SHAMANTH- K. MURTHY

INDEX

NAME: ________ STD.: ________ SEC.: ________ ROLL NO.: ________ SUB.: ________

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	24/10/24	Genetic Algorithm		10
2	07/11/24	Particle Swarm		7
3	14/11/24	Ant Colony		7
4	21/11/24	Cuckoo Search		8
5	28/11/24	Grey Wolf Optimizer		9
6	18/12/24	Parallel Cellular		10
7	18/12/24	Optimization via gene Expression		

03/10/24

1) Genetic Algorithm
→ It has a population where each individual represents a possible solution of a given problem.
→ It is based on Darwin's evolution ideas where each individual is considered as a chromosome and each bit as a character & random mutations or crossovers are done to find the best fit solution.

Applications: complex optimization problems like travelling salesman, neural network etc.

2) PSO
→ This algorithm is based on cooperative behaviour of animals like birds, ants or fish, by imitating their collective behaviour.
→ It consists of particles representing a solution to a problem. These explore the solution space by adjusting based on experience & success of others.

Applications: Neural network training, distribution networks.

3) Ant Colony Optimization.
→ It is based on ability of ants to find the shortest path between food source & nest & adapting to changes in environment.
→ The three main ideas used are preference for paths with high pheromone level, higher rate of growth of amount of pheromone on shorter path, trail mediated solution among ants.
→ All ants construct a solution & then pheromone trails are updated based on the quality of solutions found.

24/10/24

LAB - 1
Genetic Algorithm for Optimization Problems

Pseudo Code:

→ Set the parameters:
  population_size = 100
  mutation rate = 0.1
  Max value = 10   min value = -10

→ fitness function() $f(x) = x^2$

→ initialize_population()
  np.random.uniform(min, max, range)

→ select_parents (population)
  fitness_values = fitness(population)
  selected_indices =
    np.random.choice(n, size = 2, p = prob)

→ crossover (p1, p2)
  if np.rand() < crossover_rate:
    (p1 + p2)/2
  else p1

→ mutate (child, max, min)
  if np.rand() < mutation_rate
    np.random.uniform(max, min)
  else child

→ genetic_algorithm()
    initialize_population(size, max, min)
    for i in range(size)
        p1, p2 = select_parents(population)
        child = crossover(p1, p2)
        child = mutate(child, max, min)
        new_population.append(child)

    best_solution = population[np.argmax(fitness(populat))]
    return best_solution, fitness(best_solution)

Best x : -9.9130010
Maximum f(x) : 98.2735)

Snehal R
24/11/24

07/11/24

LAB - 2

Particle Swarm Optimization for function Optimization

Algorithm:

1) Initialize Parameters
   - P → no of particles c_1 → cognetive coeff
   - D → no of Dimensions c_2 → social coeff
   - T → no of iterations w → inertia weight

2) Initialise Particles
   - → For each particle randomly initialize :
     - ↳ x_i - particle's position, v_i - velocity
     - ↳ P_{best} - best position & evaluate fitness.

3) Determine Global best
   ~~Set~~ g_{best} → particle with best fitness

   Set g_{best} to the position of the particle with lowest fitness

4) In Loop, from 1 to T :
   - → Update velocity : Generate r_1, r_2

     $c = c_1 * r_1 * (P_{best} - x_i)$

     $S = c_2 * r_2 * (g_{best} - x_i)$

     $V_i = w * V_i + c + S$
   - → update position : $x_i = x_i + V_i$
   - → calculate fitness of new position x_i
   - → update P_{best} if new fitness is greater &
     - ↳ & also gbest

5) Print Solution : After all iterations print g_{best} & ~~optimal~~ fitness

14/11/24

LAB - 3

Ant Colony Optimization for Travelling Salesman Problem

Algorithm:

1) Represent the cities as nodes in a graph and construct a distance matrix

2) Initialize parameters:
   m → no of ants
   α → importance of pheromone
   β → heuristic information
   ρ → pheromone evaporation rate
   Q → " deposit constant

3) For each ant:
   → Randomly select start city
   → Build a complete tour by iteratively selecting next city
      ↳ For each unvisited city j calculate probability of moving from i to j
      ↳ Use the probability to select next city
   → Add selected city to tour & mark as visited.

