Guide of Python

内容概要： 数学建模算法

创建时间： 2022/4/8 12:41 **更新时间：** 2022/4/20 14:00

作者： **TwinkelStar**

ACO 蚁群算法

ACO Algorithm

1、操作系统相关环境

1）**硬件环境：**

- 电脑

2）**软件环境：**

- Python3.7(向下兼容 Python3)(程序设计语言)
- Numpy1.19.5(兼容大部分版本)(科学计算库)

3）**操作系统(2 选 1)：**

- Windows7
- Windows10
- Windows11

2、粒子群算法

蚁群算法是一种智能优化算法，通过蚁群优化求解复杂问题，ACO 在离散优化问题方面有比较好的优越性。蚁群算法是一种用来寻找优化路径的概率型算法。它由 Marco Dorigo 于 1992 年在他的博

士论文中提出，其灵感来源于蚂蚁在寻找食物过程中发现路径的行为。

1）基本原理

单只蚂蚁的行为及其简单，行为数量在10种以内，但成千上万只蚂蚁组成的蚁群却能拥有巨大的智慧，这离不开它们信息传递的方式——信息素。

蚂蚁在行走过程中会释放一种称为“信息素”的物质，用来标识自己的行走路径。在寻找食物的过程中，根据信息素的浓度选择行走的方向，并最终到达食物所在的地方。信息素会随着时间的推移而逐渐挥发。

在一开始的时候，由于地面上没有信息素，因此蚂蚁们的行走路径是随机的。蚂蚁们在行走的过程中会不断释放信息素，标识自己的行走路径。随着时间的推移，有若干只蚂蚁找到了食物，此时便存在若干条从洞穴到食物的路径。由于蚂蚁的行为轨迹是随机分布的，因此在单位时间内，短路径上的蚂蚁数量比长路径上的蚂蚁数量要多，从而蚂蚁留下的信息素浓度也就越高。这为后面的蚂蚁们提供了强有力的方向指引，越来越多的蚂蚁聚集到最短的路径上去，蚂蚁路径示意图如图1所示：

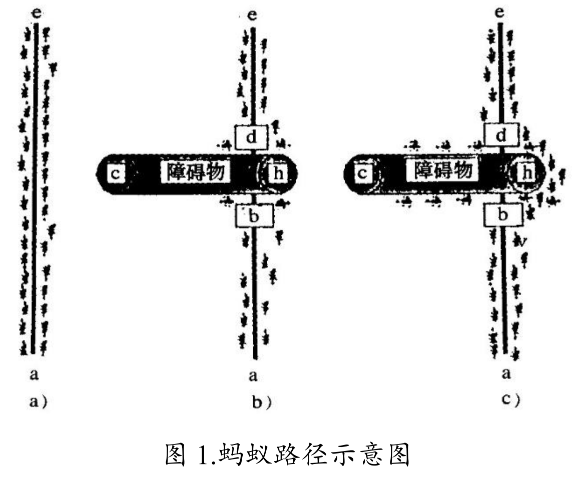

图 1.蚂蚁路径示意图

fig1. Ant path diagram

①高度结构化的组织——虽然蚂蚁的个体行为极其简单，但由个体组成的蚁群却构成高度结构化的社会组织，蚂蚁社会的成员有分工，有相互的通信和信息传递。

②自然优化——蚁群在觅食过程中，在没有任何提示下总能找到从蚁巢到食物源之间的最短路径；当经过的路线上出现障碍物时，还能迅速找到新的最优路径。

③信息正反馈——蚂蚁在寻找食物时，在其经过的路径上释放信息素（外激素）。蚂蚁基本没有视觉，但能在小范围内察觉同类散发的信息素的轨迹，由此来决定何去何从，并倾向于朝着信息素强度高的方向移动。

