Date __/__/__
Page _____

Genetic Algorithm :-

A genetic algorithm (GA) is a method to solve optimization problems by mimicking natural selection. Here's the process in short

1. Define the problem: Set the function to optimize
2. Initialise Parameters: Set population size, mutation / crossover rates and number of generations.
3. Create Initial Population: Generate random solutions.
4. Evaluate fitness: Measure how good each solution is.
5. Selection: Pick the best solutions for reproduction.
6. Crossover: Combine parts of selected solution to create new ones.
7. Mutation: Apply random changes to maintain diversity.
8. Iteration: Repeat the process for multiple generations.
9. Best Solution: Output the best solution found.

This cycle evolves better solutions over time.

Date / /
Page

Application of GA.

1. Optimization Problems
2. Machine Learning
3. Finance
4. Robotics
5. Game Development
6. Bioinformatics
7. Art and Design
8. Telecommunications.
9. Production scheduling in a Manufacturing Plant.

Algorithm

→ Start
→ Create initial population
→ Calculate fitness score for each individual
→ repeat
  ▷ ~~Solution~~ Selection
  ▷ Crossover
  ▷ Mutation
  ▷ Calculation of fitness score
  until convergence is found
→ Choose the individual with high fitness value
→ Stop.

24-10-24

Date / /
Page

Particle Swarm Optimisation for Function optimisation

Particle swarm Optimisation is inspired by the social behaviour of birds flocking or fish schooling. PSO is used to find optimal solutions by iteratively improving a candidate solution with regard to a given measure of quality.

Algorithm

Step 1: Randomly initialize swarm population of N particles Xi (i = 1, 2 ..., n)

Step 2: Select hyperparameter value
w, c1, c2

Step 3: For itr in range(max_itr):
For i in range (N):

a. Compute new velocity of ith particle
swarm[i].velocity =
w * swarm[i].velocity +
r1 * c1 * (swarm[i].bestPos - swarm[i].position) +
r2 * c2 (best_pos_swarm - swarm[i].position)

b. Compute new position of ith particle using its new velocity
swarm[i].position += swarm[i].velocity

c. If position is not in range [minx, maxx] then clip it
if swarm[i].position < minx:
swarm[i].position = minx
elif swarm[i].position > maxx:
swarm[i].position = maxx

d. Update new best of this particle and new best of swarm

if swa Insensitive do scaling of design variables. sm[i].fitness < swarm[i].bestFitness:

swarm[i].bestFitness = swarm[i].fitness

swarm[i].bestPos = swarm[i].pos

if swarm[i].fitness < best_fitness_swa
best_fitness_swarm = swarm[i].fitness
best_pos_swarm = swarm[i].position

End-for

End-for

Step 4 Return best particle for swarm.

Output

Best Position : [-7.225174070e-13
2.077081644e-13]

Best Fitness : 5.651692632044056e-25

Date / /
Page

Ant Colony Optimization

The foraging behaviour of ants has inspired the development of optimization algorithms that can solve complex problems such as the Travelling Salesman Problem (TSP).
Ant Colony Optimization simulates the way ants find the shortest path between food sources and their nest.

Algorithm

Initialise pheromone values $\forall i, j \in [1, n] : \tau_{ij} \mapsto \tau_0$
repeat
  for each ant $l \in [1 \ldots m]$ do
    initialise selection set $S \mapsto \{1 \ldots n\}$
    randomly choose starting city $i_0 \in S$ for ant l
    move to starting city $i \mapsto i_0$
    while $S \neq \phi$ do
      remove current city from selection set $S \mapsto S \setminus \{i\}$
      choose next city j in tour with probability $p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in S} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}}$
      update solution vector $\pi_l(i) \mapsto j$
      move to new city $i \mapsto j$
    end while
    finalize solution vector $\pi_l(i) \mapsto i_0$
  end for
  for each solution $\pi_l, l \in \{1, \ldots m\}$ do
    calculate tour length $f(\pi_l) \mapsto \sum_{i=1}^{n} d_{\pi(i)\pi(i+1)}$
  end for

Date __/__/__
Page

end for
for all (i,j) do
evaporate pheromone $T_{ij} \leftarrow (1-\rho) \cdot T_{ij}$
end for
determine best solution of iteration
$\pi^+ = \arg\min f(\pi_l)\ l \in [1, m]$
if π^+ better than current best π^*, ie
$f(\pi^+) < f(\pi^*)$ then set $\pi^* \leftarrow \pi^+$
end if
for all $(i,j) \in \pi^+$ do
reinforce $T_{ij} \leftarrow T_{ij} + \Delta/2$
end for
for all $(i,j) \in \pi^*$ do
reinforce $T_{ij} \leftarrow T_{ij} + \Delta/2$
end for
until condition for termination met.

