**LAB 1**

**Genetic Algorithm :**

Genetic Algorithms (GA) are inspired by the process of natural selection and genetics, where the fittest individuals are selected for reproduction to produce the next generation. GAs are widely used for solving optimization and search problems.

**CODE:**

from random import randint, random

# Objective function: maximize the number of 1s in the bitstring

def objective(bitstring):

return -sum(bitstring) # Return negative to minimize the negative count

# Tournament selection

def selection(pop, scores, k=3):

selection\_ix = randint(0, len(pop) - 1)

for ix in [randint(0, len(pop) - 1) for \_ in range(k - 1)]:

if scores[ix] < scores[selection\_ix]: # Minimize scores

selection\_ix = ix

return pop[selection\_ix]

# Crossover two parents to create two children

def crossover(p1, p2, r\_cross):

c1, c2 = p1.copy(), p2.copy()

if random() < r\_cross:

pt = randint(1, len(p1) - 2)

c1 = p1[:pt] + p2[pt:]

c2 = p2[:pt] + p1[pt:]

return [c1, c2]

# Mutation operator

def mutation(bitstring, r\_mut):

for i in range(len(bitstring)):

if random() < r\_mut:

bitstring[i] = 1 - bitstring[i] # Flip the bit

# Genetic algorithm

def genetic\_algorithm(objective, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut):

pop = [[randint(0, 1) for \_ in range(n\_bits)] for \_ in range(n\_pop)]

best, best\_eval = pop[0], objective(pop[0])

for gen in range(n\_iter):

scores = [objective(c) for c in pop]

for i in range(n\_pop):

if scores[i] < best\_eval: # Minimize scores

best, best\_eval = pop[i], scores[i]

print(f"> Generation {gen}, New best f({pop[i]}) = {scores[i]:.3f}")

selected = [selection(pop, scores) for \_ in range(n\_pop)]

children = []

for i in range(0, n\_pop, 2):

p1, p2 = selected[i], selected[i + 1]

for c in crossover(p1, p2, r\_cross):

mutation(c, r\_mut)

children.append(c)

pop = children # Replace population

return [best, best\_eval]

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

n\_bits = 20 # Length of bitstring

n\_iter = 100 # Number of generations

n\_pop = 100 # Population size

r\_cross = 0.9 # Crossover rate

r\_mut = 0.1 # Mutation rate

best, best\_eval = genetic\_algorithm(objective, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut)

print(f"Best solution: {best}, Best evaluation: {best\_eval:.3f}")

