经典运筹学问题与模型

(Models for Classical Operations Research Problems)

主讲人：朱晗

东北财经大学 管理科学与工程学院

Email：hanzhu@dufe.edu.cn

感谢东北大学系统工程研究所课程组

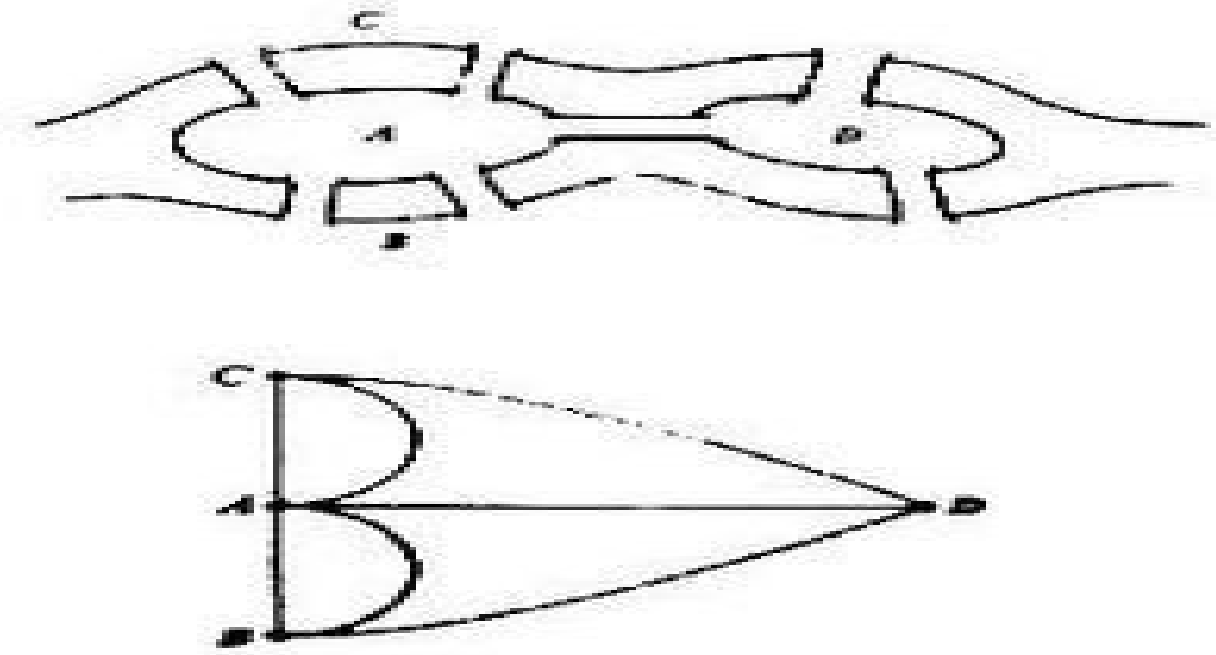
第六章　旅行商问题与模型

旅行商问题 (Traveling Salesman Problem-TSP)

- 旅行商问题的产生与起源
- 旅行商问题的分类与扩展
- TSP问题的典型应用
- 常用的求解算法
- TSP问题的遗传算法

旅行商问题

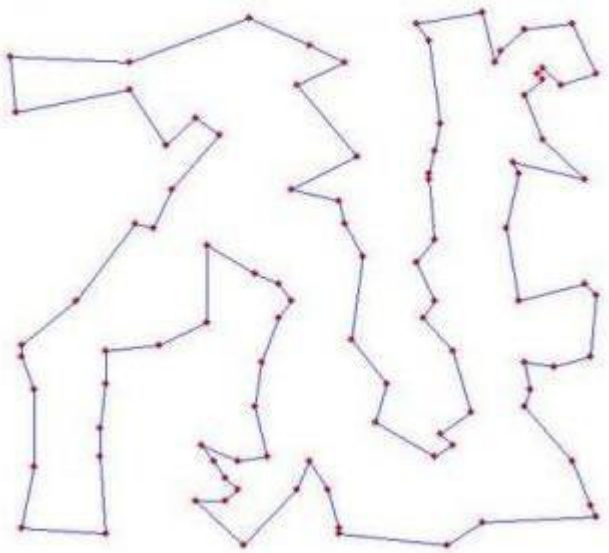
- 欧拉七桥问题
  - 18世纪著名古典数学问题之一。在哥尼斯堡的一个公园里，有七座桥将普雷格尔河中两个岛及岛与河岸连接起来(如图)。问是否可能从这四块陆地中任一块出发，恰好通过每座桥一次，再回到起点?欧拉于1736年研究并解决了此问题，他把问题归结为如右图的"一笔画"问题，证明上述走法是不可能的



