Name: Devesh Vengurlekar
Roll No: 9766
TE Comps A

AI Experiment No. 10

Title: Travelling Salesman Problem

```
# Devesh Vengurlekar
#Roll No: 9766
# TE Comps A

import random

class GeneticAlgorithm:
    def __init__(self, cities, population_size=50, mutation_rate=0.01, elitism=True):
        self.cities = cities
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.elitism = elitism

    def create_initial_population(self):
        population = []
        for _ in range(self.population_size):
            population.append(random.sample(self.cities, len(self.cities)))
        return population

    def calculate_total_distance(self, route):
        total_distance = 0
        for i in range(len(route)):
            total_distance += self.distance(route[i], route[(i + 1) % len(route)])
        return total_distance

    def distance(self, city1, city2):
        return ((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2) ** 0.5

    def crossover(self, parent1, parent2):
        start = random.randint(0, len(parent1))
        end = random.randint(0, len(parent1))
        if start > end:
            start, end = end, start
        child = [None] * len(parent1)
```

```
        for i in range(start, end):
            child[i] = parent1[i]
        idx = 0
        for i in range(len(parent2)):
            if parent2[i] not in child:
                while child[idx] is not None:
                    idx += 1
                child[idx] = parent2[i]
        return child

    def mutate(self, route):
        for i in range(len(route)):
            if random.random() < self.mutation_rate:
                j = random.randint(0, len(route) - 1)
                route[i], route[j] = route[j], route[i]
        return route

    def select_parents(self, population, fitness_scores):
        total_fitness = sum(fitness_scores)
        probabilities = [score / total_fitness for score in fitness_scores]
        parent1 = random.choices(population, probabilities)[0]
        parent2 = random.choices(population, probabilities)[0]
        return parent1, parent2

    def evolve_population(self, population):
        fitness_scores = [1 / self.calculate_total_distance(route) for route in population]
        new_population = []
        if self.elitism:
            best_route_index = fitness_scores.index(max(fitness_scores))
            new_population.append(population[best_route_index])
        while len(new_population) < len(population):
            parent1, parent2 = self.select_parents(population, fitness_scores)
            child = self.crossover(parent1, parent2)
            child = self.mutate(child)
            new_population.append(child)
        return new_population

    def run(self, generations):
        population = self.create_initial_population()
        for _ in range(generations):
            population = self.evolve_population(population)
        best_route_index = min(range(len(population)), key=lambda i:
self.calculate_total_distance(population[i]))
        return population[best_route_index]
```

```
# Example usage
cities = [(0, 0), (1, 2), (3, 1), (5, 3), (4, 6)]
ga = GeneticAlgorithm(cities)
best_route = ga.run(1000)
print("Best Route:", best_route)
print("Total Distance:", ga.calculate_total_distance(best_route))
```

Output: