FCM 聚类处理的蚁群算法优化旅行商问题

摘要：针对大规模的旅行商（TSP）问题时，可将 FCM 聚类法与蚁群或其改进算法，如精华蚂蚁系统相结合：将大规模的城市数据通过聚类，分解成多个子问题，对每个子问题独立的使用精华蚂蚁系统算法，得出每个子问题的最优解，将其连接组合之后，即为整个问题的最优解。在使用 FCM-EAS 混合算法处理大规模的旅行商问题时，能较为明显的降低时间复杂度，用一定的精确度换取更快运行速度，理论分析与仿真实验也验证了这一结论。

关键词：大规模；TSP；FCM 聚类；蚁群算法；

1 引言：（Introduction）

TSP，即旅行商问题，源自实际的运输和物流问题。该问题的主要内容是：一个旅行商需要从一个城市出发，经过一系列其他城市，每个城市只能经过一次，最后回到出发城市，旅行商希望总的旅行距离尽可能短。这个问题在运筹学和理论计算机科学中被广泛研究。

虽然 TSP 问题在定义上很简单，但它（在其最常见的对称形式中）被证明是 NP-hard 的，这意味着没有已知的多项式时间复杂度的精确解法。实际上，由于其计算复杂性，已知的 TSP 实例的最大已解决规模大约只有几千个城市。

求解 TSP 的方法可以大致分为两类：精确求解法和启发式算法。

精确求解法：这些算法可以得到问题的最优解，但计算复杂度通常非常高。像分支定界法、切割平面法和动态规划法都是精确求解法的例子。例如，分支定界法的思路是分解问题规模，通过不断界定问题的上下界，逐步减小求解范围。

启发式算法：这些算法主要用于求解规模较大的 TSP。它们通过某种启发式或元启发式来生成优良的解，虽然这些解可能不是最优的，但是在有限的时间内可以得到满意的结果。如模拟退火、遗传算法、粒子群算法等都是优秀的启发式算法。

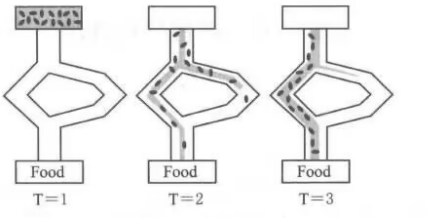
本文将要采用改进的启发式算法——蚁群算法更是求解 TSP 等组合优化问题的重要工具之一。然而，传统的蚁群算法在求解大规模的 TSP 问题时，求解的速度较慢，容易陷入局部最优解，求出解的精度不高。在蚁群算法的基础上，出现了一些变种和改进算法，如精华蚂蚁系统（Elitist Ant System）、最大最小蚂蚁系统（Max-Min Ant System）等，虽然在处理 TSP 问题有所提升，但提升有限。在这样的背景下，本文提出了在聚类

处理之后，使用精华蚂蚁系统独立求解每一子 TSP 问题，最后将各类最优解组合起来得到全局解的 FCM-EAS 混合算法。

2 传统算法介绍

2.1 蚁群算法（AS）

蚁群算法是一种基于正反馈的仿生优化算法，通过模拟蚂蚁觅食的行为来解决优化问题。



蚂蚁在运动过程中，能够在它所经过的路径上留下一种称之为外激素的物质进行信息传递，而且蚂蚁在运动过程中能够感知这种物质，并以此指导自己的运动方向，因此由大量蚂蚁组成的蚁群集体行为便表现出一种信息正反馈现象：某一路径上走过的蚂蚁越多，则后来者选择该路径的概率就越大。

蚁群算法中，最重要的两个步骤就是路径的选择和信息素的更新。

蚂蚁随机放置在一个城市中，按照一个的规则选择下一个将要访问的城市，随机概率的公式为：

$$P_k(i,j)=\begin{cases}\dfrac{[\tau(i,j)]^{\alpha}\bullet[\eta(i,j)]^{\beta}}{\sum_{u\in J_k(i)}[\tau(i,j)]^{\alpha}\bullet[\eta(i,j)]^{\beta}}, & if \quad j\in J_k(i)\\ 0 \quad , & otherwise\end{cases}$$

