly before adequately exploring the search space.

* Slow convergence in refined search.

3. Ant Colony Optimization (ACO) for TSP

ACO is a metaheuristic algorithm inspired by the foraging behaviour of ants. For the travelling Salesman Problem (TSP), ACO simulates ants finding the shortest route between cities by depositing and following pheromone trails.

Advantages:

* ACO can adapt to changes in the problem environment making it suitable for dynamic TSP scenarios.

* ACO algorithms can handle large-scale TSP instances effectively.

4) Cuckoo Search (CS)

Cuckoo search is a metaheuristic optimization algo inspired by the brood parasitism of cuckoo birds. It mimics how cuckoos lay their eggs in other bird's nests, with the host bird potentially discovering and discarding the foreign egg. CS uses this behaviour to explore the solution space with "nests" representing potential solutions & cuckoo eggs representing new, better solutions.

Advantages

* CSA typically requires fewer parameters to be tuned, which simplifies the optimization process.

* It has demonstrated effectiveness in finding global optima across a range of complex optimization problems and landscapes.

Disadvantages:

* CSA can sometimes struggle with fine-tuning solutions locally, potentially leading to slower convergence.

* For large-scale optimisation problems, the computational cost associated with evaluating solutions and performing Levy flights might become significant.

5) Grey Wolf Optimizer (GWO)

It is a population-based metaheuristic algorithm inspired by the social hierarchy and hunting behaviour of grey wolves to solve complex optimization problems. It simulates a hierarchical structure with alpha, beta, delta and omega wolves, with the fittest solutions representing the alphas. The algorithm's hunting mechanism involves three stages: encircling prey, attacking prey and searching for prey which helps the population of solutions to converge towards the optimal solution.

Biological Inspiration :

Grey wolves live and hunt in packs. Their behaviour is modeled as follows:

1) Social Hierarchy :

- Alpha (α) → Pack leader, decision maker → represents the best solution.
- Beta (β) → Advisor, second in command → represents the second-best solution.
- Delta (δ) → Scouts, hunters, caretakers → represent the third-best solution.
- Omega (ω) → Followers, lowest rank → the rest of the solutions in the population.

2) Hunting Strategy:

- Encircling prey: Wolves surround prey (exploration of solution space).
- Hunting → α, β, δ guide the pack towards prey (moving towards promising solutions).
- Attacking prey → Wolves close in when prey stops moving (convergence to * optimum).

Advantages of GWO

- Simple & easy to implement (few parameters).
- Good exploration - exploitation balance.
- Efficient global search (can escape local optima better than some algorithms).
- No gradient needed (works on non-linear, non-differentiable problems).
- Versatile (used in engineering, ML feature selection, scheduling, power systems, image processing etc.

Disadvantages

- Slower local exploitation compared to more advanced optimizers.
- Sensitive to parameters like population size and iterations.
- Being stochastic → results can vary across runs (needs multiple trials).

Applications of GWO

- Feature selection in machine learning
- Engineering optimization (structural design, power systems).
- Image processing
- Task scheduling cloud computing.
- Solving non linear equations and benchmark test functions.

6) Parallel Cellular Algorithms and Programs

PCA: A class of parallel evolutionary algorithms where the population is distributed over a cellular grid or lattice, and each individual interacts only with its local neighbours. This structure maintains diversity, allows parallel execution, and improves scalability on large computing architectures.

PCP: A parallel programming model based on a grid of independent processing cells, where each cell has its own local state and communicates only with its neighbours. This enables massive parallelism, scalability and robustness making it suitable for scientific simulations, optimization and distributed computing.

Advantages of PCAs/PCPs

- Naturally parallelizable → suitable for modern hardware
- scalability → works well on large-scale problems.
- High diversity in search algorithms.

Disadvantages

- More complex implementation compared to centralized algorithms (due to distributed nature).
- Slower convergence sometimes (due to distributed nature).
- Needs carful design of neighborhood topology (grid size, shape, connectivity).

