**Genetics Algorithm**

**Vehicle Routing Problem:**

import numpy as np

import random

class PSO\_VRP:

    def \_\_init\_\_(self, n\_vehicles, n\_customers, coordinates, capacity, n\_particles, n\_iterations, w=0.5, c1=1.5, c2=1.5):

        self.n\_vehicles = n\_vehicles

        self.n\_customers = n\_customers

        self.coordinates = coordinates

        self.capacity = capacity

        self.n\_particles = n\_particles

        self.n\_iterations = n\_iterations

        self.w = w   # Inertia weight

        self.c1 = c1  # Cognitive component

        self.c2 = c2  # Social component

        self.particles = [self.random\_solution() for \_ in range(n\_particles)]

        self.velocities = [np.zeros\_like(self.particles[0]) for \_ in range(n\_particles)]

        self.best\_particle = self.particles[0]

        self.best\_fitness = float('inf')

    def random\_solution(self):

        # Ensure vehicle assignment is between 0 and n\_vehicles - 1

        solution = np.array([random.randint(0, self.n\_vehicles - 1) for \_ in range(self.n\_customers)])

        return solution

    def fitness(self, solution):

        vehicle\_loads = np.zeros(self.n\_vehicles)

        routes = {i: [] for i in range(self.n\_vehicles)}

        total\_distance = 0

        for customer, vehicle in enumerate(solution):

            routes[vehicle].append(customer)

            vehicle\_loads[vehicle] += 1

        if np.any(vehicle\_loads > self.capacity):

            return float('inf')

        for vehicle, route in routes.items():

            if len(route) == 0:

                continue

            route\_distance = self.calculate\_route\_distance(route)

            total\_distance += route\_distance

        return total\_distance

    def calculate\_route\_distance(self, route):

        distance = 0

        start = 0  # Depot is at index 0

        for customer in route:

            distance += np.linalg.norm(self.coordinates[start] - self.coordinates[customer])

            start = customer

        distance += np.linalg.norm(self.coordinates[start] - self.coordinates[0])  # Return to depot

        return distance

    def update\_particles(self):

        for i in range(self.n\_particles):

            for j in range(self.n\_customers):

                r1 = random.random()

                r2 = random.random()

                self.velocities[i][j] = self.w \* self.velocities[i][j] + self.c1 \* r1 \* (self.best\_particle[j] - self.particles[i][j]) + self.c2 \* r2 \* (self.best\_particle[j] - self.particles[i][j])

                self.particles[i][j] = int(np.round(self.particles[i][j]))  # Ensure the vehicle assignment is integer

                self.particles[i][j] = max(0, min(self.particles[i][j], self.n\_vehicles - 1))  # Ensure it's within valid bounds

    def optimize(self):

        for iteration in range(self.n\_iterations):

            for i in range(self.n\_particles):

                fitness\_value = self.fitness(self.particles[i])

                if fitness\_value < self.best\_fitness:

                    self.best\_fitness = fitness\_value

                    self.best\_particle = self.particles[i]

            self.update\_particles()

        return self.best\_particle, self.best\_fitness

coordinates = np.array([

    [0, 0],  # Depot at origin

    [1, 2],

    [2, 4],

    [3, 1],

    [4, 3],

    [5, 5]

])

n\_vehicles = 2

n\_customers = 5

capacity = 2

n\_particles = 30

n\_iterations = 100

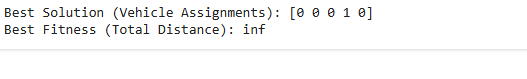
pso\_vrp = PSO\_VRP(n\_vehicles, n\_customers, coordinates, capacity, n\_particles, n\_iterations)

best\_solution, best\_distance = pso\_vrp.optimize()

print("Best Solution (Vehicle Assignments):", best\_solution)

print("Best Fitness (Total Distance):", best\_distance)

**OUTPUT:**

****