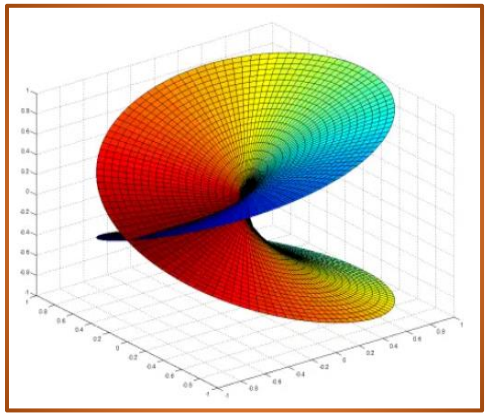


第十二讲　智能算法

云南大学

孙正宝

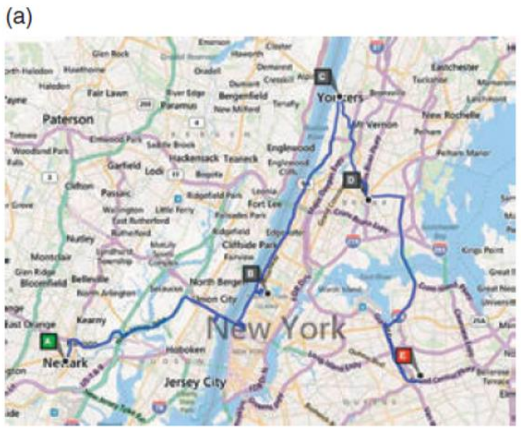
zbsun@ynu.edu.cn

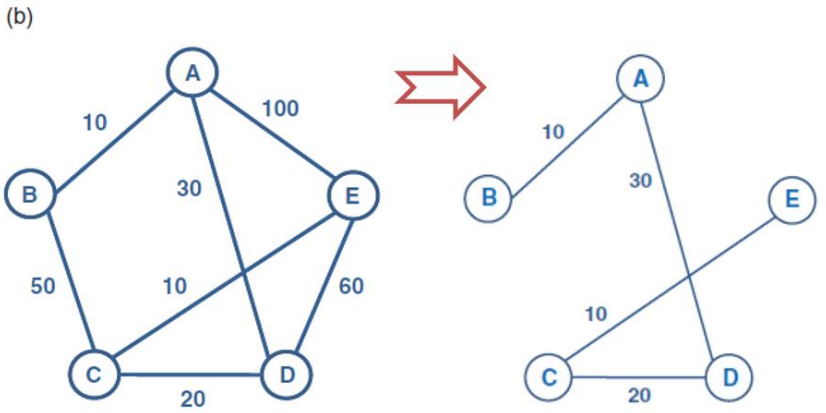
TSP问题

TSP问题描述

TSP问题（Travelling Salesman Problem，旅行商问题）是一个经典的组合优化问题。它描述了一个旅行商人需要访问多个城市，并且每个城市恰好访问一次，最后返回到起始城市，要求找到一条最短的路径。

TSP问题是一个NP-hard问题，意味着没有已知的多项式时间算法可以求解所有规模的TSP问题的最优解。因此，对于大规模的问题，通常需要使用近似算法或启发式算法求解。





Edsger Dijkstra

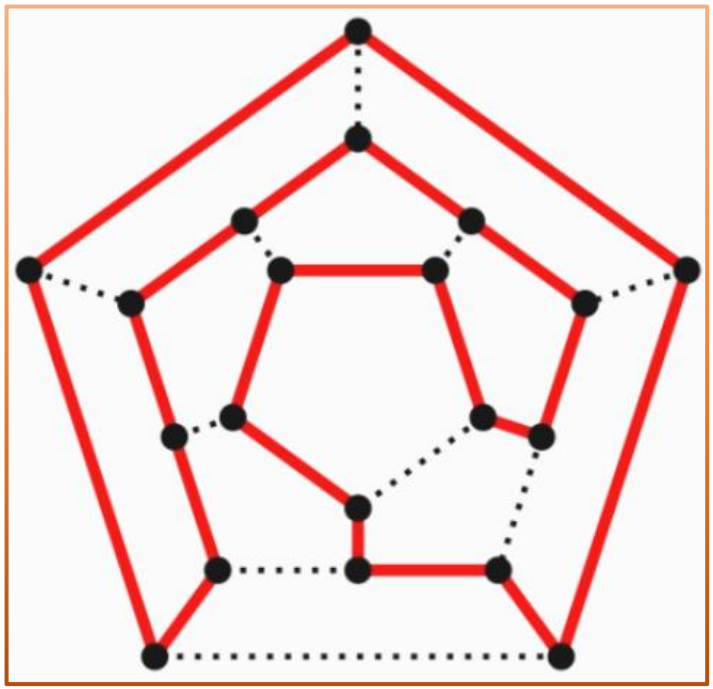
TSP问题

TSP问题数学定义

对于给定的加权无向图***G***= (***V***，***W***)，求***G***中的Hamilton回路C，使得 $w(C)$ 最小，

$$w(C) = \sum_{(v_i,v_j)\in C} w(v_i, v_j)$$

- 1859年，英国数学家、物理学家William Rowan Hamilton提出的“周游世界”问题。把一个正十二面体的二十个顶点看成地球上的二十个城市。要求游戏者沿棱线走，寻找一条经过所有结点一次且仅一次的回路。



TSP问题

TSP问题的常用求解方法

- **穷举法**：对于小规模的问题，可以通过穷举所有可能的路径组合来找到最短路径。但这种方法的时间复杂度非常高，不适合大规模问题。
- **贪婪算法**：通过局部最优选择来构建解。例如，每次选择离当前位置最近且未访问过的城市作为下一个目标。这种方法速度快，但通常只能得到近似解。
- **蚁群算法**：模拟蚂蚁觅食过程中的信息素传递和路径选择行为。蚂蚁在搜索过程中释放信息素，并根据信息素的浓度选择路径，最终找到一条较短的路径。
- **遗传算法**：借鉴生物进化过程中的遗传和变异机制来搜索最优解。通过选择、交叉和变异等操作，逐步优化解的质量。
- **模拟退火算法**：一种启发式搜索算法，通过模拟物理系统中的退火过程来寻找全局最优解，可以在一定程度上避免陷入局部最优。
- **分支限界法**：一种系统地搜索解空间的方法，通过剪枝来避免搜索不必要的分支，从而提高效率。
- **线性规划和整数规划**：将TSP问题转化为线性规划或整数规划问题，然后利用相关算法求解。这种方法可以得到精确解，但计算复杂度较高。

提　纲

一　遗传算法

二　蚁群算法

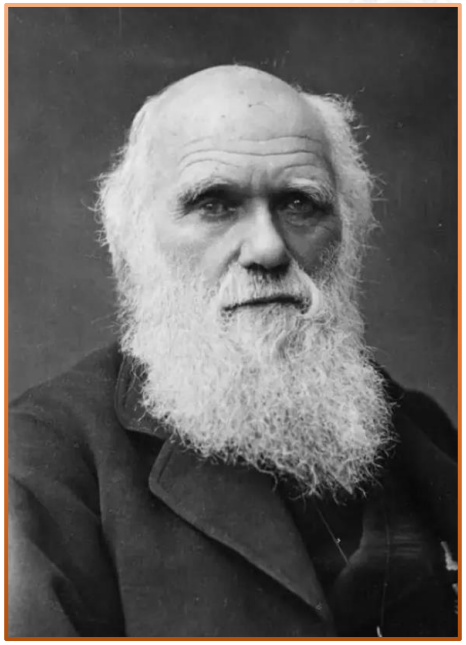
三　粒子群优化算法

四　禁忌搜索算法

五　模拟退火算法

一、遗传算法

- 遗传算法（Genetic Algorithm, GA）是受自然进化理论启发的一类搜索算法。
- 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，“物竞天择，适者生存。”
- 通过模仿自然选择和繁殖的过程，搜索最优解。
- 可以克服传统搜索和优化算法具有大量参数和复杂数学表示形式的问题。



