实 验 报 告 单

实验名称：实验 6　ACS 算法解决 TSP 问题

同组人：　无　　实验课时：4
实验室：　双创 701　　报告日期：2024.4.16

一、实验目的

二、实验内容

假设有一个旅行商人要拜访 N 个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

蚁群算法是一种模拟自然界中蚂蚁觅食行为的仿生算法，通过模拟蚂蚁在寻找食物过程中留下的信息素（一种化学物质）来指导其他蚂蚁选择路径的行为。蚂蚁倾向于选择信息素浓度较高的路径，这种行为导致信息素在最优路径上积累，最终使所有蚂蚁都选择同一条最短路径。

三、实验环境

Python 版本：Python 3 及以上
所需要的依赖包：无
可采用的软件：PyCharm
或者使用其他语言，例如 C/C++, java 等

四、实验分析与设计过程

***Abstract*—This paper introduces the ant colony system (ACS), a distributed algorithm that is applied to the traveling salesman problem (TSP). In the ACS, a set of cooperating agents called ants cooperate to find good solutions to TSP's. Ants cooperate using an indirect form of communication mediated by a pheromone they deposit on the edges of the TSP graph while building solutions. We study the ACS by running experiments to understand its operation. The results show that the ACS outperforms other nature-inspired algorithms such as simulated annealing and evolutionary computation, and we conclude comparing ACS-3-opt, a version of the ACS augmented with a local search procedure, to some of the best performing algorithms for symmetric and asymmetric TSP's.**

蚁群系统（ACS）分布式算法，该算法被应用于解决旅行商问题（TSP）。在 ACS 中，一组被称为“蚂蚁”的协作代理通过一种间接的、通过他们在构建解决方案时在 TSP 图的边缘上沉积的信息素进行的通信来协作寻找 TSP 的良好解决方案。我们通过运行实验来研究 ACS 的运行情况。结果显示，ACS 超越了其他自然启发式算法，如模拟退火和进化计算。我们得出的结论是，与一些对称和非对称 TSP 的最佳性能算法进行比较，ACS-3-opt（一种增强了局部搜索过程的 ACS 版本）表现优异。

下方 Fig1 是个有关的例子，也是算法的自然原理

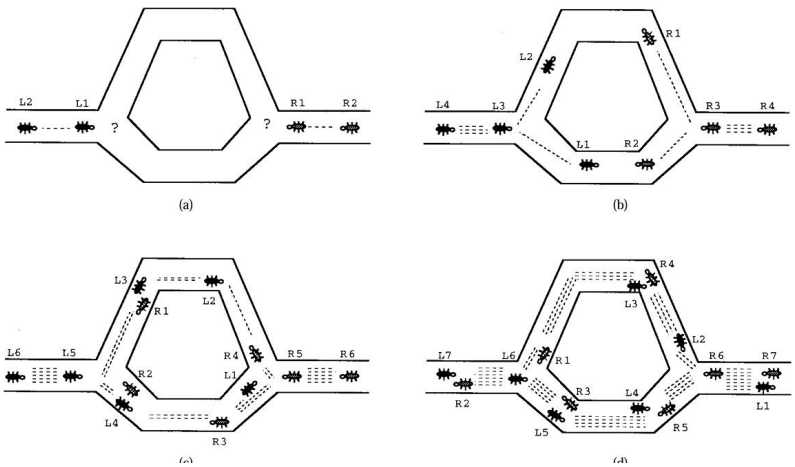


Fig. 1. How real ants find a shortest path. (a) Ants arrive at a decision point. (b) Some ants choose the upper path and some the lower path. The choice is random. (c) Since ants move at approximately a constant speed, the ants which choose the lower, shorter, path reach the opposite decision point faster than those which choose the upper, longer, path. (d) Pheromone accumulates at a higher rate on the shorter path. The number of dashed lines is approximately proportional to the amount of pheromone deposited by ants.

由自然原理得出结论：ACS 算法关键在于，蚂蚁，信息素

1. 由于蚂蚁的速度一样，由 V=S/t 得出，在路径长的蚂蚁停留时间长

2. 两端每只蚂蚁携带的信息素相同，结合 2 得出走长路径的蚂蚁在长路径上信息素密度低

3. 蚂蚁在长的路径上留下的信息素密度低，使后续蚂蚁选择该路径的可能性降低

The above behavior of real ants has inspired *ant system*, an algorithm in which a set of artificial ants cooperate to the solution of a problem by exchanging information via pheromone deposited on graph edges. The ant system has been applied to combinatorial optimization problems such as the traveling salesman problem (TSP) [7], [8], [10], [12] and the quadratic assignment problem [32], [42].

所以，算法需要实现：

1. artificial ants

2. pheromone

3. the progress of deposition and exchanging information

4. system and graph edges

来解决所谓的 TSP 和 QAP（二次分配问题），问题详情见 Fig. 2.

TSP

Let $V = \{a, \ldots, z\}$ be a set of cities, $A = \{(r,s): r,s \in V\}$ be the edge set, and $\delta(r,s) = \delta(s,r)$ be a cost measure associated with edge $(r,s) \in A$.

The TSP is the problem of finding a minimal cost closed tour that visits each city once.

In the case cities $r \in V$ are given by their coordinates (x_r, y_r) and $\delta(r,s)$ is the Euclidean distance between r and s, then we have an Euclidean TSP.

ATSP

If $\delta(r,s) \neq \delta(s,r)$ for at least some (r,s) then the TSP becomes an asymmetric TSP (ATSP).

Fig. 2. The traveling salesman problem.

在 TSP 问题中，定义了一个 V = {a,.....,z}的图代表城市，A={（r,s）:r,s∈V}边集表距离， δ（r,s）= δ（s,r）表示城市之间的关联（你到我和我到你之间的距离一样，也就是确定两点的相对位置），同时（r,s）∈A 也作为一个成本计量,每一个城市 r,s∈V，都会给出一个（X_r, Y_r）和δ（r,s）（欧几里得距离）

二维欧几里得公式：

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Ant system utilizes a graph representation which is the same as that defined in Fig. 2, augmented as follows: in addition to the cost measure $\delta(r,s)$, each edge (r,s) has also a desirability measure $\tau(r,s)$, called *pheromone*, which is updated at run time by artificial ants (*ants* for short). When ant system is applied to symmetric instances of the TSP, $\tau(r,s) = \tau(s,r)$,

同时，为了模拟信息素，我们需要引入期望测量 τ(r,s)，信息素 τ(r,s) 由人工蚂蚁进行更新（短路径上的蚂蚁），由于在 TSP 问题中只涉及点对点的期望，所以有

$$\tau(r,s) = \tau(s,r)$$

Informally, ant system works as follows. Each ant generates a complete tour by choosing the cities according to a probabilistic *state transition rule*; ants prefer to move to cities which are connected by short edges with a high amount of pheromone. Once all ants have completed their tours a *global pheromone updating rule* (global updating rule, for short) is applied; a fraction of the pheromone evaporates on all edges (edges that are not refreshed become less desirable), and then each ant deposits an amount of pheromone on edges which belong to its tour in proportion to how short its tour was (in other words, edges which belong to many short tours are the edges which receive the greater amount of pheromone). The process is then iterated.

对于人工蚂蚁，需要每一只蚂蚁都进行一次完整的遍历城市，遵循一个概率状态规则，蚂蚁会有更大概率走短路径 δ（r,s）↓ 且信息素高 τ(r,s) ↑的路，当所有蚂蚁都完全遍历并更新规则，我们得到一条最短路径。

同时，考虑到如果每条路上都会残留信息素（或多或少），算法又是由概率去进行选择路径，所以添加了一个信息素蒸发机制，让少量残留的信息素蒸发，也就是说，当某条路走的蚂蚁过少，信息素残留少，经过时间推移，信息素蒸发，会使蚂蚁走这条路的可能性变为 0，保证结果的正确性。需要注意信息素的蒸发速率的设置。

random-proportional rule

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}}, & \text{if } s \in J_k(r) \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

（1）其中 η(r,s)为 δ（r,s）的倒数，β是 r 与 s 之间的关联程度变量（β>0），J_k（r)是 k 下一阶段要去的城市数量。该公式满足了更大概率走短路径 η（r,s）↑且信息素高 τ(r,s）↑的路

In ant system, the global updating rule is implemented as follows. Once all ants have built their tours, pheromone is updated on all edges according to

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \sum_{k=1}^{m} \Delta\tau_k(r,s) \quad (2)$$

where

$$\Delta\tau_k(r,s) = \begin{cases} \dfrac{1}{L_k}, & \text{if } (r,s) \in \text{tour done by ant } k \\ 0, & \text{otherwise} \end{cases}$$

（2）：0<α<1 是蒸发变量，L_k 是每只蚂蚁走过的总路程，L_k 越小，Δ τ_k（r,s）越大，符合信息素蒸发和叠加。

* 在某些数学文献或笔记中，"←" 可能被用作一个非标准的“等于”或“定义为”的符号。但这并不是数学中的标准符号。

算法思路如下 Fig. 3

```
Initialize
Loop /* at this level each loop is called an iteration */
        Each ant is positioned on a starting node
        Loop /* at this level each loop is called a step */
                Each ant applies a state transition rule to incrementally build a solution
                and a local pheromone updating rule
        Until all ants have built a complete solution
        A global pheromone updating rule is applied
Until End_condition
```

Fig. 3. The ACS algorithm.

五、代码解读

我代码分成了四个部分

dbconn_cities.py:与存有城市经纬度的数据库相连，并负责输入天气、风向参数，选择城市存入字典中

matrix.py:将数据处理成举证，方便后续运算

aoc.py:asc 算法处理矩阵，给出结果

plt_figure.py:绘图输出

#dbconn_cities.py

```python
import time
import pandas as pd
import mysql.connector

# MySQL 数据库连接信息
config = {
    'user': 'root',
    'password': 'root',
    'host': 'localhost',  # 或者你的数据库服务器地址
    'database': 'aoc_data',
    'raise_on_warnings': True
}

# 创建数据库连接
cnx = mysql.connector.connect(**config)

# 使用 pandas 的 read_sql_query 函数从数据库中读取数据
query = "SELECT * FROM cities"
```

```
# 城市和它们的经纬度
city_names = pd.read_sql_query(query, cnx)

print("\033[36m{:=^50s}\033[0m".format("Split Line"))
print("""    天气因子:
    \033[32m100  晴天
    200  阴天\033[0m
    \033[33m300  暴雨
    400  雪\033[0m
    \033[31m500  雷雨
    600  冰雹
    700  极端天气\033[0m""")
print("\033[36m{:=^50s}\033[0m".format("Split Line"))
print("""    风向因子:
    \033[32m100  顺风\033[0m
    \033[33m200  无风\033[0m
    \033[31m500  逆风\033[0m""")
print("\033[36m{:=^50s}\033[0m".format("Split Line"))
# 城市数目
num = int(input("\033[34m请输入安排航班的城市数量: \033[0m"))
# 开始计时
start = time.time()

city_names = city_names.set_index('城市')

cities = {}
cities_weather = {}
for x in range(1, num + 1):  # 循环从1开始，但确保结束条件是num + 1来包含所有城市
    name_num = input("\033[34m请输入第{}城市名称: \033[0m".format(x))  # 不需要转换为str，因为input已经返回str
    weather_num = input("\033[34m请输入第{}城市的天气\033[0m: ".format(x))
    wind_num = input("\033[34m请输入第{}城市的风向: \033[0m".format(x))
    try:
        # 使用.at[]访问器，它更快且适用于单个值的查找
        latitude = city_names.at[name_num, '北纬']
        longitude = city_names.at[name_num, '东经']
        cities[name_num] = (longitude, latitude)
        cities_weather[name_num] = (int(weather_num),int(wind_num))
    except KeyError:
        print("找不到城市名称: '{}'。请重试。".format(name_num)
# 当你完成所有操作后，关闭数据库连接
cnx.close()
```

数据库部分展示（MySQL+DataGrip）

	省级行政区	城市	北纬	东经
1	北京	北京市	39.90469	116.40717
2	天津	天津市	39.0851	117.19937
3	上海	上海市	31.23037	121.4737
4	重庆	重庆市	29.56471	106.55073
5	香港特别行政区	九龙	22.327115	114.17495
6	香港特别行政区	新界	22.341766	114.202408
7	香港特别行政区	香港岛	22.266416	114.177314
8	澳门特别行政区	路环岛	22.116226	113.564857
9	澳门特别行政区	澳门半岛	22.198751	113.549134
10	澳门特别行政区	氹仔岛	22.156838	113.577669
11	台湾省	台中市	24.13862	120.67951
12	台湾省	台北市	25.037798	121.56517
13	台湾省	台南市	23.172478	120.279363
14	台湾省	嘉义市	23.481568	120.452538
15	台湾省	高雄市	22.620856	120.286795
16	台湾省	基隆市	25.130741	121.746248
17	台湾省	新北市	25.012366	121.465746

matrix.py

```python
import numpy as np
from dbconn_cities import cities, num, cities_weather

# 将城市坐标转换为 points
points = list(cities.values())
weather_points = list(cities_weather.values())
print("检查初代点坐标: ")
print(points)
print(weather_points)


# 计算初代点 symmetric matrix 的函数
def distance_matrix(points):
    # 初始化一个 0 matrix
    matrix = np.zeros((num, num))

    # 开始计算各点之间的距离
    for i in range(num):
        for j in range(num):
```

```
            # 如果当前点=>当前点，则认为是 infinite distance
            if i == j:
                matrix[i, j] = np.inf
            # 否则，计算两个点之间的欧式距离
            else:
                matrix[i, j] = np.sqrt((points[i][0] - points[j][0]) ** 2 +
(points[i][1] - points[j][1]) ** 2)

    # 返回计算后的 matrix
    return matrix


def weather_conditions_matrix(weather_points):
    # 初始化一个 0 matrix
    matrix = np.zeros((num, num))
    for i in range(num):
        for j in range(num):
            if i == j:
                matrix[i, j] = np.inf
            else:
                matrix[i, j] = np.sqrt((weather_points[i][0] -
weather_points[j][0]) ** 2 + (weather_points[i][1] - weather_points[j][1]) ** 2)
    # 返回计算后的 matrix
    return matrix

# 进行天气矩阵生成
weather_matrix = weather_conditions_matrix(weather_points)
# 检查天气矩阵
print(weather_matrix)
# 进行初代矩阵生成
original_matrix = distance_matrix(points)
# 检查初代矩阵
print(original_matrix)
```

