人工智能作业三
TSP 问题的 3 种解决方法

姓名：王丹
学号：2120151036

目录

一、实验目的

1.熟悉人工智能系统中的问题求解过程；
2.熟悉状态空间的启发式搜索算法的应用；
3.熟悉神经网络、群智能、进化算法的基本原理及应用方法。

二、实验内容

1.学习理解连续型 Hopfiled 神经网络原理，编程求解 TSP 问题；
2.学习理解蚁群算法原理，编程求解 TSP 问题；
3.学习理解遗传算法原理，编程求解 TSP 问题；

三、实验环境

编程平台：NetBeans IDE8.1
JAVA 环境：JDK1.8
编程语言：JAVA

四、实验原理

1、TSP 问题介绍

旅行商问题， 即 TSP 问题，是数学领域中著名问题之一。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。TSP 问题是一个组合优化问题。该问题可以被证明具有 NP 计算复杂性。

TSP 问题可以分为两类，一类是对称 TSP 问题（Symmetric TSP），另一类是非对称问题（Asymmetric TSP）。所有的 TSP 问题都可以用一个图来描述：

令$\mathrm{V}=\{c_1,c_2,\ldots,c_i,\ldots,c_n\},\mathrm{i}=1,2,\ldots,\mathrm{n}$是所有城市的集合。 c_i表示第 i 个城市， n 为城市的数目；

$\mathrm{E}=\{(\mathrm{r},\mathrm{s}):\mathrm{r},\mathrm{s}\in\mathrm{V}\}$是所有城市之间连接的集合；

$\mathrm{C}=\{c_{rs}:\mathrm{r},\mathrm{s}\in\mathrm{V}\}$是所有城市之间连接的成本度量（一般为城市之间的距离）;如果$c_{rs}=c_{sr}$ 那么该 TSP 问题为对称的，否则为非对称的。

一个 TSP 问题可以表达为：求解遍历图$\mathrm{G}=(\mathrm{V},\mathrm{E},\mathrm{C})$，所有的节点一次并且回到起始节点，使得连接这些节点的路径成本最低。

2、连续型 Hopfield 神经网络求解 TSP 问题

(1)连续型 Hopfield 网络简介

一般神经网络（RBF 或 BP）是单向的，而 Hopfield 网络特点在于有反馈，而最终得到的结果是一个网络的收敛值。对于一个 Hopfield 网络来说，关键是在于确定它在稳定条件下的权系数。

(2) 连续型 Hopfield 神经网络求解 TSP 问题原理

TSP 问题是一个组合优化问题，就是在给定约束条件下，求出使目标函数极小（或极

大）的变量组合问题。将 Hopfield 网络应用于求解组合优化问题包括“将目标函数转化为网络的能量函数”和“把问题的变量对应于网络的状态”两个方面。这样，当网络的能量函数收敛于极小值时，问题的最优解也随之解出。

a)换位矩阵

将 TSP 问题映射到一个神经网络。假定每个神经元的放大器有一个很高的放大倍数，神经元的输出限制在二值 0 和 1 上，则映射问题可以用一个换位矩阵(PM, Permutation Matrix)来进行，换位矩阵可如下图所示：

次序 / 城市	1	2	3	4	5
A	0	0	0	1	0
B	1	0	0	0	0
C	0	0	0	0	1
D	0	1	0	0	0
E	0	0	1	0	0

对于n个城市需用由$n*n$个神经元构成的$n*n$阶 PM 来表示旅行路线。在该换位矩阵中每一列只有一个个元素为 1，其余为 0，列的大小表示对某城市访问的次序。同样每一行也只有一个元素为 1，其余为 0。通过这样的 PM，可唯一确定一条旅行路线。

b)约束条件和最优条件

1) 一个城市只能被访问一次→换位矩阵每行只有一个 1。

2)一次只能访问一个城市→换拉矩阵每列只有一个 1。

3)总共有 N 个城市→换位矩阵元素之和为 N。

4)求巡回路径最短→网络能量函数的最小值对应于 TSP 的最短路径。

c)网络能量函数

$$E=\frac{A}{2}\sum_{x=1}^{N}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\upsilon_{xi}\upsilon_{xj}+\frac{B}{2}\sum_{i=1}^{N}\sum_{x=1}^{N-1}\sum_{y=x+1}^{N}\upsilon_{xi}\upsilon_{yi}$$

$$+\frac{C}{2}(\sum_{x=1}^{N}\sum_{i=1}^{N}\upsilon_{xi}-N)^2$$

$$+\frac{D}{2}\sum_{x=1}^{N}\sum_{y=1}^{N}\sum_{i=1}^{N}d_{xy}\upsilon xi(\upsilon_{y,i+1}+\upsilon_{y,i-1})$$

当 E 达到极小时，由网络状v_{ij}构成的换位矩阵表达了最佳旅行路线。

d)网络加权及阈值

确定网络神经元之间的连接权及神经元之间输出的阈值。

(x, i)神经元与(y, j)神经元之间的连接权为$w_{xi,yj}$，神经元(x, i)输出的阈值为I_{xi}，则有

$$\omega_{xi,yj} = -A\delta_{xy}(1-\delta_{ij}) - B\delta_{ij}(1-\delta_{xy}) - C - Dd_{xy}(\delta_{j,i+1}+\delta_{j,i-1})$$

$$I_{xi} = CN \qquad \delta_{ij} = \begin{cases} 1(i=j) \\ 0(i\neq j) \end{cases}$$

e)TSP 网络的迭代方程

$$c_{xi}\frac{du_i}{dt} = -\frac{u_{xi}}{R_{xi}} - A\sum_{\substack{j=1 \\ j\neq 1}}^{N}\upsilon_{xi} - B\sum_{\substack{y=1 \\ y\neq x}}^{N}\upsilon_{yi} - C(\sum_{x=1}^{N}\sum_{y=1}^{N}\upsilon_{xy} - N) - D\sum_{y=1}^{N}d_{xy}(\upsilon_{y,i+1}+\upsilon_{y},i-1)$$

$$\upsilon_{xi} = \varphi_{xi}(u_{xi}) = \frac{1}{2}[1+\tanh(\frac{u_{xi}}{u_0})]$$

3、蚁群算法求解 TSP 问题

(1)蚁群算法简介

蚁群算法（Ant Clony Optimization， ACO）是一种群智能算法，它是由一群无智能或有轻微智能的个体（Agent）通过相互协作而表现出智能行为，从而为求解复杂问题提供了一个新的可能性。

蚁群算法是一种仿生学算法，是由自然界中蚂蚁觅食的行为而启发的。在自然界中，蚂蚁觅食过程中，蚁群总能够按照寻找到一条从蚁巢和食物源的最优路径。下图展示了觅食的过程：

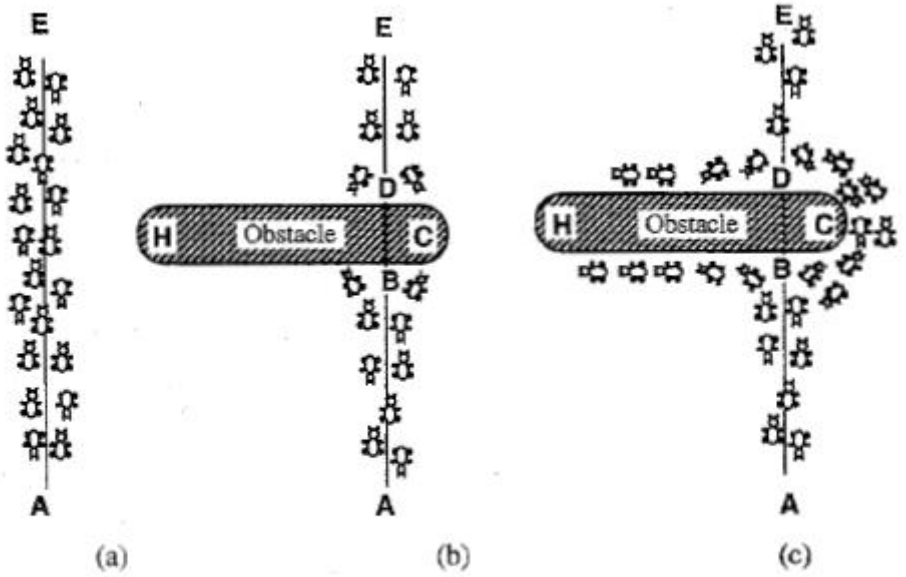


Fig. 1. An example with real ants. (a) Ants follow a path between points *A* and *E*. (b) An obstacle is interposed; ants can choose to go around it following one of the two different paths with equal probability. (c) On the shorter path more pheromone is laid down.