（Charles Robert Darwin，1809年2月12日—1882年4月19日）

- 其主要特点是直接对结构对象进行操作，不存在求导和函数连续性的限定；具有内在的隐并行性和更好的全局寻优能力；采用随机寻优方法，不需要确定的规则就能自动获取和指导优化的搜索空间，自适应地调整搜索方向。

一、遗传算法

达尔文进化论的原理可概括为：

- 变异：种群中单个样本的特征(性状，属性) 不同，导致样本彼此之间有一定程度的差异；
- 遗传：某些特征可以遗传给其后代，导致后代与双亲样本具有一定程度的相似性；
- 选择：种群通常在给定的环境中争夺资源，更适应环境的个体在生存方面更具优势，因此会产生更多的后代。

- 遗传算法以一种群体中的所有个体为对象，利用随机化技术指导对一个被编码的参数空间进行搜索。

一、遗传算法

- 进化维持了种群中个体样本彼此不同。那些适应环境的个体更有可能生存，繁殖并将其性状传给下一代。这样，随着世代的更迭，物种变得更加适应其生存环境。而进化的重要推动因素是交叉 (crossover) 或重组 (recombination) 或杂交——结合双亲的特征产生后代。交叉有助于维持群体的多样性，并随着时间的推移将更好的特征融合在一起。此外，变异 (mutations) 或突变(特征的随机变异)可以通过引入偶然性的变化而在进化中发挥重要作用。
- 选择、交叉和变异（突变）构成了遗传算法的遗传操作；
- 参数编码、初始群体的设定、适应度函数的设计、遗传操作设计、控制参数设定五个要素组成了遗传算法的核心内容。

一、遗传算法

□ 基本概念：

- **基因型 (Genotype)**

在自然界中，通过基因型表征繁殖和突变，基因型是组成染色体的一组基因的集合。在遗传算法中，每个个体都由代表基因集合的染色体构成。例如，一条染色体可以表示为二进制串，其中每个位代表一个基因：

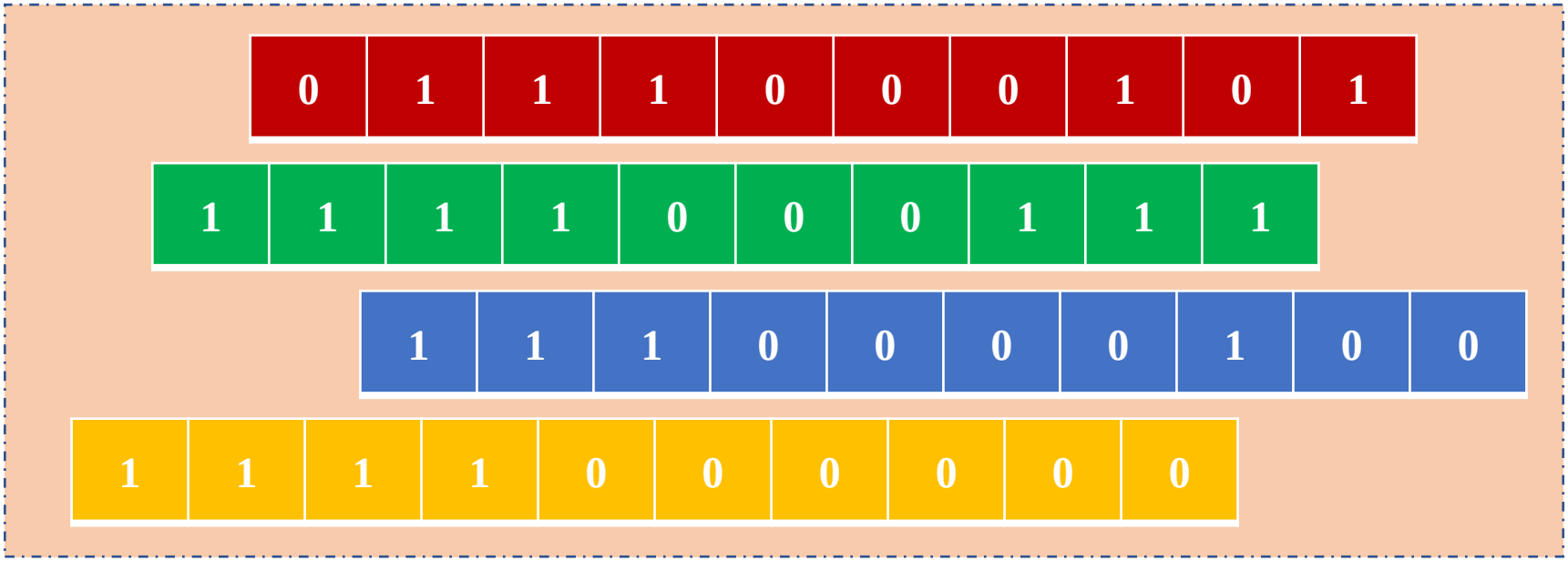
0	1	1	1	0	0	0	1	0	1

一、遗传算法

□ 基本概念：

- **种群 (Population)**

遗传算法保持大量的个体 (individuals) —— 针对当前问题的候选解集合。由于每个个体都由染色体表示，因此种群 可以看作是个体 (individuals)的集合：



一、遗传算法

基本概念：

- **适应度函数 (Fitness function)**

在算法的每次迭代中，使用适应度函数(即目标函数)对个体进行评估。适应度得分更高的个体代表了更好的解，其更有可能被选择繁殖并且其性状会在下一代中得到表现。随着遗传算法的进行，解的质量会提高，适应度会增加，一旦找到具有满意的适应度值的解，终止遗传算法。

- **选择 (Selection)**

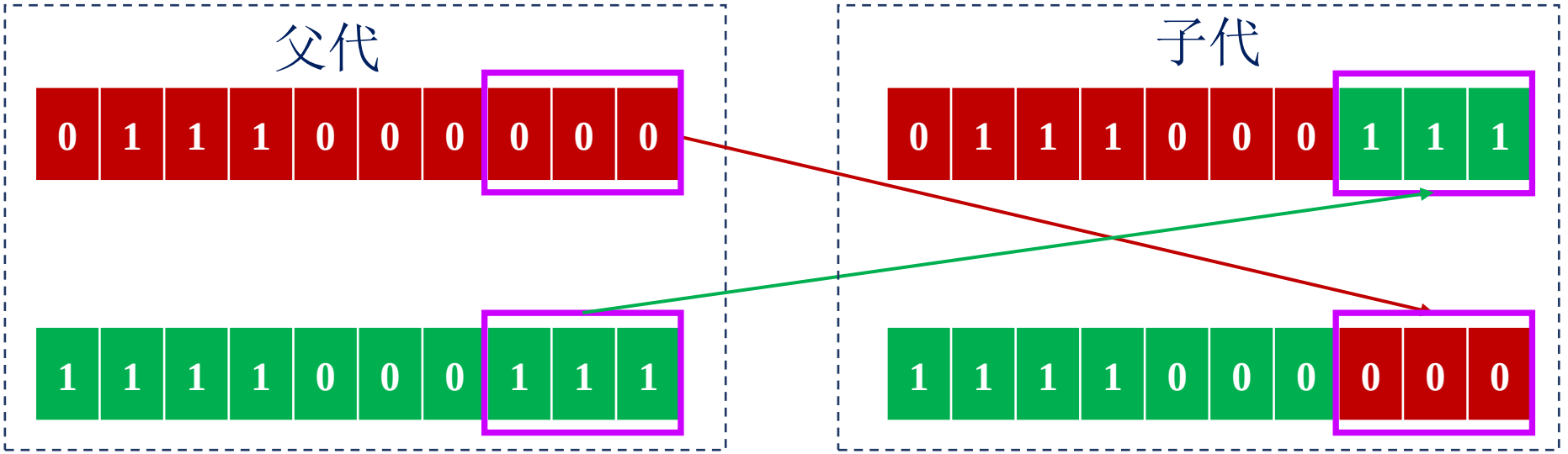
在计算出种群中每个个体的适应度后，使用选择过程来确定种群中的某个个体将用于繁殖并产生下一代，具有较高值的个体更有可能被选中，并将其遗传物质传递给下一代。选择低适应度值个体的概率较低。