aoc. py

```
import time
import numpy as np
from dbconn_cities import num, start
from matrix import original_matrix, points, weather_matrix


# 蚁群算法类
class ACO:
    # 初始化（包括城市数目、距离矩阵、蚂蚁数目、迭代次数、信息素因子、距离因子、
```

```
挥发因子、信息素强度）
    def __init__(self, num_cities, distance_matrix, weather_matrix, num_ants,
max_iter=200, alpha=2, beta=5, rho=0.5, q=100):
        # 城市数目
        self.num_cities = num_cities
        # 距离矩阵
        self.distance_matrix = distance_matrix
        #天气矩阵
        self.weather_matrix = weather_matrix
        # 信息素矩阵，其中 shape 属性返回数组的维度
        self.pheromone = np.ones(self.distance_matrix.shape) / num_cities
        # 蚂蚁数目
        self.num_ants = num_ants
        # 迭代次数
        self.max_iter = max_iter
        # 信息素因子
        self.alpha = alpha
        # 距离因子
        self.beta = beta
        # 挥发因子
        self.rho = rho
        # 信息素强度
        self.q = q

    # 更新信息素
    def _update_pheromone(self, ants):
        self.pheromone *= (1 - self.rho)
        epsilon = 1e-10  # 避免除以零
        for ant in ants:
            for i in range(self.num_cities - 1):
                self.pheromone[ant[i]][ant[i + 1]] += self.q / (
                            self.distance_matrix[ant[i]][ant[i + 1]] + epsilon) +
self.q / (

self.weather_matrix[ant[i]][ant[i + 1]] + epsilon)
            self.pheromone[ant[-1]][ant[0]] += self.q /
(self.distance_matrix[ant[-1]][ant[0]] + epsilon) + self.q / (
                        self.weather_matrix[ant[-1]][ant[0]] + epsilon)

    # 选择下一个城市
    def _choose_next_city(self, current_city, visited):
        available_cities = [i for i in range(self.num_cities) if i not in visited]
        probabilities = np.zeros(len(available_cities))
        epsilon = 1e-10  # 避免除以零
```

```
        for i, city in enumerate(available_cities):
            probabilities[i] = self.pheromone[current_city][city] ** self.alpha *
(
                    (1.0 / (self.distance_matrix[current_city][city] + epsilon))
** self.beta +
                    (1.0 / (self.weather_matrix[current_city][city] + epsilon))
** self.beta
            )
        if np.all(probabilities == 0):
            return np.random.choice(available_cities)  # 所有概率都是零时，随机
选择一个城市
        probabilities /= np.sum(probabilities)
        chosen_city_index = np.random.choice(len(available_cities),
p=probabilities)
        return available_cities[chosen_city_index]
    # 解决 TSP 问题
    def solve(self):
        # 初始化最好路线和距离
        best_route = None
        best_distance = float('inf')

        # 进行每一次的迭代
        for _ in range(self.max_iter):
            # 初始化蚂蚁，令他们随机分配到一个城市
            ants = [[np.random.randint(self.num_cities)] for _ in
range(self.num_ants)]

            # 遍历每一只蚂蚁
            for ant in ants:
                # 生成完整的路径
                for _ in range(self.num_cities - 1):
                    # 当前路径中的最后一个城市
                    current_city = ant[-1]
                    # 调用函数，选择下一个访问的城市
                    next_city = self._choose_next_city(current_city, ant)
                    # push 到蚂蚁的数组里面
                    ant.append(next_city)

            # 更新信息素
            self._update_pheromone(ants)

            # 遍历每一只蚂蚁
            for ant in ants:
                # 生成距离
```

```
                distance = sum([self.distance_matrix[ant[i]][ant[i + 1]] for i in
range(-1, self.num_cities - 1)]+[self.weather_matrix[ant[i]][ant[i + 1]] for i in
range(-1, self.num_cities - 1)])
                # 更新最好距离
                if distance < best_distance:
                    best_distance = distance
                    best_route = ant

        # 返回最优解
        return best_route, best_distance


# 蚂蚁的数量（城市数量的 1.5 倍）
num_ants = int(num * 15)

# 初始化 aco 类
aco = ACO(num, original_matrix, weather_matrix,num_ants)

# 利用类中的 solve 函数，进行 TSP 问题的求解
best_route, best_distance = aco.solve()

# 结束计时
end = time.time()

# 计算算法的耗时
during = end - start
print(f"Time consumption: {during}s")

# 输出最佳路线和距离
print(f"Best route: {best_route}")
print(f"Best distance: {best_distance}")

# 使用 zip 分离坐标
x, y = zip(*points)

# 初始化新的 x 和 y 坐标的数组
newx = []
newy = []

# 根据 best-route 更新 newx 和 newy
for i in range(num):
    # 当前点的 id 是 best-route 中的编号
    id = best_route[i]
    # 新增点的坐标信息
```

```
    newx.append(x[id])
    newy.append(y[id])
```