④自催化行为——某条路径上走过的蚂蚁越多，留下的信息素也越多（随时间蒸发一部分），后来蚂蚁选择该路径的概率也越高，如图 2 所示：

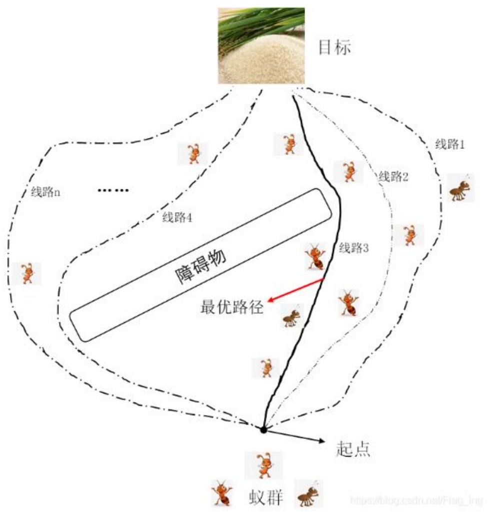

图 2.ACO 算法原理图

fig2.ACO Algorithm theory

2）算法思路

①根据具体问题设置多只蚂蚁，分头并行搜索。

②每只蚂蚁完成一次周游后，在行进的路上释放信息素，信息素量与解的质量成正比。

③蚂蚁路径的选择根据信息素强度大小（初始信息素量设为相等），同时考虑两点之间的距离，采用随机的局部搜索策略。这使得距离较短的边，其上的信息素量较大，后来的蚂蚁选择该边的概率也较大。

④每只蚂蚁只能走合法路线（经过每个城市 1 次且仅 1 次），为此设置禁忌表 p 来控制，如图 3 所示：

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}{\sum\limits_{s\in J_k(i)}[\tau_{is}(t)]^{\alpha}\cdot[\eta_{is}(t)]^{\beta}}=\dfrac{X}{Y}, & \text{如果}j\in J_k(i)\\ 0, & \text{否则}\end{cases}\qquad \eta_{ij}=\frac{1}{d_{ij}}$$

图 3.状态转移矩阵更新公式

fig3. Update formula of state transition matrix

⑤所有蚂蚁都搜索完一次就是迭代一次，每迭代一次就对所有的边做一次信息素更新，原来的蚂蚁死掉，新的蚂蚁进行新一轮搜索。

⑥更新信息素包括原有信息素的蒸发和经过的路径上信息素的增加。更新公式如式（1）所示：

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

⑦达到预定的迭代步数，或出现停滞现象（所有蚂蚁都选择同样的路径，解不再变化），则算法结束，以当前最优解作为问题的最优解。

3、例题与程序设计

1）例题

求解下列函数的最小值：

$$f(\mathrm{x}) = -x_1^4 + x_2^4 - \cos(3x_1) - 0.4 \times \cos(4 \times x_2) + 0.6 \tag{2}$$

2）算法步骤：

算法流程图如图 4 所示：

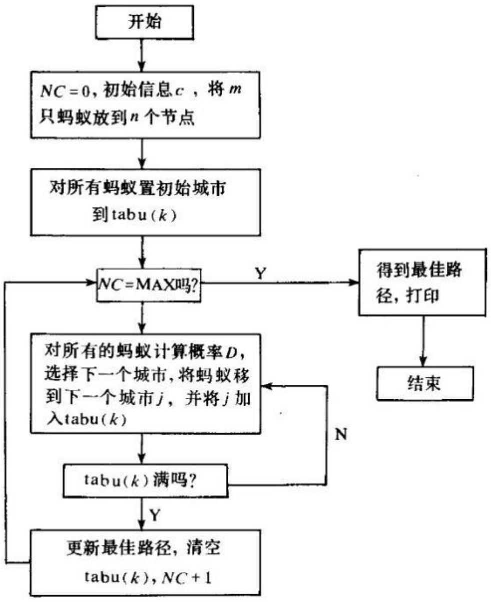

图 4.ACO 算法流程图

fig4. ACO algorithm flow chart

3）Python 代码：

在程序编写之前，对符号做出约束说明，如表 1 所示：

表 1.符号描述图表
table1.Symbol Description

Symbol	Description
Ant	蚂蚁的数量
Times	蚂蚁的移动次数
Rou	信息数的挥发系数
P0	转移概率常数

图 5.ACO 算法实现

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv

Ant=100 # 蚂蚁的数量
Times=505 # 蚂蚁的移动次数
Rou=0.9 # 信息数的挥发系数
p0=0.2 # 转移概率常数
Lower_1=-1 # 设置搜索范围
Upper_1=1
Lower_2=-1
Upper_2=1