一、遗传算法

基本概念：

- **交叉 (Crossover)**

为了创建一对新个体，通常将从当前代中选择的双亲样本的部分染色体互换(交叉)，以创建代表后代的两个新染色体。此操作称为交叉或重组：



- 交叉概率：多数情况下，交叉以一定的概率发生。

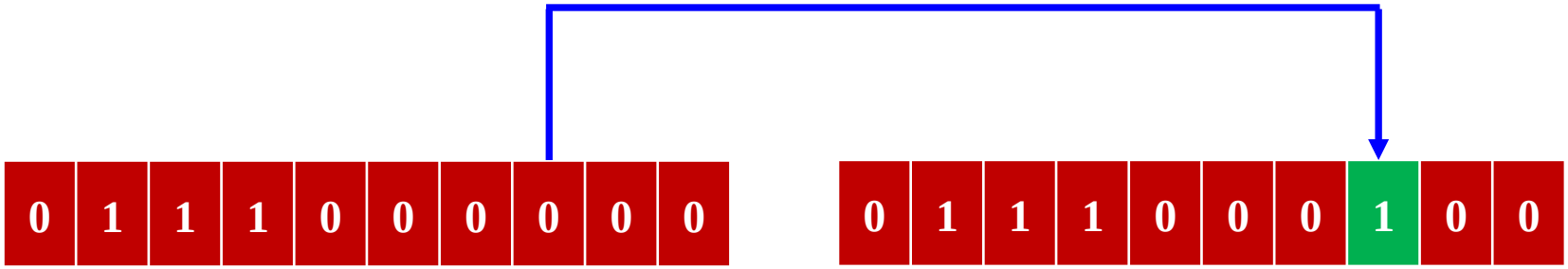
一、遗传算法

□ 基本概念：

- **突变 (Mutation)**

突变可能表现为基因的随机变化，通过随机改变一个或多个染色体值来实现的。

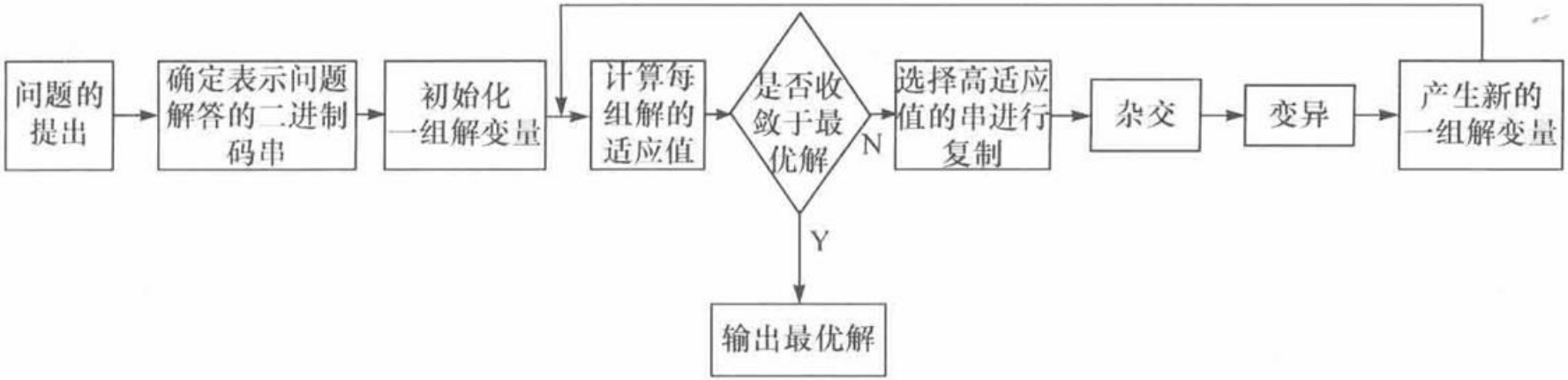
突变操作的目的是定期随机更新种群，将新模式引入染色体，并鼓励在解空间的未知区域中进行搜索。



- 突变概率：随机选择某一基因位，进行翻转。通常取值较小，如0.001。

一、遗传算法

算法思想与设计：



- ✓ 二进制编码（编码与解码）
- ✓ 初始群体设置
- ✓ 适应度函数的设计
- ✓ 控制参数设定（主要是交叉、突变概率）
- ✓ 遗传操作设计（选择、交叉、突变操作）
- ✓ 评价

一、遗传算法

算法的优缺点：

优点：

- **全局最优解：**与其他传统优化算法相比，遗传算法更有可能找到全局最优解；
- **并行性：**适应度是针对每个个体独立计算的，因此可以同时评估种群中的所有个体；而且选择、交叉和突变的操作可以分别在种群中的个体和个体对上同时进行；
- **适用性：**基于认为设计的适应度函数，对缺乏数学表达和含噪声问题具有良好的适用性；
- ……

缺点：

- 人为设定适应度函数和染色体结构；
- 超参数调优；
- 计算密集；
- ……
- ……

一、遗传算法

示例

- 遗传算法实现云南省129个县市区的遍历
- 以旅程总距离最小为目标
- 假设：任意两城市之间有直线连通路径
- 输入：129个县市区的地理坐标

```
#参数
CityNum = 129 #城市数量
MinCoordinate = 0 #二维坐标最小值
MaxCoordinate = 360 #二维坐标最大值
#GA参数
generation = 100  #迭代次数
popsize = 100   #种群大小
tournament_size = 5 #锦标赛小组大小
pc = 0.95   #交叉概率
pm = 0.1    #变异概率
```

```
#适应度函数
def calFitness(line,dis_matrix):
    dis_sum = 0
    dis = 0
    for i in range(len(line)):
        if i<len(line)-1:
            dis = dis_matrix.loc[line[i],line[i+1]]
            dis_sum = dis_sum+dis
        else:
            dis = dis_matrix.loc[line[i],line[0]]
            dis_sum = dis_sum+dis
    return round(dis_sum,1)
```

一、遗传算法

示例

选择

```python
#竞标赛选择
def tournament_select(pops,popsize,fits,tournament_size):
    new_pops,new_fits = [],[]
    while len(new_pops)<len(pops):
        tournament_list = random.sample(range(0,popsize),tournament_size)
        tournament_fit = [fits[i] for i in tournament_list]
        #转化为df方便索引
        tournament_df = pd.DataFrame([tournament_list,tournament_fit]).transpose().sort_values(by=1).reset_index(drop=True)
        #选出获胜者
        fit = tournament_df.iloc[0,1]
        pop = pops[int(tournament_df.iloc[0,0])]
        new_pops.append(pop)
        new_fits.append(fit)
    return new_pops,new_fits
```

交叉

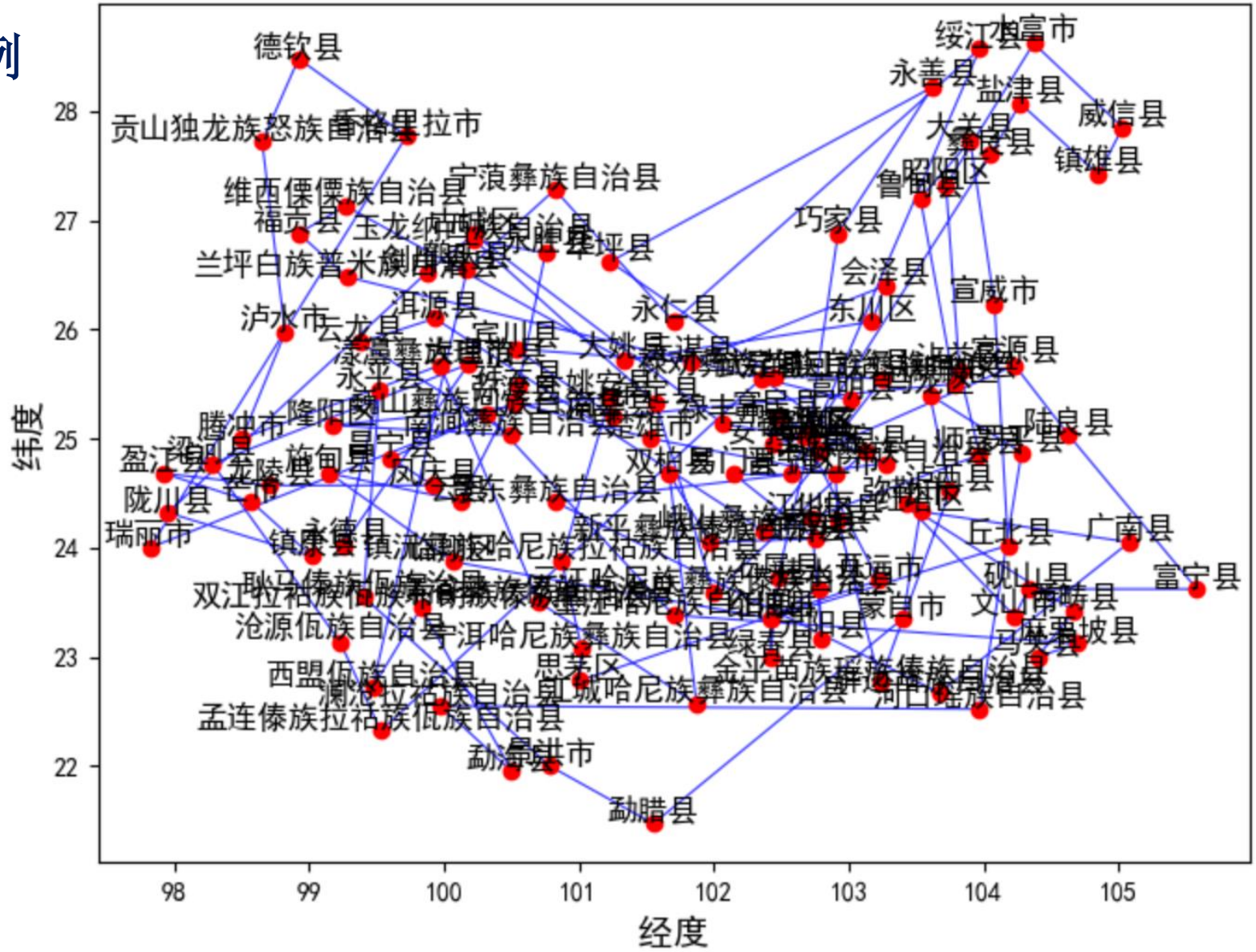
```python
#顺序交叉函数
def crossover(popsize,parent1_pops,parent2_pops,pc):
    child_pops = []
    for i in range(popsize):
        #初始化
        child = [None]*len(parent1_pops[i])
        parent1 = parent1_pops[i]
        parent2 = parent2_pops[i]
        if random.random() >= pc:
            child = parent1.copy()#随机生成一个（或者随机保留父代中的一个）
            random.shuffle(child)
        else:
            #parent1
            start_pos = random.randint(0,len(parent1)-1)
            end_pos = random.randint(0,len(parent1)-1)
            if start_pos>end_pos:
                tem_pop = start_pos
                start_pos = end_pos
                end_pos = tem_pop
            child[start_pos:end_pos+1] = parent1[start_pos:end_pos+1].copy()
            # parent2 -> child
            list1 = list(range(end_pos+1,len(parent2)))
            list2 = list(range(0,start_pos))
            list_index = list1+list2
            j = -1
            for i in list_index:
                for j in range(j+1,len(parent2)):
                    if parent2[j] not in child:
                        child[i] = parent2[j]
                        break
        child_pops.append(child)
    return child_pops
```

突变

```python
#变异
def mutate(pops,pm):
    pops_mutate = []
    for i in range(len(pops)):
        pop = pops[i].copy()
        #随机多次成对变异
        t = random.randint(1,5)
        count = 0
        while count < t:
            if random.random() < pm:
                mut_pos1 = random.randint(0,len(pop)-1)
                mut_pos2 = random.randint(0,len(pop)-1)
                if mut_pos1 != mut_pos2:
                    tem = pop[mut_pos1]
                    pop[mut_pos1] = pop[mut_pos2]
                    pop[mut_pos2] = tem
            pops_mutate.append(pop)
            count +=1
    return pops_mutate
```

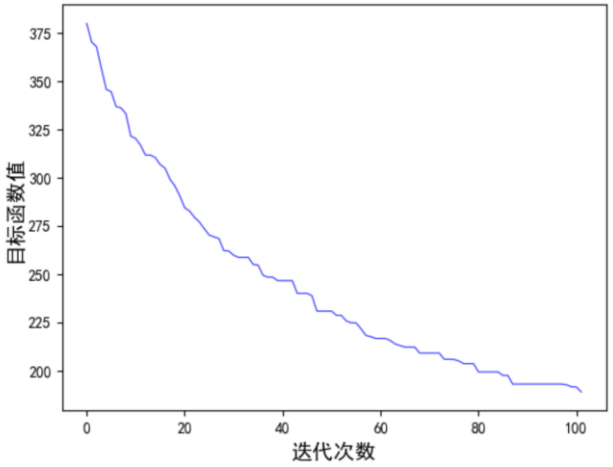
一、遗传算法

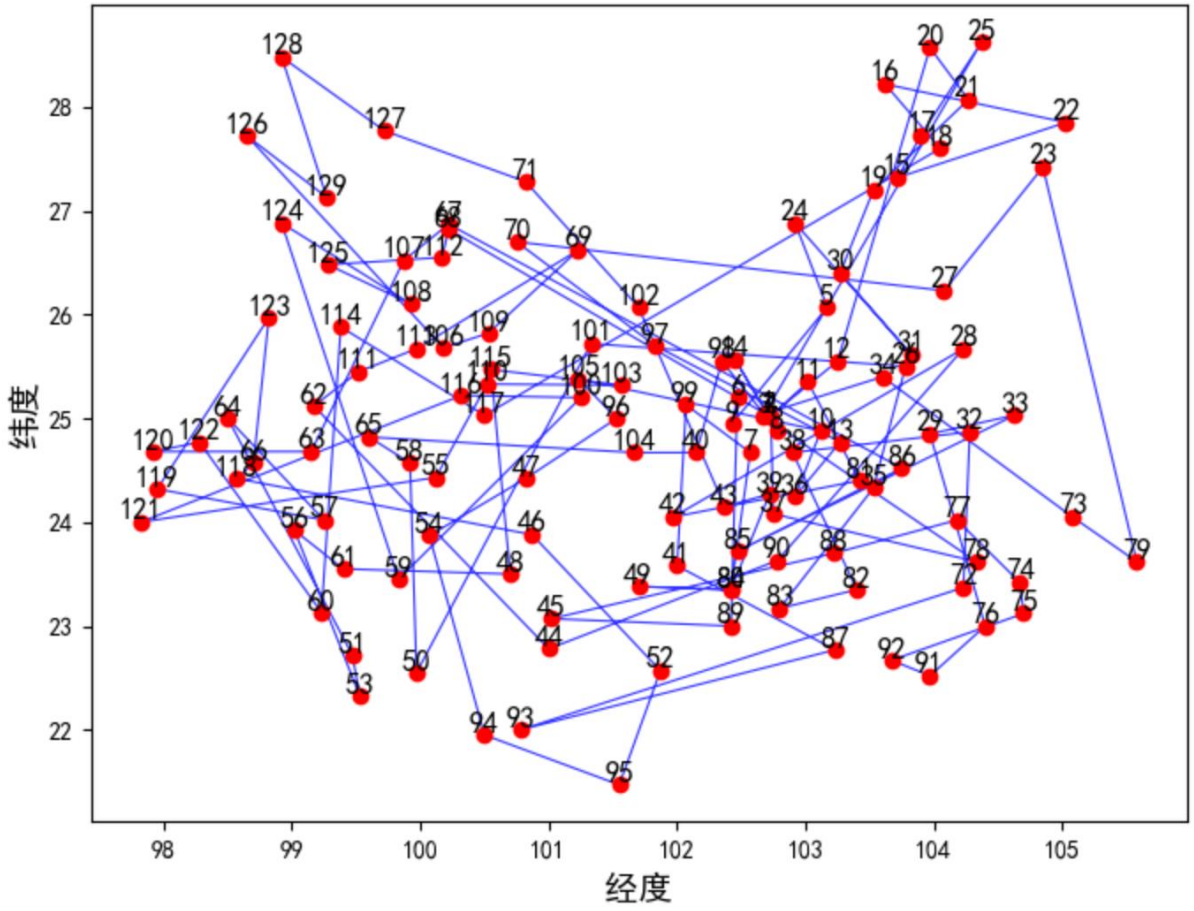
示例


德钦县
贡山独龙族怒族自治县
香格里拉市
维西傈僳族自治县
福贡县
兰坪白族普米族自治县
泸水市
宁蒗彝族自治县
巧家县
永善县
绥江县
水富市
盐津县
威信县
镇雄县
鲁甸县
会泽县
宣威市
永仁县
东川区
腾冲市
盈江县
陇川县
瑞丽市
双江拉祜族佤族布朗族傣族自治县
沧源佤族自治县
西盟佤族自治县
孟连傣族拉祜族佤族自治县
勐腊县
丘北县
广南县
富宁县
砚山县
文山壮族苗族自治州
麻栗坡县
纬度
经度
28
27
26
25
24
23
22
98
99
100
101
102
103
104
105

一、遗传算法

示例





- 最优路径为：

[11, 4, 37, 78, 3, 25, 17, 15, 22, 16, 18, 117, 114, 60, 122, 120, 63, 107, 67, 86, 85, 33, 29, 76, 91, 92, 75, 74, 77, 88, 45, 89, 6, 9, 5, 24, 31, 30, 19, 21, 20, 12, 2, 68, 112, 125, 108, 124, 59, 100, 116, 121, 55, 110, 103, 115, 48, 61, 56, 53, 51, 64, 57, 119, 123, 66, 118, 46, 52, 95, 94, 54, 47, 96, 105, 10, 36, 13, 97, 70, 27, 23, 79, 73, 34, 1, 14, 8, 98, 40, 104, 65, 58, 50, 101, 26, 35, 83, 82, 38, 32, 72, 93, 87, 41, 99, 7, 80, 49, 84, 39, 43, 102, 71, 127, 128, 129, 126, 106, 109, 69, 113, 111, 62, 44, 90, 28, 42, 81]

智能算法实验（1/2）

- 实验一：教材351页，习题1。
- 实验结果分析与评价。

提　纲

一　遗传算法

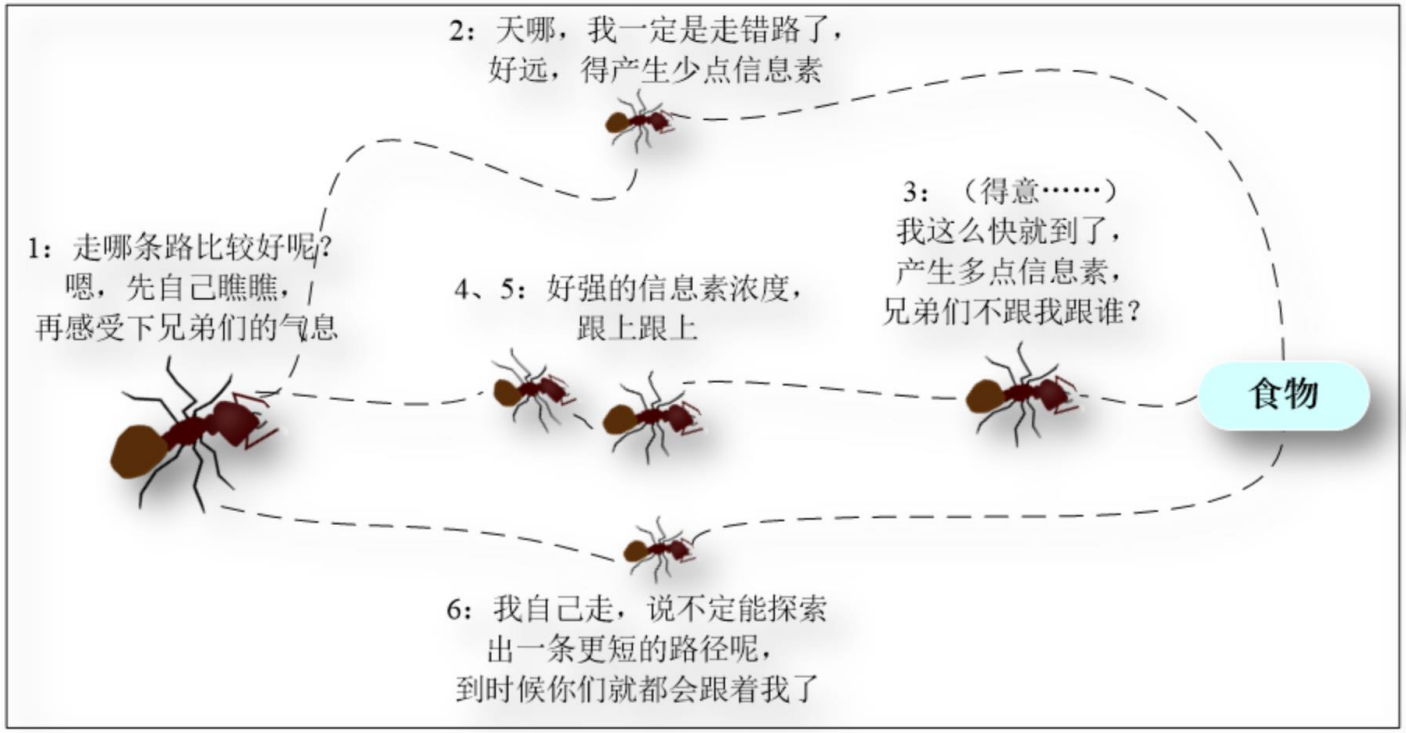
二　蚁群算法

三　粒子群优化算法

四　禁忌搜索算法

五　模拟退火算法

二、蚁群算法



- 蚁群算法是一种用来寻找优化路径的概率型算法，由意大利学者Marco Dorigo在1992年提出。这种算法的设计灵感来源于蚂蚁在寻找食物过程中发现路径的行为。
- 自然界蚂蚁群体在寻找食物的过程中，通过一种被称为信息素（Pheromone）的物质实现相互的间接通信，从而能够合作发现从蚁穴到食物源的最短路径。

二、蚁群算法

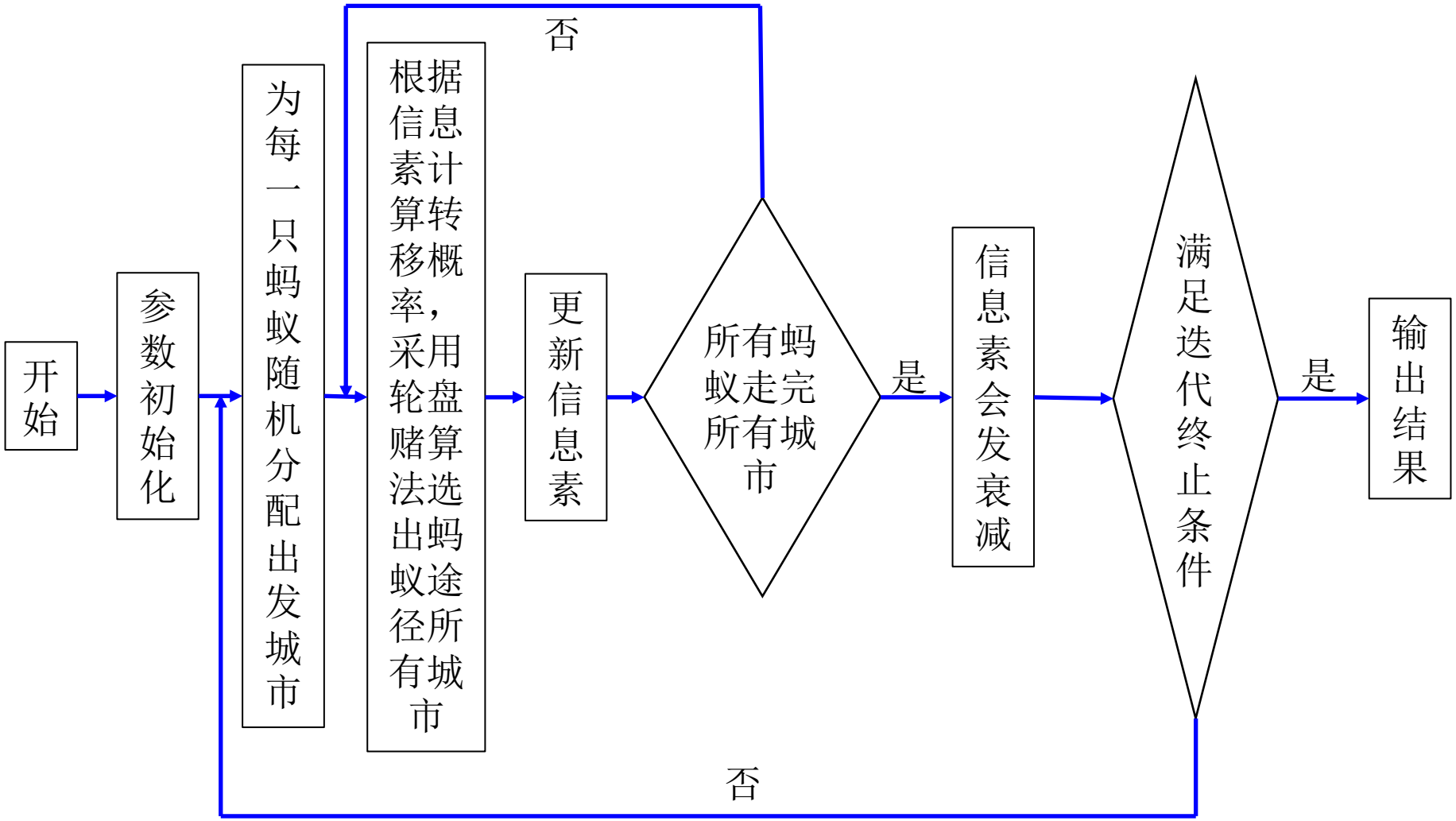
算法原理：

- 1. 蚂蚁在行走过程中会依据信息素浓度来选择道路，选择信息素较浓的路走，并且在行走的路径中会释放信息素，对于所有蚂蚁都没经过的路，则随机选择一条路走；
- 2. 蚂蚁释放的信息素浓度与长度相关，通常与路径长度成反比；
- 3. 信息素浓的路径会受到蚂蚁更大概率的选择，形成正向反馈，最短路径上的信息素浓度会越来越大，最终蚁群就都按这条最短路径走。

- 路径选择规则（转移概率）和信息素更新规则是蚁群算法的关键。

二、蚁群算法

算法设计


开始
参数初始化
为每一只蚂蚁随机分配出发城市
根据信息素计算转移概率，采用轮盘赌法选出蚂蚁途径所有城市
更新信息素
所有蚂蚁走完所有城市
否
是
信息素会发衰减
满足迭代终止条件
否
是
输出结果

二、蚁群算法

□ 算法主要步骤——路径选择规则

设蚂蚁的数量为 m，$d_{ij}(i,j=1,2,3,\cdots,n)$ 表示城市 i 和城市 j 之间的距离，$\tau_{ij}(t)$ 表示 t 时刻在城市 i 与城市 j 连线上信息素的强度。初始时刻，各条路径上信息素的强度相同，设 $\tau_{ij}(0)=C$ (C 为常数)。蚂蚁 k(k=1, 2, …, m)在运动过程中，根据各条路径上的信息素的强度决定转移方向。$P_{ij}^{k}(t)$ 表示 t 时刻蚂蚁 k 从城市 i 转移到城市 j 的概率，其计算公式为

$$P_{ij}^{k}=\begin{cases}\dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum\limits_{s\in \text{allowed}_k}\tau_{is}^{\alpha}(t)\eta_{is}^{\beta}(t)} & j\notin \text{tabu}_k\\ 0 & \text{其他}\end{cases} \tag{5.2.1}$$

其中，tabu_k(k=1, 2, …, m)表示蚂蚁 k 已走过城市的集合；$\text{allowed}_k=\{0,1,\cdots,n-1\}-\text{tabu}_k$ 表示蚂蚁 k 下一步允许选择的城市(开始时 tabu_k 中只有一个元素，即蚂蚁 k 的出发城市，随着选择的进行，tabu_k 中的元素不断增加)；η_{ij} 表示能见度(又称期望程度)，在 TSP 问题中通常取路径 i–j 长度(城市 i 和 j 之间距离)的倒数，即 $\eta_{ij}=1/d_{ij}$；α、β 分别表示调节信息素强度 τ_{ij} 与能见度 η_{ij} 的相对重要程度。因此转移概率 P 是能见度 η_{ij} 和信息素强度 τ_{ij} 的权衡。

二、蚁群算法

算法主要步骤——信息素更新规则

随着时间的推移，以前留在各条路径上的信息素逐渐消失，设信息素的保留系数为 $\rho(0<\rho<1)$，它体现了信息素强度的持久性；而 $1-\rho$则表示信息素的挥发程度。经过 Δt 时刻，蚂蚁完成一次循环，各路径上信息素的强度要根据式(5.2.2)和式(5.2.3)进行调整：

$$\tau_{ij}(t+\Delta t)=\rho\cdot\tau_{ij}(t)+\Delta\tau_{ij}(\Delta t) \tag{5.2.2}$$

$$\Delta\tau_{ij}(\Delta t)=\sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{5.2.3}$$

其中，Δt 表示第 k 只蚂蚁在本次循环中留在路径 i–j 上的信息素的强度，$\Delta\tau_{ij}(\Delta t)$ 表示本次循环中 m 只蚂蚁在路径 i–j 上所留下的信息素强度之和。

二、蚁群算法

算法主要步骤——信息素更新规则

- 蚁周系统模型（Ant Cycle System, ACS）
- 蚁量系统模型（ Ant Quantity System, AQS ）
- 蚁密系统模型（ Ant Density System, ADS ）
- 区别主要在于信息更新的时刻和量不同。