7) Gene Expression Programming (GEP)

GEP is an evolutionary algorithm that optimizes complex problems by mimicking biological evolution, specifically the process of gene expression. It separates the genotype (linear chromosome) from the phenotype (expression tree), allowing for efficient genetic operations while ensuring all resulting solutions are syntactically valid programs.

Applications of GEP

1) Symbolic Regression - Discovering mathematical expressions or models from data.

2) Classification & Prediction: Used in machine learning tasks

3) Data Mining & Knowledge Discovery - Extracting patterns, relationships and rules from large datasets.

4) Bioinformatics: Gene regulatory, protein structure prediction.

5) Engineering Optimization - Circuit design, resource allocation.

6. Image & Signal Processing - feature selection.

Advantages of GEP

- Powerful search & optability → can evolve complex models.
- Fixed-length chromosomes + flexible expression tree
- Simplicity - fewer parameters to tune than some evolutionary methods.
- Robustness - handles noisy, incomplete or nonlinear data well.

Disadvantages

- Computationally expensive → requires large populations
- Solution interpretability → evolved expressions may be very complex and hard to understand.

classmate
Date
Page

GEP	GA
• Fixed-length chromosomes expressed as variable-length expression trees (phenotype)	Fixed-length chromosomes (bit strings, real numbers)
• Produces explicit models/ equations	Produces optimized parameter values.
• Can evolve functions, models, rules symbolic regression.	Best suited for parameter optimization and search.

Genetic Algorithm

28/8

classmate
Date 28/08/25
Page

LAB-2

Algorithms

Genetic Algorithm Pseudocode

```
Genetic_Algorithm()
// Initialization
// Initialize the individual in the given population.
P ← Initialize_population(size = N)

# fitness
for each individual x in P:
    fitness[x] ← Evaluate_fitness(x)

while not Termination_condition():   // Selection
    parents ← Select(P, method = "tournament" or "roulette")

    // Crossover
    offspring ← []
    for i in 1 to |parents|/2:
        if rand() < Pc:   // Pc crossover probability
            child1, child2 ← crossover(parents[2*i-1], parents[2*i])
        crossover of two parents
        else:
            child1, child2 ← parents[2*i-1], parents[2*i]
        offspring.append(child1)
        offspring.append(child2)

    // Mutation
    for each in offspring:
        if rand() < Pm   // Mutation probability
            child ← Mutate(child)

    // Fitness Evaluation
    for each child in offspring:
        fitness[child] ← Evaluate_fitness(child)
```

// Termination.

P ← selection_of_successor(P, offspring)
best ← fitness[x] over x ∈ P
return best.

- Execute WRT - Robotic process.

Generic version of Genetic Algorithm
1) Randomly initialize population P
2) Determine fitness of population
3) Until convergence repeat:
   a) Select parents from population
   b) Perform mutation on new population
   c) Crossover and generate new population
   d) Calculate fitness for new population.

