Date 18/10/24
Page 11

LAB 2

Genetic Algorithm

$F(x) = x^2$

Step 1 : Initialization

1 Population size: Choose a population size, say 6 individuals

2 Binary Representation: Each individual will be a 5-bit binary string

3 Randomly Initialize Population: Generate 6 random binary strings.

Step 2 : Fitness Evaluation

convert binary to decimal

10101 = 441

00111 = 49

01100 = 144

11001 = 625

10010 = 324

00010 = 4

Fitness Values : [441, 49, 144, 625, 324, 4

Step 3 : selection

Use a method like tournament selection, we'll use tournament selection

selected individuals : ["11001", "10101", "10010"]

Step 4 : crossover

Pairs selected individuals to create offspring (Single-point crossover)

Pair 1: 11001 & 10101 → crossover at point 2
→ offspring : 11001, 10101.

Pair 2: 10010 & a random parent → 10010 & 00011
→ crossover → offspring : 10011

offspring after crossover

New offspring : ["11001", "10101", "10011", "00110"]

Step 5 : Mutation

Introduce mutations to some of the offspring, flip a random bit with a small probability

- If we mutate 10011 → 10001

Date __/__/__
Page 13

Step 6 : Replacement

Replace the old population with the new offspring

New population ["11001", "10101", "10001", "00110"]

step 7: Iteration

Repeat steps 2-6 for a predetermined number of generations or until convergence

Code

```
import random
import numpy as np

def fitness_function(x):
    return x**2

population_size = 10
mutation_rate = 0.01
crossover_rate = 0.8
num_generations = 100
gene_length = 16

def create_population(size, gene_length):
    return [np.random.randint(0, 2, gene_length)
            .tolist() for _ in range(size)]
```

```
def binary_to_decimal(binary):
    binary_str = ''.join(str(bit) for bit in binary)
    return int(binary_str, 2) / (2**gene_length - 1) * 20 - 10

def evaluate_population(population):
    return [fitness_function(binary_to_decimal(individual)) for individual in population]

def select(population, fitness_scores):
    total_fitness = sum(fitness_scores)
    selection_probs = [fitness/total_fitness for fitness in fitness_scores]
    return population[np.random.choice(range(len(population)), p=selection_probs)]

def crossover(parent1, parent2):
    if random() < crossover_rate:
        child1 = parent1[:crossover_point] + parent2
        child2 = parent2[:crossover_point] + parent1
        return [child1, child2]

def mutate(individual):
    for i in range(gene_length):
        if random.random() < mutation_rate:
            return individual
```

```
def genetic_algorithm():
    population = create_population(population_size,
                                   gene_length)
    for generation in range(num_generations):
        fitness_scores = evaluate_population(population)
        best_fitness = max(fitness_scores)
        print(f"Generation {generation}: Best Fitness
              = {best_fitness:.4f}")

        new_population = []
        population = new_population[:population
                                    _size]

        best_fitness = max(fitness_scores)
        best_individual = population[fitness_scores.
                                     index(best_fitness)]
        best_solution = binary_to_decimal(best_individu
                                          al)

        print(f"\nBest solution found: x = {best_solution
              :.4f}, f(x) = {fitness_function
              (best_solution):.4f}")

genetic_algorithm()
```

Output

```
Gen 0 : BF = 78.2611
Gen 1 : BF = 80.3503
Gen 2 : BF = 90.8360
Gen 3 : BF = 98.8304
Gen 4 : BF = 81.0508
```

Gen 5 : BF = ~~78.2611~~ 82.1123
Gen 6 : BF = 82.1123
Gen 7 : BF = 82.1123
Gen 8 : BF = 83.1258
Gen 9 : BF = 88.1273
Gen 10 : BF = 88.1371

Best solution found: x = 8.8856, f(x) = 78.9544

Date / /
Page 17

Lab 3

Particle swarm.

1 → Define Objective Function - Specify the function $f(x)$ to minimize

2 → Initialize Parameters -
Set num_particles, num_iterations, inertia_weight, cognitive_coefficient, & social_coefficient.

3 → Initialize Particles -
- Randomly set positions & velocities for num_particles.
- Initialize pBest & pBest_fitness.
- Determine gBest & its fitness

4 → Set Best known solutions -
For each particle keep track of its best known position & corresponding fitness score.

5 → Select particles to swap
- Swap pos. of selected particles
- Update
- calculate the fitness of new positions of both particles.

4 → Update Best known
If the new pos of particle has better fitness score than previous

Date __/__/__
Page 18

best

- Update the best known position.

Pseudocode

step 1: def f(x)
return x**2

step 2: Initilize parameters
N = 30, W = 0.5, C1 = 1.5, C2 = 1.5

step 3: for each particle initialise their positions & velocity randomly with the range [10, 10], [-1, 1]

step 4: In this step we evaluate the fitness by sending values to f

step 5: updating values if current value < best value update the best of its own particle.
If curr value < global best update global best values that is velocity based on entire swarm.

step 6: update velocity

step 7: display the best value found.

Date __/__/__
Page 19

→ optimizes shapes & materials
(↳ by adjusting design parameters)

to achieve
- minimizing weight
- maximizing strength
- reduced costs.

shape → is represented using variables.
(length, width, thickness)

Date __/__/__
Page 13

Lab 4

Ant colony optimization.

Step 1: Initialization: set the nodes or elements according to the problem like cities in tsp.

no. of ants, n

no. of iterations m

pheremon evaphoration rate P
pheremon influence factor, α
heuristic influence factor β.

