LAB 07 **(16-12-24)**

Optimization via Gene Expression Algorithms:

Gene Expression Algorithms (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. In GEA, solutions to optimization problems are encoded in a manner similar to genetic sequences. The algorithm evolves these solutions through selection, crossover, mutation, and gene expression to find optimal or near-optimal solutions. GEA is effective for solving complex optimization problems in various domains, including engineering, data analysis, and machine learning.

Implementation Steps:

1. Define the Problem: Create a mathematical function to optimize.
2. Initialize Parameters: Set the population size, number of genes, mutation rate, crossover rate, and number of generations.
3. Initialize Population: Generate an initial population of random genetic sequences.
4. Evaluate Fitness: Evaluate the fitness of each genetic sequence based on the optimization function.
5. Selection: Select genetic sequences based on their fitness for reproduction.
6. Crossover: Perform crossover between selected sequences to produce offspring.
7. Mutation: Apply mutation to the offspring to introduce variability.
8. Gene Expression: Translate genetic sequences into functional solutions.
9. Iterate: Repeat the selection, crossover, mutation, and gene expression processes for a fixed number of generations or until convergence criteria are met.
10. Output the Best Solution: Track and output the best solution found during the iterations.

ALGORITHM/LOGIC-

OPTIMIZATION VIA GENE EXPRESSIONS

PURPOSE:

The gene expression Algorithm (GEA) is inspired by biological processes where genetic information is expressed to produce functional proteins. Primary goal is to solve complex optimization problems by simulating genetic evolution.

APPLICATION:

1. Engineering: Optimizing design parameters for systems
2. Data Analysis: Feature selection and predictive modelling
3. Machine Learning: Training and feature optimization in models
4. Mathematical Optimization: Finding solutions to non-linear and combinatorial optimization problems.

ALGORITHM:

~~Input: f(x), M, N, C, G~~
~~Output~~

1. Initialize population P with N random chromosomes
2. Evaluate fitness of each chromosome in P using f(x)
3. For generation g = 1 to G:
   a. Select parent from P based on fitness
   b. Perform crossover on selected parents with probability C to produce offspring.

Date 16/12/24
Page 26

c. Apply mutation to offspring with probability m
d. evaluate fitness of the new population
e. Update P with offspring population
f. Track the best chromosome found so far.
4. End loop
5. Return the best chromosome x^* and its corresponding solution.

IMPLEMENTATION:
1. Define the problem: formulate mathematical function
2. Initialize the parameters: set variables like population size etc.
3. Initialize population
4. Evaluate fitness
5. Selection: select fittest individuals for reproduction
6. Crossover: combine genetic material of parents to create offspring
7. Mutation: introduce variability in offspring
8. Gene Expression: Translate genetic sequence into functional solution
9. Iterate: Repeate steps till convergence is reached
10. Output best solution

Date 16/12/24
Page 27

OUTPUT:

Generation 1: Best fitness = 0.99799,
Best solution = -0.04489

Generation 2: Best fitness = 0.99799,
Best solution = -0.04489

.
.
.