其中 $\tau(i,j)$ 表示边 (i,j) 上的信息素量，$\eta(i,j)$ 是启发因子，取 (i,j) 距离的倒数。长度越短，信息素浓度越大的路径，蚂蚁选择的概率就越大。α,β 为权重因子，分别用来控制信息素浓度和启发式信息的权重关系。

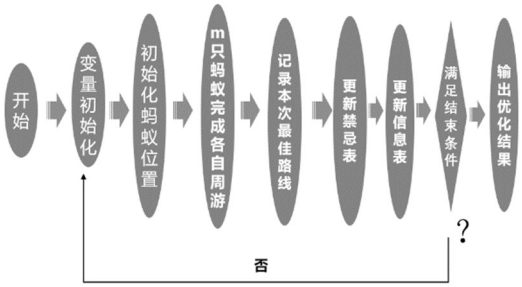
信息素更新也有两个步骤，一是所有路径上的信息素都会蒸发，我们可以设置一个蒸发系数 ρ，其次，每只蚂蚁在本轮自己走过的路径上，均会释放信息素，蚂蚁走过的路径越短，释放的信息素就越多。信息素的更新公式为：

$$\tau(i,j)=(1-\rho)\tau(i,j)+\sum_{k=1}^{m}\Delta\tau_k(i,j)$$

$$\Delta\tau_k(i,j)=\begin{cases}1/C_k \quad , & if(i,j)\in R^k\\ 0 \quad , & otherwise\end{cases}$$

其中，m 为蚂蚁的数量，C_k 为路径的长度，是 R^k 中所有边的长度之和。

算法流程图如下：



2.2 精华蚂蚁系统（EAS）

精华蚂蚁系统（Elitist Ant System）是在普通蚂蚁系统的基础上加以改进，其主要改进方式为更改信息素的迭代方式。

在此次迭代中，若有蚂蚁所走的路径为当次最短路径 T^{bs} 时，为增强其正反馈的效果，人工释放额外的信息素 e。信息素修改公式为：

$$\Delta\tau_{ij}=\sum_{k=1}^{m}\Delta\tau_{ij}^{k}+e\Delta\tau_{ij}^{bs}$$

$$\Delta\tau_{ij}^{bs}=\begin{cases}1/L_{bs}\ , & (i,j)\in T^{bs}\\ 0\ , & other\end{cases}$$

L_{bs} 即为最优路径 T^{bs} 的长度。

2.3 模糊聚类（FCM）

模糊 C 均值聚类算法（Fuzzy C-Means，简称 FCM）是一种常用的模糊聚类算法，用于将一组数据点划分为若干个模糊的聚类。

在传统的 K 均值聚类算法中，每个数据点只能属于一个聚类，而在 FCM 中，每个数据点可以以一定的隶属度（membership degree）属于多个聚类，表示其与每个聚类的关联程度。

初始化过程中，先选定聚类的数量 K，再设置好模糊因子 m（通常取大于 1 的数，例如 2），最后设置终止条件，如到达最大迭代次数。

在迭代过程中，对于每个数据点 i 和个聚类中心 j，计算隶属度矩阵 U，表示数据点 i 属于聚类中心 j 的程度

$$u_{ij}=1\Big/\sum_{r=1}^{c}\left(\frac{d_{ij}(k)}{d_{rj}(k)}\right)^{\frac{2}{m-1}}$$

根据隶属度矩阵 U，更新聚类中心向量。对于每个聚类中心 j，计算其更新后的聚类中心 $v(k+1)$

$$v(k+1)=\sum_{j=1}^{n}u_{ij}^{m}(k)x_{j}\Big/\sum_{j=1}^{n}u_{ij}^{m}(k)$$

重复更新隶属度矩阵和聚类中心，直到满足聚类中心的变化小于某个阈值或者达到最大迭代次数。

根据最终的聚类中心向量，将数据点分配到相应的聚类中心，形成聚类结果。FCM 算法通过引入隶属度矩阵和模糊因子，使得每个数据点可以部分属于多个聚类中心，而不是严格划分到一个聚类中心。这种模糊性使得 FCM 算法对于某些数据集更加适用，尤其是在存在数据点边界模糊或重叠的情况下，表现情况往往更优。