~~Genetic Alg~~

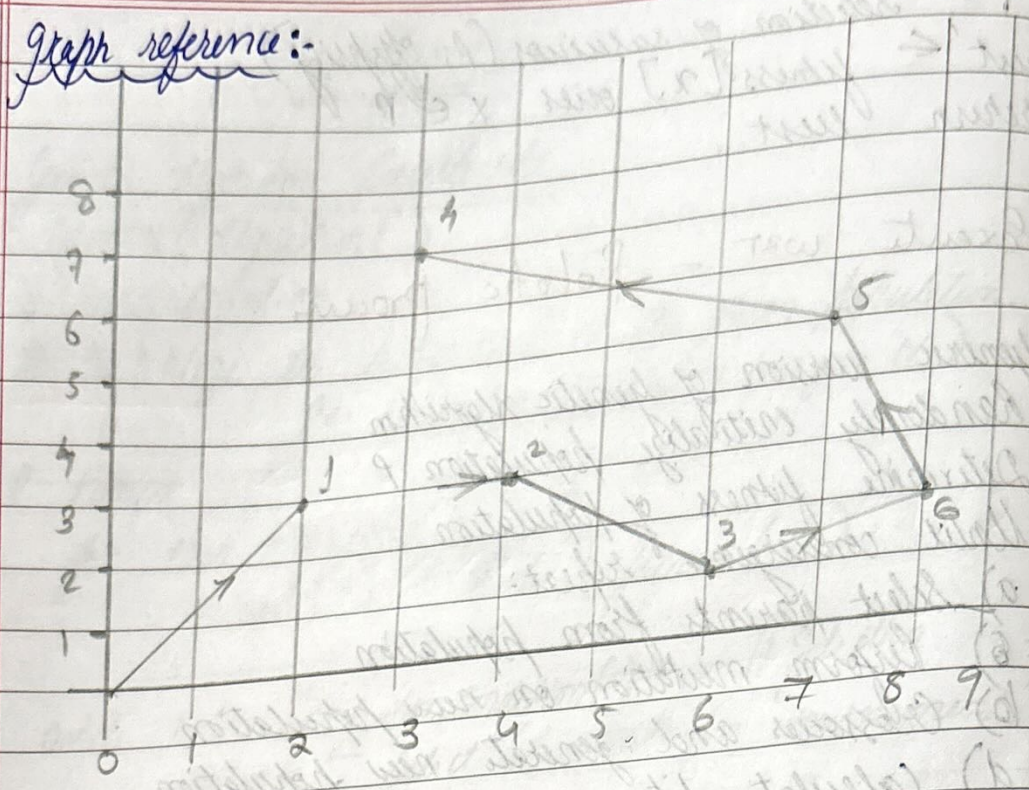
Robotic Process implementation:

Problem: Find the shortest path for a robot that visits all given waypoints starting from (0,0) and returns the best possible sequence of waypoints.

Input: [(0,0), (2,3), (4,3), (6,1), (3,7), (9,5), (8,2)]

P.T.O →

Graph reference:-



Output:

1) Generation 0 : Best distance = 23.7
2) Generation 10 : Best distance = 16.3
3) Generation 20 : Best distance = 16.3
4) Generation 30 : Best distance = 19.2
5) Generation 40 : Best distance = 20.1
6) Generation 50 : Best distance = 16.4
7) Generation 60 : Best distance = 16.3
8) Generation 70 : Best distance = 16.3
9) Generation 80 : Best Distance = 16.3
10) Generation 90 : Best Distance = 16.3

Best path : [0, 1, 2, 3, 6, 5, 4]

Shortest distance path : 16.3044577772

classmate
Date 4/09/25
Page

Particle Swarm Optimization for function Optimization

Gene Expression Programming (GEP)

Input:
fitness function f(n)
population size, num_genes,
mutation_rate, generations,
crossover_rate.

Output:
best solution found
Initialize population
population = [random.chromosome(num_gene)
for i in i..... population size].

best solution = ϕ

for g = 1 to generation do
fitness_list []

for chromosome in population:
sol = express(chromosome)
fit = f(solution)
if fit > best fitness
best fitness = fit
fit_solution = chromosome

# Selection:
parents = [ ]
for i in 1 ... population size:

c1, c2 = random individuals
if fitness(c1) > fitness(c2)
    parents.add(c1)
else:
    parents.add(c2)

crossover
offspring ← []
for i in [illegible] to len(parents) // 2
    p1, p2 = parent[i], parent[i+1]
    if rand() < [illegible] pc // crossover prob
        point = random index
        child1 = p1[0:point] + p2[point:]
        child2 = p2[0:point] + p1[point:]
else
    child1, child2 = p1, p2

# mutation:
for gene in child1:
    if rand() < [illegible] pm // mutation prob: flip
for gene in child2:
    if rand() < pm: flip