旅行商问题

- TSP的历史很久
  - 最早的描述是 1759 年欧拉研究的骑士周游问题，即对于国际象棋棋盘中的 64 个方格，走访 64 个方格一次且仅一次，并且最终返回到起始点。

- 货担郎问题、推销员问题。。。
- 最早的数学模型是1959年的Dantzig（1959）提出的；
- TSP由美国RAND公司于1948年引入，该公司的声誉以及线性规划这一新方法的出现使得TSP成为一个知名且流行的问题。



旅行商问题

- 一般性描述：
  - 有一个推销员，要到 n 个城市推销商品，他要找出一个包含所有 n 个城市的具有最短路程的环路。
- 中国邮递员问题（Chinese Postman Problem CPP）：
  - ：一个邮递员从邮局出发，到所辖街道投递邮件，最后返回邮局，如果他必须走遍每条街道至少一次，应如何选择投递路线，使所走的路程最短。
  - 1962年中国数学家管梅谷先生提出中国邮递员问题(简称CPP)，是著名图论问题之一。
- 在过去的几十年中，在求旅行商问题的最优解方面取得了极大的进展。
  - 48个城市的问题、120、318、532、666、2392、24978个城市的问题
- 尽管有这些成就，但旅行商问题还远未解决。问题的许多方面还要研究，很多问题还在期待满意的回答。

问题描述

-给定

无向图 $G=(V,E,d)$。其中，V是顶点集合，$|V|=m$，E 则是边集合，d 是顶点之间的距离矩阵。注意：旅行商问题的无向图 G 是一个完全图。

- 约束

每个顶点必须被访问一次。

- 目标

寻找一条路径，使得总路程长度最短。

决策变量：$x_{ij} = \begin{cases} 1, \text{如果解中包含边}[i,j] \in E \\ 0, \text{否则} \end{cases}$

数学表达：**Dantzig-Fulkerson-Johnson**

$$\textbf{Min.} \quad \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij}$$

$$\sum_{h \in V} x_{hi} = 1 \quad \forall i \in V$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V$$

仅考虑这两个约束会产生子回路，下面的约束用于破解子回路

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \quad \forall S \subset V: |S| \geq 2$$

$$x_{ij} \in \{0,1\} \quad \forall [i,j] \in E$$

这两种破解子回路方法不好，约束太多了

上述破解子回路约束可以换成：

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subset V: |S| \geq 2$$

single commodity formulation：另外一种破解子回路方法，约束更少

y_{ij}：边 $[i,j]$上的流量, $\forall [i,j] \in E$

$$y_{ij} \leq (\mathrm{m}-1)x_{ij} \quad \forall [i,j] \in E$$

$$\sum_{j=2}^{m} y_{1j} = \mathrm{m}$$

$$-1$$

$$\sum_{i \in V} y_{ij} - \sum_{k \in V} y_{jk} = 1 \quad \forall j \in V \setminus \{1\}$$

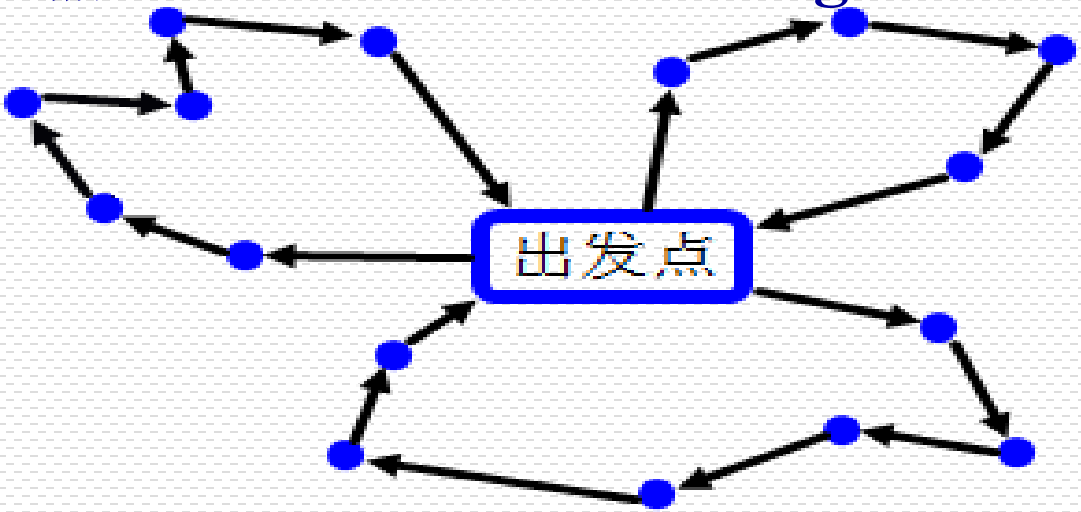
$$y_{ij} \geq 0 \quad \forall [i,j] \in E$$

