
# 东北大学秦皇岛分校

# 计算机与通信工程学院

# 数据结构课程设计

设计题目 蚁群算法在旅行商问题中的应用

| | |
|---|---|
| 专业名称 | 计算机科学与技术 |
| 班级学号 | 计科 1803　20188117 |
| 学生姓名 | **项溢馨** |
| 指导教师 | 刘杰民 |
| 设计时间 | **2019 年 12 月 30 日—2020 年 1 月 5 日** |

# 课程设计任务书

专业：计算机科学与技术 学号：20188117 学生姓名（签名）：

## 设计题目：蚁群算法在旅行商问题中的应用

## 一、 设计实验条件

自主完成

## 二、设计任务及要求

1. 应用蚁群算法求解 TSP 问题；
2. TSP 中的城市数量不少于 30 个，组成完全图，边上的权值自定；
3. 蚂蚁数量可配置，迭代次数可配置；
4. 给出较全面的实验结果：结果路经及长度；蚁群算法执行时间；不同参数值（蚂蚁数量，迭代次数）的影响等；

## 三、设计报告的内容

**1. 设计题目与设计任务（设计任务书）**
**2. 前言（绪论）(设计的目的、意义等)**
**3. 设计主体（各部分设计内容、分析、结论等）**

**3.1 需求分析**

以无歧义的陈述说明程序设计的任务，强调的是程序要做什么？给出功能模块图和流程图。同时明确规定：输入的形式和输出值的范围；输出的形式；程序所能够达到的功能；测试数据：包括正确的输入及其输出结果和含有错误的输入及其输出结果。

**3.2 系统设计**

说明本程序中所有用到的数据及其数据结构的定义，包含基本操作及其伪码算法。画出函数之间的调用关系图；写出主程序及其主要模块的伪码流程。

**3.3 系统实现**

给出算法的实现；程序调试过程中遇到的问题是如何解决的；对设计与实现的回顾和分析；算法的时空分析和改进思想。

**3.4 用户手册**

说明任何使用你编写的程序，详细列出每一步的操作步骤。

**3.5 测试**

给出测试过程及结果。

**4. 结束语（设计的成果，展望等）**

**5. 参考资料**

**6. 附录**

带注释的源程序。

## 四、设计时间与安排

**1、设计时间：1 周**

**2、设计时间安排：**

熟悉实验设备、收集资料：2 天

设计图纸、实验、计算、程序编写调试：1 天

编写课程设计报告：1 天

答辩：1 天

# 课程设计报告

## 前言

蚁群算法是由 M.Dorigo 等人于 20 世纪 90 年代初提出的一种智能算法。M.Dorigo 等人将其运用于求解 TSP，取得了很好的效果。

蚂蚁在寻找食物过程中通过彼此留在路径上的信息素进行交流合作，进而发现一条最短觅食路径。以图 1 为例说明蚁群算法原理。如图 1(a)所示，巢穴内有两只蚂蚁，以黄色圆与绿色圆表示。两只蚂蚁选择了两条路线，如图 1(b)所示，假设它们的速度相等，那么经过 t 时间后，黄色蚂蚁找到食物，而绿色蚂蚁继续寻找；又经过 t 时间，黄色蚂蚁返回巢穴，绿色蚂蚁依旧没有到达食物位置，如图 1(c)所示。3t 时间后，绿色蚂蚁找到食物，此时黄色蚂蚁已第二次到达，如图 1(d)所示。此时两条路径上的信息素比值为 3:1，继续进行，越来越多的蚂蚁选择走黄色蚂蚁的觅食路径，此条路线上信息素不断增加。最终所有蚂蚁选择黄色蚂蚁的觅食路径，如图 1(e)所示。