FCM 的模糊因子 m 取值情况一般为 $[1,\infty)$，m 的不同取值，对 FCM 算法的影响较大，如

(1). $m\to 1^{+}$ 时，FCM 算法会退化为 HCM 算法

(2). $m=1$ 时，FCM 算法即为 HCM 算法（K-means 算法）

(3). $m\to\infty$ 时，FCM 算法的聚类结果是最模糊的

但是本篇文章并非是详细的研究 FCM 聚类算法的各项指标性能，仅仅用为对城市进行分类的工具，故具体分析不在赘述，读者若感兴趣可仔细查阅相关资料。

3 基于 FCM 聚类的蚁群改进算法

在求解大规模城市的 TSP 问题时，传统蚁群算法运行速度慢，收敛性较差，容易陷入局部最优解。因此，这里将一个大规模的城市，通过 FCM 聚类分析，转化为 K 个小

规模的子 TSP 问题，对于每个子 TSP 问题，独立的使用蚁群算法，求出每个子 TSP 问题的最优或较优路径，通过一定的规则，把每个子 TSP 问题的路径相连组合，进而得到全局最优或较优解。同时，对于每个子 TSP 问题进行求解时，可进一步使用蚁群算法的改进算法，本文采用精华蚂蚁系统算法进行求解。

在传统的蚁群算法中，设外层的最大迭代次数为 $\max\_count$，蚂蚁的数量为 m，城市的数量为 n，在每次迭代中，需要进行 m 次蚂蚁的路径选择和更新信息素的操作，路径选择的时间复杂度为 $O(m\cdot n^2)$，更新信息素的操作时间复杂度为 $O(m\cdot n^2)$，故传统蚁群算法的总时间复杂度为

$$O_1 = O\left(\max\_count\cdot m\cdot n\right)$$

基于 FCM 聚类的蚁群改进算法中，迭代次数，蚂蚁数量，城市数量，均与传统算法一致，但在分为 k 类之后，假设每类子 TSP 问题的蚂蚁数量和城市数量取分类均值，即分别取 m/k，n/k，则该问题的时间复杂度为：

$$O_2 = O\left(k\cdot\max\_count\cdot\frac{m}{k}\cdot\left(\frac{n}{k}\right)^2\right)$$
$$= O\left(\max\_count\cdot m\cdot n/k^2\right)$$

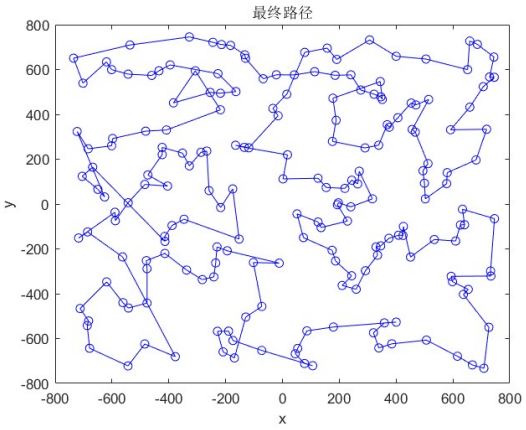
可以看出传统算法的时间复杂度相对于基于 FCM 聚类的蚁群改进算法的时间复杂度差了 k^2 倍，可以做出猜测：改进后的算法能明显缩短处理大规模 TSP 问题的时间。

4　仿真实验与结果分析

我们使用 MATLAB R2021 先后随机生成 30 个，100 个，197 个城市坐标，分别存入相对应的 txt 文件中，横纵坐标范围均为 $[-750,750]$。

城市的数量可通过读取上述文件所得，为使得城市能被蚂蚁随机的分布，设置蚂蚁的数量为城市的 5/3 倍，$\alpha=2$，$\beta=1$，蒸发系数 $\rho=0.1$，迭代次数设置为 200。

