TSP 问题 ACO

1.实验环境：

- OS: Windows 10
- CPU: intel CORE i5
- Python 3.6
- Sckikit_learn 0.18.1
- 其他 Python 科学计算模块
- 编译器：PythonCharm 2017.3.3

2.主要参考文档：

- Sklearn 0.18 官方文档

3.实验过程

3.1.实验问题描述

TSP 问题（Traveling Salesman Problem），是数学领域著名的问题之一，设有一个旅行商人要拜访 n 个城市，他必须选择要走的路径，路径的限制是每一个城市只能拜访 1 次，而且最后要回到原来的城市。路径的选择目标是要求得到的路径为所有路径之中的最小值。

TSP 问题是一个组合优化问题。该问题被证明具有 NPC 计算复杂度。迄今为止，还没有一个有效的算法，所以大家认为这类问题的大型实例不能用精确算法求解，必须寻求这类问题的有效的近似算法。

本实验使用 ACO(ant colony optimization)，即蚁群算法来解决 TSP 问题。

3.2.实验资源库：

本实验在测试过程中，使用到了一些数据来自于 TSPLIB 网站，但是这个网站的很多数据来自于实际的生活中，所以坐标点集合有些过于排列整齐，我对其中的数据进行了选择以及一些修改。

TSPLIB 网站网址：

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/

3.3.实验原理与过程：

这个实验参考了网上的博客，在充分理解的基础上做出了自己的修改和一些必要的补充。

首先简单阐述一下 ACO 算法的原理：

蚁群算法是由意大利学者 M. 多瑞格在 20 世纪 90 年代提出的一种模拟蚂蚁群体觅食行为的优化算法，这个算法具有思路新颖，鲁棒性，并行性等优点，得到了广泛关注与应用。

ACO 算法在 TSP 中运用的原理可以使用一下几个公式来进行说明：

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum_{l\in N_i^k(t)}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}} & j\in N_i^k(t)\\ 0 & j\notin N_i^k(t)\end{cases}$$

在这个式子中，$P_{ij}^{k}(t)$表示的是一只蚂蚁选择下一个要走城市节点的概率，对于能够到达的城市节点集合$N_i^k(t)$中，不能到达的城市节点是当前已经遍历的城市节点。$\tau_{ij}(t)$表示信息素浓度，$\eta_{ij}(t)$表示启发式信息，一般都是取路径长度倒数；α 和 β 为参数，分别反应了信息素浓度和启发式信息的影响程度。根据得到的概率，以轮盘赌的方式得到下一个要走的城市。

对于信息素浓度的更新，这个实验采取的方法是每迭代一次就更新一次。迭代一次的蚂蚁数量为 50 只，在 50 只蚂蚁走过了一遍以后，根据路径长度和走过的路径，对信息素浓度进行更新。更新信息素的基本方法是：

$$\Delta\tau_{ij}=\sum_{k=1}^{m}\Delta\tau_{ij}(t)$$

$$\Delta\tau_{ij}(t+1)=(1-\rho)\cdot\tau_{ij}(t)+\Delta\tau_{ij}(t)$$

在这个公式中，ρ表示的是信息素挥发的速率。根据实际情况，在每迭代一次之后，上一次的信息素应该以一定的比例挥发，这样才不至于使得最后的所有路径的信息素浓度都在升高。

信息素浓度的更新采用蚁周模型。设 Q 为常量，L^k为第 k 只蚂蚁遍历城市的路径长度和，则蚁周模型对应的$\Delta\tau_{ij}(t)$如下：

$$\Delta\tau_{ij}(t)=\begin{cases}\dfrac{Q}{L^k} & \text{第 k 只蚂蚁在本次遍历中经过的路径}c_i-c_j\\ 0 & \textit{其他}\end{cases}$$

根据选择方法和信息素浓度更新公式，蚂蚁系统解决旅行商问题的算法流程如下：

- 将参数初始化
- 将 m 只蚂蚁随机放在 n 个城市上
- 针对每只蚂蚁执行以下步骤：
  根据选择下一个城市概率的公式，得到选择城市的概率
  使用轮盘赌的方法，选择下一个城市
  重复以上的两个步骤
- 一次迭代完成之后，更新所有路径上的信息素浓度
- 如果满足终止条件，算法停止，输出最优的结果，否则继续迭代，选择下一个城市

