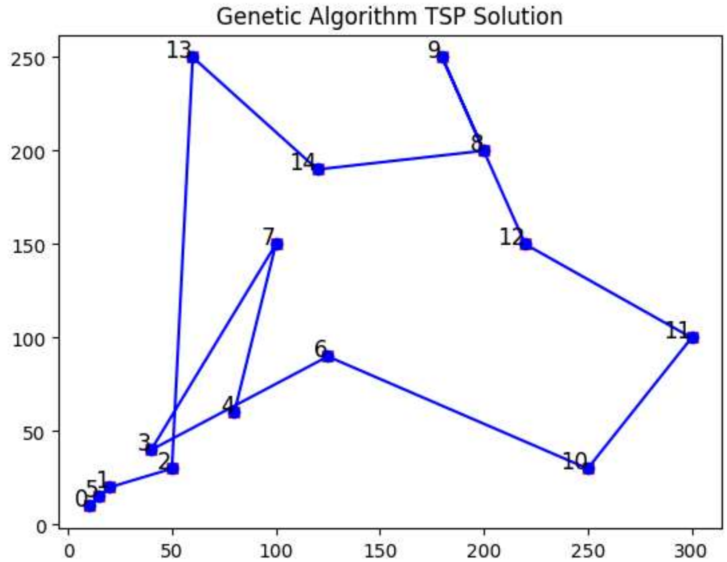
```
import numpy as np
import random
import matplotlib.pyplot as plt


def define_cities():
    return np.array([
        [10, 10], [20, 20], [50, 30], [40, 40], [80, 60],
        [15, 15], [125, 90], [100, 150], [200, 200], [180, 250],
        [250, 30], [300, 100], [220, 150], [60, 250], [120, 190]
    ])


def compute_distance(city1, city2):
    return np.linalg.norm(city1 - city2)


def compute_distance_matrix(cities):
    num_cities = len(cities)
    dist_matrix = np.zeros((num_cities, num_cities))
    for i in range(num_cities):
        for j in range(num_cities):
            dist_matrix[i, j] = compute_distance(cities[i], cities[j])
    return dist_matrix


def calculate_tour_length(tour, dist_matrix):
    length = 0
    for i in range(len(tour) - 1):
        length += dist_matrix[tour[i], tour[i + 1]]
    length += dist_matrix[tour[-1], tour[0]]
    return length


def initialize_population(num_individuals, num_cities):
    population = [np.random.permutation(num_cities) for _ in range(num_individuals)]
    return population


def tournament_selection(population, fitness, tournament_size=3):
    selected = np.random.choice(len(population), tournament_size, replace=False)
    best_idx = selected[np.argmin([fitness[i] for i in selected])]
    return population[best_idx]


def crossover(parent1, parent2):
    size = len(parent1)
    start, end = sorted(random.sample(range(size), 2))
    child = [-1] * size
    child[start:end] = parent1[start:end]

    idx = end
    for gene in parent2:
        if gene not in child:
            if idx >= size:
                idx = 0
            child[idx] = gene
            idx += 1
    return child


def mutate(tour, mutation_rate):
    if random.random() < mutation_rate:
        i, j = random.sample(range(len(tour)), 2)
        tour[i], tour[j] = tour[j], tour[i]


def genetic_algorithm(cities, num_individuals, num_generations, mutation_rate):
    dist_matrix = compute_distance_matrix(cities)
    num_cities = len(cities)
    population = initialize_population(num_individuals, num_cities)
    best_tour = None
    best_length = float('inf')

    for generation in range(num_generations):
        fitness = [calculate_tour_length(tour, dist_matrix) for tour in population]
```

```
        new_population = []


        for _ in range(num_individuals // 2):
            parent1 = tournament_selection(population, fitness)
            parent2 = tournament_selection(population, fitness)
            child1 = crossover(parent1, parent2)
            child2 = crossover(parent2, parent1)
            mutate(child1, mutation_rate)
            mutate(child2, mutation_rate)
            new_population.extend([child1, child2])


        population = new_population


        current_best_idx = np.argmin(fitness)
        current_best_length = fitness[current_best_idx]

        if current_best_length < best_length:
            best_length = current_best_length
            best_tour = population[current_best_idx]

    return best_tour, best_length


def plot_tour(cities, best_tour):
    tour_cities = cities[best_tour]
    plt.plot(tour_cities[:, 0], tour_cities[:, 1], 'bo-', markersize=6)
    plt.scatter(cities[:, 0], cities[:, 1], color='red', marker='x')

    for i, city in enumerate(cities):
        plt.text(city[0], city[1], f'{i}', fontsize=12, ha='right')

    plt.title("Genetic Algorithm TSP Solution")
    plt.show()

if __name__ == "__main__":
    cities = define_cities()
    num_individuals = 50
    num_generations = 200
    mutation_rate = 0.1


    best_tour, best_length = genetic_algorithm(cities, num_individuals, num_generations, mutation_rate)


    print("Tour Order:", best_tour)
    print(f"Best Tour Length: {best_length:.2f}")
    plot_tour(cities, best_tour)


print("Vatsal - 1BM22CS323")
```

```
Tour Order: [5, 0, 1, 2, 13, 14, 8, 9, 12, 11, 10, 6, 3, 7, 4]
Best Tour Length: 1003.07
```



Vatsal - 1BM22CS323