旅行商问题


哈尔滨
长春
沈阳
北京
天津
济南
郑州
合肥
武汉
杭州
南昌
福州
台北
长沙
广州
南宁
海口
昆明
贵阳
成都
西宁
兰州
银川
太原
西安

问题的扩展

- 对称、非对称旅行商问题
- 多旅行商问题（MTSP）
- 带约束（如时间约束）的旅行商问题
- 瓶颈旅行商问题
- 多目标旅行商问题
- 奖金收集旅行商问题
- 多回路运输问题（Vehicle Routing Problem, VRP）
- ……

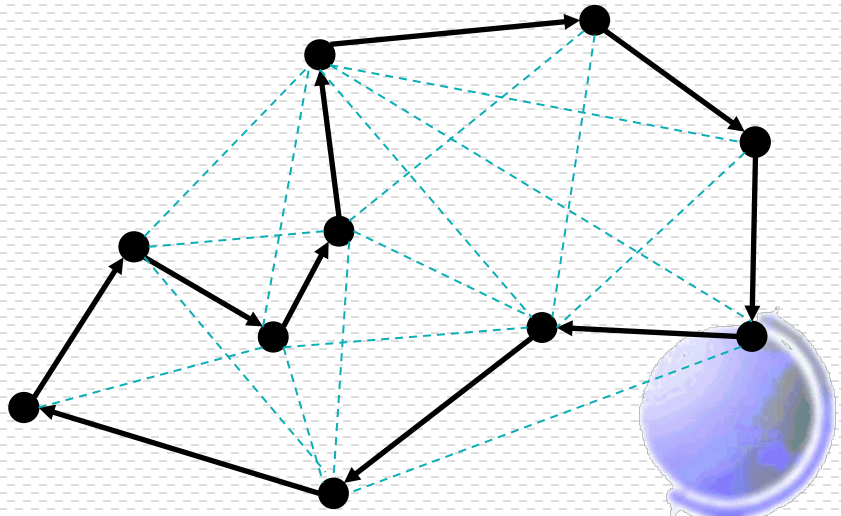


特点

- NP完全问题
- 它的解是多维的、多局部极值的
- 很难用数学公式描述 TSP 问题
- 吸引了许多不同领域的研究者，包括
  - 数学、
  - 运筹学、
  - 物理、
  - 生物、
  - 人工智能等领域。
- TSP展示了组合优化的所有方面，它已经成为并将继续成为测试新算法的标准问题，如：
  - 模拟退火、
  - 禁忌搜索、
  - 神经网络以及进化算法等都用 TSP 来测试。

应用

- 旅行路线的确定
- 配送路线问题（Route of Distribution）
  - TSP问题在物流中的描述是对应一个物流配送公司(快递员），欲将 *n* 个客户的订货（快件）沿最短路线全部送到，如何确定最短路线
  - 例如：连锁店的货物配送路线确定问题等
  - 例如：金融押运车辆的调度问题（机构网点相当于城市，这类问题属于多旅行商问题。）
- “一笔画”问题（Drawing by one line）
  - 平面上有 *n* 个点，用最短的线将全部的点连起来。称为“一笔画”问题。
  - 类似的问题：机械加工当中，要在一个零件（或者一个电路板）上加工很多孔，如何安排刀具的走刀路径，能使走刀距离最短，（孔相当于城市、孔之间的距离相当于城市间的距离）。