以上步骤最为核心的步骤就是选择下一个城市和信息素浓度的更新。包括蚂蚁系统在内的所有蚁群优化算法都可以认为是由解路径构建和信息素浓度更新两个关键过程构成的。

下面是具体的代码的实现：

首先是对所有参数的初始化。

```
def __init__(self, cities, num_ants=50):
    '''

    :param cities: 城市的数量，初始化为 50
    :param num_ants: 蚂蚁的数量，初始化为 50
    '''
    self.cities = cities
    self.num_cities = len(cities)

    #初始化距离矩阵
    self.Distance = np.empty((self.num_cities, self.num_cities))
    for i in range(self.num_cities):
        for j in range(self.num_cities):
            V = (cities[i][0] - cities[j][0], cities[i][1] - cities[j][1])
            self.Distance[i, j] = np.linalg.norm(V)
    self.Pher = np.ones((self.num_cities, self.num_cities))

    self.num_ants = num_ants
    self.alpha = 1
    self.beta = 2
    self.rho = 0.5
    self.Q = 100

    self.bestPath_Length = float('inf')
    self.bestPath = None
```

对所有城市点之间的距离，对所有路径上的信息素浓度初始化，对其他相关各个参数初始化。

接下来是一个关键的函数——选择下一个城市的函数：

```
def _NextCity(self, current_city, available_cities):
    '''

    :param current_city: 当前蚂蚁所在的城市
    :param available_cities: 当前蚂蚁可以到达的城市集合
    :return: 返回蚂蚁选择的下一个城市
    '''
    Probs = []
    for i in available_cities:
        # 根据选择城市的概率公式，得到分子的信息素浓度的 alpha 次方与到下一个城市启发式信息的 beta 次方
        Pher = pow(self.Pher[current_city][i], self.alpha)
        Distance = pow(self.Distance[current_city][i], self.beta)
        Probs.append(Pher / Distance)
    sum_weight = sum(Probs)
```

```
    Probs = list(map(lambda x: x / sum_weight, Probs))
    #这里根据轮盘赌方法随机选择下一个城市
    next_city = np.random.choice(available_cities, p=Probs)
    return next_city
```

这个函数基本是按照公式来写的，并且调用了 numpy 中的一个轮盘赌选择的函数来选择下一个城市，返回类型就是选择出来的下一个城市。
整个算法的主要流程函数如下：

```
def _Ant(self):
    '''''
    蚂蚁不断根据当前的城市节点选择下一个节点，直到走遍所有的节点，返回最终的路径
    :return:
    '''
    available_cities = list(range(self.num_cities))
    path = []

    current_city = np.random.choice(available_cities)
    available_cities.remove(current_city)
    path.append(current_city)

    for i in range(self.num_cities - 1):
        next_city = self._NextCity(current_city, available_cities)
        available_cities.remove(next_city)
        path.append(next_city)
        current_city = next_city

    return path
```

每一只蚂蚁的运行过程都有两个步骤,：循环的选择下一个城市节点，移动到下一个城市节点；然后在这只蚂蚁的路径的集合中添加进下一个城市节点。整个过程的结束为蚂蚁遍历完所有的节点。整个函数返回一只蚂蚁遍历的路径。

接下来，就是最后一个公式，如何去更新信息素浓度，由于这个实验我们采用了蚁周模型，所以，需要在一个周期之后才去更新各个路径的信息素浓度，并且信息素还有挥发机制，这些都需要在函数中体现出来。

```
def _UpdatePher(self, path_list):
        '''''
        :param path_list: 一次迭代的路径集合
        :return:
        '''
        Phers = np.zeros((self.num_cities, self.num_cities))
        for path_Length, path in path_list:
            amount_Pher = self.Q / path_Length
```

```
            for i in range(-1, self.num_cities - 1):
                Phers[path[i], path[i + 1]] += amount_Pher
                Phers[path[i + 1], path[i]] += amount_Pher
        self.Pher = self.Pher * self.rho + Phers
```

