```
import random
import numpy as np
import math

def generate_cities(num_cities):
    return [(random.randint(0, 100), random.randint(0, 100)) for _ in range(num_cities)]

def compute_distance_matrix(cities):
    num_cities = len(cities)
    distances = [[0] * num_cities for _ in range(num_cities)]
    for i in range(num_cities):
        for j in range(num_cities):
            if i != j:
                distances[i][j] = math.sqrt(
                    (cities[i][0] - cities[j][0])**2 + (cities[i][1] - cities[j][1])**2
                )
    return distances

class TSP:
    def __init__(self, distances):
        self.distances = distances
        self.num_cities = len(distances)

    def fitness(self, route):

        total_distance = sum(
            self.distances[route[i]][route[i + 1]] for i in range(len(route) - 1)
        )
        total_distance += self.distances[route[-1]][route[0]]

class GeneticAlgorithm:
    def __init__(self, tsp, population_size=100, generations=500, mutation_rate=0.1):
        self.tsp = tsp
        self.population_size = population_size
        self.generations = generations
        self.mutation_rate = mutation_rate
        self.population = self._initialize_population()

    def _initialize_population(self):
        return [random.sample(range(self.tsp.num_cities), self.tsp.num_cities) for _ in range(self.population_

    def _select_parents(self):

        fitnesses = [self.tsp.fitness(route) for route in self.population]
        total_fitness = sum(fitnesses)
        probabilities = [f / total_fitness for f in fitnesses]
        return random.choices(self.population, probabilities, k=2)

    def _crossover(self, parent1, parent2):

        size = len(parent1)
        start, end = sorted(random.sample(range(size), 2))
        child = [-1] * size
        child[start:end] = parent1[start:end]

        p2_idx = 0
        for i in range(size):
            if child[i] == -1:
                while parent2[p2_idx] in child:
                    p2_idx += 1
                child[i] = parent2[p2_idx]
        return child
```

```
    def _mutate(self, route):

        if random.random() < self.mutation_rate:
            i, j = random.sample(range(len(route)), 2)
            route[i], route[j] = route[j], route[i]

    def evolve(self):
        for _ in range(self.generations):
            new_population = []
            for _ in range(self.population_size):
                parent1, parent2 = self._select_parents()
                child = self._crossover(parent1, parent2)
                self._mutate(child)
                new_population.append(child)
            self.population = new_population

    def get_best_solution(self):

        best_route = min(self.population, key=lambda route: 1 / self.tsp.fitness(route))
        best_distance = 1 / self.tsp.fitness(best_route)
        return best_route, best_distance

if __name__ == "__main__":
    num_cities = 5
    cities = generate_cities(num_cities)
    distances = compute_distance_matrix(cities)

    print("City Coordinates:")
    for i, city in enumerate(cities):
        print(f"City {i}: {city}")

    tsp = TSP(distances)
    ga = GeneticAlgorithm(tsp, population_size=50, generations=100, mutation_rate=0.2)
    ga.evolve()
    best_route, best_distance = ga.get_best_solution()

    print("\nBest route:", best_route)
    print("Best distance:", best_distance)
```

```
City Coordinates:
City 0: (22, 31)
City 1: (33, 46)
City 2: (89, 0)
City 3: (65, 0)
City 4: (3, 17)

Best route: [2, 1, 0, 4, 3]
Best distance: 202.96101904990562
```