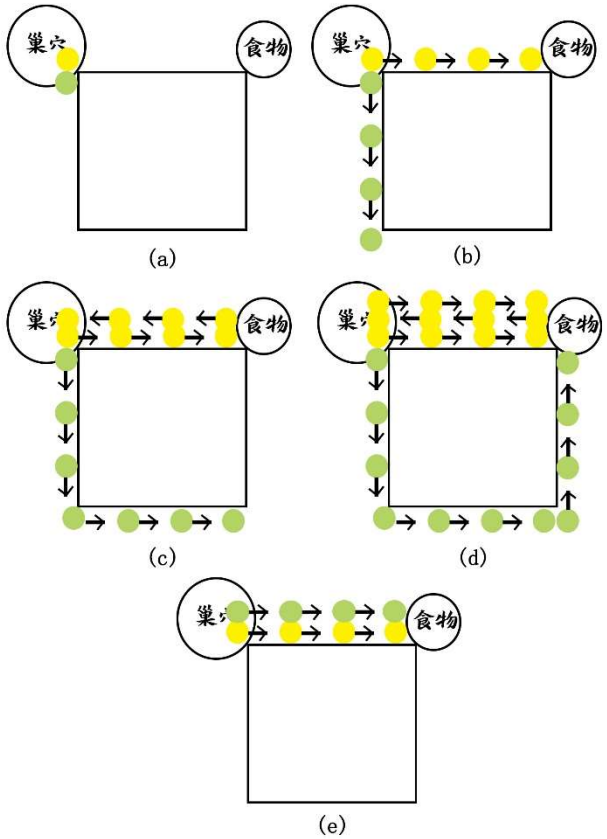


图 1 蚁群算法原理

在用蚁群算法求解优化问题时，每条蚂蚁的觅食路径就表示相应优化问题的

一个解，所有蚂蚁的觅食路径构成优化问题的解空间，其中最优路径就是最优化问题的最优解。蚁群算法求解优化问题基本流程如图 2 所示：

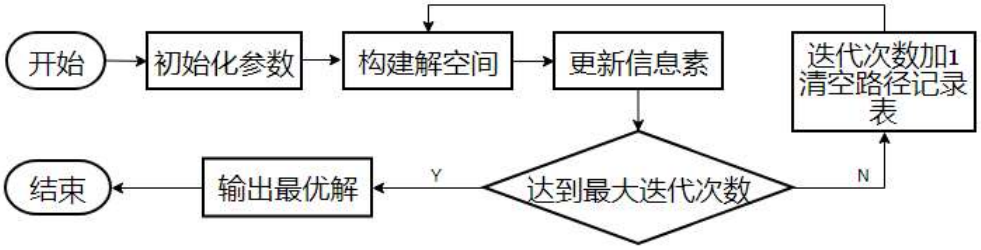


图 2 蚁群算法流程图

旅行商问题是指已知 N 座城市以及任意两座城市之间的距离，任意选择一个城市作为起点，从该城市出发，逐渐到达其余 N-1 个城市后回到出发城市，N-1 个城市只能到达一次，寻找一条最短路径。

旅行商问题是简单易描述，但是要精确求解却非常困难，学术界公认其为 NP-Hard 问题。对于旅行商问题，所有可行路径共(N-1)!/2 条。对于现在每秒可执行 150 亿次浮点的计算机来说，当 N=10 时需要 1.27E-5s 就能找到最优解；当 N=20 时需要 47d 才能找到最优解；而当 N=30 时需要 9.3E12 a 才能找到最优解。由此可见，求解对于复杂大规模旅行商问题，枚举法是不可取的。