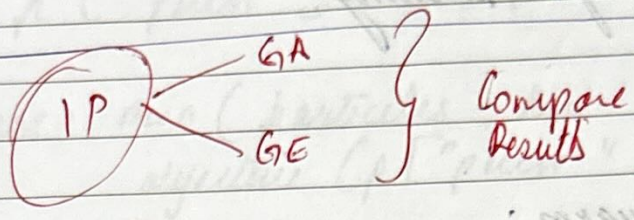
Gene Expression: decoding to real solution
done in evaluation step.
offsprings.append(child1)
offsprings.append(child2)

# Replace old population
population = offspring

Reflex test

return best solution, best fitness

end



GE → "Fails" — Best results.

Output

Successfully loaded target image
Starting with a random canvas mismatched pixels = 20007
Generation 191 : Mismatched pixels = 19907
Generation 382 : Mismatched pixels = 19807

Generation 208722 : Mismatched pixels = 607
Generation 313380 : Mismatched pixels = 7

Output Successful

★ This program was faster in comparision to genetic algorithms even for 500×500 images, it effortlessly finishes iterating in seconds.

- Fails for small images

Lab-4

PARTICLE SWARM OPTIMIZATION

Optimization algorithm inspired by social behaviour of birds flocking or fish schooling.

Algorithm

1. Initialize the swarm:
Each particle has its position, velocity and remembers its pbest.

```
import random

def objective(x):
    # according to requirement
particles = []

for _ in range(num_particles):
    pos = [random.uniform(-10, 10)]
    vel = [random.uniform(-1, 1)]

    particles.append({
        "position": pos,
        "velocity": vel,
        "pbest": pos[:]})

gbest = min(particles, key=lambda p:
            objective(p["pbest"]))["pbest"]
```

2. Evaluate fitness:
update pbest and gbest

```
for p in particles:
    if objective(p["position"]) < objective(p["pbest"]):
        p["pbest"] = p["position"][:]

gbest = min(particles, key = lambda p:
            objective(p["pbest"]))["pbest"][:]
```

3. Update velocity and position

```
for p in particles:
    new_velocity = []
    new_position = []

    for d in range(dimensions):
        new_vel.append(vel_d)
        new_pos.append(pos_d)
```

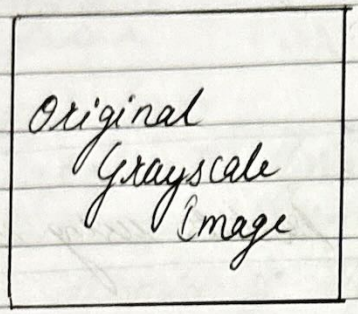
4. Run loop

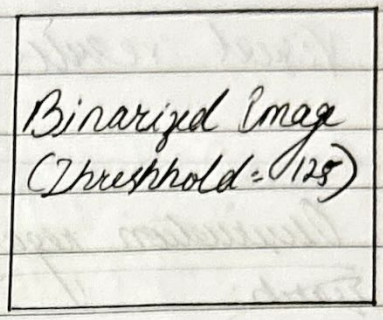
The same iteration is done for specific number of times or until minimum convergence.

PSO
V
Img processing

PSO in Image Processing

Output:





Iteration 10/50, best Otsu variance : 1423.8369
Iteration 20/50, best Otsu variance, : 1423.8369
Iteration 30/50, best Otsu variance, : 1423.8369
Iteration 40/50, best Otsu variance, : 1423.8369
Iteration 50/50, best Otsu, variance, : 1423.8369

Threshold for all: 125
=
Optimal Threshold found by PSO : 125

Observation: The PSO algorithm successfully determined an optimal threshold value for binarizing the grayscale image. By maximizing Otsu's between-class variance, PSO identified a threshold that effectively separates the foreground from the background in the image.