Step 2: Initializing the state

Initilize pheremon levels for all edges between the nodes

define a visibility heuristic if applicable

Iteration starts from 1 and goes to m

Ant solo solution construction

Step 3: execution

for each ant start at a random node.

travell between all nodes & find out the probability of the best path.

using the evaporation rate ρ, pheromon influence factor, α, heuristic influence factor β we choose the max and the choose the best path. Incrementing its values respectively.

Higher the pheromon the greater is the chance of choosing that path and the other ant(solutions) following that path.

ACO Robotics ✓

→ to find the shortest, safest, and most efficient path from start point to a distination

→ robot - ant → deposits pheromones (marking better paths)
→ robot uses a probabilistic rule to decide its next step, balancing pheromone strength & unexplored paths.
→ over multiple iterations, gives optimal path, adjusting to obstacles.

Date __/__/__
Page 20

Lab 5

Cuckoo Search.

Step 1 Initialization :- . Set the no. of nests N and
. Set the max iterations value.

Step 2 Solution :- Generate a solution.

Step 3 calculation :- calculate the fitness of that particular solution

Step 4 Iterations :- Keep repeating this procedure and find different solution

Step 5 Compare :- Keep comparing the solution with a randomly choosen existing nest.

Step 6 Choosing the best :- If it is better than the previous make that the optimal solution.

→ In this algorithm the find the best nest just like the cuckoo. i.e to find the best solution.

→ the best nest should be able to accept the cuckoo.

→ nests is the solution space

→ egg is the solution.

→ optimagation ~~function~~

→ Robotics

Write Pseudocode / Implant in Data mining

Implementation on Data Mining.

process of discovering patterns, trends, getting useful info from large datasets

extracting.

→ for feature selection, clustering,

~~text~~ clustering

- to group data points into clusters such that points in the same cluster are similar & points in different clusters are dissimilar.
- each nest → represented → clustering configuration → centroids
- algo adjusts centroids over the iterations → optimizing clusters
- follows random walk

Pseudocode

Initialize population of nests (solutions)
Define discovery rate (p-a), no. of iterations
& objective functions → probability of nests to abandon and replace with new solution

FOR each iteration:
    FOR each nest:
        Generate a new solution by Levy flight → generate a new solution & random walk
        Evaluate the fitness of the new solution
        If new solution is better:
            Replace the innocent nest → compares solutions to keep with the new solution the best one.
    Abandon a fraction (p-a) of nests & replace with new random solutions.
    Update the best solution found so far

Return best solution

15/11

[illegible]

[illegible]

[illegible]

→ [illegible]

→ [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

→ [illegible]

(i) [illegible]

(ii) [illegible]

(iii) [illegible]

(iv) [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

→ [illegible]

→ [illegible]

→ [illegible]

[illegible]

[illegible]

[illegible]

~~Lab 6~~

Grey wolf optimization

→ Robotics - it helps the robots to find the best path using travelling Salesman

- the robot goes from A to B for the 1st time - (α)
- 2nd time (β)
- 3rd time (δ)
- 4 time ... nth time the values keeps updating as it finds better paths.

Implementation in data Mining + Disadvantages

→ extracting data

Implementation in Data Mining

→ process of discovering patterns, trends, getting useful information from large dataset.

→ feature selection, clustering & classification optimization.

clustering → done by using similar measures
↳ eg Manhattan distance

- to group data points into clusters such that points in the same cluster are more similar & points in different clusters are dissimilar

- ~~each input~~ → (represented) clustering configuration
↳ centroid
↳ central point of a cluster

- this algorithm helps to adjust centroids over the iterations
↳ to get a optimizing cluster

each wolf → potential clustering solution
centroid → wolf's position

Disadvantages of Grey wolf algorithm

1] Premature convergence : can get stuck in local optima for complex problems

2] Lack of Diversity : Wolves often converge too quickly, reducing exploration

3] Scalability issues : Performance declines with large-scale dimensional problems

4] No guarantee of global optimum

Date __/__/__
Page 26

Improvements

→ hybrid approaches

combine GWO with GA, PSO to improve exploration & avoid local optima.

→ Increased population Diversity
↳ introduce mutation-like strategies

10/10

Date 29/11/24
Page 27

Lab 7

Parallel cellular Algorithm.

Output

Iteration 1 - Fitness : 5377.712222222223

[[93 80 63 81 70 79 121 89 109 45]
[100 85 91 86 117 129 131 99 74 81]
[95 82 95 119 135 135 116 105 84 79]]

Iteration 2 - Fitness : 473.0400000000001

[[88 85 85 89 100 119 113 105 80 78]
[87 88 91 103 116 119 113 96 88 81]
[85 89 98 108 116 114 109 98 86 85]]

Iteration 10 - Fitness : 1.5288888888888892

[[92 93 93 94 93 93 92 91 91]
[92 92 93 93 93 92 91 91 91]
[92 92 92 93 93 92 92 91 91]]

Best solution Fitness : 1.5288888888888892

Date ___/___/___
Page ___

Gene Expression Algorithm.

Generation	1 : Best Fitness = -4.093999946842503, Best solution = [-4.28309512 0.43487975]
-11-	2 : -11- = -4.093999946842503, -11- [-4.28309612 0.43487975]
-11-	3 : -11- = -4.262850914664476 -11- [-4.28309512 -0.14228212]
-11-	4 : -11- = -4.277432567035412 -11- [-4.28309512 0.07524998]
-11-	9 : -11- = -4.386638398263757 -11- [-4.39230095 0.07524998]
-11-	10 : -11- = -4.386638398263757 -11- [-4.39230095 0.07524997]