应用

- 机器人焊接路径规划问题
  - 例如汽车制造的车身焊接，焊点多达4-5千个，一条合理的焊接路径可以减少机器人工位的生产时间，缩短整体工时，提高生产效率。这里焊点相当于城市、焊点间的距离相当于城市间的距离。
  - 再如一个印刷线路板需要安装大量的元器件，不同的安装点可以看做是不同的城市，问题是：如何规划机器人的路径,以缩短运动时间,提高工作效率。
- 多品种装配顺序确定问题
  - 装配企业中，对许多品种的产品进行装配时，一个产品装配完以后要卸下夹具并换上装配另一个产品的夹具，更换两个不同的夹具所需的时间是不一样的。问题是如何安排多品种的装配顺序，能使更换夹具的时间最短。这里的品种相当于城市、更换时间相当于距离
- 车间配送路径优化问题
  - 工厂当中要将一匹零件分配到很多个车间，如何选择一条路径使得每个车间走一遍后回到起点，且所走的路径最短，构成TSP问题。

应用

- 投币电话的硬币收集问题
  - 在一个城市当中，如何收集投币电话的硬币，能使行走的路程最短。投币电话就相当于城市（由于有些道路是单向的，所以这是一个非对称TSP问题）。
- 管道铺设问题
  - 如铺设一些半径大小不等的圆形管道,要求所有管道与底边相切,管道之间可以相切,尽可能使铺好后的总宽度最窄。此类问题通常也称为圆排列问题。把管道看做城市、管道圆心之间的距离看做是城市间的距离，这个问题可以归结为TSP问题。（增加一个城市0，到所有管道之间的距离就是管道的半径）
- 中药自动配药系统中，取药路径规划问题
  - 不同的中药放置在不同的地方，不同的药方涉及不同的中药，相当于不同的城市，如何把所有的中药都放入取药车，而取药车移动距离最短。

应用

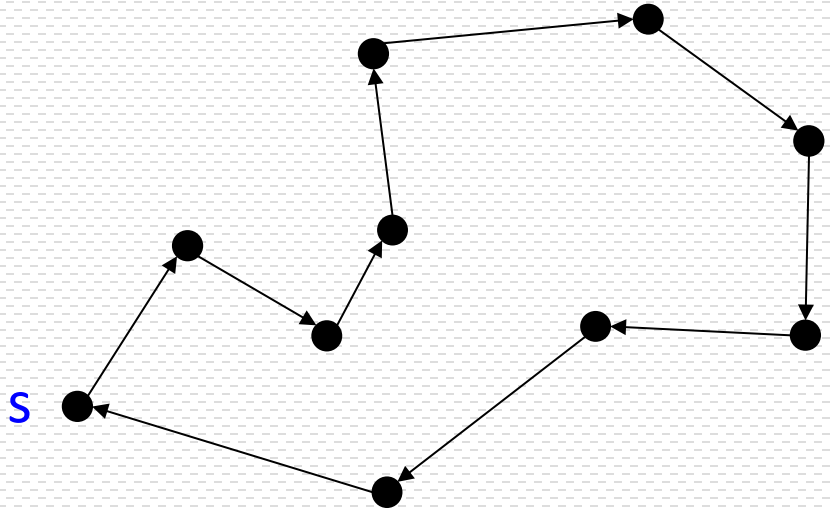
- 海上航标巡回检查问题
  - 航标相当于城市，目标是所有的航标都要检查一次而行走的距离最短，构成TSP问题。
- 垃圾运输问题
  - 每天从垃圾处理厂出发，到各个垃圾集中点将垃圾运回，也构成TSP问题。
- 超市商品的上架问题
- 图书馆的图书上架问题
- ……

一般的求解方法

- 精确算法
  - TSP问题最简单的求解方法是枚举法
  - 动态规划法
  - 分支定界法
  - 回溯求解算法
- 近似算法
  - 最近邻点法（Nearest Neighbor）
  - 最近插入法（Nearest Insertion）
  - 2OPT→k-OPT
  - 节约里程法（Saving Algorithm）
  - 扫描算法（Sweep Algorithm）
- …

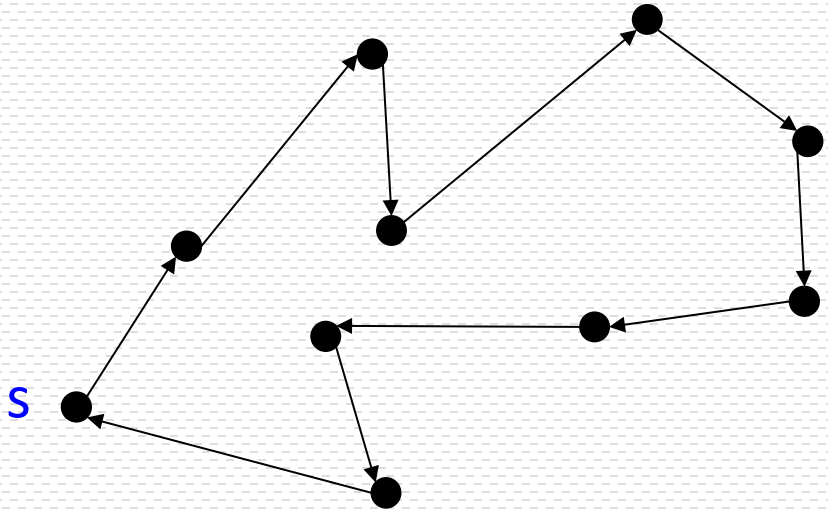
一般的求解方法

- 最近邻点法（Nearest Neighbor）
  - 首先从零点开始，作为整个回路的起点，
  - 然后找到离刚刚加入到回路的上一节点最近的一个节点，并将其加入到回路中。
  - 重复上一步，直到所有的节点都加入到回路中，
  - 最后，将最后一个加入的节点和起点连接起来，构成了一个TSP问题的解。



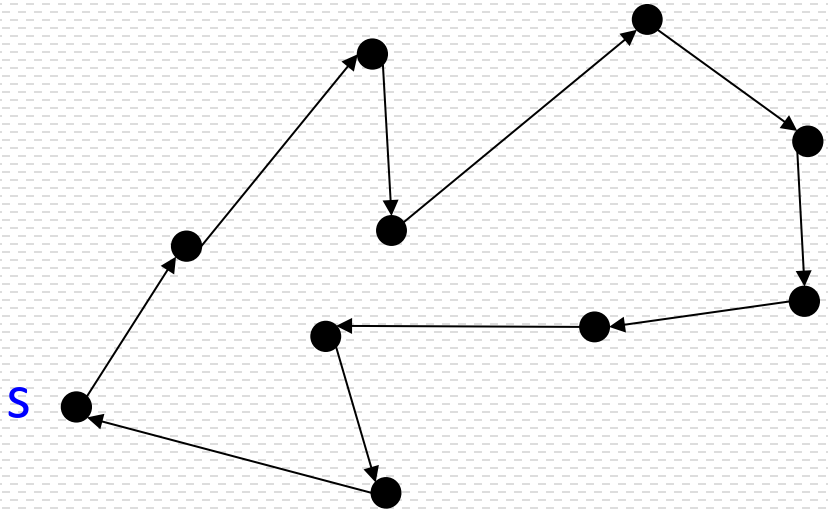
一般的求解方法

- 最近插入法（Nearest Insertion）
  - 首先从一个节点出发，找到一个最近的节点，形成一个往返式子回路；
  - 在剩下的节点中，寻找一个离子回路中某一节点最近的节点，再在子回路中找到一个弧，使弧的两端节点到刚寻找到的最近节点的距离之和减去弧长的值最小，实际上就是把新找到的节点加入子回路以后使得增加的路程最短，就把这个节点增加到子回路中，并去掉子回路中相应的弧。
  - 重复以上过程，直到所有的节点都加入到子回路中。



一般的求解方法

- 最近插入法（Nearest Insertion）
  - 首先从一个节点出发，找到一个最近的节点，形成一个往返式子回路；
  - 在剩下的节点中，寻找一个离子回路中某一节点最近的节点，再在子回路中找到一个弧，使弧的两端节点到刚寻找到的最近节点的距离之和减去弧长的值最小，实际上就是把新找到的节点加入子回路以后使得增加的路程最短，就把这个节点增加到子回路中，并去掉子回路中相应的弧。
  - 重复以上过程，直到所有的节点都加入到子回路中。



一般的求解方法

- 近似算法：C-W节约法（C-W Saving Algorithm）

节约算法是用来解决运输车辆数目不确定的VRP问题的最有名的启发式算法。它的核心思想是依次将运输问题中的两个回路合并为一个回路，每次使合并后的总运输距离减小得幅度最大，直到达到一辆车的装载限制时，再进行下一辆车的优化。优化过程分为并行方式和串行方式两种。

C-W节约算法

算法的思想

假定有n个访问地，把每个访问地看成一个点，并取其中的一个点作为基点。首先将每个点与基点相联接，构成线路1 - j - 1(j = 2，3,…,n)这样就得到一个具有n-1条线路的图。旅行者按照此路线访问的n个点所走的路程总为 $z=2\sum c_{1j}$ 其中c_{1j}为点1到点j(j = 2，3,…,n)的路段长度，这里假定$c_{1j} = c_{j1}$（对所有点j）。若联接点i和j，即使旅行者走弧(i,j),所节约的路程值(i,j)可计算如下：$s(i,j)=2\ c_{1j}+2\ c_{1j}-(c_{1j}+c_{1j}+c_{ij})$。对不同的点对s(i,j)越大，所节约的路程越多，因此应优先将这段弧插入到旅行线路中。

C-W节约算法

算法的步骤

(1)选取基点，将基点与其他各点联接，得到n-1条线路
1-j-1(j＝2，3,…,n)

(2)对不违背条件的所有可联接点对(i,j)计算节约值
$s(i,j)=c_{1i}+c_{1j}-c_{ij}$

(3)将所有的s(i,j)按其值由大到小排列。

(4)按s(i,j)值的上述顺序，逐个考察其端点i和j，若满足以下条件，就将弧(i,j)插入到线路中。其条件是：

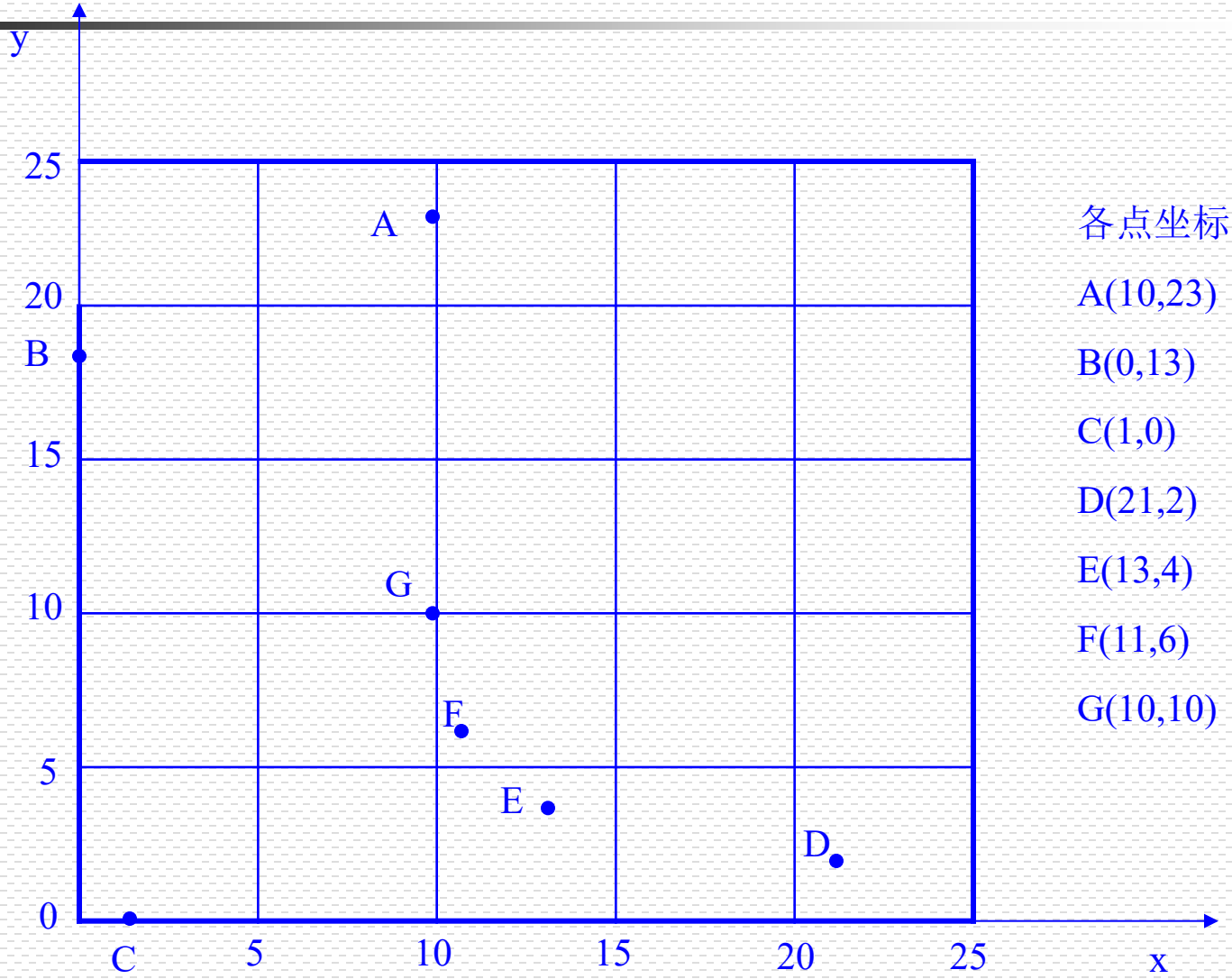
1) 点i和点j不在一条线路上

2) 点i和点j均与基点相邻。

(5)返回步骤(4),直至考察完所有的弧为止。

通过上面的步骤，使问题的解逐步得到改善，最后达到满意解。

例：用C-W节约算法求解下述TSP问题，已知访问点的位置如下所示



各点坐标：

A(10,23)