在上图（a）中，有一群蚂蚁，假如 A 是蚁巢，E 是食物源（反之亦然）。这群蚂蚁将沿着蚁巢和食物源之间的直线路径行驶。假如在 A 和 E 之间突然出现了一个障碍物（b），那么，在 B 点（或 D 点）的蚂蚁将要做出决策，到底是向左行驶还是向右行驶？由于一开始路上没有前面蚂蚁留下的信息素（pheromone），蚂蚁朝着两个方向行进的概率是相等的。但是当有蚂蚁走过时，它将会在它行进的路上释放出信息素，并且这种信息素会议一定的速率散发掉。信息素是蚂蚁之间交流的工具之一。它后面的蚂蚁通过路上信息素的浓度，做出决策，往左还是往右。很明显，沿着短边的的路径上信息素将会越来越浓（c），从而吸引了越来越多的蚂蚁沿着这条路径行驶。

(2)蚁群算法求解 TSP 问题原理

a)基本参数、信息素浓度公式、择路概率

设蚂蚁的数量为m，城市的数量为n，城市 i 与城市 j 之间的距离为d_{ij}，t 时刻城市 i 与城市 j 之间的信息素浓度为$t_{ij}(t)$，初始时刻，各个城市间连接路径上的信息素浓度相同，不妨记为$t_{ij}(0) = c$。

蚂蚁$k(k = 1,2,..,m)$根据各城市间连接路径上的信息素浓度，决定其下一个要访问的城市，设$P_{ij}{}^{k}(t)$表示 t 时刻，蚂蚁 k 从城市 i 到城市 j 的概率，其计算公式为如下：

$$P_{ij}^{k} = \begin{cases} \dfrac{[t_{ij}(t)]^{\alpha} \bullet [\eta_{ij}(t)]^{\beta}}{\sum_{s \in allow} [t_{ij}(t)]^{\alpha} \bullet [\eta_{ij}(t)]^{\beta}} & s \in allow_k \\ 0 & s \notin allow_k \end{cases}$$

其中：

$\eta_{ij}(t)$为启发式函数，$\eta_{ij}(t)=1/d_{ij}$，表示蚂蚁从城市 i 转移到城市 j 的期望程序；

$allow_k(k = 1,2,\ldots,m)$表示蚂蚁 k 待访问的城市的集合，开始时$allow_k$为其他 n-1 城市，随着时间推进，其中的元素不断减少，直至为空，表示所有城市访问完，即遍历所有城市。

α为信息素的重要程度因子，其值越大，转移中起的作用越大。

β为启发函数的重要程度因子，其值越大，表示启发函数在转移中的作用越大，即蚂蚁以较大的概率转移到距离短的城市。

蚂蚁释放的信息素会随时间的推进而减少，设参数$\rho(0<\rho<1)$表示信息素的挥发度，当所有蚂蚁完成一次循环后，各个城市间连接路径上的信息素浓度，需要实时更新。

$$t_{ij}(t+1)=(1-\rho)t_{ij}(t)+\Delta t_{ij},\Delta t_{ij}=\sum_{k=1}^{n}\Delta t_{ij}^{k}$$

其中：

$\Delta t_{ij}{}^{k}$表示蚂蚁 k 在城市 i 与城市 j 的连接路径上，释放的信息素浓度

Δt_{ij}表示所有蚂蚁在城市 i 与城市 j 的连接路径上，释放的信息素浓度。

b) Δt_{ij}的计算方法

$$\Delta t_{ij}^{k}=\begin{cases}Q/L_k & \text{第}k\text{只蚂蚁从城市i访问城市j}\\ 0 & \text{其他}\end{cases}$$

其中：

Q 为常数，表示蚂蚁循环一次释放的信息素的总量；

L_k为第 k 只蚂蚁经过路径的总长度。

4、遗传算法求解 TSP 问题

(1)遗传算法简介

遗传算法（Genetic Algorithm）是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群开始的，而一个种群则由经过基因编码的一定数目的个体组成。每个个体实际上是染色体带有特征的实体。染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现（即基因型）是某种基因组合，它决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此，在一开始需要实现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂，我们往往进行简化，如二进制编码，初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代演化产生出越来越好的近似解，在每一代，根据问题域中个体的适应度大小选择个体，并借助于自然遗传学的遗传算子进行组合交叉和变异，产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境，末代种群中的最优个体经过解码，可以作为问题近似最优解。

(1)遗传算法求解 TSP 原理

用遗传算法求解 TSP 问题，最困难的是要解决三个方面的问题，即编码，交叉操作以及变异操作。

a)编码

TSP 问题编码一般有五种不同的方式：基于二进制的编码、基于矩阵的编码、基于邻接的编码、基于索引（Ordinary）的编码、基于路径的编码。

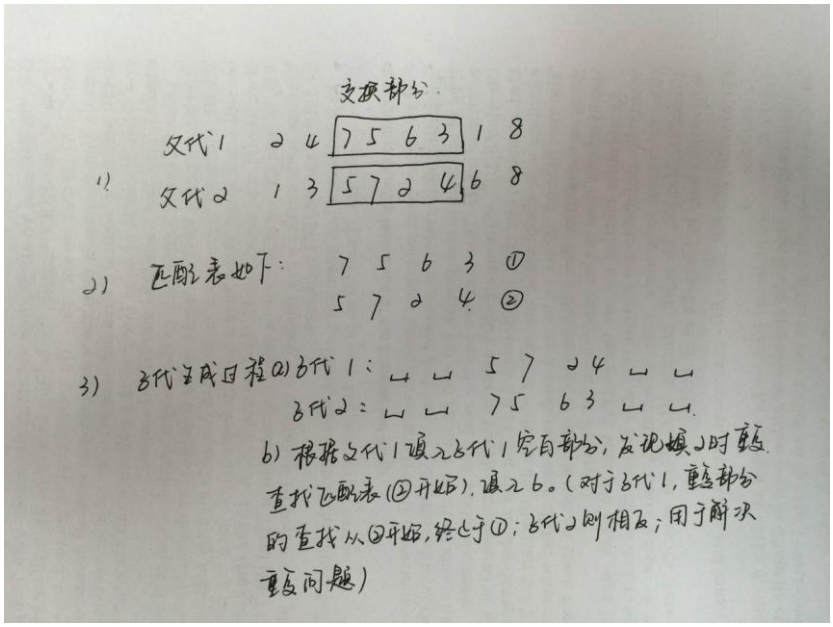
基于二进制的编码是一种传统的编码方式，但是这种方式的编码，在经过遗传操作以后很难保证后代还是一个可行解，还需要另外的修正操作；基于矩阵的编码、邻接的编码以及索引的编码均比较复杂，并且需要占用大量的内存，应用的不是很广泛；目前最普遍用的编码方式就是基于路径的编码方式，因为它最直观最容易理解，操作起来也比较方便。

以下将主要介绍基于路径的编码方式以及相关的遗传操作。（本实验编程采用基于路径的编码方式）

b) 交叉操作

交叉操作主要有部分匹配法、循环交叉法、次序交叉法、基于位置的交叉法、交替位置的交叉法等。不同的交叉操作实现难度不同。本实验编程采用的是部分匹配法。以下是对部分匹配法（Partially Matching Crossover， PMX）的介绍。