Output shows:

* Convergence over iterations - The PSO improved threshold value, indicated by increasing between-class variance and stabilization of the best threshold.

LAB-5

Date 9/10/20

ANT COLONY OPTIMIZATION FOR THE TRAVELLING SALESMAN PROBLEM

Algorithm

Input :

// ~~Initializing~~
// no. of cities
dist [n][n] // distance matrix
num_ants // number of the ants
num_iteration // maximum no. of iterations
alpha, beta // pheromones & heuristic importance
rho // evaporation rate | ~~constant~~
Q // pheromone deposit concent

// Initializing
pheromone [i][j] = small positive rate for all city ~~trails~~ pairs.

best_length ← ∞
best_tour ← ϕ

for ~~each~~ each num_iteration ← 1 to num_iteration
    for each ant K ← 1 to num_ants
        start_city ← random_city
        tour [n] ← start_city
        visited ← {start_city}

~~while city (visited) < N {~~
    ~~current ← last city on tour [K]~~
~~for~~

for each unvisited cities:
current ← last city in tour[k]
for each city j not in visited do:
probability[j] ← (pheromone[current][j], alpha) * (1/dist[current][j], beta)

normalize probability[j] so sum(probability) = 1

next_city ← random city based on the probability
tour[k].append(next_city)
visited.append(next_city)
}
tour[k].append(start_city). // Returning back to initial node
length[k] ← distance_total(tour[k])

}

TSP

// Evaporation of the pheromones
{
for i ← 1 to n {
for j ← 1 to n {

pheromone[i][j] ← (1 - rho) * pheromones[i][j]

}

P.T.O ⇒

// pheromones (density and the trail) update

```
for each ant k in num_ants {
    for each edges[i][j] in tour(k) {
        pheramones[i][j] = pheramone +
                           (Q / length[k])
    }
}

// Update global best
for each ant k in num_ants {
    if length[k] < best_length {
        best_length ← length[k]
        best_tour ← tour[k]
    }
}
```

Output: best_length, best_tour.

ACO

General Algorithm for ACO

Initialize pheromone values let's say τ on all of the solution components set parameters α (pheromone influence), β (heuristic influence), evaporation rate ρ, no. of ants m.

while stopping criteria not met:
    for each ant k in 1 to m:
        initialize an empty solution S_k
        while solution S_k is incomplete:
            select next solution component c based on probability proportional to:
            $[\tau(c)]^\alpha * [\eta(c)]^\beta$
            where $\eta(c)$ is heuristic desirability of component c.
        Add component c to solution S_k evaluate the quality of solution S_k.

    for each solution component c:
        evaporate pheromone:
        $\tau(c) = (1-\rho) * \tau(c)$

    for each ant k
        deposit pheromone on components in solution S_k:
        $\tau(c) = \tau(c) + \Delta\tau_k(c)$
        (ie. amount $\Delta\tau_k(c)$ depends on quality of the solution.

return best solution found.

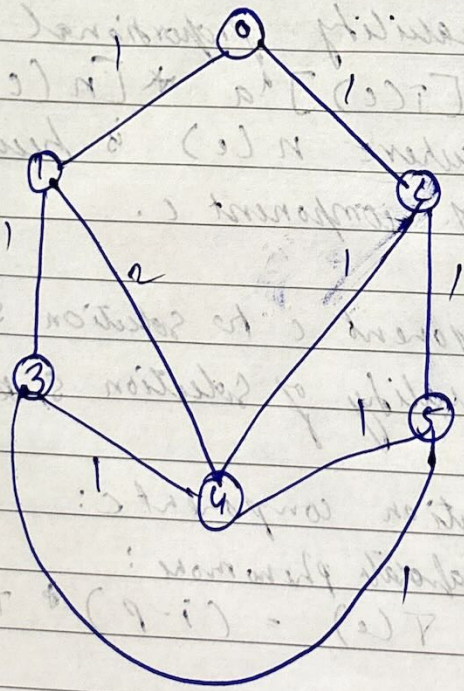
Packets transmission via Routers

Packets Transmission via Router

Input : adjacency matrix

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 2 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Graph is :



start node : 0
Destination node : 5

output : 0 → 1 → 3 → 5

cost : 3

Goal: To implement a routing strategy using the Ant Colony Optimization algorithm. ~~that~~ ~~simulates~~ how

classmate
Date 16/10/25
Page

Cuckoo Search Algorithm

Algorithm

Cuckoo Search ()