plt_figure.py

```
import matplotlib.pyplot as plt
from matplotlib import font_manager
from aoc import newx, newy, best_route
from dbconn_cities import num, city_names, cities

font_path = 'C:/Windows/Fonts/msyh.ttc'  # 请确保路径是正确的
prop = font_manager.FontProperties(fname=font_path)

# 设置 matplotlib 的全局字体为 SimHei
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的
问题

plt.figure()
plt.scatter(newx, newy, color='blue', s=2)
for i in range(num):
    # 下一个点需要对 num 进行取余
    next_i = (i + 1) % numdcmatr
    plt.plot([newx[i], newx[next_i]], [newy[i], newy[next_i]], color='red')

# 绘制图形并添加标签、标题等
plt.xlabel("经度", fontproperties=prop)  # 使用支持中文的字体
plt.ylabel("纬度", fontproperties=prop)
plt.title("蚁群算法路径", fontproperties=prop)

# 背景含网格
plt.grid(True)

# 假设您已经有了 best_route 和对应的城市坐标 newx, newy
city_names = list(cities.keys())
for i in range(num):
    plt.text(newx[i], newy[i], city_names[best_route[i]], fontsize=10,
ha='center', va='center', fontproperties=prop)

# 展示图片
plt.show()
```

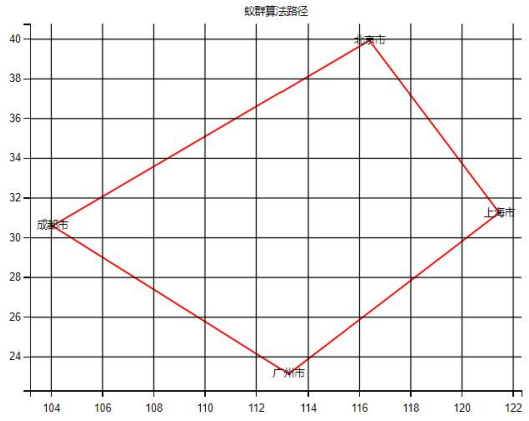
六、实验测试

输入格式：

```
====================Split Line====================
    天气因子：
    0   晴天
    10   阴天
    30   暴雨
    40   雪
    50   雷雨
    60   冰雹
    100   极端天气
====================Split Line====================
    风向因子：
    0   顺风
    20   无风
    40   逆风
====================Split Line====================
请输入安排航班的城市数量：4
请输入第1城市名称：北京市
请输入第1城市的天气：0
请输入第1城市的风向：0
```

输出格式：

```
检查初代点坐标：
[(116.40717, 39.90469), (121.4737, 31.23037), (104.06476, 30.5702), (113.26436, 23.12908)]
[(0, 0), (10, 20), (30, 0), (20, 40)]
[[       inf 22.36067977 30.         44.72135955]
 [22.36067977        inf 28.28427125 22.36067977]
 [30.         28.28427125        inf 41.23105626]
 [44.72135955 22.36067977 41.23105626        inf]]
[[       inf 10.04557384 15.47474679 17.06746453]
 [10.04557384        inf 17.42145276 11.53361014]
 [15.47474679 17.42145276        inf 11.83228241]
 [17.06746453 11.53361014 11.83228241        inf]]
Time consumption: 131.448062419891365
Best route: [2, 3, 1, 0]
Best distance: 164.83862898210583
```

