实验2：遗传算法求解TSP问题

实验目标

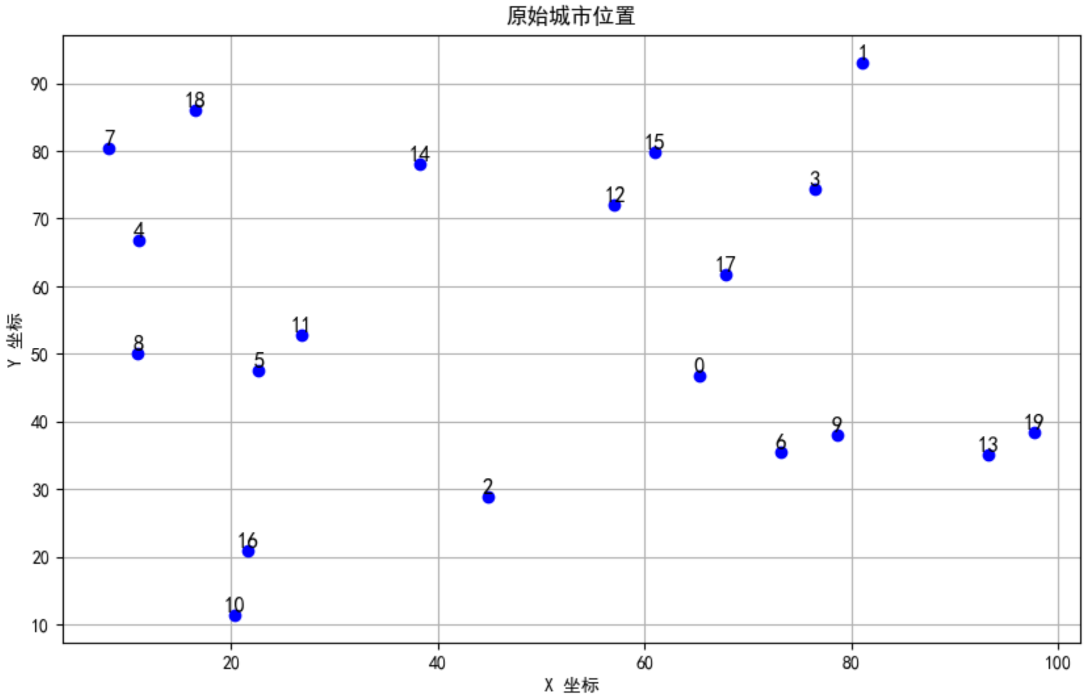
- 使用遗传算法解决TSP问题，找到经过20个城市的最短路径。

实现步骤

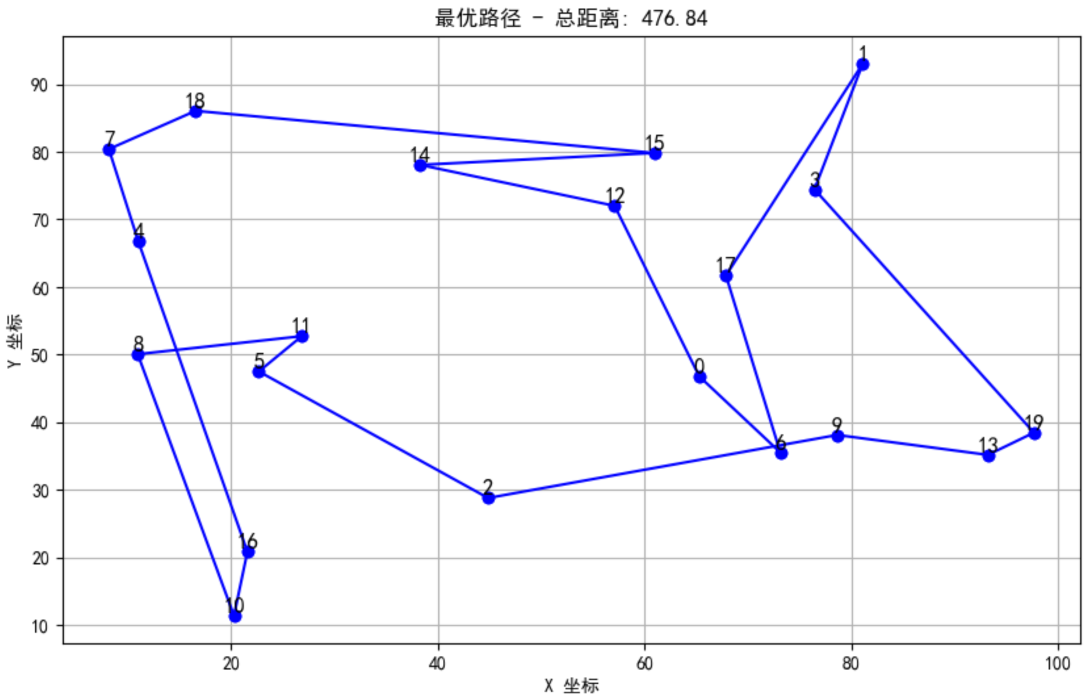
1. **编码设计**：采用排列编码表示路径。
2. **适应度函数**：路径长度的倒数作为适应度。
3. **选择、交叉与变异**：
    - 选择：采用锦标赛选择。
    - 交叉：使用部分映射交叉（PMX）。
    - 变异：采用交换变异。