在 ACS 中；$\Delta\tau_{ij}^{k}$ 为

$$\Delta\tau_{ij}^{k}=\begin{cases}\dfrac{Q}{L_k} & \text{若第}k\text{只蚂蚁经过路径}i-j\\ 0 & \text{否则}\end{cases}$$

在 AQS 和 ADS 中，$\Delta\tau_{ij}^{k}$ 分别为

$$\Delta\tau_{ij}^{k}=\begin{cases}\dfrac{Q}{d_{ij}} & \text{若第}k\text{只蚂蚁经过路径}i-j\\ 0 & \text{否则}\end{cases}$$

$$\Delta\tau_{ij}^{k}=\begin{cases}Q & \text{若第}k\text{只蚂蚁经过路径}i-j\\ 0 & \text{否则}\end{cases}$$

二、蚁群算法

例题

- 4个城市的TSP问题，距离矩阵如下：

$$D=(d_{ij})=\begin{bmatrix}0 & 3 & 1 & 2\\ 3 & 0 & 5 & 4\\ 1 & 5 & 0 & 2\\ 2 & 4 & 2 & 0\end{bmatrix}$$

图 5-3　距离矩阵

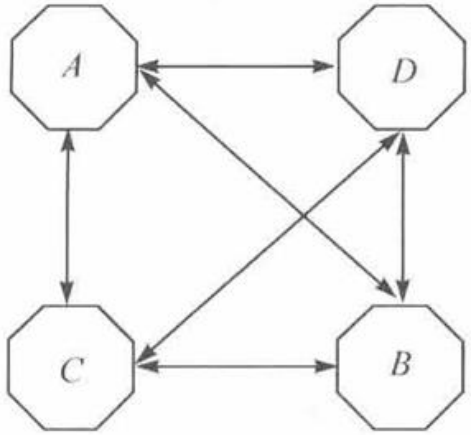


图 5-4　有权重边的图

二、蚁群算法

例题

假设共 m=3 只蚂蚁，参数 $\alpha=1,\beta=2,\rho=0.5$ 。

步骤 1：初始化。

首先使用贪婪算法得到路径($ACDBA$)，则 C_{nn}=1+2+4+3=10，求得 $\tau_0=m/C_{nn}=0.3$ ，则

$$\tau(0)=\left(\tau_{ij}(0)\right)=\begin{bmatrix}0 & 0.3 & 0.3 & 0.3\\ 0.3 & 0 & 0.3 & 0.3\\ 0.3 & 0.3 & 0 & 0.3\\ 0.3 & 0.3 & 0.3 & 0\end{bmatrix}$$

步骤 2：为每只蚂蚁随机选择出发城市，假设蚂蚁 1 选择城市 A，蚂蚁 2 选择城市 B，蚂蚁 3 选择城市 D。

步骤 3.1：为每只蚂蚁选择下一个访问城市，仅以蚂蚁 1 为例，当前城市 i=A，可访问城市集合 $J_1(i)=\{B,C,D\}$，计算蚂蚁 1 访问各个城市的概率：

$$D=\left(d_{ij}\right)=\begin{bmatrix}0 & 3 & 1 & 2\\ 3 & 0 & 5 & 4\\ 1 & 5 & 0 & 2\\ 2 & 4 & 2 & 0\end{bmatrix}$$

$$A\Rightarrow\begin{cases}B:\tau_{AB}^{\alpha}\times\eta_{AB}^{\beta}=0.3^1\times(1/3)^2=0.033\\ C:\tau_{AC}^{\alpha}\times\eta_{AC}^{\beta}=0.3^1\times(1/1)^2=0.300\\ D:\tau_{AD}^{\alpha}\times\eta_{AD}^{\beta}=0.3^1\times(1/2)^2=0.075\end{cases}$$

$$P(B)=\frac{0.033}{0.033+0.3+0.075}=0.08$$

$$P(C)=\frac{0.300}{0.033+0.3+0.075}=0.74$$

$$P(D)=\frac{0.075}{0.033+0.3+0.075}=0.18$$

用轮盘赌法选择下一个访问城市，假设产生的随机数 q=0.05，则蚂蚁 1 会选择城市 B，用同样的方法为蚂蚁 2 和蚂蚁 3 选择下一访问城市，假设蚂蚁 2 选择城市 D，蚂蚁 3 选择城市 A。

二、蚁群算法

例题

步骤 3.2：为每只蚂蚁选择下一访问城市，仅以蚂蚁 1 为例，当前城市 i=B，路径记忆向量 $R'=(AB)$，可访问城市集合 $J_1(i)=\{C, D\}$，计算蚂蚁 1 访问 C，D 城市的概率。

$$B \Rightarrow \begin{cases} C: \tau_{BC}^{\alpha} \times \eta_{BC}^{\beta} = 0.3^1 \times (1/5)^2 = 0.012 \\ D: \tau_{BD}^{\alpha} \times \eta_{BD}^{\beta} = 0.3^1 \times (1/4)^2 = 0.019 \end{cases}$$

$$P(C) = \frac{0.012}{0.012 + 0.019} = 0.39$$

$$P(D) = \frac{0.019}{0.012 + 0.019} = 0.61$$

用轮盘赌法选择下一个访问城市，假设产生的随机数 q=0.67，则蚂蚁 1 会选择城市 D，用同样的方法为蚂蚁 2 和蚂蚁 3 选择下一访问城市，假设蚂蚁 2 选择城市 C，蚂蚁 3 选择城市 C。

此时，所有蚂蚁的路径都已构造完毕。

蚂蚁 1：$A \to B \to D \to C \to A$

蚂蚁 2：$B \to D \to C \to A \to B$

蚂蚁 3：$D \to A \to C \to B \to D$

二、蚁群算法

例题

步骤 4：信息素更新。

计算每只蚂蚁构建的路径长度：C_1=3+4+2+1=10；C_2=4+2+1+3=10；C_3=2+1+5+4=12。更新每条边上的信息素：

$$\tau_{AB}=(1-\rho)\times\tau_{AB}+\sum_{k=1}^{3}\Delta\tau_{AB}^{k}=0.5\times0.3+\left(\frac{1}{10}+\frac{1}{10}\right)=0.35$$

$$\tau_{AC}=(1-\rho)\times\tau_{AC}+\sum_{k=1}^{3}\Delta\tau_{AC}^{k}=0.5\times0.3+\frac{1}{12}=0.16$$

……

步骤 5：如果满足结束条件，则输出全局最优结果并结束程序，否则，转向步骤 2 继续执行。

二、蚁群算法

示例

- 蚁群算法实现云南省129个县市区的遍历
- 以旅程总距离最小为目标
- 假设：任意两城市之间有直线连通路径
- 输入：129个县市区的地理坐标

```
#计算路径距离，即目标函数
def calFitness(line,dis_matrix):
    dis_sum = 0
    dis = 0
    for i in range(len(line)-1):
        dis = dis_matrix.loc[line[i],line[i+1]]#计算距离
        dis_sum = dis_sum+dis
    dis = dis_matrix.loc[line[-1],line[0]]
    dis_sum = dis_sum+dis

    return round(dis_sum,1)
```

```
#参数
CityNum = 129 #城市数量
MinCoordinate = 0 #二维坐标最小值
MaxCoordinate = 360 #二维坐标最大值
iterMax = 100 #迭代次数
iterI = 1 #当前迭代次数
#ACO参数
antNum = 50 #蚂蚁数量
alpha = 2 #信息素重要程度因子
beta = 1 #启发函数重要程度因子
rho = 0.2 #信息素挥发因子
Q = 100.0 #常数
```