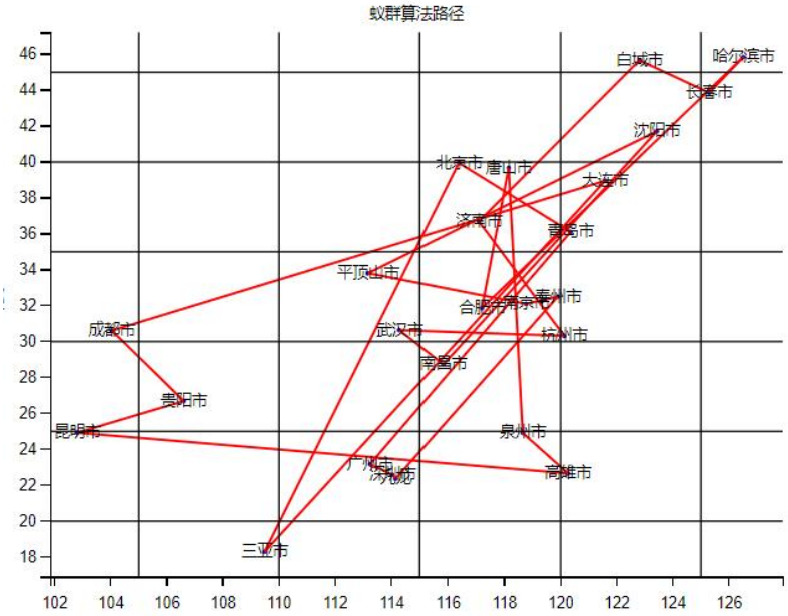


由结果可知，最佳线路为：成都市->广州市->上海市->北京市

实验测试案例：

a. 20+以上随机城市，天气，因子

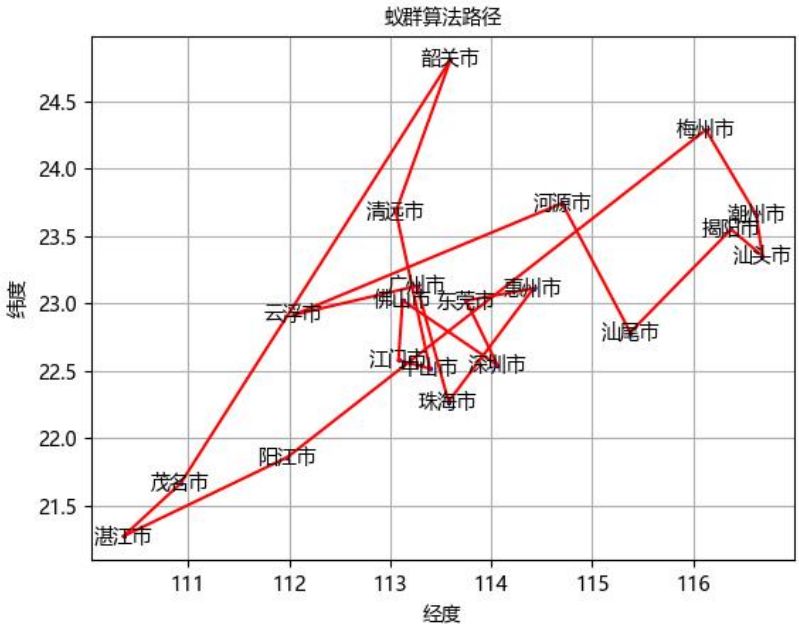
城市	天气因子	风向因子
北京市	10	0
九龙	0	20
高雄市	30	10
唐山市	50	20
平顶山市	10	10
济南市	20	30
青岛市	0	0
沈阳市	10	10
大连市	10	0
长春市	30	20
白城市	10	30
哈尔滨市	50	10
南京市	10	10
泰州市	10	20
杭州市	20	30
合肥市	70	10
泉州市	50	20
南昌市	10	10
武汉市	20	30
广州市	0	0
深圳市	10	10
三亚市	10	0
成都市	10	0
贵阳市	0	20
昆明市	30	10




```
Time consumption: 355.2465810775757s
Best route: [3, 16, 2, 24, 23, 22, 8, 21, 0, 6, 19, 20, 1, 13, 12, 4, 7, 17, 18, 14, 5, 10, 9, 11, 15]
Best distance: 493.92133581160886
```

b. 密集多城市

城市	天气因子	风向因子
广州市	10	10
韶关市	30	0
深圳市	10	10
珠海市	0	10
汕头市	10	0
佛山市	10	10
江门市	10	20
湛江市	20	20
茂名市	30	0
惠州市	0	10
梅州市	30	10
汕尾市	0	0
河源市	0	0
阳江市	20	20
清远市	40	0
东莞市	10	10
中山市	50	10
潮州市	60	0
揭阳市	0	0
云浮市	0	10