B(0,13)

C(1,0)

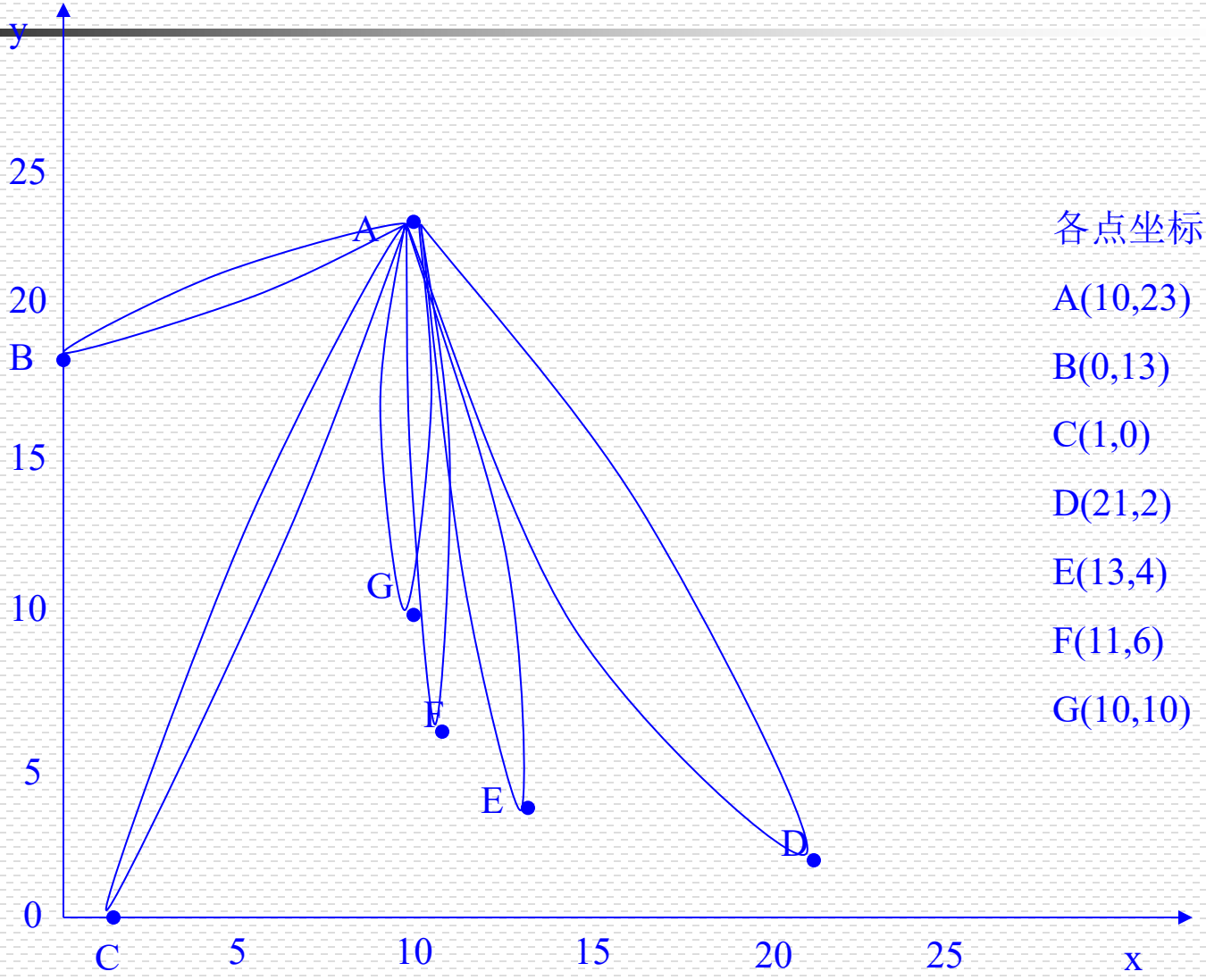
D(21,2)