Output
Best path : [0, 4, 9, 5, 1, 3, 2, 6, 8, 7, 0]
Best distance : 258.41696.

14.11.24

Date __/__/__
Page

Cuckoo Search (CS)

Purpose :- This is a nature optimised inspired optimization algorithm based on the brood parasitism of some cuckoo species. This involves laying eggs in the nests of other birds, leading to the optimization of survival strategies. CS uses levy flights to generate new solutions promoting global search capabilities and avoiding local minima.

Parameters :-

1. Number of nests (n_nests) : Controls how many candidate solutions (nests) are maintained throughout the algorithm.

2. Probability of Discovery (Pa) : Controls how often a cuckoos egg is discovered by the host bird.

3. Number of Iterations (n_itee) : Defines how many times the algorithm will run to evolve the population of nests.

4. levy Flight Parameter (beta) : This controls the step size distribution during levy flights.

5. Dimension of the ~~pbo~~ problem (dim) : Defines the number of variables or decision parameters in the problem.

Date
Page

Applications of Cuckoo Search :-

- Engineering Design
- Machine learning
- Data Mining
- Control Systems
- Robotics
- Scheduling

Algorithm (Optimizes the Rastrigin function)

1. Define the objective function : func(x)
2. Initialise parameters :
   - n_nests : Number of nests (candidate solutions)
   - pa : Probability of discovery (how often eggs are discovered by birds)
   - n_iter : Number of iterations
   - beta : levy flight parameter (controls step size distribution)
   - dim : no. of decision variables
3. Initialise nests with random solutions in search space
   Nests = Initialise random population (size n_nests, dimension dim)
4. Evaluate fitness of each nest :
   Fitness [i] = func(Nests [i]) for all i from 1 to n_nests
5. Set the best solution as the initial best
   - Best_nest = Nests [best index]
   - Best_fitness = Fitness [best index]
6. For each iteration from ~~to~~ 1 to n_iter :
   a. Generate new solutions using levy flights :
      - for each nest i in nests :

Step = levy_flight(beta, dim)
New_nest[i] = Nests[i] + step * (Nests[i] - Best_nests)
New_nest[i] = Clip new nest within search space bounds
Evaluate fitness of the new nest:
New_fitness = func(New_nest[i])
If New_fitness < fitness[i], replace Nests[i] with New_nest[i]

b. Abandon the worst nests with probability pa:
For each nest i in nests:
if rand() < pa:
- Replace Nests[i] with new random solution
Evaluate fitness: Fitness[i] = func(Nests[i])

c. Update the best solution
Best_fitness = min(Fitness)
Best_nest = Nests[best index]

7. Output the best solution & its fitness
Return Best_nest, Best_fitness.

Output :-
Best_nest = [1.8805 0.10537]
Best_fitness = 1.3661853769607717

o/p seen (G.H)

21.11.24

Date
Page

Grey Wolf Optimizer

Pse The (Grey Wolf Optimizer) algorithm is a swarm intelligence algorithm inspired by the social hierarchy and hunting behaviour of grey wolves. It mimics the leadership structure of alpha, beta, delta and omega wolves and their collaborative hunting strategies.

Parameters :-

① N (Population Size) :
Number of wolves (solutions)
Range : 10-50

② Tmax :-
Maximum iterations (stopping condition)
Range : 100 - 1000

③ a
Linearly decreases from 2 to 0 to control A
Range : 2 → 0

④ A
Exploration - exploitation balance
Range : [-2, 2]

⑤ C
Stochastic influence
Range : [0, 2]

⑥ Xmin, Xmax
Defines the solution space
Range : Problem dependent

⑦ Objective function
Evaluates the fitness of wolves
Problem dependent

Date / /
Page

Applications

1. Feature Selection in Machine learning - Xup
2. Parameter Tuning in Neural Network
3. Power Systems Optimization
4. Scheduling and Logistics
5. Portfolio Optimization
6. Design Optimization

Algorithm

1. Initialize the population of wolves (position) randomly in the solution space.
2. Evaluate the fitness of each wolf using the objective function
3. Identify the top three wolves
   - Alpha (best solution)
   - Beta (second-best solution)
   - Delta (third-best selection)
4. Repeat until the max number of iterations is reached or convergence criteria are met:

a. For wolf in the population:

i. Update the distance to alpha, beta and delta:

$$D\_alpha = |C1 * X\_alpha - X\_wolf|$$
$$D\_beta = |C2 * X\_beta - X\_wolf|$$
$$D\_delta = |C3 * X\_delta - X\_wolf|$$

ii. Calculate new positions relative to the leaders:

$$X1 = X\_alpha - A1 * D\_alpha$$
$$X2 = X\_beta - A2 * D\_beta$$
$$X3 = X\_delta - A3 * D\_delta$$

iii. Update the wolf's position as the

average of X1, X2, X3 :
X_new = (X1 + X2 + X3) / 3

iv. Ensure the new position is within valid bounds.

b. Reevaluate the fitness of each wolf in the updated population.

c. Update alpha, beta, and delta wolves if better solutions are found

Return the position and fitness of the alpha wolf as the best solution.