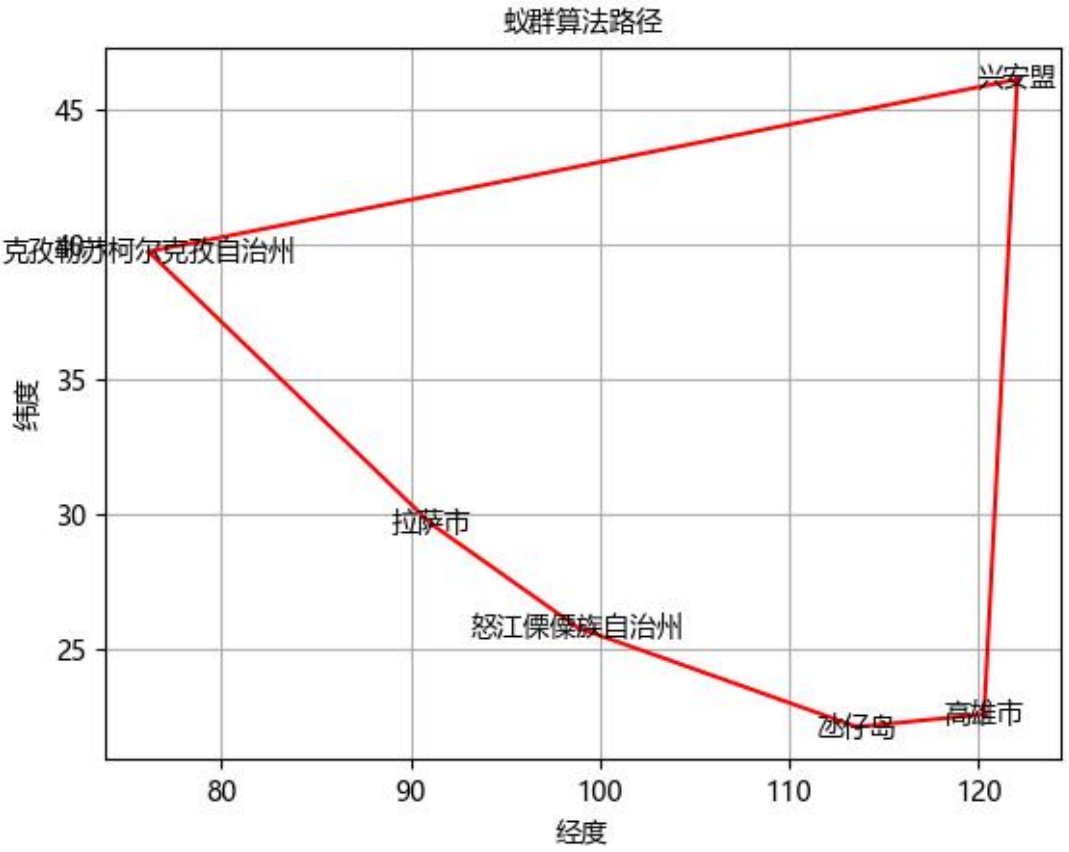



```
Time consumption: 293.188129901886s
Best route: [19, 0, 3, 9, 15, 2, 5, 6, 16, 14, 1, 8, 7, 13, 10, 17, 4, 18, 11, 12]
Best distance: 250.39420663278437
```

c. 远距离城市

城市	天气因子	风向因子
高雄市	0	0
拉萨市	0	0
冰仔岛	0	0
克孜勒苏柯尔克孜自治州	0	0
兴安盟	0	0
怒江傈僳族自治州	0	0

```
Time consumption: 52.15990114212036s
Best route: [0, 2, 5, 3, 1, 4]
Best distance: 118.39256069742974
```



七、参考文献

[1]:

Marco Dorigo, Luca Maria Gambardella.Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1, APRIL 1997