通过运行传统蚁群算法（AS）之后,可以得到如下路径图：



迭代 200 次后，可以看到，所得的路径图较为无序，有较多交叉、反复，与期望最优解不符。

下面将传统蚁群算法（AS），精华蚂蚁系统（EAS），基于 FCM 聚类的蚁群改进算法（FCM-EAS）三种算法进行比较，绘制出如下表格并做出分析：

从表格中可以看出，当城市的坐标为 30

时，三种算法的运行速度均很快，AS 和 EAS 均在 10 秒左右找出了最短路径。

当问题的规模增大到 100 时，三种算法的运行时间便有了较大的差异，最慢的 AS 用时 48.59 秒，最快的 FCM-EAS 用时仅为 13.63 秒，虽然所得的最短距离较 AS 和 EAS 得出的结果有所偏大，但牺牲一定的精确度换取运行时间是可以接受的

当问题的规模增大到 197 时，传统 AS 算法与 EAS 和 FCM-EAS 有了非常大的差异，最慢的运行时间与最快运行时间相差三倍多，但得出的最短路径长度相差不大。与前文做出的猜测吻合。

城市数量	算法	聚类 K	路径长度	运行时间/s
30	AS	5	6846.94	10.25
	EAS		6846.94	9.25
	FCM-EAS		6664.86	0.97
100	AS	5	11902.33	48.59
	EAS		12044.22	32.37
	FCM-EAS		12505.47	13.63
197	AS	5	17921.20	208.59
	EAS		18197.13	81.43
	FCM-EAS		17725.90	62.67

注： 由于作者并未找到好的方法，将聚类处理后的各子 TSP 问题的最优路径连接起来，故这里取各子问题最短路径的和来近似看做整个问题的最短路径长度

5 评价与不足

在选择聚类的 K 值时，是由自己直接给出，可使用循环遍历 $k \in \left[2, \sqrt{n}\right]$，得出各自的轮廓系数，选取轮廓系数最高相对应的 k 值作为聚类的数量。但需要注意的是，遍历求取轮廓系数的时间复杂度较高。

在问题规模极大时，可尝试考虑使用 K-means 聚类法替代 FCM 聚类法。

本文在计算 FCM-EAS 的最短路径时，只是简单的把各个路径长度相加视为最短路径。可制定规则，当把聚类之后的各个最短路径连接起来时，可将各个类中心作为中心城市，对其再次使用蚁群算法获取其连接顺序，由于聚类之后的中心城市数量不太多，此项步骤或可以用普通的遍历所替代。得到中心城市的连接顺序之后，可遍历相邻两个类中各个点之间的距离，选取最短的将其连接起来，同时原类中断开的另一点，可向此类的另一相邻类做遍历，连接其最短路径，如此往复直到形成完整回路，即视为最优路径。

6 结语(Conclusions)

本文通过传统算法 FCM 聚类法和精华蚂蚁系统（EAS），构建出一个适用于解决大规模 TSP 问题的 FCM-EAS 混合算法。该算法原理较为简单，容易实现，在处理问题时，与传统算法相比，较大的缩短了运行时间,提高运行的效率。理论分析和仿真部分的结果也验证了这个结论。

参考文献（References）：

[1]. 刘新宇, 谭力铭, 杨春曦, 等. 未知环境下的蚁群-聚类自适应动态路径规划[J]. 计算机科学与探索, 2019, 13: 846.

[2]. 陈永胜. 基于 k-means 聚类与蚁群算法的物流配送路径优化[R]. 2022.

[3]. 张硕航, 郭改枝, 张朋. K-means 聚类下的改进蚁群算法优化 TSP 问题[R]. 2021.

[4]. 杨燕, 王全根, 黄波. 蚁群聚类算法的并行化设计与实现[J]. 控制工程, 2013, 20: 411–414.

[5]. 莫锦萍, 陈琴, 马琳, 等. 一种新的 K—Means 蚁群聚类算法[J]. 广西科学院学报, 2008, 24: 284–286.

[6]. 孙吉贵 [1, 刘杰 [1, 赵连宇 [1. 聚类算法研究[J]. 软件学报, Citeseer, 2008, 19: 48–61.

[7]. 刘芳, 李义杰. 改进的种群分类蚁群算法及其应用 ①[J]. 计算机系统应用, 2010, 19.