运用蚁群算法求解著名的旅行商问题，从实验上探索了优化能力，获得了满意的效果。

# 设计主体

## 3.1 需求分析

基本蚁群算法求解旅行商问题的流程图如下：

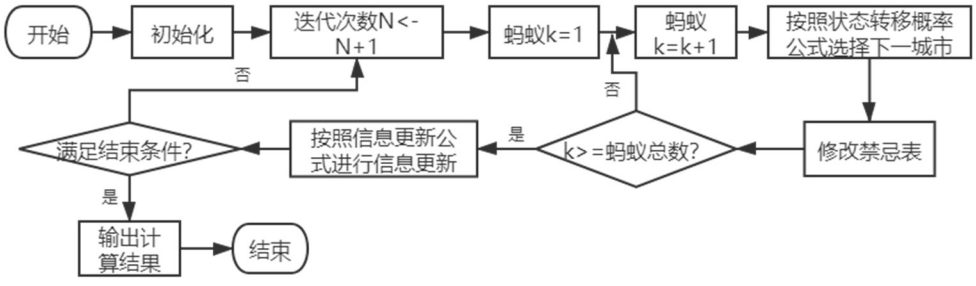


图 3 蚁群算法求解旅行商问题流程图

给出 30 个城市构成的完全图，部分输入数据如下：

| | | | | | |
|---|---|---|---|---|---|
| 1 2 1 | 1 24 10 | 2 18 21 | 3 13 13 | 4 9 34 | 5 6 15 |
| 1 3 38 | 1 25 30 | 2 19 50 | 3 14 30 | 4 10 21 | 5 7 38 |
| 1 4 48 | 1 26 22 | 2 20 34 | 3 15 46 | 4 11 10 | 5 8 24 |
| 1 5 49 | 1 27 30 | 2 21 17 | 3 16 4 | 4 12 10 | 5 9 11 |
| 1 6 25 | 1 28 2 | 2 22 22 | 3 17 33 | 4 13 7 | 5 10 9 |
| 1 7 15 | 1 29 46 | 2 23 33 | 3 18 27 | 4 14 4 | 5 11 42 |
| 1 8 10 | 1 30 28 | 2 24 13 | 3 19 47 | 4 15 31 | 5 12 47 |
| 1 9 46 | 2 3 17 | 2 25 21 | 3 20 47 | 4 16 26 | 5 13 50 |
| 1 10 28 | 2 4 36 | 2 26 2 | 3 21 19 | 4 17 1 | 5 14 8 |
| 1 11 34 | 2 5 44 | 2 27 39 | 3 22 28 | 4 18 36 | 5 15 33 |
| 1 12 24 | 2 6 36 | 2 28 11 | 3 23 14 | 4 19 36 | 5 16 27 |
| 1 13 49 | 2 7 47 | 2 29 12 | 3 24 42 | 4 20 47 | 5 17 32 |
| 1 14 38 | 2 8 39 | 2 30 13 | 3 25 40 | 4 21 43 | 5 18 11 |
| 1 15 1 | 2 9 49 | 3 4 45 | 3 26 17 | 4 22 12 | 5 19 48 |
| 1 16 24 | 2 10 8 | 3 5 5 | 3 27 11 | 4 23 10 | 5 20 2 |
| 1 17 45 | 2 11 6 | 3 6 22 | 3 28 17 | 4 24 10 | 5 21 10 |
| 1 18 50 | 2 12 23 | 3 7 19 | 3 29 12 | 4 25 20 | 5 22 3 |
| 1 19 3 | 2 13 5 | 3 8 5 | 3 30 36 | 4 26 18 | 5 23 16 |
| 1 20 25 | 2 14 44 | 3 9 8 | 4 5 4 | 4 27 44 | 5 24 29 |
| 1 21 30 | 2 15 49 | 3 10 13 | 4 6 32 | 4 28 48 | 5 25 21 |
| 1 22 8 | 2 16 41 | 3 11 29 | 4 7 17 | 4 29 15 | 5 26 14 |
| 1 23 11 | 2 17 48 | 3 12 7 | 4 8 13 | 4 30 30 | 5 27 3 |

图 4 完全图数据

为保证完全图的无序性与无规律性，数据采用随机数生成程序生成。输入数据每一行数据前两个代表无向图两个顶点，第三个数据为边权。为简化计算过程，边权数据范围为[1,50)，且为整数。

每次迭代过程输出数据第一行为求解的路径总长度最小值，第二行为算法执行时间大小，第三行输出按顺序经历的城市编号。

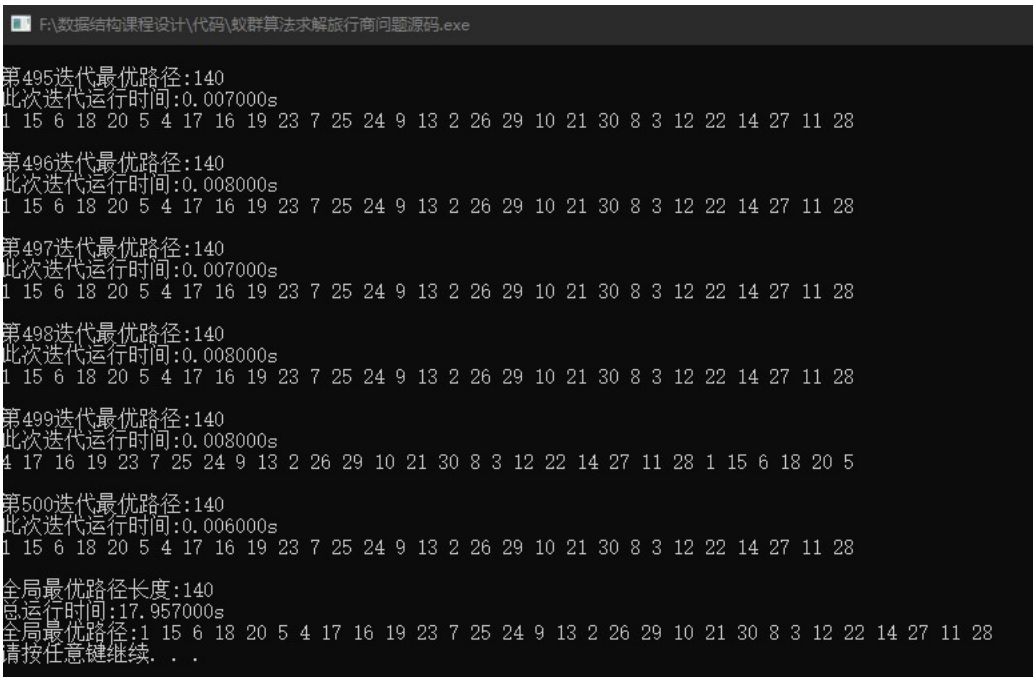


图 5 输出结果示例

### 3.2 系统设计

```
class AntColonySystem
{
private:
        double info[N][N], visible[N][N];
public:
        AntColonySystem(){}

        //计算当前节点到下一节点转移的概率
        double Transition(int i, int j);

        //根据式4局部更新
        void UpdateLocalPathRule(int i, int j);

        //初始化
        void InitParameter(double value);

        //全局信息素更新
        void UpdateGlobalPathRule(int*bestTour,int globalBestLength);
};
```

图 6 蚁群系统定义

蚁群系统包含两个 private 定义：info 数组表示节点之间的信息素浓度，visible 数组表示节点之间的启发式信息量。Public 中有五个函数，包括一个构造函数。Transition 函数计算当前节点到下一节点转移的概率，相当于算法式（2）中大括号内部分的计算；UpdateLocalPathRule 函数用于局部更新信息素，相当于算法式（4）；InitParameter 函数用于初始化信息素浓度与启发式信息量，相当于算法式（1）和启发式信息量定义；UpdateGlobalPathRule 函数用于全局信息素更新，相当于算法式（5）（6）（7）。

```
class ACSAnt
{
private:
        AntColonySystem* antColony;
protected:
        int startCity, cururentCity;
        int allowed[N];
        int Tour[N][2];
        int currentTourIndex;
public:
        ACSAnt(AntColonySystem* acs, int start)
        {
                antColony=acs;
                startCity=start;
        }

        //选择下一节点
        int Choose();

        //开始搜索
        int* Search();

        //移动到下一节点
        void MoveToNextCity(int nextCity);
};
```

图 7 蚂蚁系统定义

蚂蚁系统包含一个 private 类说明自身所属蚁群，protected 中包括：startCity 代表蚂蚁初始城市编号；cururentCity 代表蚂蚁当前城市编号；allowed 数组代表禁忌表，值为 1 即表示蚂蚁还未曾走过这座城市；Tour 数组表示当前路径，是一个个路径段序列组成，即（currentcity，nextcity），用(Tour[i][0],Tour[i][1])表示；currentTourIndex 代表当前路径索引，从 0 开始存储蚂蚁经过的城市编号。Public 下包含构造函数在内的四个函数：Choose 函数用于决策蚂蚁下一个选择的城市，其决策方式如算法步骤 2)；Search 函数用于不同蚂蚁对图的不同遍历方式，目的为寻找局部最优长度；MoveToNextCity 函数主要在 Choose 函数后执行禁忌表与 Tour 数组的更新。

```
const int N=30,M=50;
const int Max=500;
double alpha=2,beta=5,rou=0.1,alpha1=0.1;
double dis[N][N];
double Lnn;

int ChooseNextNode(int currentNode, int visitedNode[]);
double CalAdjacentDistance(int node);
void calculateAllDistance();
double calculateSumOfDistance(int* tour);
```

图 8 全局变量与函数

定义 N 为城市数量，M 为蚂蚁数量，Max 为最大迭代次数；alpha 为信息启发因子α，beta 为期望启发式因子β，rou 为全局信息素挥发系数ρ，alpha1 为局部信息素挥发系数ξ；dis 数组表示完全图上两两城市之间的距离；Lnn 为最近邻长度 $L_{min}$。ChooseNextNode 函数用于选择下一个最近邻节点，CalAdjacentDistance 函数用于计算通过最近邻方法得到的总长度 Lnn，calculateAllDistance 函数用邻接矩阵存储输入的两两城市之间的距离，calculateSumOfDistance 函数用于获得经过给定 n 个城市的路径总长度。

