**LAB 07**

**GA mechanism for optimization process:** The following is a sequence of steps of GA mechanism when used for optimization of problems.

Step 1: Generate the initial population randomly.   
Step 2: Select the initial solution with best fitness values.   
Step 3: Recombine the selected solutions using mutation and crossover operators.   
Step 4: Insert an offspring into the population.   
Step 5: Now, if the stop condition is met, return the solution with their best fitness value. Else go to step 2.

import random

from deap import base, creator, tools

def eval\_func(individual):

    target\_sum = 15

    return len(individual) - abs(sum(individual) - target\_sum),

def create\_toolbox(num\_bits):

    creator.create("FitnessMax", base.Fitness, weights=(1.0,))

    creator.create("Individual", list, fitness=creator.FitnessMax)

    toolbox = base.Toolbox()

    toolbox.register("attr\_bool", random.randint, 0, 1)

    toolbox.register("individual", tools.initRepeat,

                     creator.Individual, toolbox.attr\_bool, num\_bits)

    toolbox.register("population", tools.initRepeat, list, toolbox.individual)

    toolbox.register("evaluate", eval\_func)

    toolbox.register("mate", tools.cxTwoPoint)

    toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)

    toolbox.register("select", tools.selTournament, tournsize=3)

    return toolbox

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

    num\_bits = 45

    toolbox = create\_toolbox(num\_bits)

    random.seed(7)

    population = toolbox.population(n=500)

    probab\_crossing, probab\_mutating = 0.5, 0.2

    num\_generations = 10

    print('\nEvolution process starts')

    fitnesses = list(map(toolbox.evaluate, population))

    for ind, fit in zip(population, fitnesses):

        ind.fitness.values = fit

    print('\nEvaluated', len(population), 'individuals')

    for g in range(num\_generations):

        print("\n- Generation", g)

        offspring = toolbox.select(population, len(population))

        offspring = list(map(toolbox.clone, offspring))

        for child1, child2 in zip(offspring[::2], offspring[1::2]):

            if random.random() < probab\_crossing:

                toolbox.mate(child1, child2)

                del child1.fitness.values

                del child2.fitness.values

        for mutant in offspring:

            if random.random() < probab\_mutating:

                toolbox.mutate(mutant)

                del mutant.fitness.values

        invalid\_ind = [ind for ind in offspring if not ind.fitness.valid]

        fitnesses = list(map(toolbox.evaluate, invalid\_ind))

        for ind, fit in zip(invalid\_ind, fitnesses):

            ind.fitness.values = fit

        print('Evaluated', len(invalid\_ind), 'individuals')

        population[:] = offspring

        fits = [ind.fitness.values[0] for ind in population]

        length = len(population)

        mean = sum(fits) / length

        sum2 = sum(x\*x for x in fits)

        std = abs(sum2 / length - mean\*\*2)\*\*0.5

        print('Min =', min(fits), ', Max =', max(fits))

        print('Average =', round(mean, 2),

              ', Standard deviation =', round(std, 2))

    print("\nEvolution ends")

    best\_ind = tools.selBest(population, 1)[0]

    print('\nBest individual:\n', best\_ind)

    print('\nNumber of ones:', sum(best\_ind))

**OUTPUT:**



