Program 3 - Ant Colony Optimization for the Traveling Salesman Problem:

Design and implement an Ant Colony Optimization (ACO) algorithm in Python to solve the Traveling Salesman Problem (TSP). The algorithm will simulate the foraging behavior of ants, where they explore and identify the shortest path that visits all cities exactly once and returns to the starting city.

Algorithm:

1. **Define the Problem**: Create a set of cities, represented by their coordinates.
2. **Initialize Parameters**:
    - Number of ants NNN
    - Importance of pheromone (α\alphaα)
    - Importance of heuristic information (β\betaβ)
    - Evaporation rate (ρ\rhoρ)
    - Initial pheromone value
3. **Construct Solutions**:
    - For each ant, start from a random city.
    - Probabilistically choose the next city using:
    - Repeat until all cities are visited.
4. **Evaluate Solutions**:
    - Calculate the total distance for each ant's tour.
    - Track the shortest distance and its route.
5. **Update Pheromones**:
    - Evaporate pheromone: τij=(1−ρ)⋅τij\tau_{ij} = (1 - \rho) \cdot \tau_{ij}τij=(1−ρ)⋅τij
    - Deposit new pheromone: τij+=∑antsQtour length\tau_{ij} += \sum_{\text{ants}} \frac{Q}{\text{tour length}}τij+=ants∑tour lengthQ
6. **Iterate**: Repeat steps 3–5 for a fixed number of iterations.
7. **Output the Best Solution**: Return the shortest route and its distance.

code:

```
import random

# Function to optimize: f(x) = -x^2 + 5x + 20
def fitness_function(x):
    return -x**2 + 5*x + 20

# Particle Swarm Optimization
def particle_swarm_optimization():
    # Input parameters
    num_particles = int(input("Enter number of particles: "))
    num_iterations = int(input("Enter number of iterations: "))
    inertia_weight = float(input("Enter inertia weight (w): "))
    cognitive_coeff = float(input("Enter cognitive coefficient (c1): "))
```

```
    social_coeff = float(input("Enter social coefficient (c2): "))
    x_min = float(input("Enter minimum value of x: "))
    x_max = float(input("Enter maximum value of x: "))

    # Initialize particles
    particles = [random.uniform(x_min, x_max) for _ in range(num_particles)]
    velocities = [random.uniform(-1, 1) for _ in range(num_particles)]
    personal_best_positions = particles[:]
    personal_best_fitness = [fitness_function(x) for x in particles]
    global_best_position =
particles[personal_best_fitness.index(max(personal_best_fitness))]

    # PSO main loop
    for _ in range(num_iterations):
        for i in range(num_particles):
            # Update velocity
            r1, r2 = random.random(), random.random()
            velocities[i] = (
                inertia_weight * velocities[i]
                + cognitive_coeff * r1 * (personal_best_positions[i] -
particles[i])
                + social_coeff * r2 * (global_best_position - particles[i])
            )
            # Update position
            particles[i] += velocities[i]
            particles[i] = max(min(particles[i], x_max), x_min)  # Clamp to
range

            # Evaluate fitness
            fitness = fitness_function(particles[i])
            if fitness > personal_best_fitness[i]:
                personal_best_fitness[i] = fitness
                personal_best_positions[i] = particles[i]

        # Update global best
        global_best_position =
personal_best_positions[personal_best_fitness.index(max(personal_best_fitness
))]

    # Output the best solution
    return global_best_position, fitness_function(global_best_position)
```

```python
# Run the PSO algorithm
best_solution, best_fitness = particle_swarm_optimization()
print(f"Best Solution: {best_solution}, Fitness: {best_fitness}")
print("Name-pooja Gaikwad (1BM22CS194)")
```

Output:

```
Enter number of particles: 10
Enter number of iterations: 15
Enter inertia weight (w): 0.5
Enter cognitive coefficient (c1): 2.0
Enter social coefficient (c2): 2.0
Enter minimum value of x: -10
Enter maximum value of x: 10
Best Solution: 2.499487628110455, Fitness: 26.249999737475047
Name-pooja Gaikwad (1BM22CS194)
```

Observation:

(5)* Ant colony :-

* Algorithm :-

1.* Start

2. Initiazlize parameter

R ← no of ants

Maxiteration ← Max no of iterations

ρ ← pheromone level

Q ← pheromone deposit constant

α ← pheromons influence

β ← Heuristic value

3. For iteration 1 to max :

Construct solution :

for each ant (R) :

place the ant at the starting node

choose the next node using transition probability

$$P_{ij} = \frac{(\tau_{ij})^{\alpha} (\eta_{ij})^{\beta}}{\Sigma ((\tau_{ij})^{\alpha} (\eta_{ij})^{\beta})}$$

update path ← Mark node j as visted

Store the solution for ant R.

4. calculate the pheromone deposted on the path

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta \tau_{ij}$$

where $\Delta \tau_{ij} = Q / \text{cost of edges used by ant}$

5. compare from all ants and give best solution.