算法的主要的步骤就是这些，当然还包含一些细节的处理函数，比如需要画出图表的分析。

4.实验结果与分析

这个实验的各个参数为

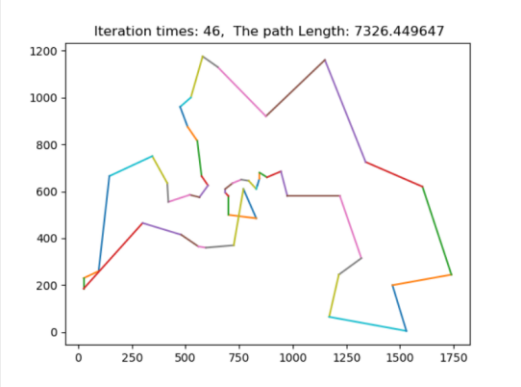
```
Cities=50, n_ants=50, alpha=1, beta=2, rho=0.5, Q=100
```

最后的结果如图所示

```
Iterate times: 1, Length: 12031.224425, Path: [39, 20, 33, 7, 6, 19, 45, 36, 4, 14, 15, 16, 42, 34, 26, 43, 37, 11, 0, 8, 28, 27, 30, 18, 5, 38, 1, 3, 12, 17, 23, 2, 2
Iterate times: 2, Length: 11988.998588, Path: [21, 37, 11, 43, 0, 8, 26, 2, 3, 9, 6, 4, 7, 5, 19, 18, 42, 14, 15, 16, 33, 45, 36, 20, 34, 41, 46, 48, 30, 35, 47, 31, 3
Iterate times: 3, Length: 11699.382419, Path: [18, 20, 45, 36, 21, 42, 6, 14, 15, 16, 8, 37, 9, 34, 4, 7, 5, 33, 19, 11, 26, 17, 43, 0, 24, 25, 10, 2, 22, 23, 12, 44,
Iterate times: 4, Length: 10166.107750, Path: [42, 34, 11, 37, 9, 23, 12, 2, 44, 1, 38, 15, 16, 6, 14, 4, 7, 5, 36, 45, 19, 33, 20, 18, 46, 21, 47, 48, 30, 3, 10, 25,
Iterate times: 5, Length: 10175.432001, Path: [6, 5, 7, 4, 16, 15, 14, 42, 20, 33, 19, 45, 36, 34, 25, 10, 24, 2, 12, 23, 3, 9, 11, 18, 46, 39, 30, 47, 48, 29, 40, 31,
Iterate times: 6, Length: 9843.641341, Path: [15, 16, 6, 4, 5, 19, 33, 45, 36, 20, 18, 46, 47, 48, 31, 35, 32, 40, 29, 39, 30, 17, 23, 3, 9, 11, 37, 14, 42, 21, 38, 1
Iterate times: 7, Length: 8834.061877, Path: [24, 2, 17, 28, 27, 26, 8, 0, 43, 11, 37, 6, 16, 15, 14, 42, 7, 4, 36, 45, 19, 33, 20, 18, 46, 34, 5, 41, 13, 48, 31, 32,
Iterate times: 8, Length: 8258.797564, Path: [19, 33, 20, 18, 46, 36, 45, 5, 14, 16, 15, 7, 6, 4, 34, 9, 37, 11, 3, 12, 23, 17, 22, 2, 24, 25, 10, 44, 1, 38, 21, 42, 4
Iterate times: 9, Length: 8304.243195, Path: [13, 30, 39, 29, 40, 32, 31, 48, 47, 35, 37, 11, 3, 9, 44, 1, 38, 21, 42, 14, 15, 16, 6, 7, 4, 5, 36, 45, 19, 33, 20, 18,
Iterate times: 10, Length: 7773.970067, Path: [10, 25, 24, 2, 22, 17, 23, 12, 44, 1, 38, 21, 34, 42, 14, 15, 16, 6, 7, 4, 5, 36, 45, 19, 33, 20, 18, 46, 30, 39, 29, 40
Iterate times: 11, Length: 8463.560106, Path: [46, 18, 20, 33, 19, 45, 36, 4, 6, 14, 15, 16, 37, 11, 43, 0, 8, 26, 27, 28, 41, 13, 30, 39, 29, 40, 32, 31, 48, 47, 35,
Iterate times: 12, Length: 8227.717825, Path: [17, 22, 25, 10, 24, 44, 2, 23, 12, 3, 11, 37, 9, 38, 1, 21, 34, 42, 14, 15, 16, 6, 4, 5, 33, 20, 19, 45, 36, 7, 18, 46,