Initialize population of n host nests (solutions)
x_i ($i = 1, 2, \ldots n$)
Define the objective function $f(x)$, $x = (x_1, x_2, \ldots x_d)^T$

Find the current best solution x_best among the nests
while (t < MaxGeneration or stopping criteria is not met) do:
    for each cuckoo (i = 1 to n) do
        Generate a new solution x_i_new by Lévy flight
from x_i
        Evaluate fitness $f(x_i$_new)
        Randomly choose a nest j among n
        if ($f(x_i$_new) < $f(x_i)$) then
            replace x_j with x_i_new
        end if
    end for
    A fraction (pa) of worse nests are abandoned and replaced
        with new random solutions.
    Keep the best solution (x_best) among current nests.
    Rank the nests and find the new x_best
    t = t + 1
end while
Return x_best as the optimal solution found.

Optimize n in Nko packets

Problem Statement for Cuckoo Search

Optimization → Power Consumption

Output:

Iteration 1/100 : Best Power Consumption : 21.478
Iteration 10/100 - Best Power Consumption : 20.7345
Iteration 20/100 - Best Power Consumption : 20.314
Iteration 30/100 - Best Power Consumption : 19.7958
Iteration 40/100 - Best Power Consumption : 19.6848
Iteration 50/100 - Best Power Consumption : 19.716
Iteration 60/100 - Best Power Consumption : 19.716
Iteration 70/100 - Best Power Consumption : 19.716
Iteration 80/100 - Best Power Consumption : 19.716
Iteration 90/100 - Best Power Consumption : 19.612
Iteration 100/100 - Best Power Consumption : 19.612

Optimization Results

Optimal Parameters : [4.5885625 0.57364008
1.395972

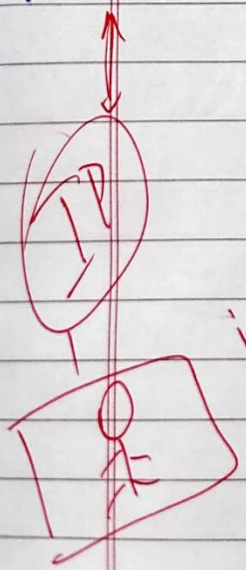
Minimum Power Consumption : 19.61250 39

Grey Wolf search optimization

Grey Wolf search Optimizer

Algorithm

1. Begin GWO
2. Define the objective function $f(x)$, $x = (x_1, x_2, \ldots x_n)$
3. Initialize parameters:
   - Number of wolves (N)
   - Number of Dimensions (D)
   - Number of maximum iterations (Max_iters)
   - Search space bounds (lb, ub)
4. Initialize the population of grey wolves
   x_i [i = 1, 2 ..., N]
   - For each wolf i:
     For each dimension j:
     X_{ij} = random (lb, ub)
5. Evaluate the fitness of every wolf
   $f_{x_i} = f(x_i)$
6. Identify the top 3 wolves based on fitness: distance to alpha
   alpha = best (lowest fitness)
   beta = second best
   delta = third best
7. For each iteration t = 1 to Max_iter do:
   - Update the control parameter
     $a = 2 - t * (2 / max\_iters)$
   - For each wolf i = 1 to N do:



ing outline looping

- For each dimension j = 1 to D do:
  Generate random numbers r1, r2 ∈ [0,1]
  Compute coefficients:
    A1 = 2 * a * r1 - a
    C1 = 2 * r2
  Compute the distance to Alpha:

