Lab Experiment – 7

GEA [Gene Expression Algorithm]

Lab - 7

GEA.

Gene expression Algorithm: 6 Main Phases.

- Initialization
- Fitness Assignment.
- Selection
- Cross over
- Gene expression
- Termination

Steps: Fitness $(x) = x^2$

1) Select encoding Technique: 0 - 31

Use chromosome of fixed length with terminals (variables, constants and functions (+, −, *, /)).

2) Initial Population:

S.NO	Initial chromosome	Phenotype	Value	Fitness	P
1.	* x x	x^2	12	144	0.1247
2.	+ x x	2x	25	625	0.541
3.	x	x	5	25	0.0216
4.	− x	x−2	19	361	0.8125

$\Sigma P(x) = 1155$

Avg = 288.75

Actual cost	Exp. cost
1	0.5
2	2.1
0	0.08
1	1.25

3) Selection of Mating Pool.

S.No	Selected chromosome	crossover point	offspring	Phenotype
1	*xx	2	*x+	x*(x+...)
2	+xx	1	+xx	2x
3	+xx	3	+x-	x+(x...)
4	-x2	1	+x2	x+2

x values	Fitness
13	169
24	576
27	729
17	289

4) Crossover:

Perform crossover randomly chosen gene. positions (not raw bits)

Max fitness after crossover = 729.

5) Mutation:

S.No	offspring before mutation	Mutation Applied	Offspring after mutation	Phenotype
1	*x+	+ → -	+x-	x*(x...)
2	+xx	None	+xx	2x
3	+x-	- → *	+x*	x+x*x
4	+x2	None	+x2	x+2

x value	Fitness f(x)
29	841
24	576
27	729
20	400

6) Gene expression and evaluation:

Decode each genetype → phenotype.

$\Sigma f(x) = 841 + 576 + 729 + 400 = 2546$

Avg = 636.5

Max = 841

7) Iterate until convergence:

Repeat step 3-6 until fitness improvement is negligible or generation limit has reached.

Pseudocode / Algorithm:

* Start
* Define Fitness function.
* Define parameters
* Select mating pool,
* Mutation after mating.
* Gene expression and evaluation.
* Iterate.
* Output best value.

Output:

Genes: [29.53, 29.82, 29.84, 28.57, 15.09, 21.83, 23.43, 30.81, 22.51, 26.22].

x : 26.37

$f(x)$: 695.45

Output:

```
Enter 4 chromosomes (each of 5 bits, e.g., '10101'):
Chromosome 1: 01101
Chromosome 2: 11000
Chromosome 3: 11011
Chromosome 4: 10001
Generation 1: Best Fitness = 729, Best x = 27
Generation 2: Best Fitness = 729, Best x = 27
Generation 3: Best Fitness = 729, Best x = 27
Generation 4: Best Fitness = 729, Best x = 27
Generation 5: Best Fitness = 729, Best x = 27
Generation 6: Best Fitness = 729, Best x = 27
Generation 7: Best Fitness = 729, Best x = 27
Generation 8: Best Fitness = 729, Best x = 27
Generation 9: Best Fitness = 729, Best x = 27
Generation 10: Best Fitness = 729, Best x = 27
Generation 11: Best Fitness = 729, Best x = 27
Generation 12: Best Fitness = 729, Best x = 27
Generation 13: Best Fitness = 729, Best x = 27
Generation 14: Best Fitness = 729, Best x = 27
Generation 15: Best Fitness = 729, Best x = 27
Generation 16: Best Fitness = 729, Best x = 27
Generation 17: Best Fitness = 729, Best x = 27
Generation 18: Best Fitness = 729, Best x = 27
Generation 19: Best Fitness = 729, Best x = 27
Generation 20: Best Fitness = 729, Best x = 27
...
Best solution found:
Chromosome: 11011
x = 27
f(x) = 729
```

Code:

```
import random

# Define the function to maximize

def fitness_function(x):

    return x ** 2

# Convert binary string to integer

def decode(chromosome):

    return int(chromosome, 2)

# Evaluate fitness for the entire population

def evaluate_population(population):

    return [fitness_function(decode(individual)) for individual in population]


# Selection: Roulette Wheel
```

```python
def select(population, fitnesses):
    total_fitness = sum(fitnesses)
    if total_fitness == 0:
        return random.choice(population)
    pick = random.uniform(0, total_fitness)
    current = 0
    for individual, fitness in zip(population, fitnesses):
        current += fitness
        if current > pick:
            return individual

# Crossover: Single-point crossover
def crossover(parent1, parent2):
    if random.random() < CROSSOVER_RATE:
        point = random.randint(1, CHROMOSOME_LENGTH - 1)
        return (parent1[:point] + parent2[point:], parent2[:point] + parent1[point:])
    return parent1, parent2

# Mutation: Flip random bit
def mutate(chromosome):
    new_chromosome = ''
    for bit in chromosome:
        if random.random() < MUTATION_RATE:
            new_chromosome += '0' if bit == '1' else '1'
        else:
            new_chromosome += bit
    return new_chromosome
```

```
# Get user input for initial population
def get_initial_population(size, length):
    population = []
    print(f"Enter {size} chromosomes (each of {length} bits, e.g., '10101'):")
    while len(population) < size:
        chrom = input(f"Chromosome {len(population)+1}: ").strip()
        if len(chrom) == length and all(bit in '01' for bit in chrom):
            population.append(chrom)
        else:
            print(f"Invalid input. Please enter a {length}-bit binary string.")
    return population

# Run the Genetic Algorithm
def genetic_algorithm():
    population = get_initial_population(POPULATION_SIZE, CHROMOSOME_LENGTH)
    best_solution = None
    best_fitness = float('-inf')
    for generation in range(GENERATIONS):
        fitnesses = evaluate_population(population)
        # Update best solution
        for i, individual in enumerate(population):
            if fitnesses[i] > best_fitness:
                best_fitness = fitnesses[i]
                best_solution = individual
        print(f"Generation {generation + 1}: Best Fitness = {best_fitness}, Best x =
{decode(best_solution)}")
        new_population = []
        while len(new_population) < POPULATION_SIZE:
```

```
            parent1 = select(population, fitnesses)
            parent2 = select(population, fitnesses)
            offspring1, offspring2 = crossover(parent1, parent2)
            offspring1 = mutate(offspring1)
            offspring2 = mutate(offspring2)
            new_population.extend([offspring1, offspring2])
        population = new_population[:POPULATION_SIZE]
    print("\nBest solution found:")
    print(f"Chromosome: {best_solution}")
    print(f"x = {decode(best_solution)}")
    print(f"f(x) = {fitness_function(decode(best_solution))}")
# Parameters
POPULATION_SIZE = 4
CHROMOSOME_LENGTH = 5
MUTATION_RATE = 0.01
CROSSOVER_RATE = 0.8
GENERATIONS = 20

# Run it
if __name__ == "__main__":
    genetic_algorithm()
```