### 3.3 系统实现

运用蚁群算法求解旅行商问题，算法步骤如下：

1）信息素初始化

$$\tau_{ij}(0) = \frac{1}{L_{min}*m} \tag{1}$$

其中：$\tau_{ij}(0)$表示初始时城市 i 到 j 路径上的信息素强度；m 为蚂蚁的数量；

$L_{min}$为从任一随机源节点出发，每次选择一个距离最短的点来遍历所有的节点得到的路径长度和。

2）状态转移

在 ACS 算法中，每只蚂蚁 k 在 t 时刻从城市 i 走向城市 j 的过程分两步进行：首先通过式(2)进行随机采样和比较，然后再结合式(3)确定状态变迁的规则

$$\mathrm{j}=\begin{cases} arg_{k\in allowed_k}\max\left\{\tau_{ik}^{\alpha}(t)*\eta_{ik}^{\beta}\right\}, q<q_0 \\ J, \qquad otherwise \end{cases} \tag{2}$$

其中：$allowed_k$为可行域，记录蚂蚁没有走过的城市；$q_0$是初始设定的参数，q 是一个随机采样的数，且$\mathrm{q}, q_0\in[0,1]$；J 是一个随机变量，其值按式(3)计算得到的概率经随机采样来确定。

蚂蚁 k 从城市 i 走向城市 j 的转移概率为：

$$p_{\mathrm{y}}^{k}(t)=\begin{cases} \dfrac{\tau_{ij}^{\alpha}(ij)*\eta_{ij}^{\beta}}{\sum_{s\in allowed_k}\tau_{ij}^{\alpha}(ij)*\eta_{ij}^{\beta}}, & j\notin tabu_k \\ 0, & 其他 \end{cases} \tag{3}$$

其中：$\tau_{ij}(t)$为 t 时刻在路径(i,j)上存在的信息素的浓度；$\eta_{ij}(t)=\frac{1}{d_{ij}}$为 t 时刻路径(i,j)上的能见度，路径距离越大能见度越小，也称作启发函数。$d_{ij}$为路径(i,j)的距离；$tabu_k$为禁忌表，记录蚂蚁走过的城市；α为信息启发因子，表示轨迹的相对重要性，反映蚂蚁在运动过程中积累信息素的重要性，其值越大表示蚂蚁越倾向选择其他蚂蚁经过的路径；β为期望启发式因子，表示能见度的重要性，反映了蚂蚁在运动过程中企发信息的重视程度。

3）信息素的更新

蚂蚁在寻找食物的过程中，会在所走的路径上释放信息素，根据式( 4)对信息素进行局部更新。

$$\tau_{ij}(t+1)=(1-\xi)\tau_{ij}(t)+\xi\tau_0 \tag{4}$$

通常设定ξ=0.1，$\xi\in(0,\ 1)$。

为了防止过了一段时间之后，残留路径上的信息素过多，从而使启发信息变得不再重要，在蚂蚁行走固定时间或者固定长度或者到达食物源后，要对整个路径上的信息素进行更新。在 t+n 时刻 d(i,j)按照下面公式进行更新。

$$\tau_{ij}(t+n)=(1-\rho)\tau_{ij}(t)+\rho*\tau_{ij}(t) \quad (5)$$

$$\tau_{ij}(t)=\sum_{k=1}^{m}\tau_{ij}^{k}(t) \quad (6)$$

ρ表示全局信息素挥发因子，取值为[0,1)；$\Delta\tau_{ij}(t)$表示当前路径城市 i 到城市 j 的信息素增量；$\Delta\tau_{ij}^{k}(t)$表示第 k 只蚂蚁在路径(i,j)上信息素的浓度。

信息素更新策略满足：

$$\tau_{ij}^{k}(t)=\begin{cases}\frac{Q}{L}, & 若第k只蚂蚁在本次循环中经过ij\\ 0, & 其他\end{cases} \quad (7)$$

其中 Q 表示信息素强度增加的系数；L 表示第 k 只蚂蚁在本次循环中所走路径的长度。

### 3.4 用户手册

采用随机生成的完全图导入格式要求 N*N/2 行输入，每行包含 i，j，k 代表 i 与 j 之间有权值为 k 的路径相连。所有参数都可以人为配置，根据完全图的性质决定。

输出组数为最大更迭次数 Max 组，每组第一行输出最优距离，第二行输出更迭时间，第三行输出最优路径。

### 3.5 测试

蚁群算法中有些参数在算法运行前需要人为设定其值，这些参数分别为：蚂蚁的数目 m、信息素浓度因子α、期望启发式因子β、信息素挥发系数ρ、信息素浓度$q_0$，而参数的设定对算法性能影响很大，如果参数设置不当，会导致求解速度很慢且解的质量差。

当 m 较大时，表面上来看，较多的蚂蚁数量会提高蚁群算法的全局搜索能力，但实际上蚂蚁数目过多时，会使曾经被搜索过的路径上的信息素浓度趋于平均，信息素正反馈作用降低，增加了算法的时间消耗；当 m 较小时，较小规模的 TSP 还能够解决，当规模上升到足够大时，蚂蚁数量较少，经过每个路径上的信息素太低，全局搜索概率降低，从而出现早熟或者停滞，影响算法的性能。

已给定数据为例，最大迭代次数为 500，分别取 m=10，20，30，40，50，统计平均用时与平均误差，绘制下表和下图。

表 1 m 值对平均误差和平均用时的影响

| M 值 | 平均误差 | 平均用时 |
|---|---|---|
| 10 | 1.6 | 14.2495 |
| 20 | 1.3 | 15.5945 |
| 30 | 1 | 16.497 |
| 40 | 0.3 | 16.8029 |
| 50 | 0.6 | 17.9782 |

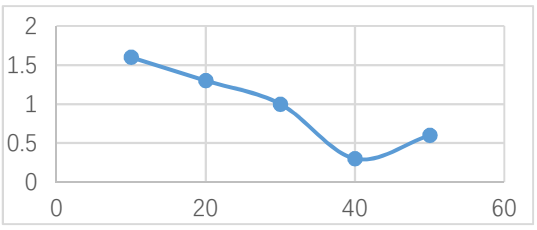


图 9 m 值对平均误差的影响

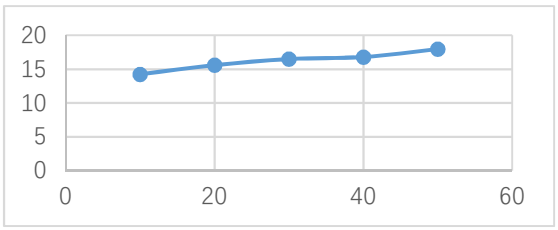


图 10 m 值对平均用时的影响

可以发现，在给定输入数据条件下，m 在 40 左右为最佳，继续增大 m 值并不会明显提高准确率，但会增加用时。

## 结束语

蚁群算法的搜索是建立在有指导的基础上，蚂蚁以路径上留存的信息素量为指导进行搜索，使得蚁群算法逐渐趋近于目标值。蚁群算法求解的过程并不是一步到位的，在该过程中很容易与其它方法结合，进而改善算法的性能。

在实验层次上蚁群算法可谓是十分成功的，然而却缺乏与之相对应的理论对蚁群算法解决问题的原因进行深入的解释。蚁群算法如果参数设置不当，会导致求解速度慢且解的质量差。然而算法参数的设置却靠的是研究者的经验，如果算法参数值设置不当，这会对算法的求解质量和执行效率产生巨大的影响。

在算法开始运行之后，由于其中一些路径上存在大量的信息素，在后续的搜索中，大量的蚂蚁受信息素浓度较大的影响会选择相同的路径，这样一来其余路径的探索就会停止，很难做到深入全面搜索所有的路径，使算法陷

入局部最优。蚂蚁不会去积累自身的经验，而几乎完全依靠于其它蚂蚁的信息反馈进行强化学习，使得行为缺乏自主性，算法的收敛速度也因此无法得到提升。因而往往需要嵌入一些专门的辅助技巧。

## 参考资料

[1]许凯波. 蚁群算法的改进及其在若干优化问题中的应用[D].江南大学,2018.
[2]张于贤,丁修坤,薛殿春,王晓婷.求解旅行商问题的改进蚁群算法研究[J].计算机工程与科学,2017,39(08):1576-1580.
[3]贾燕花.蚁群算法在旅行商问题(TSP)中的应用研究[J].计算机与数字工程,2016,44(09):1664-1667.
[4]姜坤霖,李美安,张宏伟.面向旅行商问题的蚁群算法改进[J].计算机应用,2015,35(S2):114-117.
[5]马振. 改进蚁群算法及其在TSP中的应用研究[D].青岛理工大学,2016.
[6]冀俊忠,黄振,刘椿年.一种快速求解旅行商问题的蚁群算法[J].计算机研究与发展,2009,46(06):968-978.