Iterate times: 13, Length: 8166.232248, Path: [22, 17, 23, 12, 3, 44, 1, 38, 21, 42, 14, 15, 16, 6, 7, 5, 36, 19, 33, 20, 18, 46, 30, 39, 29, 48, 31, 40, 32, 35, 47, 3
Iterate times: 14, Length: 8479.310772, Path: [42, 14, 15, 16, 6, 7, 5, 36, 45, 19, 33, 20, 18, 46, 4, 34, 21, 38, 1, 44, 12, 2, 24, 25, 10, 22, 17, 23, 3, 9, 37, 11,
Iterate times: 15, Length: 8552.220712, Path: [16, 15, 14, 42, 34, 21, 38, 1, 44, 3, 9, 23, 12, 2, 10, 25, 24, 22, 17, 26, 8, 0, 43, 11, 37, 6, 7, 5, 36, 45, 19, 20, 3
Iterate times: 16, Length: 8089.121597, Path: [48, 47, 35, 32, 31, 40, 29, 39, 30, 46, 18, 20, 33, 19, 4, 7, 5, 36, 45, 34, 21, 38, 1, 44, 12, 23, 17, 22, 2, 10, 25, 2
Iterate times: 17, Length: 8120.817122, Path: [2, 44, 1, 38, 21, 42, 34, 10, 25, 24, 22, 17, 23, 12, 3, 9, 37, 11, 43, 0, 8, 26, 27, 28, 41, 13, 39, 29, 40, 31, 32, 35
Iterate times: 18, Length: 7475.427042, Path: [7, 6, 16, 15, 14, 42, 21, 34, 4, 5, 36, 45, 19, 20, 33, 18, 46, 30, 48, 47, 35, 32, 31, 40, 29, 39, 13, 41, 28, 27, 26,
Iterate times: 19, Length: 8511.123841, Path: [18, 20, 19, 45, 36, 5, 7, 6, 16, 15, 14, 42, 21, 38, 1, 44, 2, 12, 23, 17, 22, 25, 10, 24, 3, 9, 37, 11, 43, 0, 8, 26, 2
Iterate times: 20, Length: 7631.335104, Path: [6, 7, 4, 5, 36, 45, 19, 33, 20, 18, 46, 30, 39, 29, 40, 32, 31, 48, 47, 35, 34, 21, 38, 1, 44, 2, 24, 25, 10, 22, 17, 23
Iterate times: 21, Length: 8256.970584, Path: [20, 18, 33, 19, 45, 36, 5, 7, 6, 16, 15, 14, 42, 21, 38, 1, 44, 2, 10, 25, 24, 22, 17, 23, 12, 3, 9, 37, 11, 43, 0, 8, 2
Iterate times: 22, Length: 8061.914606, Path: [22, 17, 23, 12, 3, 9, 37, 11, 43, 0, 8, 26, 27, 28, 13, 41, 18, 20, 19, 33, 5, 7, 4, 34, 21, 42, 14, 15, 16, 6, 45, 36,
Iterate times: 23, Length: 7731.439881, Path: [4, 5, 7, 6, 16, 15, 14, 42, 34, 21, 38, 1, 44, 9, 3, 12, 23, 17, 22, 24, 25, 10, 2, 43, 0, 8, 26, 27, 28, 41, 13, 39, 29
Iterate times: 24, Length: 7991.251099, Path: [20, 33, 19, 45, 36, 5, 4, 7, 6, 16, 15, 14, 42, 21, 38, 1, 44, 2, 12, 23, 17, 22, 10, 25, 24, 3, 9, 37, 11, 43, 0, 8, 26
Iterate times: 25, Length: 7553.293463, Path: [23, 17, 22, 10, 25, 24, 2, 44, 1, 38, 21, 42, 14, 15, 16, 6, 4, 7, 5, 33, 19, 45, 36, 20, 18, 46, 30, 39, 29, 40, 32, 31
Iterate times: 26, Length: 7522.947587, Path: [41, 27, 28, 26, 8, 0, 43, 11, 37, 9, 3, 12, 23, 17, 22, 25, 24, 10, 2, 44, 1, 38, 21, 42, 14, 15, 16, 6, 4, 7, 5, 36, 45