Generation 50: Best fitness = 1.00000
Best solution = 0.00132

Optimal solution:
Best solution: 0.0013201
Best fitness: 0.999998

16/12/24

INPUT-

```
import random
import numpy as np

# Objective Function to Optimize (Example: Minimize x^2)
def objective_function(x):
    return x ** 2

# Initialize Population
def initialize_population(pop_size, lower_bound, upper_bound):
    return [random.uniform(lower_bound, upper_bound) for _ in range(pop_size)]
```

```
# Evaluate Fitness
def evaluate_fitness(population):
    return [1 / (1 + objective_function(ind)) for ind in population]  # Fitness is inverse of objective

# Select Parents (Roulette Wheel Selection)
def select_parents(population, fitness):
    total_fitness = sum(fitness)
    probabilities = [f / total_fitness for f in fitness]
    parents = random.choices(population, probabilities, k=len(population))
    return parents

# Perform Crossover (Single-Point)
def crossover(parents, crossover_rate):
    offspring = []
    for i in range(0, len(parents), 2):
        if i + 1 < len(parents) and random.random() < crossover_rate:
            # Single-point crossover
            alpha = random.random()  # Blending factor
            child1 = alpha * parents[i] + (1 - alpha) * parents[i + 1]
            child2 = alpha * parents[i + 1] + (1 - alpha) * parents[i]
            offspring.extend([child1, child2])
        else:
            offspring.extend([parents[i], parents[i + 1] if i + 1 < len(parents) else parents[i]])
    return offspring

# Perform Mutation

def mutate(offspring, mutation_rate, lower_bound, upper_bound):
    for i in range(len(offspring)):
        if random.random() < mutation_rate:
            offspring[i] += random.uniform(-1, 1)  # Small random change
            offspring[i] = max(lower_bound, min(offspring[i], upper_bound))  # Keep within bounds
    return offspring

# Main Gene Expression Algorithm
def gene_expression_algorithm(
    pop_size, generations, crossover_rate, mutation_rate, lower_bound, upper_bound
):
    # Step 1: Initialize Population
```

```
    population = initialize_population(pop_size, lower_bound, upper_bound)
    best_solution = None
    best_fitness = float('-inf')

    for gen in range(generations):
        # Step 2: Evaluate Fitness
        fitness = evaluate_fitness(population)

        # Step 3: Track the best individual
        max_fitness = max(fitness)
        if max_fitness > best_fitness:
            best_fitness = max_fitness
            best_solution = population[fitness.index(max_fitness)]

        # Step 4: Select Parents
        parents = select_parents(population, fitness)

        # Step 5: Crossover
        offspring = crossover(parents, crossover_rate)

        # Step 6: Mutation
        population = mutate(offspring, mutation_rate, lower_bound, upper_bound)

        # Log Progress
        print(f"Generation {gen+1}: Best Fitness = {best_fitness:.5f}, Best Solution =
{best_solution:.5f}")
    return best_solution, best_fitness

# Parameters
POPULATION_SIZE = 20
GENERATIONS = 50
CROSSOVER_RATE = 0.7
MUTATION_RATE = 0.1
LOWER_BOUND = -10
UPPER_BOUND = 10
# Run the Algorithm
best_solution, best_fitness = gene_expression_algorithm(
    POPULATION_SIZE, GENERATIONS, CROSSOVER_RATE, MUTATION_RATE,
LOWER_BOUND, UPPER_BOUND
)
```

```
print("\nOptimal Solution:")
print(f"Best Solution: {best_solution}")
print(f"Best Fitness: {best_fitness}")
```

OUTPUT:

```
Generation 1: Best Fitness = 0.95741, Best Solution = -0.21091
Generation 2: Best Fitness = 0.95741, Best Solution = -0.21091
Generation 3: Best Fitness = 0.95741, Best Solution = -0.21091
Generation 4: Best Fitness = 0.95741, Best Solution = -0.21091
Generation 5: Best Fitness = 0.99006, Best Solution = 0.10020
Generation 6: Best Fitness = 0.99006, Best Solution = 0.10020
Generation 7: Best Fitness = 0.99999, Best Solution = 0.00270
Generation 8: Best Fitness = 0.99999, Best Solution = 0.00270
Generation 9: Best Fitness = 0.99999, Best Solution = 0.00270
Generation 10: Best Fitness = 0.99999, Best Solution = 0.00270
Generation 11: Best Fitness = 0.99999, Best Solution = 0.00270
Generation 12: Best Fitness = 0.99999, Best Solution = 0.00270
Generation 13: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 14: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 15: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 16: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 17: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 18: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 19: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 20: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 21: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 22: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 23: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 24: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 25: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 26: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 27: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 28: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 29: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 30: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 31: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 32: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 33: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 34: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 35: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 36: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 37: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 38: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 39: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 40: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 41: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 42: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 43: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 44: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 45: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 46: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 47: Best Fitness = 1.00000, Best Solution = -0.00043
```


```
Generation 45: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 46: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 47: Best Fitness = 1.00000, Best Solution = -0.00043
```


```
Generation 48: Best Fitness = 1.00000, Best Solution = -0.00043
Generation 49: Best Fitness = 1.00000, Best Solution = -0.00001
Generation 50: Best Fitness = 1.00000, Best Solution = -0.00001

Optimal Solution:
Best Solution: -8.15520154000618e-06
Best Fitness: 0.9999999999334928
```