二、蚁群算法

示例

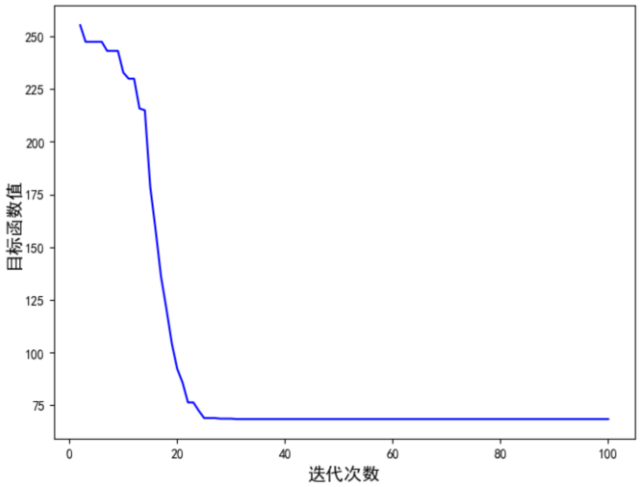
- 蚁周系统模型

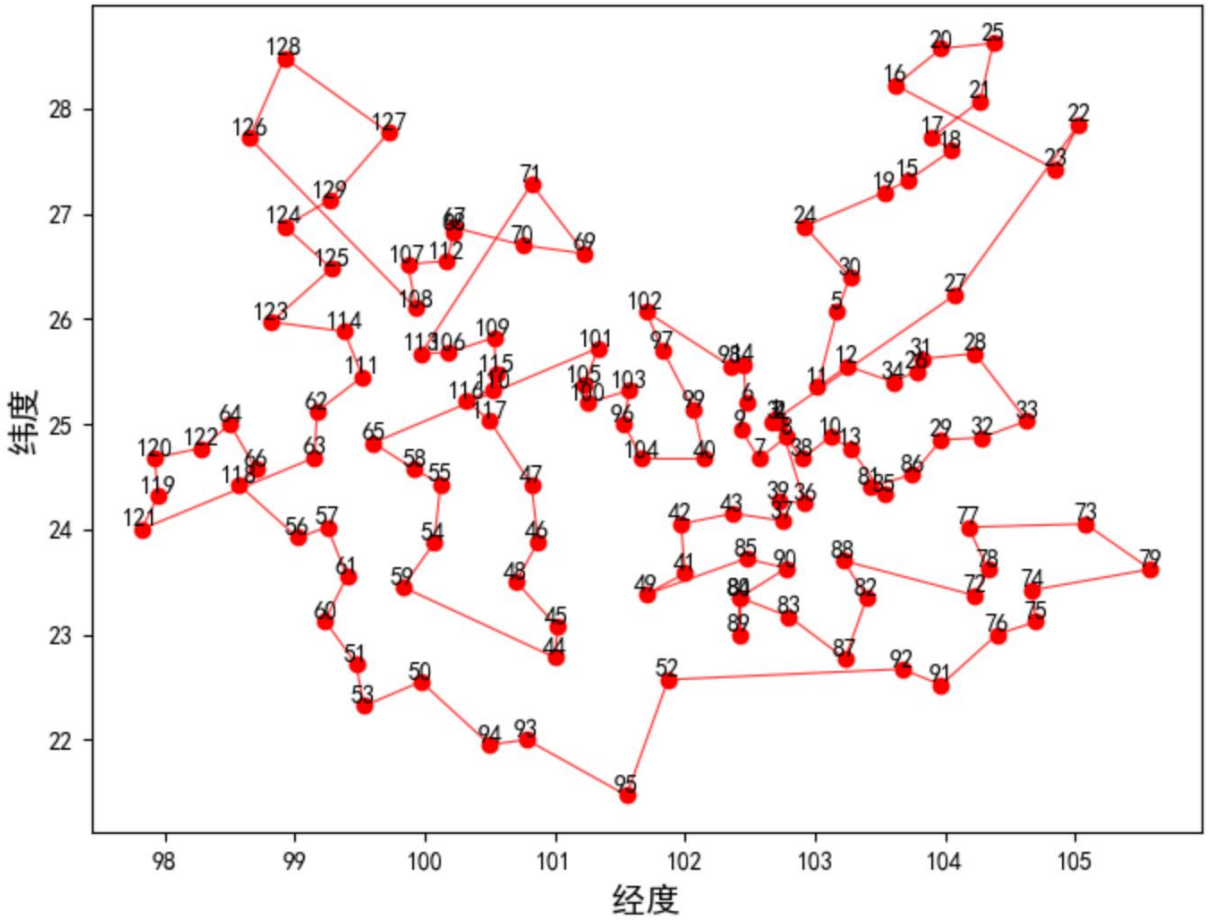
```
while iterI <= iterMax:
    '''
    每一代更新一次环境因素导致的信息素减少，每一代中的每一个蚂蚁完成路径后，都进行信息素增量更新（采用蚁周模型）
    每一代开始都先初始化蚂蚁出发城市；
    '''
    antCityList,antCityTabu = intialize(CityCoordinates,antNum)#初始化城市
    fitList = [None]*antNum#适应值列表

    for i in range(antNum):#根据转移概率选择后续途径城市，并计算适应值
        antCityList[i] = select(antCityList[i],antCityTabu[i],trans_p)
        fitList[i] = calFitness(antCityList[i],dis_matrix)#适应度，即路径长度
        pheromone = updatePheromone(pheromone,fitList[i],antCityList[i],rho,Q)#更新当前蚂蚁信息素增量
        trans_p = calTrans_p(pheromone,alpha,beta,dis_matrix,Q)

    if best_fit >= min(fitList):
        best_fit = min(fitList)
        best_line = antCityList[fitList.index(min(fitList))]

    print(iterI,best_fit)#打印当前代数和最佳适应度值
    iterI += 1 #迭代计数加1
    pheromone = pheromone*(1-rho) #信息素挥发更新
```

```
def select(antCityList,antCityTabu,trans_p):
    '''
    轮盘赌选择，根据出发城市选出途径所有城市
    输入:trans_p-概率矩阵;antCityTabu-城市禁忌表，即未经过城市;
    输出:完整城市路径-antCityList;
    '''
    while len(antCityTabu) > 0:
        if len(antCityTabu) == 1:
            nextCity = antCityTabu[0]
        else:
            fitness = []
            for i in antCityTabu:fitness.append(trans_p.loc[antCityList[-1],i])#取出a
            sumFitness = sum(fitness)
            randNum = random.uniform(0, sumFitness)
            accumulator = 0.0
            for i, ele in enumerate(fitness):
                accumulator += ele
                if accumulator >= randNum:
                    nextCity = antCityTabu[i]
                    break
        antCityList.append(nextCity)
        antCityTabu.remove(nextCity)

    return antCityList
```

二、蚁群算法

示例





- 最优路径为：

[52, 95, 93, 94, 50, 53, 51, 60, 61, 57, 56, 118, 66, 64, 122, 120, 119, 121, 63, 62, 111, 114, 123, 125, 124, 129, 127, 128, 126, 108, 107, 112, 68, 67, 70, 69, 71, 113, 106, 109, 115, 110, 116, 117, 47, 46, 48, 45, 44, 59, 54, 55, 58, 65, 101, 105, 100, 103, 96, 104, 40, 99, 97, 102, 98, 14, 6, 9, 7, 8, 38, 10, 13, 35, 81, 86, 29, 32, 33, 28, 31, 26, 34, 12, 11, 5, 30, 24, 19, 15, 18, 17, 21, 25, 20, 16, 23, 22, 27, 3, 1, 2, 4, 36, 39, 37, 43, 42, 41, 49, 85, 90, 80, 89, 84, 83, 87, 82, 88, 72, 78, 77, 73, 79, 74, 75, 76, 91, 92]