Output

Choose dataset : (1) Iris [default] ,
(2) Custom : 1

Enter the number of wolves in the population : 10

Enter the maximum number of iterations: 20

Optimization Results :
Selected Features : [0 2 3]
Best Accuracy : 1.0000

1.a.b.B
21/4/24

28.11.24

Date / /
Page

Parallel Cellular Algorithm

Inspired by the functioning of biological cells that operate in a highly parallel and distributed manner.

Parameters

→ Grid Size (N×M)
Represents no. of cells in the grid

→ Neighbourhood Size
No. of neighbours each cells interact with

→ Iterations (T)
No. of iterations to run the algorithm

→ Fitness Function
Used to evaluate how good the state of cell is

→ Update Rule
Defines how cell updates its state based on its neighbours

→ Population Size (N)
No. of potential solutions (cells) in the grid

Applications

1. Optimisation
2. Image Segmentation
3. Clustering and Classification
4. Combinatorial Problems
5. Machine Learning
6. Path finding and Robotics

Algorithm

Initialize parameters
- Define the grid size (number of cells)
- Define the no. of iterations (T)
- Define the neighbourhood structure
- Define the fitness function
- Initialize population of cells randomly

For t = 1 to T do:
Evaluate the fitness of each cell in the grid:
- For each cell i, j in the grid:
  - Calculate the fitness value of the cell based on fitness function

Parallel update the state of each cell.
- For each cell i, j in the grid:
  - Get the fitness value of its neighbors
  - Update the state of the cell based on the best neighbouring solution or predefined rules

Track the best solution found:
- If a better solution (fitness) is found, store the current state of the grid as the best solution

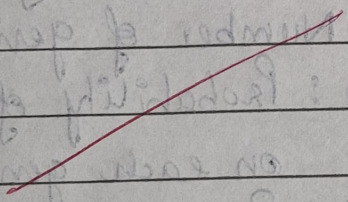
End for

Return the best solution found during the iterations.

Date __/__/__
Page _____

Output:

Enter the no. of rows : 4
Enter the no. of columns : 2
Enter the no. of iterations for segmentation: 2
Enter the neighbourhood size : 3



28-11-24

Date
Page

Gene Expression Algorithms

Genetic Expression Algorithm (GEA) are inspired by the biological process of gene expression in living organisms. This process involves the translation of genetic information encoded in DNA into functional proteins. The algorithm evolves these solutions through selection, crossover, mutation and gene expression to find optimal or near optimal solutions.

Parameters

→ Population Size : Number of individuals
→ Gene Size : Number of genes in each individual
→ Mutation Rate : Probability of mutation occurring on each gene
→ Crossover Rate : Probability of crossover occuring bhw two individuals
→ Generations : No. of iterations
→ Fitness function : Function used to evaluate the quality of a solution

Applications of

→ Engineering Optimization
→ Knapsack Problem (0/1)
→ Data Mining
→ Machine learning
→ Control Systems
→ Game theory

Date / /
Page

Algorithm

1. Define the problem
   Objective function :
   $f(x) \rightarrow$ maximize/minimize
   x is the vector of decision variables

   Fitness function :
   Fitness $(x) \rightarrow$ evaluate $(f(x))$
   Determines how good a solution is

2. Initialize Parameters
   N ⇒ Population Size
   G ⇒ Gene Length
   M ⇒ Mutation Rate
   C ⇒ Crossover Rate
   T ⇒ No. of generations

3. Initialize Population
   for i = 1 to N:
   Individual = Random solution (G)
   population . add (individual)

4. Evaluate Fitness
   For each individual in population :
   fitness = evaluate Fitness (Individual)
   fitnes_values . add (fitness)

5. Selection
6. Crossover
7. Mutation
8. Translate genetic representations into actual solns
9. Iterate
10. Output the Best solution

Output:-

Enter the number of items : 5
Enter the weight and value of each item
10 20
30 40
50 60
90 80
90 100
Enter knapsack capacity:
90
Enter the population size:
100
Enter mutation rate
0.2
Enter crossover rate
0.9
Enter no. of generations
10

Best solution (items selected): [1, 1, 1, 0, 0]
Best fitness (total value) : 120