  D_alpha = | C1 * Alpha[j] - X_i[j] |
  X1 = Alpha[j] - A1 * D_Alpha
  Repeat for Beta and Delta:
  (A2, C2, D_beta, X2)
  (A3, C3, D_delta, X3)
  Update position:
    X_i[j] = (X1 + X2 + X3) / 3

8. Record best fitness
9. End for
10. Return Alpha and f(Alpha) (Best fitness)

Image Outline Deepening

Iteration 1/20, Best fitness : 0.0047
Iteration 2/20, Best fitness : 0.0047
Iteration 3/20, Best fitness : 0.0059
Iteration 4/20, Best fitness : 0.0060
Iteration 5/20, Best fitness : 0.0060
Iteration 6/20, Best fitness : 0.0081
Iteration 7/20, Best fitness : 0.0175
Iteration 8/20, Best fitness : 0.0175
Iteration 9/20, Best fitness : 0.0175
Iteration 10/20, Best fitness : 0.0175
Iteration 11/20, Best fitness : 0.0175
Iteration 12/20, Best fitness : 0.0175
Iteration 13/20, Best fitness : 0.0175
Iteration 14/20, Best fitness : 0.0175
Iteration 15/20, Best fitness : 0.0175
Iteration 16/20, Best fitness : 0.0175
Iteration 17/20, Best fitness : 0.0175
Iteration 18/20, Best fitness : 0.0175
Iteration 19/20, Best fitness : 0.0175
Iteration 20/20, Best fitness : 0.0175

Optimized thresholds : [0. 1.88649006]

Fitness : 0.0175.

LAB-8

Parallel Cellular Algorithm

Algorithm:

1) Start
2) Define the optimization problem:
   A function is defined $f(x)$ to be minimized or maximized.

3) Initialize parameters:
   Set NUM_CELLS ← no number of cells
   Set GRID_SIZE ← size of grid
   Set MAX_ITER ← maximum number of iteration
   Define NEIGHBORHOOD structure (e.g., 4 neighbors, 8 neighbors)

4) Initialize population:
   For each cell i in the grid:
   Assign a random position $x[i]$ in the solution space.
   Compute fitness[i] = $f(x[i])$

5) Repeat for ITER = 1 to MAX_ITER:
   For each cell i in parallel:
   Find neighboring cells based on NEIGHBORHOOD
   Update state of cell i using update rule:
   $x[i] \leftarrow$ updated rule ($x[i]$, neighbors)
   Calculate fitness[i] = $f(x[i])$

   Find the best cell with highest (or lowest) fitness

6) Until convergence or ITER = MAX_ITER

7) Output: Print the best solution and its fitness value

8) Stop.

Sol:

Output:

Initial Image:

10 20 30 40 50
20 60 120 60 20
30 120 255 120 30
20 60 120 60 20
10 20 30 40 50

Iteration 1:

17 29 40 45 47
29 65 106 68 33
38 107 191 107 38
29 65 108 68 33
17 29 40 45 47

Iteration 2:

24 36 47 49 47
36 66 98 71 42
45 97 157 97 45
36 66 98 71 42
24 36 47 49 47

Iteration 3:

30	42	52	53	49
41	66	90	71	48
50	89	130	90	51
41	66	90	71	48
30	42	52	53	49

Iteration 4:

35	46	56	56	51
45	65	84	70	52
53	82	111	84	55
45	65	84	70	52
35	46	56	56	51

Improvement of Algorithm Over Time:

Era	Implementation	True Parallelism	TC	SC	Key Improvement
Early (2000)	CPU / Theoretical Parallel CA	Concept only	$O(n^2)$	$2 \cdot n^2$	Basic rules
Modern (2010-2025)	GPU / RL-integrated CA	Fully Parallel (real time reformance)	$O(n^2)$ less	n^2 of	Adaptive rules

30/10