4) Calculate total length of each tour & keep track of best tour found so far.

5) Update pheromones:
   pheromones *= $(1 - \rho)$
   pheromone_increase = Q / tour_length

6) Return the shortest tour & its length as the best solution

21/11/24

Date
Page

LAB - 4

Cuckoo Search

Algorithm:

1) Define the objective function $f(x)$ to optimize & bounds of search space x_{min}, x_{max}.

$$f(x) = x^2$$

2) Initialize parameters:
$n \rightarrow$ no of nests $\quad P_a \rightarrow$ discovery probability
max no of iterations

3) Generate an initial population of nests with random positions within search space.

4) Evaluate fitness ~~within~~ of each nest using the objective function.

5) Generate new solutions:
→ For each nest, generate new sol^n

$$x_{new} = x_{curr} + \text{step size} \times \text{Levy flight}$$

→ Levy flight is a random walk with step sizes from a Lévy distribution.

6) Abandon worst nests:
→ Using probability P_a, abandon a fraction of worst nests & replace with new ones.

7) Repeat steps 4 - 6 for specified no of iterations

8) Return the nest with best fitness & corresponding position

08/11/24

LAB - 5

Grey Wolf Optimizer

Algorithm:

1) Define the objective function $f(x)$ & specify bounds of search space x_{min}, x_{max}

2) Initialize parameters:
   $n \rightarrow$ no of wolves in a pack
   max iterations

3) Generate initial population of wolves with random positions

4) Evaluate fitness:
   → calculate fitness of each wolf using objective function
   → identify three best wolves : α, β, γ

5) For each wolf:
$$\vec{D_\alpha} = |\vec{C_1} \cdot \vec{X_\alpha} - \vec{X}|$$
$$\vec{X_1} = \vec{X_\alpha} - \vec{A_1} \cdot \vec{D_\alpha}$$
   similarly for β & γ
$$\vec{X}(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3}$$
   $\vec{A}$ & $\vec{C}$ are coefficient vectors

6) Return the position of α wolf & its fitness

classmate

Date
Page

18/12/24

LAB-6

Parallel Cellular Algorithms & Programs

Algorithm:

1) Objective function : $f(x) = \Sigma x_i^2$

2) Initialize parameters:
no of cells, grid size, dimensions, bounds
no of iterations
np.random.uniform(bounds[0], bounds[1])

3) Initialize population with initial positions of all cells

4) Evaluate Fitness for each cell & put it into a 1D array

5) Identify the neighbours using Moore neighbourhood structure

6) Update each cell's state by copying the position of its best neighbour

7) Main Algorithm:
Print the best solution & its fitness.

5)
```
for dx in [-1, 0, 1]:
    for dy in [-1, 0, 1]:
        if dx == 0 and dy == 0:
            continue.
        nx, ny = (x + dx) % grid_size.
        neighbours.append(nx * grid_size + ny)
```

18/12/24

LAB - 7

Optimization via Gene Expression Algorithm

Algorithm :

1) Define the objective function $f(x) = \sum x_i^2$

2) Initialize parameters:
no of genes, bounds, mutation rate
crossover rate, no of generations

3) Generate a population P with G genes each
$x_i^j \rightarrow$ j^{th} gene of i^{th} individual

4) Evaluate fitness using the objective function $f(x)$

5) Filter the population with lower fitness values

6) Choose two parents at a time & perform crossover

7) Introduce variability in offspring by randomly altering genes.

8) Combine the offsprings to the new population.

9) Output the genetic sequence with the best fitness