附录：

MATLAB 代码：

fcm_EAS.m

```
%% 导入城市坐标
load citys_30.txt
citys=citys_30;

%% 聚类处理
k = 5; % 设置聚类的群组数量
[idx, centroids] = f_fcm(citys,k,2); % 使用 fcm 聚类算法
%[idx, centroids] = Kmeans(citys,k); % 使用 kmeans 算法
%% 初始化
n = size(citys, 1); % 城市数量
m = ceil(5*n/3);    % 蚂蚁数量
max_count = 200;    % 最大迭代次数
Short_length=zeros(k,1);
Short_path=cell(k,1);
cov=zeros(k,max_count);

%% 根据群组数量初始化相关变量
D = cell(k, 1);     % 存储每个群组内城市之间的距离
v = cell(k, 1);     % 存储每个群组内的启发因子
start = cell(k, 1); % 存储每个群组内蚂蚁的初始位置
Tabu = cell(k, 1);  % 存储每个群组内蚂蚁的禁忌表
Tao = cell(k, 1);   % 存储每个群组内的信息素浓度

%% 迭代寻优
for class = 1:k
    % 获取当前群组的城市数据和索引
    class_cities = citys(idx == class, :);
    class_idx = find(idx == class);
    class_n = length(class_idx);

    % 计算当前群组内城市之间的距离
    class_D = zeros(class_n);
    for i = 1:class_n
        for j = 1:class_n
            if i == j
                class_D(i, j) = eps;
```

```
            else
                class_D(i, j) = norm(class_cities(i, :) -
class_cities(j, :), 2);
            end
        end
    end
    D{class} = class_D;

    % 计算当前群组内的启发因子
    class_v = 1 ./ class_D;
    v{class} = class_v;

    % 初始化当前群组的蚂蚁初始位置、禁忌表和信息素浓度
    class_start = randi(class_n, m, 1);
    start{class} = class_start;
    Tabu{class} = zeros(m, class_n);
    Tao{class} = ones(class_n);
end
    % 在每个群组内独立运行蚁群算法
 for class = 1:k
        class_cities = citys(idx == class, :);
        class_idx = find(idx == class);
        class_n = length(class_idx);
        class_D = D{class};
        class_v = v{class};
        class_start = start{class};
        class_Tabu = Tabu{class};
        class_Tao = Tao{class};

        % 每个群组内的蚁群算法
        for i = 1:m
            % 初始化蚂蚁位置和禁忌表
            curr_city = class_start(i);
            class_Tabu(i, 1) = curr_city;
        end

        % 调用 EAS 算法
       [shortestRoute, shortestLength,cov(class,:)] = ...
       f_EAS(class_cities, m, max_count,class_D,class_v,class_Tabu);
       Short_length(class)=shortestLength;
       Short_path{class}= shortestRoute;
 end

sum(Short_length)
```

f_fcm.m

```
function [idx, centers] = f_fcm(data,C,m)
    max_count=200;
    N = size(data,1);
    U = rand(N,C);
    U = U ./ sum(U,2);
    for iter = 1:max_count
        centers = (U .^ m)' * data ./ sum(U .^ m)';
        distances = pdist2(data,centers);
        U_new = 1 ./ distances .^ (2 / (m - 1));
        U_new = U_new ./ sum(U_new,2);
        if norm(U_new-U) < 1e-6
            break;
        end
        U = U_new;
    end
    [~, idx] = max(U, [], 2); % 获取每个数据点的最大隶属度的类别
end
```

f_EAS.m(信息素更新部分)

```
        pos=find(L==min(L));
        % 更新信息素
        dt = zeros(n);
        e = 30;
        for k = 1:m
            for j = 1:n-1
                if ismember(k, pos)
                    dt(Tabu(k,j), Tabu(k,j+1)) = dt(Tabu(k,j),
Tabu(k,j+1)) + e * Q / L(k);
                end
                dt(Tabu(k,j), Tabu(k,j+1)) = dt(Tabu(k,j), Tabu(k,j+1)) +
Q / L(k);
            end
            dt(Tabu(k,n), Tabu(k,1)) = dt(Tabu(k,n), Tabu(k,1)) + Q /
L(k);
        end
        Tao = (1 - rho) * Tao + dt
```