以两个父代个体为例：（2 4 7 5 6 3 1 8）和（1 3 5 7 2 4 6 8），随机选择两个交叉的点，假如第一个点为位置 3，第二个交叉点为位置 6，那么在两个点之间的位置将进行交叉，其它的位置进行复制或者用相匹配的数进行替换。在此实例中，第一个父代个体中 **7 5 6 3** 被选中，第二个父代个体中，**5 7 2 4** 被选中。那么 7 与 5，5 与 7，6 与 2，3 与 4 相匹配。匹配过程和插入过程及重复处理过程下图所示。

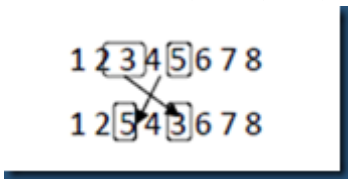


首先将 **7 5 6 3** 与 **5 7 2 4** 分别加入到子代 2 和子代 1 中相应的位置，然后将其他位置上的数字直接复制到相应的后代中，如果该数字已经在该子代中已经存在，则用相应的匹配法则进行替换，例如子代 1 中将 2 复制进去的时候，发现 2 已经存在在子代中，通过查找相应的匹配法则，发现，2 与 6 匹配，然后复制 6，6 不存在在该子代中，所以可以将 6 复制进去。如此反复，知道子代中城市的数目达到定义的长度，该子代创建完成。最终得到的子代是（6 3 5 7 2 4 1 8）和（1 4 7 5 6 3 2 8）

c) 变异操作

常见的变异操作方法有替换变异、交换变异、插入变异、简单倒位变异、倒位变异、争夺变异等。本实验编程选择交换变异（Exchange Mutation， EM）。

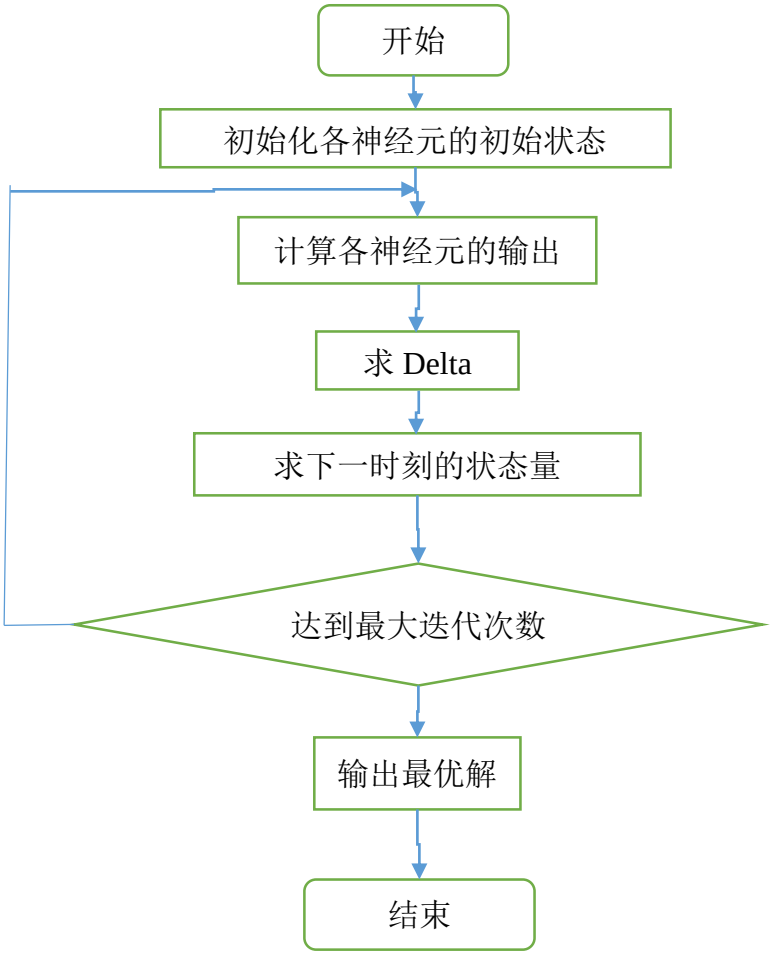
EM 是从父代个体中选择两个基因位置，然后互换这两个位置的基因。如下图所示。



五、算法实现流程

1.TSP 的 Hopfield 神经网络求解

（1）Hopfield 求解 TSP 流程图



（2） 流程详解

a)初始化：给定一个u_0值。这保证收敛于正确解，按下式取网络各神经元的初始状态：

$$u_{xi} = u_{00} + \delta_{uxi}$$

式中，$u_{00} = \frac{1}{2} u_0 \ln(N-1)$，其中 N 为网络神经元个数；$\delta_{uxi}$为（-1，+1）区间的随机值。

b)按 a)中的结果求出各神经元的输出

$$v_{xi} = u_{xi}(t_0) = \frac{1}{2}\left[1 + \tanh\left(\frac{u_{xi}(t_0)}{u_0}\right)\right]$$

c)将$\upsilon_{xi}(t_0)$代入求得$\left.\frac{du_{xi}}{dt}\right|_{t=t_0}$。

d)求下一时刻的状态量

$$u_{xi}(t_0 + \Delta t) = u_{xi}(t_0) + \left.\frac{du_{xi}}{dt}\right|_{t=t_0} \Delta t$$

e)返回步骤 b)

(3)伪代码

输入：迭代次数 T，城市数目 citynum，控制参数A, B, C, D, u_0

预处理：初始化城市距离矩阵$distance[citynum][citynum]$

Hopfield 算法迭代：

初始化网络各神经元的初始状态u_0

for i=0.05:0.31;i+=0.01{

A=B=D = i;

while(没有达到最大迭代次数 T){

求出各神经元的输出v_{xi}；

计算能量函数；

计算 $\frac{du_{xi}}{dt}\Big|_{t=t_0}$ ；

updatePM_u():更新全局状态矩阵

findBestRoad():判定路径合法性并寻求最优解

if(能量减小值<0.01)

break;

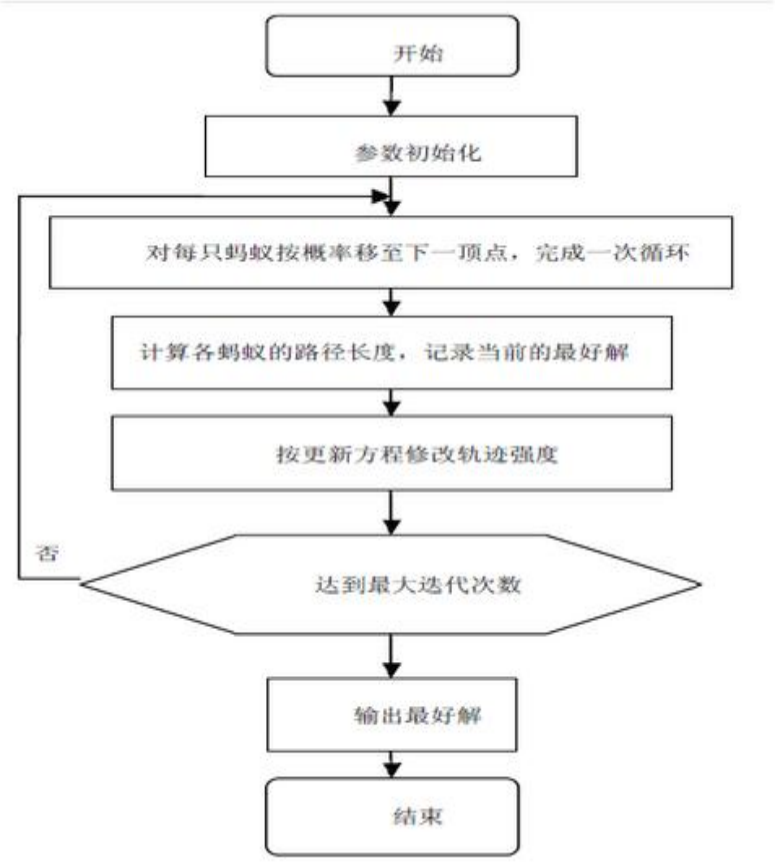
}

}

输出：整个过程的最优解

2.TSP 的蚁群算法求解

（1）蚁群算法求解 TSP 流程图



（2）流程讲解

假如蚁群中所有蚂蚁的数量为 m，所有城市之间的信息素用矩阵 pheromone 表示，最短路径为bestLength，最佳路径为bestTour。每只蚂蚁都有自己的内存，内存中用一个禁忌表Tabu来存储该蚂蚁已经访问过的城市，表示其在以后的搜索中将不能访问这些城市；还有用

另外一个允许访问的城市表（Allowed）来存储它还可以访问的城市；另外还用一个矩阵（Delta）来存储它在一个循环（或者迭代）中给所经过的路径释放的信息素；还有另外一些数据，例如一些控制参数(α, β, ρ, Q)，该蚂蚁行走玩全程的总成本或距离(tourLength），等等。假定算法总共运行 MAX_GEN 次，运行时间为 t。

蚁群算法计算过程如下：

a)初始化

设 t=0，初始化bestLength为一个非常大的数（正无穷），bestTour为空。初始化所有的蚂蚁的Delt矩阵所有元素初始化为 0，Tabu表清空，Allowed 表中加入所有的城市节点。随机选择它们的起始位置。在Tabu中加入起始节点，Allowed 中去掉该起始节点。

b)为每只蚂蚁选择下一个节点

为每只蚂蚁选择下一个节点，该节点只能从 Allowed 中以

$$P_{ij}^{k}=\begin{cases}\dfrac{[t_{ij}(t)]^{\alpha}\bullet[\eta_{ij}(t)]^{\beta}}{\sum_{s\in allow}[t_{ij}(t)]^{\alpha}\bullet[\eta_{ij}(t)]^{\beta}} & s\in allow_k\\ 0 & s\notin allow_k\end{cases}$$

搜索到，每搜到一个，就将该节点加入到Tabu中，并且从 Allowed 中删除该节点。该过程重复 n-1 次，直到所有的城市都遍历过一次。遍历完所有节点后，将起始节点加入到Tabu中。此时Tabu表元素数量为 n+1（n 为城市数量），Allowed 元素数量为 0。（参数介绍参见实验原理部分）

接下来按照

$$\Delta t_{ij}^{k}=\begin{cases}Q/L_k & \text{第}k\text{只蚂蚁从城市i访问城市j}\\ 0 & \text{其他}\end{cases}$$

计算每个蚂蚁的 Delta 矩阵值。最后计算最佳路径，比较每个蚂蚁的路径成本，然后和bestLength比较，若它的路径成本比bestLength小，则将该值赋予bestLength，并且将其Tabu赋予BestTour。（参数介绍参见实验原理部分）。

c)更新信息素矩阵

按照以下更新信息素矩阵 pheromone。

$$t_{ij}(t+1)=(1-\rho)t_{ij}(t)+\Delta t_{ij},\Delta t_{ij}=\sum_{k=1}^{n}\Delta t_{ij}^{k}$$

（参数介绍参见实验原理部分）

d)检查终止条件

如果达到最大代数 MAX_GEN，算法终止，转到第 e)步；否则，重新初始化所有的蚂蚁的 Delta 矩阵所有元素初始化为 0，Tabu表清空，Allowed 表中加入所有的城市节点。随机选择它们的起始位置。在Tabu中加入起始节点，Allowed 中去掉该起始节点，重复执行 b)、c)、d)步。

e)输出最优值

（3）伪代码

输入：迭代次数 T，城市数目 citynum，蚂蚁数目 antnum，信息素量 Q，控制参数α，β，γ

预处理：初始化城市距离矩阵distance[$citynum$][$citynum$],初始化信息素矩阵pheromone[$citynum$][$citynum$]

蚁群算法迭代：

while(没有达到最大迭代次数 T){

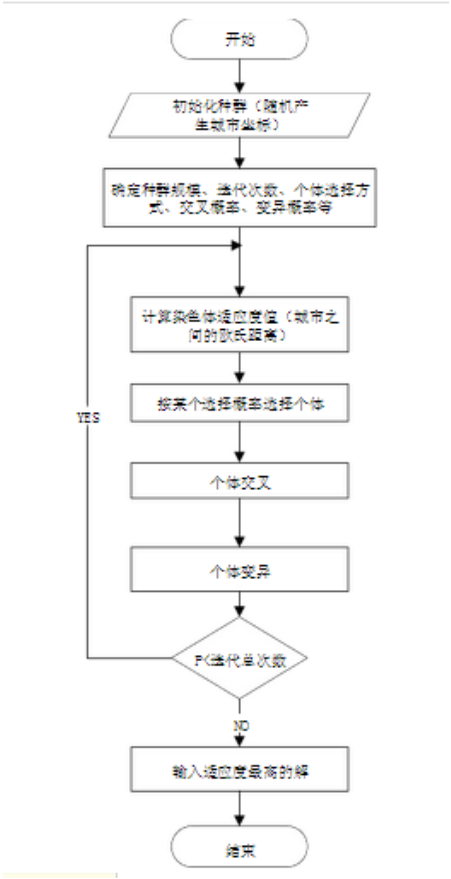
初始化蚂蚁群Ants[$antnum$];
for i=1:antnum{
while(Allowed.size()>0){
MovetoNextCity():轮盘赌求蚂蚁移动的下一节点，更新 Tabu 表和 Allowed 表；
}
根据 Tabu 表计算每只蚂蚁的路径长度：

$$Ants[i].roadlength = \sum_{n=0}^{citynum-1} distance[Tabu[n]][Tabu[n+1]] + disance[Tabu[citynum-1]][Tabu[0]]$$

}
updatePheromone():更新全局信息素矩阵
findBestRoad():寻找并更新当前最好解
}
输出：整个过程的最优解

3.TSP 的遗传算法求解

(1)遗传算法求解 TSP 问题流程图



注：本实验编程中并未采用欧式距离作为城市距离计算公式。而是为每对邻居城市随机产生 2~100 之间的浮点数作为城市距离。

(2)流程讲解

设最大进化代数为 T，每一代的群体表示为$P(1), P(2), \dots, P(i), \dots, P(T)$; $i = 1,2, \dots$， T。最短

路径表示为shortestLength，最优个体表示为bestEntity。城市之间的距离用矩阵 distance 表示。distance(i, j)表示城市 i 和城市 j 之间的距离。

a)初始化

设置进化代数计数器$t = 0$，随机生成 M 个个体作为初始群体$P(0)$。

b)个体评价

计算群体$P(t)$中各个个体的适应度 Adaptability 和幸存程度 Lucky，作为选择运算的依据。采用路径总距离作为适应度。

$$\text{Adaptability(i)} = \sum_{n=1}^{M-1} distance(n, n+1) + distance(M, 1)$$

幸存程度计算如下：

$$\text{Lucky(i)} = 1 - \text{Adaptability(i)} / \sum_{n=1}^{M} Adaptability\,(n)$$

实际编程中，对 Lucky 进行归一化：

$$\text{All}_{\text{Lucky}} = \sum_{n-1}^{M} Lucky(i)$$

$$\text{Lucky(i)} = \frac{Lucky(i)}{All_{lucky}}$$

$$(\text{i} = 1,2, \dots, \text{M})$$

以上过程是有意义的，Adaptability 大的个体表示路径长，这并不适合个体生存，所以用幸存程度修正使用度。最终 Adaptability 大的个体 Lucky 小，不易生存；Adaptability 小的个体 Lucky 大，容易存活。

c)选择运算

将选择算子作用于群体。选择的目的是把优化的个体直接遗传到下一代或通过配对交叉产生新的个体再遗传到下一代。选择操作是建立在群体中个体的适应度评估基础上的。

d)交叉运算

将交叉算子作用于群体。所谓交叉是指把两个父代个体的部分结构加以替换重组而生成新个体的操作。遗传算法中起核心作用的就是交叉算子。

e)变异运算

将变异算子作用于群体。即是对群体中的个体串的某些基因上的基因值作变动。

群体$P(t)$经过选择、交叉、变异运算之后得到下一代群体$P(t+1)$。

f)终止条件判断

选择本次迭代中 Adaptability 最小的个体 G，和bestEntity对比。

$$\text{bestEntity} = \begin{cases} G & if\ \ Adaptability < shortestLength \\ bestEntity & otherwise \end{cases}$$

$$\text{shorestLength} = \text{bestEntity.Adaptability}$$

随时保证当前最优解是所有已进行迭代过程中的最优解。

若$t = T$,则以进化过程中所得到的具有最大适应度个体作为最优解输出，终止计算。否则重复 b)、c)、d)、e)步骤。

（3）伪代码

输入：迭代次数 T，城市数目 citynum，种群数目 num，

预处理：初始化城市距离矩阵distance[*citynum*][*citynum*]，初始化种群gaEntity[*num*];

遗传算法迭代：

while(没有达到最大迭代次数 T){

Cal_AdaptabilityAndLucky()计算个体适应度值(每个染色体的路径总和)和幸存程度；

chooseSample()轮盘赌选择种群数量的待操作个体；

Mating()部分匹配法进行交叉；

Variating()交换变异；

chooseBestSoulution():寻找并更新当前最优解

}

输出：整个过程的最优解

六、实验结果及分析

(1)TSP 的 Hopfield 神经网络求解

a)实验结果

程序错误，暂无实验结果。

b)结果分析

程序错误，暂无。

(2)TSP 的蚁群算法求解

a)实验结果

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	4	93	41	2	66	60	29	81	65	93	17	73	13	63	80	52	54	11	11	76	2	78	89	48	78	85	81	46	57
1	81	0	81	94	28	23	84	29	87	50	87	60	93	75	78	20	21	47	78	96	6	89	38	88	89	3	56	96	17	47
2	60	57	0	93	33	55	49	65	97	87	77	66	63	48	52	94	29	97	42	61	5	58	42	87	66	24	19	34	19	15
3	6	74	29	0	98	82	45	87	99	19	8	79	67	11	57	45	91	13	14	27	68	22	81	48	81	65	65	97	11	19
4	84	97	26	68	0	35	62	86	56	88	35	77	9	17	91	87	11	4	72	82	86	45	79	36	61	73	36	47	42	28
5	56	23	5	53	21	0	13	58	78	73	35	33	14	3	2	71	26	49	9	81	55	43	97	20	15	85	28	17	89	84
6	99	81	46	13	70	99	0	97	30	49	3	18	7	74	78	92	36	12	16	36	9	52	69	34	88	90	92	56	20	3
7	45	73	25	81	92	43	48	0	85	33	41	11	89	76	44	62	60	86	77	16	86	27	64	38	96	6	95	23	76	84
8	44	29	26	82	17	64	87	80	0	90	96	76	30	6	88	12	32	18	72	17	13	23	79	64	31	96	22	97	74	48
9	93	39	55	25	71	67	44	44	91	0	61	60	38	2	6	56	81	65	88	40	66	86	94	38	26	94	2	67	82	83
10	32	12	61	88	67	56	88	72	33	50	0	22	93	15	98	58	3	3	74	16	63	91	69	18	96	63	60	3	30	44
11	97	37	36	62	24	36	39	18	70	63	2	0	8	81	87	4	59	83	23	50	91	23	26	8	7	78	54	75	21	11
12	79	73	41	34	84	54	41	32	65	64	91	66	0	33	77	38	51	16	70	84	13	96	45	54	21	37	28	82	47	91
13	50	34	83	22	63	67	88	18	75	20	42	52	37	0	56	90	73	6	28	5	92	58	41	11	71	96	99	12	53	88
14	7	14	90	90	2	10	67	45	77	61	82	30	96	43	0	22	73	68	74	31	27	65	51	31	68	55	74	5	13	85
15	7	41	39	99	58	76	31	38	39	9	59	84	37	93	78	0	91	22	83	50	43	67	5	5	72	13	97	81	78	39
16	95	7	39	78	46	89	39	94	4	12	92	57	81	48	66	20	0	66	99	31	10	6	42	31	5	70	24	82	96	80
17	36	45	64	92	43	16	65	31	22	5	96	30	50	83	23	91	89	0	87	84	78	82	72	56	13	87	34	84	23	31
18	42	5	62	17	69	59	7	91	85	98	44	5	53	45	25	31	63	34	0	53	60	63	58	63	52	48	21	22	81	38
19	87	38	80	29	51	33	82	57	21	62	21	38	69	5	59	15	99	2	27	0	4	12	64	14	29	23	11	20	35	90
20	83	87	73	6	75	25	11	21	25	11	49	95	76	6	10	2	19	31	16	99	0	63	42	9	20	7	90	46	42	11
21	59	46	32	71	80	97	5	85	11	17	39	66	67	87	84	55	66	45	66	21	90	0	23	85	96	78	56	97	54	60
22	35	26	15	6	52	53	70	13	10	11	60	19	39	78	73	10	8	45	14	32	77	12	0	35	66	22	48	65	32	61
23	34	34	67	23	94	17	76	28	7	17	99	77	21	95	14	57	82	46	95	48	42	31	27	0	36	22	66	91	9	97
24	45	9	57	40	85	28	25	98	71	30	13	3	39	41	7	38	86	30	91	19	12	46	4	14	0	19	79	76	94	17
25	90	16	70	79	69	17	77	60	29	50	39	26	27	17	56	44	56	18	32	45	72	74	48	78	4	0	76	61	26	65
26	38	58	22	73	58	43	60	99	82	90	45	69	26	20	83	45	32	62	14	54	63	26	41	53	54	38	0	22	35	82
27	66	56	65	27	94	16	66	6	83	30	98	39	71	4	16	53	35	88	92	76	82	33	19	54	6	72	42	0	18	79
28	87	51	55	36	22	55	20	2	23	67	77	31	13	5	91	49	49	84	84	68	39	63	70	70	44	30	28	53	0	56
29	51	43	41	32	55	92	62	17	14	97	72	71	21	73	79	70	28	84	99	43	58	33	24	23	9	23	15	21	96	0

图 1 随机产生的 30*30 的城市距离矩阵，
实际距离值为浮点数，打印输出时仅输出整数部分

```
第0次迭代：
当前最优解是：26;10;28;4;23;16;5;19;13;18;14;21;1;25;15;8;17;24;20;7;11;22;27;0;2;12;6;3;9;29;
该路径下的最低消耗：1213.9725434971026
第1次迭代：
当前最优解是：17;3;2;21;19;5;4;27;0;11;7;28;12;6;18;23;29;10;16;9;15;8;26;1;25;13;22;14;20;24;
该路径下的最低消耗：1183.5580641966785
第2次迭代：
当前最优解是：3;0;21;11;10;29;4;26;18;22;23;24;15;2;20;7;13;9;27;1;25;12;5;6;28;19;14;17;8;16;
该路径下的最低消耗：1076.9293216660221
第3次迭代：
当前最优解是：16;21;6;4;2;19;25;17;20;3;9;11;22;15;0;28;7;12;24;29;1;5;8;14;27;26;18;10;13;23;
该路径下的最低消耗：1040.8398198309276
第4次迭代：
当前最优解是：13;17;5;21;29;0;18;27;14;6;11;10;19;23;26;28;7;3;24;25;20;16;8;9;1;15;22;4;2;12;
该路径下的最低消耗：1033.2573534246233
第5次迭代：
当前最优解是：13;17;5;21;29;0;18;27;14;6;11;10;19;23;26;28;7;3;24;25;20;16;8;9;1;15;22;4;2;12;
该路径下的最低消耗：1033.2573534246233
第6次迭代：
当前最优解是：13;17;5;21;29;0;18;27;14;6;11;10;19;23;26;28;7;3;24;25;20;16;8;9;1;15;22;4;2;12;
该路径下的最低消耗：1033.2573534246233
第7次迭代：
当前最优解是：13;17;5;21;29;0;18;27;14;6;11;10;19;23;26;28;7;3;24;25;20;16;8;9;1;15;22;4;2;12;
该路径下的最低消耗：1033.2573534246233
第8次迭代：
当前最优解是：13;17;5;21;29;0;18;27;14;6;11;10;19;23;26;28;7;3;24;25;20;16;8;9;1;15;22;4;2;12;
该路径下的最低消耗：1033.2573534246233
第9次迭代：
当前最优解是：13;17;5;21;29;0;18;27;14;6;11;10;19;23;26;28;7;3;24;25;20;16;8;9;1;15;22;4;2;12;
该路径下的最低消耗：1033.2573534246233
```

当前最优解是：18;10;23;5;13;11;20;7;27;24;22;3;6;29;4;16;15;2;28;12;26;25;8;17;1;9;19;21;14;0;
该路径下的最低消耗：857.1764784833808
第809次迭代：
当前最优解是：18;10;23;5;13;11;20;7;27;24;22;3;6;29;4;16;15;2;28;12;26;25;8;17;1;9;19;21;14;0;
该路径下的最低消耗：857.1764784833808
第810次迭代：
当前最优解是：18;10;23;5;13;11;20;7;27;24;22;3;6;29;4;16;15;2;28;12;26;25;8;17;1;9;19;21;14;0;
该路径下的最低消耗：857.1764784833808
第811次迭代：
当前最优解是：18;10;23;5;13;11;20;7;27;24;22;3;6;29;4;16;15;2;28;12;26;25;8;17;1;9;19;21;14;0;
该路径下的最低消耗：857.1764784833808
第812次迭代：
当前最优解是：18;10;23;5;13;11;20;7;27;24;22;3;6;29;4;16;15;2;28;12;26;25;8;17;1;9;19;21;14;0;
该路径下的最低消耗：857.1764784833808
第813次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第814次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第815次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第816次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第817次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第992次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第993次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第994次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第995次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第996次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第997次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第998次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441
第999次迭代：
当前最优解是：0;4;28;21;19;13;10;17;12;6;29;1;25;22;9;27;7;2;26;18;14;15;20;5;23;8;16;24;11;3;
该路径下的最低消耗：826.7000321977441

图 2 迭代求解过程

b)结果分析

蚁群算法充分体现了自组织这个过程。当算法开始的初期，单个的人工蚂蚁无序的寻找解，算法经过一段时间的演化，人工蚂蚁间通过信息激素的作用，自发的越来越趋向于寻找

到接近最优解的一些解，这就是一个无序到有序的过程。

蚁群算法是一种正反馈的算法。从真实蚂蚁的觅食过程中我们不难看出，蚂蚁能够最终找到最短路径，直接依赖于最短路径上信息激素的堆积，而信息激素的堆积却是一个正反馈的过程。对蚁群算法来说，初始时刻在环境中存在完全相同的信息激素，给予系统一个微小扰动，使得各个边上的轨迹浓度不相同，蚂蚁构造的解就存在了优劣，算法采用的反馈方式是在较优的解经过的路径留下更多的信息激素，而更多的信息激素又吸引了更多的蚂蚁，这个正反馈的过程使得初始的不同得到不断的扩大，同时又引导整个系统向最优解的方向进化。因此，正反馈是蚂蚁算法的重要特征，它使得算法演化过程得以进行。

蚁群算法具有较强的鲁棒性。相对于其它算法，蚁群算法对初始路线要求不高，即蚁群算法的求解结果不依赖子初始路线的选择，而且在搜索过程中不需要进行人工的调整。其次，蚁群算法的参数数目少，设置简单，易于蚁群算法应用到其它组合优化问题的求解。

(3)TSP 的遗传算法求解

a)实验结果

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	31	31	39	80	78	72	80	10	93	82	48	62	96	70	52	90	91	91	41	61	78	46	5	57	23	10	97	24	68
1	31	0	50	41	37	80	24	82	71	70	46	61	83	54	42	92	5	80	33	57	24	62	64	89	63	34	21	22	15	57
2	31	50	0	57	23	51	17	66	6	93	22	67	5	81	75	23	83	15	47	26	13	88	31	73	19	4	73	37	20	22
3	39	41	57	0	79	53	29	92	21	54	66	4	30	65	14	97	41	87	48	35	28	59	83	59	82	86	64	74	49	26
4	80	37	23	79	0	96	58	41	98	57	14	60	18	93	78	19	11	47	32	96	15	94	78	50	88	22	19	98	72	9
5	78	80	51	53	96	0	80	48	19	55	66	90	43	30	16	71	73	55	94	16	53	67	60	32	78	63	57	51	61	11
6	72	24	17	29	58	80	0	63	79	60	63	25	67	50	14	42	56	63	62	65	99	21	86	7	73	23	64	63	64	78
7	80	82	66	92	41	48	63	0	92	33	74	78	37	31	12	55	10	93	98	3	40	22	18	83	90	30	24	54	89	44
8	10	71	6	21	98	19	79	92	0	62	29	50	93	85	96	47	31	14	45	47	33	89	41	69	20	46	4	33	68	55
9	93	70	93	54	57	55	60	33	62	0	80	66	44	63	22	14	97	67	65	40	42	78	77	21	27	35	99	9	33	90
10	82	46	22	66	14	66	63	74	29	80	0	95	92	99	53	83	43	19	66	42	22	4	47	10	53	13	89	55	11	39
11	48	61	67	4	60	90	25	78	50	66	95	0	34	5	20	95	72	7	44	12	15	80	54	60	11	70	73	55	16	8
12	62	83	5	30	18	43	67	37	93	44	92	34	0	21	89	40	40	66	22	19	49	71	74	31	22	65	82	45	22	65
13	96	54	81	65	93	30	50	31	85	63	99	5	21	0	81	17	92	68	19	94	44	72	13	49	16	21	16	65	18	97
14	70	42	75	14	78	16	14	12	96	22	53	20	89	81	0	22	32	24	68	74	74	41	56	76	61	43	40	16	75	42
15	52	92	23	97	19	71	42	55	47	14	83	95	40	17	22	0	93	95	84	99	25	13	54	3	47	78	62	37	86	24
16	90	5	83	41	11	73	56	10	31	97	43	72	40	92	32	93	0	16	75	59	16	73	28	38	76	27	54	19	53	25
17	91	80	15	87	47	55	63	93	14	67	19	7	66	68	24	95	16	0	73	22	25	55	31	79	51	37	31	27	97	30
18	91	33	47	48	32	94	62	98	45	65	66	44	22	19	68	84	75	73	0	34	80	45	15	64	14	7	16	69	69	99
19	41	57	26	35	96	16	65	3	47	40	42	12	19	94	74	99	59	22	34	0	48	70	5	68	61	75	96	47	73	92
20	61	24	13	28	15	53	99	40	33	42	22	15	49	44	74	25	16	25	80	48	0	98	73	87	71	58	99	11	19	10
21	78	62	88	59	94	67	21	22	89	78	4	80	71	72	41	13	73	55	45	70	98	0	3	8	77	64	21	42	48	12
22	46	64	31	83	78	60	86	18	41	77	47	54	74	13	56	54	28	31	15	5	73	3	0	61	88	12	88	66	10	57
23	5	89	73	59	50	32	7	83	69	21	10	60	31	49	76	3	38	79	64	68	87	8	61	0	64	14	37	3	27	51
24	57	63	19	82	88	78	73	90	20	27	53	11	22	16	61	47	76	51	14	61	71	77	88	64	0	47	4	58	96	9
25	23	34	4	86	22	63	23	30	46	35	13	70	65	21	43	78	27	37	7	75	58	64	12	14	47	0	85	49	86	60
26	10	21	73	64	19	57	64	24	4	99	89	73	82	16	40	62	54	31	16	96	99	21	88	37	4	85	0	46	73	75
27	97	22	37	74	98	51	63	54	33	9	55	55	45	65	16	37	19	27	69	47	11	42	66	3	58	49	46	0	19	50
28	24	15	20	49	72	61	64	89	68	33	11	16	22	18	75	86	53	97	69	73	19	48	10	27	96	86	73	19	0	50
29	68	57	22	26	9	11	78	44	55	90	39	8	65	97	42	24	25	30	99	92	10	12	57	51	9	60	75	50	50	0

图 1 随机产生的 30*30 的城市距离矩阵，
实际是浮点数，输出时仅输出了整数部分

初始种群0: 24; 9; 28; 1; 14; 10; 19; 23; 21; 12; 15; 22; 13; 18; 6; 25; 17; 4; 20; 29; 16; 5; 0; 3; 27; 2; 7; 11; 8; 26;
初始种群1: 7; 18; 12; 5; 10; 19; 14; 4; 0; 15; 17; 26; 8; 16; 29; 13; 20; 21; 28; 1; 23; 2; 3; 25; 22; 6; 27; 9; 11; 24;
初始种群2: 13; 16; 6; 7; 0; 12; 1; 24; 25; 3; 19; 17; 14; 20; 15; 8; 5; 18; 21; 22; 2; 28; 10; 9; 26; 23; 11; 4; 29; 27;
初始种群3: 10; 16; 7; 24; 8; 23; 25; 5; 13; 12; 2; 29; 1; 20; 15; 0; 6; 19; 4; 26; 27; 3; 14; 18; 28; 22; 9; 17; 21; 11;
初始种群4: 27; 18; 12; 25; 16; 23; 1; 3; 20; 2; 19; 11; 7; 0; 9; 29; 14; 6; 26; 10; 17; 15; 22; 28; 13; 8; 24; 5; 4; 21;
初始种群5: 11; 26; 13; 14; 4; 18; 23; 6; 16; 17; 9; 25; 15; 5; 0; 21; 29; 28; 2; 7; 24; 20; 12; 1; 27; 3; 8; 19; 10; 22;
初始种群6: 14; 27; 10; 11; 25; 13; 24; 3; 9; 29; 20; 0; 26; 18; 19; 23; 22; 5; 12; 2; 28; 16; 15; 7; 4; 8; 6; 21; 17; 1;
初始种群7: 14; 17; 23; 6; 29; 8; 20; 9; 12; 13; 19; 0; 28; 7; 16; 4; 15; 3; 26; 10; 5; 24; 11; 27; 21; 25; 2; 22; 1; 18;
初始种群8: 23; 10; 12; 8; 25; 9; 19; 16; 3; 2; 20; 28; 17; 15; 7; 21; 0; 18; 27; 24; 4; 11; 29; 6; 26; 5; 22; 13; 14; 1;
初始种群9: 5; 28; 13; 4; 19; 23; 29; 21; 17; 16; 3; 15; 24; 7; 26; 27; 0; 18; 6; 8; 11; 20; 12; 14; 10; 9; 22; 2; 25; 1;
初始种群10: 26; 5; 12; 2; 10; 24; 17; 22; 21; 16; 3; 27; 11; 28; 29; 0; 8; 23; 18; 6; 7; 4; 14; 25; 9; 13; 20; 19; 1; 15;
初始种群11: 27; 1; 5; 14; 4; 24; 8; 11; 28; 2; 17; 19; 25; 29; 26; 16; 20; 0; 12; 3; 6; 22; 15; 18; 7; 10; 21; 9; 23; 13;
初始种群12: 11; 6; 12; 26; 21; 1; 8; 9; 2; 28; 22; 7; 24; 13; 15; 3; 20; 18; 5; 29; 23; 27; 25; 4; 19; 14; 17; 0; 16; 10;
初始种群13: 4; 3; 12; 2; 10; 17; 14; 25; 1; 5; 29; 11; 21; 15; 16; 26; 19; 20; 22; 9; 24; 6; 23; 0; 7; 13; 28; 27; 8; 18;
初始种群14: 17; 18; 14; 0; 2; 23; 27; 10; 12; 21; 1; 26; 19; 5; 15; 25; 7; 8; 3; 6; 28; 13; 22; 16; 4; 20; 29; 9; 24; 11;
初始种群15: 17; 15; 0; 27; 14; 3; 18; 1; 12; 21; 9; 29; 5; 24; 16; 8; 7; 25; 20; 4; 10; 13; 19; 22; 2; 26; 23; 6; 28; 11;
初始种群16: 12; 20; 6; 13; 8; 4; 17; 25; 27; 24; 21; 0; 29; 14; 7; 18; 23; 2; 22; 26; 15; 10; 28; 11; 3; 9; 5; 1; 16; 19;
初始种群17: 13; 18; 5; 19; 29; 26; 23; 20; 28; 16; 8; 4; 3; 0; 25; 22; 21; 11; 6; 14; 2; 1; 27; 15; 7; 9; 17; 10; 12; 24;
初始种群18: 18; 24; 10; 12; 21; 23; 1; 8; 26; 16; 17; 22; 29; 20; 27; 28; 3; 25; 5; 14; 19; 4; 0; 9; 15; 6; 11; 7; 2; 13;
初始种群19: 23; 13; 6; 17; 14; 5; 4; 28; 15; 9; 19; 20; 24; 3; 2; 16; 27; 25; 26; 1; 0; 18; 10; 12; 11; 21; 7; 22; 29; 8;
初始种群20: 21; 7; 12; 19; 5; 29; 23; 25; 28; 11; 27; 20; 9; 24; 26; 8; 16; 1; 18; 17; 14; 13; 10; 2; 22; 0; 3; 6; 4; 15;
初始种群21: 13; 0; 25; 6; 5; 8; 12; 22; 15; 2; 29; 14; 19; 26; 9; 1; 24; 16; 4; 28; 7; 10; 11; 18; 17; 3; 23; 21; 20; 27;
初始种群22: 7; 0; 20; 18; 13; 21; 5; 22; 29; 25; 14; 23; 1; 27; 12; 10; 24; 28; 26; 16; 3; 15; 6; 19; 9; 11; 8; 4; 17; 2;
初始种群23: 1; 18; 9; 13; 28; 5; 7; 11; 26; 25; 22; 15; 0; 12; 4; 24; 17; 23; 16; 2; 10; 29; 19; 20; 14; 3; 27; 8; 21; 6;
初始种群24: 20; 2; 26; 17; 15; 9; 5; 24; 18; 3; 25; 21; 6; 13; 1; 4; 14; 8; 7; 22; 10; 16; 29; 12; 19; 28; 23; 11; 27; 0;
初始种群25: 4; 25; 18; 17; 9; 21; 8; 3; 6; 1; 12; 13; 29; 24; 11; 0; 22; 23; 7; 14; 16; 5; 2; 10; 27; 19; 15; 26; 28; 20;
初始种群26: 24; 3; 10; 17; 7; 21; 0; 18; 14; 5; 15; 27; 23; 12; 11; 2; 1; 16; 22; 9; 8; 28; 26; 4; 20; 6; 13; 25; 19; 29;
初始种群27: 13; 25; 28; 26; 7; 19; 8; 16; 18; 14; 21; 1; 12; 11; 27; 15; 23; 17; 24; 22; 2; 3; 10; 9; 20; 0; 6; 5; 29; 4;

图 2 初始化种群，个体数量为 100

第0次迭代：
该次迭代最好的路径： 21; 7; 12; 19; 5; 29; 23; 25; 28; 11; 27; 20; 9; 24; 26; 8; 16; 1; 18; 17; 14; 13; 10; 2; 22; 0; 3; 6; 4; 15;幸存概率：0.010033900864022423
该次迭代最低消耗：1034.5789484848017
第1次迭代：
该次迭代最好的路径： 21; 7; 12; 19; 5; 29; 23; 25; 28; 11; 27; 20; 9; 24; 26; 8; 16; 1; 18; 17; 14; 13; 10; 2; 22; 0; 3; 6; 4; 15;幸存概率：0.010032995646942182
该次迭代最低消耗：1034.5789484848017
第2次迭代：
该次迭代最好的路径： 21; 7; 12; 19; 5; 29; 23; 25; 28; 11; 27; 20; 9; 24; 26; 8; 16; 1; 18; 17; 14; 13; 10; 2; 22; 0; 3; 6; 4; 15;幸存概率：0.010032855445125157
该次迭代最低消耗：1034.5789484848017
第3次迭代：
该次迭代最好的路径： 21; 7; 12; 19; 5; 29; 23; 25; 28; 11; 27; 20; 9; 24; 26; 8; 16; 1; 18; 17; 14; 13; 10; 2; 22; 0; 3; 6; 4; 15;幸存概率：0.010032602925102338
该次迭代最低消耗：1034.5789484848017
第4次迭代：
该次迭代最好的路径： 3; 28; 23; 27; 26; 8; 4; 25; 1; 7; 6; 21; 22; 13; 24; 11; 12; 2; 5; 14; 19; 18; 0; 9; 15; 17; 16; 20; 29; 10;幸存概率：0.010026538468320637
该次迭代最低消耗：1161.8784258970604
第5次迭代：
该次迭代最好的路径： 29; 3; 11; 16; 27; 17; 6; 5; 10; 20; 8; 19; 21; 7; 15; 23; 25; 4; 12; 2; 0; 1; 26; 18; 13; 14; 9; 28; 24; 22;幸存概率：0.010024486173706366
该次迭代最低消耗：1179.5698606665949
第6次迭代：
该次迭代最好的路径： 20; 4; 15; 27; 17; 0; 2; 24; 18; 3; 25; 23; 29; 14; 26; 8; 16; 11; 22; 7; 21; 10; 28; 12; 5; 19; 9; 1; 6; 13;幸存概率：0.010031735395273104
该次迭代最低消耗：1075.979454713347
第7次迭代：
该次迭代最好的路径： 20; 9; 28; 27; 17; 0; 2; 24; 18; 3; 25; 23; 29; 14; 26; 8; 16; 11; 22; 7; 21; 10; 5; 15; 19; 13; 6; 1; 4; 12;幸存概率：0.010018398222367364
该次迭代最低消耗：1287.5948206666887
第8次迭代：
该次迭代最好的路径： 25; 10; 26; 24; 3; 27; 4; 28; 0; 15; 16; 22; 29; 20; 8; 5; 14; 9; 7; 21; 6; 1; 18; 12; 19; 2; 13; 11; 17; 23;幸存概率：0.010024596862459408
该次迭代最低消耗：1186.7954430714642
第9次迭代：
该次迭代最好的路径： 22; 25; 9; 14; 20; 0; 5; 7; 17; 11; 28; 27; 15; 13; 1; 4; 2; 8; 19; 10; 3; 16; 21; 6; 23; 24; 26; 18; 12; 29;幸存概率：0.010025760476273 5
第93次迭代：
该次迭代最好的路径： 9; 5; 8; 14; 3; 20; 17; 13; 1; 26; 7; 12; 16; 21; 15; 0; 24; 11; 19; 18; 22; 25; 23; 2; 4; 29; 10; 28; 27; 6;幸存概率：0.010029870437380758
该次迭代最低消耗：1085.7876284385554
第94次迭代：
该次迭代最好的路径： 6; 11; 17; 26; 2; 20; 7; 8; 13; 28; 10; 4; 16; 3; 14; 5; 19; 15; 0; 9; 18; 27; 24; 29; 12; 23; 25; 22; 21; 1;幸存概率：0.01002330785212398
该次迭代最低消耗：1176.8352044113992
第95次迭代：
该次迭代最好的路径： 6; 11; 17; 26; 2; 20; 7; 8; 13; 28; 10; 4; 16; 3; 14; 5; 19; 15; 0; 12; 25; 29; 24; 27; 9; 23; 18; 22; 21; 1;幸存概率：0.010029062772890755
该次迭代最低消耗：1125.2006760138477
第96次迭代：
该次迭代最好的路径： 1; 21; 6; 19; 11; 17; 2; 0; 12; 3; 25; 22; 10; 20; 28; 15; 13; 23; 5; 9; 7; 16; 24; 18; 4; 14; 29; 8; 26; 27;幸存概率：0.010026530500489544
该次迭代最低消耗：1155.8899100107433
第97次迭代：
该次迭代最好的路径： 1; 21; 6; 19; 11; 17; 2; 0; 12; 3; 25; 22; 10; 20; 28; 15; 13; 23; 5; 9; 7; 16; 24; 18; 4; 14; 29; 8; 26; 27;幸存概率：0.010026381041999575
该次迭代最低消耗：1155.8899100107433
第98次迭代：
该次迭代最好的路径： 1; 21; 6; 19; 11; 17; 2; 0; 12; 3; 25; 22; 10; 20; 28; 15; 13; 23; 5; 9; 7; 16; 24; 18; 4; 14; 29; 8; 26; 27;幸存概率：0.010026679723900351
该次迭代最低消耗：1155.8899100107433
第99次迭代：
该次迭代最好的路径： 11; 26; 2; 23; 15; 9; 24; 21; 27; 1; 16; 4; 10; 12; 28; 5; 17; 14; 3; 8; 22; 25; 13; 20; 29; 0; 18; 7; 19; 6;幸存概率：0.010022227915974262
该次迭代最低消耗：1216.8210147404095

图 3 100 次迭代求解

最好的路径： 2; 28; 27; 16; 1; 5; 8; 4; 20; 0; 17; 11; 24; 29; 3; 21; 22; 12; 18; 13; 15; 23; 10; 25; 7; 19; 14; 9; 26; 6;幸存概率：1.0134413609798742E-4
最低消耗：1024.8995264989378

图 4 在 100 次迭代中找到的最优解

b)结果分析

遗传算法是从问题的解的集合中开始搜索，而不是从单个解开始。传统优化算法从单个初始值迭代求解，这就容易陷入局部最优解。遗传算法从串集开始搜索，覆盖面大，利于全局择优。

遗传算法有极强的容错能力。它的初始串集本身就带有与最优解甚远的信息，通过选择、交叉、变异操作能迅速排除与最优解相差极大的串。

遗传算法中的选择、交叉和变异都是随机操作，而不是确定的精确规则。这说明遗传算法是采用随机方法进行最优解搜索，选择体现了最优解迫近，交叉体现了最优解产生，变异体现了全局最优阶的覆盖。

七、实验小结

这次实验最大的收获是搞懂了蚁群算法和遗传算法。在参照网上的博客后，自己动手编写了这两个算法。在编写过程中加深了对算法原理的理解。这是一件让自己兴奋的事情。这两个算法最大的不足是不能从 txt 文件中读入既定城市距离矩阵。当然这这很容易修正。因为要省时间写报告这个小问题就没有改。

至于神经网络这一块，就没有蚁群算法和遗传算法那么幸运了。代码编写完运行并不能输出好的结果。换位矩阵在少量迭代后会变成全 0 矩阵，显然是不正确的。经过调试，发现系数对网络影响很大。但是还是难求出一个好结果。会继续想这个问题的。

最后一点是，用 java 编程还是不如 matlab 来得方便。以后可以尝试用 matlab 或者调用已有库吧。