x = np.zeros((Ant,2))

for i in range(Ant):
    x[i, 0] = (Lower_1 + (Upper_1 - Lower_1) * np.random.randn(1)[0])
    x[i, 1] = (Lower_2 + (Upper_2 - Lower_2) * np.random.randn(1)[0])

step = 0.05

def F(x1, x2):
    """
    函数值的计算

    Parameters
    ----------
    x : TYPE
        蚁群的坐标.

    Returns
    -------
    f : TYPE
        返回函数值.

    """
    # x1 = x[:,0]
    # x2 = x[:,1]
    f = -(x1**4 + x2**4 - np.cos(3*np.pi*x1) - 0.4*np.cos(4*np.pi*x2) + 0.6)
    f = np.reshape(f,(1,-1))#以行的形式输出
    return f

f = F(x[:, 0], x[:, 1])

tau_Best = np.zeros((1,Times))
bestIndex = [0]
p = np.zeros((Times,Ant))
xx = []
for T in range(Times):
    lamda = 1 / (T+1)
    tau_Best[0,T] = np.max(f)
    bestIndex[0] = np.argmax(f)

    for i in range(Ant):
        p[T, i] = (f[0, bestIndex[0]] - f[0,i]) / f[0, bestIndex[0]]

    for i in range(Ant):
        if p[T, i] < p0:
            temp1 = x[i, 0] + (2 * np.random.randn(1)[0] - 1) * lamda
            temp2 = x[i, 1] + (2 * np.random.randn(1)[0] - 1) * lamda
        else:
            temp1 = x[i, 0] + (Upper_1 - Lower_1) - (np.random.randn(1)[0] * 0.5)
            temp2 = x[i, 1] + (Upper_2 - Lower_2) - (np.random.randn(1)[0] * 0.5)

        if temp1 < Lower_1:
            temp1 = Lower_1

        if temp1 > Upper_1:
            temp1 = Upper_1

        if temp2 < Lower_2:
            temp2 = Lower_2

        if temp2 > Upper_2:
            temp2 = Upper_2

        if F(temp1,temp2)[0][0] > F(x[i, 0], x[i, 1])[0][0]:
            x[i, 0] = temp1
            x[i, 1] = temp2
    for i in range(Ant):
        f[0, i] = (1 - Rou) * f[0, i] + F(x[i, 0], x[i, 1])[0][0]
    print(np.max(f))
```

图 5.ACO 算法实现

fig5. ACO algorithm implementation

4）函数调用方式：

直接运行代码，运行结果如图 6 所示：

图 6.ACO 算法结果

fig5. ACO algorithm result

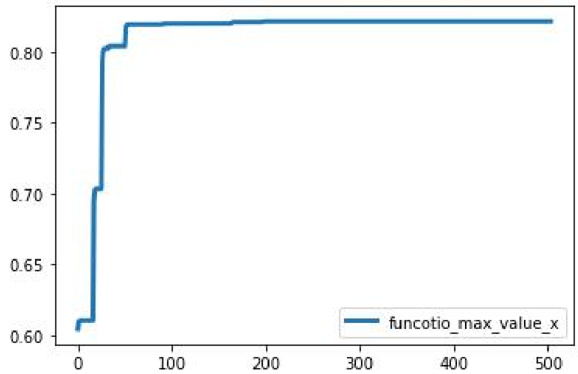

图 7.ACO 算法结果可视化

fig7. ACO algorithm result chart

经过程序计算可知，目标函数的最大值为 0.8215284206082081。当出现停滞现象的时候，如图 7 所示，在 100 次迭代之后最大值收敛时，说明已经得到最优解，算法结束。