Iterate times: 27, Length: 8425.133186, Path: [14, 15, 16, 6, 7, 5, 36, 45, 19, 33, 20, 18, 46, 30, 39, 29, 40, 32, 31, 48, 47, 35, 34, 42, 21, 38, 1, 44, 2, 12, 3, 9,
Iterate times: 28, Length: 8260.462296, Path: [14, 15, 16, 6, 7, 5, 4, 36, 45, 19, 33, 20, 18, 46, 34, 42, 21, 38, 1, 44, 3, 9, 37, 11, 17, 23, 12, 2, 22, 24, 25, 10,
Iterate times: 29, Length: 7558.832066, Path: [1, 38, 21, 34, 42, 14, 15, 16, 6, 7, 5, 4, 36, 45, 19, 33, 20, 18, 46, 30, 48, 31, 32, 35, 47, 40, 29, 39, 13, 41, 28, 2
Iterate times: 30, Length: 8001.050892, Path: [22, 2, 44, 1, 38, 21, 42, 34, 36, 5, 7, 6, 16, 15, 14, 4, 45, 19, 33, 20, 18, 46, 30, 39, 29, 40, 32, 35, 47, 48, 31, 13
Iterate times: 31, Length: 7906.237906, Path: [11, 37, 9, 3, 12, 23, 17, 22, 2, 44, 1, 38, 21, 34, 42, 14, 15, 16, 6, 4, 7, 5, 36, 45, 19, 33, 20, 18, 46, 30, 48, 47,
Iterate times: 32, Length: 8241.215245, Path: [38, 1, 44, 3, 9, 37, 11, 43, 0, 8, 26, 27, 28, 41, 13, 30, 39, 29, 40, 31, 32, 35, 47, 48, 21, 42, 14, 15, 16, 6, 4, 34,
Iterate times: 33, Length: 7988.561833, Path: [34, 21, 42, 14, 15, 16, 6, 7, 4, 5, 36, 45, 19, 33, 20, 18, 46, 30, 39, 29, 40, 32, 35, 47, 48, 31, 13, 41, 28, 27, 26,
Iterate times: 34, Length: 8186.084631, Path: [0, 43, 11, 37, 9, 3, 12, 23, 17, 22, 24, 25, 10, 2, 44, 1, 38, 21, 42, 14, 15, 16, 6, 7, 5, 4, 36, 45, 19, 33, 20, 18, 4
```

```
Iterate times: 35, Length: 7685.342741, Path: [42, 14, 15, 16, 6, 7, 5, 36, 45, 19, 20, 33, 18, 46, 30, 39, 29, 40, 32, 31, 48, 47, 35, 34, 21, 38, 1, 44, 2, 24, 10, 2
Iterate times: 36, Length: 8085.696825, Path: [9, 3, 11, 37, 43, 0, 8, 26, 27, 28, 41, 13, 30, 39, 29, 40, 32, 31, 48, 47, 35, 34, 46, 18, 20, 33, 19, 45, 36, 5, 7, 6,
Iterate times: 37, Length: 8452.472791, Path: [30, 39, 29, 40, 32, 31, 48, 47, 35, 34, 46, 45, 36, 5, 7, 6, 16, 15, 14, 42, 21, 38, 1, 44, 2, 10, 25, 24, 22, 17, 23, 1
Iterate times: 38, Length: 7906.791496, Path: [10, 25, 24, 12, 3, 9, 37, 11, 43, 0, 8, 26, 28, 27, 41, 13, 39, 29, 40, 32, 31, 48, 47, 35, 30, 46, 18, 20, 33, 19, 45,
Iterate times: 39, Length: 7652.290541, Path: [29, 40, 31, 32, 35, 47, 48, 30, 39, 13, 41, 28, 27, 26, 8, 0, 43, 11, 37, 9, 3, 12, 23, 17, 22, 24, 25, 10, 2, 44, 1, 38