实验结果

初始化城市位置



城市连接图



- 最优路径: [18, 15, 14, 12, 0, 6, 17, 1, 3, 19, 13, 9, 2, 5, 11, 8, 10, 16, 4, 7]
- 最短距离: 476.83840809953165

附录

```
import numpy as np

import matplotlib.pyplot as plt

# 设置中文字体

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode_minus'] = False

# 随机生成20个城市的坐标

City_Map = 100 * np.random.rand(20, 2)



# 计算两个城市之间的距离

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



# 计算路径的总长度

def calculate_total_distance(path):

    total_distance = 0

    for i in range(len(path) - 1):

        total_distance += calculate_distance(City_Map[path[i]], City_Map[path[i + 1]])

    total_distance += calculate_distance(City_Map[path[-1]], City_Map[path[0]])

    return total_distance



# 初始化种群

def initialize_population(pop_size, num_cities):

    population = []

    for _ in range(pop_size):

        individual = np.random.permutation(num_cities).tolist()

        population.append(individual)

    return population



# 选择操作

def selection(population, fitness, num_parents):

    parents = []

    for _ in range(num_parents):

        max_fitness_idx = np.where(fitness == np.max(fitness))[0][0]

        parents.append(population[max_fitness_idx])
```

```python
        fitness[max_fitness_idx] = -1  # 避免重复选择
    return parents


# 交叉操作
def crossover(parents, offspring_size):
    offspring = []
    for _ in range(offspring_size):
        parent1_idx = _ % len(parents)
        parent2_idx = (_ + 1) % len(parents)
        crossover_point = np.random.randint(1, len(parents[0]))
        child = parents[parent1_idx][:crossover_point]
        for city in parents[parent2_idx]:
            if city not in child:
                child.append(city)
        offspring.append(child)
    return offspring


# 变异操作
def mutation(offspring, mutation_rate):
    for individual in offspring:
        if np.random.rand() < mutation_rate:
            idx1, idx2 = np.random.choice(len(individual), 2, replace=False)
            individual[idx1], individual[idx2] = individual[idx2], individual[idx1]
    return offspring
```

```python
# 遗传算法主函数
def genetic_algorithm(pop_size, num_generations, mutation_rate):
    num_cities = len(City_Map)
    population = initialize_population(pop_size, num_cities)
    best_distance = float('inf')
    best_path = []

    for generation in range(num_generations):
        fitness = np.array([1 / calculate_total_distance(individual) for individual in population])
        best_idx = np.argmax(fitness)
        current_best_distance = calculate_total_distance(population[best_idx])
        if current_best_distance < best_distance:
            best_distance = current_best_distance
            best_path = population[best_idx]

        parents = selection(population, fitness, pop_size // 2)
        offspring = crossover(parents, pop_size - len(parents))
        offspring = mutation(offspring, mutation_rate)
        population = parents + offspring

    return best_path, best_distance

# 绘制原始城市位置
def plot_original_cities(city_map):
    plt.figure(figsize=(10, 6))

    for i, city in enumerate(city_map):

        plt.plot(city[0], city[1], 'bo')  # 绘制城市点

        plt.text(city[0], city[1], str(i), fontsize=12, ha='center',
va='bottom')  # 标注城市索引

    plt.title("原始城市位置")

    plt.xlabel("X 坐标")

    plt.ylabel("Y 坐标")

    plt.grid(True)

    plt.show()



# 参数设置

pop_size = 100

num_generations = 500

mutation_rate = 0.01



# 展示原始城市位置

plot_original_cities(City_Map)



# 运行遗传算法

best_path, best_distance = genetic_algorithm(pop_size, num_generations,
mutation_rate)



# 输出结果

print("最优路径:", best_path)

print("最短距离:", best_distance)
```

```
# 可视化展示最优路径
plt.figure(figsize=(10, 6))
for i in range(len(best_path) - 1):
    plt.plot([City_Map[best_path[i]][0], City_Map[best_path[i + 1]][0]],
             [City_Map[best_path[i]][1], City_Map[best_path[i + 1]][1]], 'bo-')
plt.plot([City_Map[best_path[-1]][0], City_Map[best_path[0]][0]],
         [City_Map[best_path[-1]][1], City_Map[best_path[0]][1]], 'bo-')
for i, city in enumerate(City_Map):
    plt.text(city[0], city[1], str(i), fontsize=12, ha='center', va='bottom')
plt.title(f"最优路径 - 总距离: {best_distance:.2f}")
plt.xlabel("X 坐标")
plt.ylabel("Y 坐标")
plt.grid(True)
plt.show()
```