## 附录

```
#include<bits/stdc++.h>
#include <ctime>
#define ll long long
#define U unsigned
#define sqr(x) ((x)*(x))
#define rep0(x,n) for(int x=0;x<n;x++)
#define endl "\n"
using namespace std;

const int N=30,M=40;
const int Max=500;
double alpha=2,beta=5,rou=0.1,alpha1=0.1;
//信息启发因子,期望启发式因子,全局信息素挥发系数,局部信息素挥发系数

double dis[N][N];
double Lnn;

//蚁群系统
class AntColonySystem
{
private:
    double info[N][N], visible[N][N];//节点之间的信息素量,节点之间的启发式信息量
public:
    AntColonySystem(){}

    //计算当前节点到下一节点转移的概率
```

```
    double Transition(int i, int j)
    {
        if(i!=j)    return (pow(info[i][j],alpha)*pow(visible[i][j],beta));
        else    return 0.0;
    }

    //根据式 4 局部更新
    void UpdateLocalPathRule(int i, int j)
    {
        info[i][j]=info[j][i]=(1.0-alpha1)*info[i][j]+alpha1*(1.0/(N*Lnn));
    }

    //初始化
    void InitParameter(double value)
    {
        rep0(i,N)
        {
            rep0(j,N)
            {
                info[i][j]=info[j][i]=value;
                if(i!=j) visible[i][j]=visible[j][i]=1.0/dis[i][j];
            }
        }
    }

    //全局信息素更新
    void UpdateGlobalPathRule(int* bestTour, int globalBestLength)
    {
        rep0(i,N)
        {
            int row=*(bestTour+2*i);
            int col=*(bestTour+2*i+1);
            info[row][col]=info[col][row]=(1.0-rou)*info[row][col]+rou*(1.0/globalBestLength);
        }
    }
};

//蚂蚁个体
class ACSAnt
{
private:
    AntColonySystem* antColony;
protected:
    int startCity, cururentCity;//初始城市编号，当前城市编号
```

```
    int allowed[N];                 //禁忌表
    int Tour[N][2];                 //一个个路径段序列组成，即（currentcity，nextcity）
    int currentTourIndex;           //当前路径索引，从 0 开始，存储蚂蚁经过城市的编号
public:
    ACSAnt(AntColonySystem* acs, int start)
    {
        antColony=acs;
        startCity=start;
    }

    //选择下一节点
    int Choose()
    {
        int nextCity=-1;
        double q=rand()/(double)RAND_MAX;       //产生一个 0~1 之间的随机数 q

        if(q<=0.1)                                  //随机采样和比较
        {
            double probability=-1.0;
            rep0(i,N)
            {
                if(allowed[i])
                {
                    double prob=antColony->Transition(cururentCity,i);   //计算当前节点转移
到下一节点的概率
                    if(prob>probability)
                    {
                        nextCity=i;
                        probability=prob;
                    }
                }
            }
        }
        else
        {
            //按概率转移
            //生成一个随机数,用来判断落在哪个区间段
            double p=rand()/(double)RAND_MAX;
            double sum=0.0;
            //概率的区间点，p 落在哪个区间段，则该点是转移的方向
            double probability=0.0;
            rep0(i,N)
            {
                if(allowed[i])    sum+=antColony->Transition(cururentCity,i);
```

```
            }

            rep0(j,N)
            {
                if(allowed[j]&&sum>0)
                {
                    //往城市 j 转移的概率
                    probability+=antColony->Transition(cururentCity,j)/sum;
                    if(probability>=p||(p>0.9999&&probability>0.9999))
                    {
                        nextCity=j;
                        break;
                    }
                }
            }
        }
        return nextCity;
}

//开始搜索
int* Search()
{
        cururentCity=startCity;
        int toCity;
        currentTourIndex=0;
        rep0(i,N)
        {
            allowed[i]=1;
        }
        allowed[cururentCity]=0;
        int endCity;
        int count=0;
        do
        {
            count++;
            endCity=cururentCity;
            toCity=Choose();                                    //选择下一个节点
            if(toCity>=0)
            {
                MoveToNextCity(toCity);                         //移动到下一个节点
                antColony->UpdateLocalPathRule(endCity, toCity); //进行局部更新
                cururentCity=toCity;
            }
        }while(toCity>=0);
```

```
        MoveToNextCity(startCity);
        antColony->UpdateLocalPathRule(endCity,startCity);
        return *Tour;
    }

    //移动到下一节点