Iterate times: 40, Length: 8024.656971, Path: [14, 15, 16, 6, 7, 5, 4, 36, 45, 19, 33, 20, 18, 46, 34, 42, 21, 38, 1, 44, 2, 12, 23, 17, 22, 10, 25, 24, 35, 47, 48, 31
Iterate times: 41, Length: 8126.525504, Path: [10, 25, 24, 21, 38, 1, 44, 2, 22, 17, 23, 12, 3, 9, 37, 11, 43, 0, 8, 26, 27, 28, 41, 13, 39, 29, 40, 32, 31, 48, 47, 35
Iterate times: 42, Length: 8221.523421, Path: [41, 28, 27, 26, 8, 0, 43, 11, 37, 9, 3, 4, 5, 7, 6, 16, 15, 14, 42, 1, 44, 2, 12, 23, 17, 22, 24, 25, 10, 38, 21, 34, 45
Iterate times: 43, Length: 7668.037825, Path: [36, 45, 19, 33, 20, 18, 46, 34, 4, 5, 7, 6, 16, 15, 14, 42, 21, 38, 1, 44, 2, 24, 25, 10, 22, 17, 23, 12, 3, 9, 37, 11,
Iterate times: 44, Length: 7846.114163, Path: [22, 17, 23, 12, 3, 9, 37, 11, 43, 0, 8, 26, 27, 28, 41, 45, 36, 4, 14, 15, 16, 6, 7, 5, 33, 19, 20, 18, 46, 30, 48, 47,
Iterate times: 45, Length: 7375.428425, Path: [30, 48, 47, 35, 32, 31, 40, 29, 39, 13, 41, 27, 28, 26, 8, 0, 43, 11, 37, 9, 3, 12, 23, 17, 22, 25, 10, 24, 2, 44, 1, 38
Iterate times: 46, Length: 7326.449647, Path: [34, 42, 14, 15, 16, 6, 7, 5, 36, 45, 19, 33, 20, 18, 46, 30, 48, 47, 35, 32, 31, 40, 29, 39, 13, 41, 28, 27, 26, 8, 0, 4
Iterate times: 47, Length: 8050.132784, Path: [23, 12, 2, 22, 17, 6, 7, 5, 4, 36, 45, 19, 33, 20, 18, 46, 30, 48, 47, 35, 32, 40, 31, 29, 39, 13, 41, 28, 27, 26, 8, 0,
Iterate times: 48, Length: 8030.798495, Path: [12, 23, 17, 22, 2, 44, 1, 38, 21, 34, 42, 14, 15, 16, 6, 7, 5, 4, 36, 45, 19, 33, 20, 18, 46, 30, 39, 29, 40, 31, 32, 35
Iterate times: 49, Length: 7619.196038, Path: [17, 23, 12, 3, 9, 37, 11, 43, 0, 8, 26, 28, 27, 41, 13, 39, 29, 40, 31, 32, 35, 47, 48, 30, 46, 18, 20, 36, 5, 33, 19, 4
Iterate times: 50, Length: 7926.352302, Path: [42, 14, 15, 16, 6, 7, 4, 5, 36, 45, 19, 20, 33, 18, 46, 30, 48, 47, 35, 32, 31, 40, 29, 39, 13, 41, 28, 27, 26, 8, 0, 43
The best Path is:  Iterate:46, Length: 7326.449647
```

得到的最佳的路径的长度为 7326.449647



整个迭代过程，一开始的时候，路径长度下降较快，在一定的迭代的次数之

后，则开始波动，根据得到的路径来看效果还是较为理想的。

当然，这只是在一般情况下出现的较好的结果，ACO 算法还是有不好的地方需要改进和思考的：

首先 ACO 算法在迭代搜索到一定的程度以后就容易出现停滞的状态，从而陷入局部最优的情况，并且 ACO 总体来说还是搜索的时间较长。

ACO 算法是否能够找到全局最优，还需要在一下几个方面做一些思考：

信息的散播方式，信息素的挥发的参数，以及是否需要在每一只蚂蚁自身也存在信息素挥发的机制。这是需要尝试和思考的。是否还可以模拟自然界的生物，给蚂蚁和环境一定的记忆能力，用这种启发式的信息来帮助减少搜索空间。