    void MoveToNextCity(int nextCity)
    {
        allowed[nextCity]=0;
        Tour[currentTourIndex][0]=cururentCity;
        Tour[currentTourIndex][1]=nextCity;
        currentTourIndex++;
        cururentCity=nextCity;
    }
};

//选择下一个最邻近节点
int ChooseNextNode(int currentNode, int visitedNode[])
{
    int nextNode=-1;
    double shortDistance=0.0;
    rep0(i,N)
    {
        if(visitedNode[i])
        {
            if(shortDistance==0.0)
            {
                shortDistance=dis[currentNode][i];
                nextNode=i;
            }
            if (shortDistance<dis[currentNode][i])
            {
                nextNode=i;
            }
        }
    }
    return nextNode;
}

//给一个节点由最近邻距离方法计算长度 Lnn
double CalAdjacentDistance(int node)
{
    double sum=0.0;
    int visitedNode[N];
```

```
    rep0(j,N)
    {
        visitedNode[j]=1;
    }
    visitedNode[node]=0;
    int currentNode=node;
    int nextNode;
    do
    {
        nextNode=ChooseNextNode(currentNode, visitedNode);
        if(nextNode>=0)
        {
            sum+=dis[currentNode][nextNode];
            currentNode=nextNode;
            visitedNode[currentNode]=0;
        }
    }while(nextNode>=0);
    sum+=dis[currentNode][node];
    return sum;
}

//由矩阵表示两两城市之间的距离
void calculateAllDistance()
{
    freopen("in.txt","r",stdin);
    rep0(i,N)
    {
        for(int j=i+1;j<N;j++)
        {
            int a,b,c;cin>>a>>b>>c;
            dis[i][j]=dis[j][i]=c;
        }
    }
}

//获得经过 n 个城市的路径长度
double calculateSumOfDistance(int* tour)
{
    double sum=0;
    rep0(i,N)
    {
        int row= *(tour+2*i);
        int col= *(tour+2*i+1);
        sum+=dis[row][col];
```

```
    }
    return sum;
}

int main()
{
    calculateAllDistance();

    AntColonySystem* acs = new AntColonySystem();
    ACSAnt* ants[M];
    for (int k = 0; k < M; k++)    ants[k] = new ACSAnt(acs, (int)(k%N));

    time_t timer;time(&timer);
    unsigned long seed = timer;
    seed%=56000;
    srand((unsigned int)seed);

    int node=rand()%N;                    //随机选择一个节点计算由最近邻方法得到 Lnn
    Lnn = CalAdjacentDistance(node);
    double initInfo = 1 / (N * Lnn);
    acs->InitParameter(initInfo);         //根据式 1 初始化蚁群信息素强度

    int globalTour[N][2];                 //全局最优路径序列
    double globalBestLength=0.0;          //全局最优路径长度

    static clock_t Start,Finish,s,f;
    s=clock();

    rep0(i,Max)
    {
        Start=clock();

        int localTour[N][2];              //局部最优路径序列
        double localBestLength=0.0;       //局部最优路径长度
        double tourLength;                //当前路径长度

        rep0(j,M)
        {
            int* tourPath=ants[j]->Search();
            tourLength=calculateSumOfDistance(tourPath);

            //局部比较，并记录路径和长度
            if(tourLength<localBestLength||abs(localBestLength-0.0)<0.000001)
            {
```

```
                rep0(m,N)
                {
                    int row=*(tourPath+2*m);
                    int col=*(tourPath+2*m+1);
                    localTour[m][0]=row;
                    localTour[m][1]=col;
                }
                localBestLength=tourLength;
            }
        }

        //全局比较，并记录路径和长度
        if(localBestLength<globalBestLength||abs(globalBestLength-0.0)<0.000001)
        {
            rep0(m,N)
            {
                globalTour[m][0]=localTour[m][0];
                globalTour[m][1]=localTour[m][1];
            }
            globalBestLength=localBestLength;
        }

        acs->UpdateGlobalPathRule(*globalTour,globalBestLength);

        Finish = clock();
        double time_second=double(Finish-Start)/CLOCKS_PER_SEC;

        cout<<"第"<<i+1<<"迭代最优路径:"<<localBestLength<<" "<<endl;
        printf("此次迭代运行时间:%fs\n",time_second);
        rep0(m,N)    cout<<localTour[m][0]+1<<" ";
        cout<<endl;
        cout<<endl;
    }
    f=clock();
    double t=double(f-s)/CLOCKS_PER_SEC;

    cout<<"全局最优路径长度:"<<globalBestLength<<endl;
    printf("总运行时间:%fs\n",t);
    cout<<"全局最优路径:";
    rep0(m,N)    cout<<globalTour[m][0]+1<<" ";
    cout<<endl;
    system("pause");
    return 0;
}
```