E(13,4)

F(11,6)

G(10,10)

例：用C-W节约算法求解下述TSP问题，已知访问点的位置如下所示



各点坐标：

A(10,23)

B(0,13)

C(1,0)

D(21,2)

E(13,4)

F(11,6)

G(10,10)

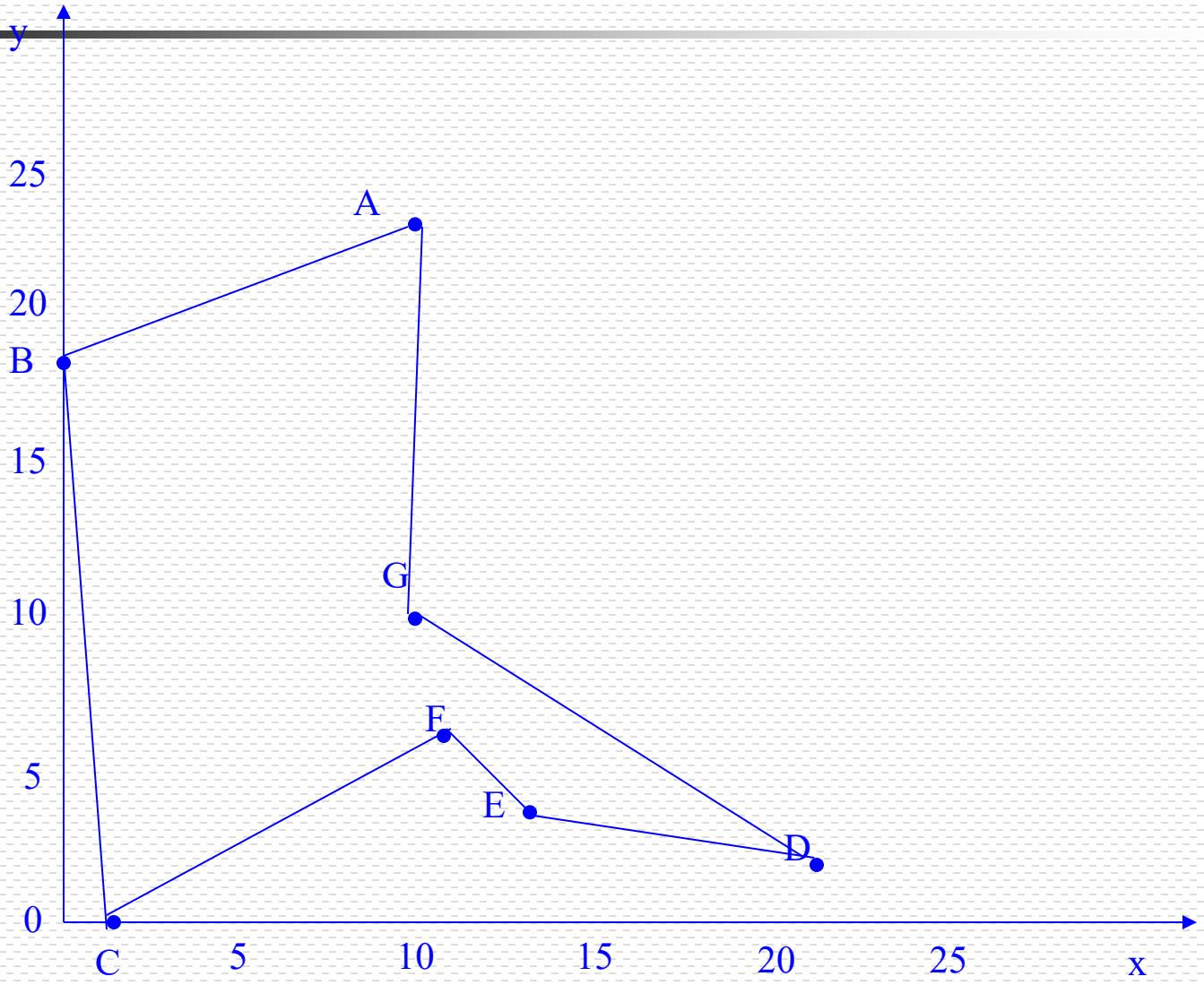
各点对之间的距离，$c_{ij}=c_{ji}$

从 \ 到	A	B	C	D	E	F	G
A	0	14.14	24.7	23.71	19.24	17.03	13.00
B		0	13.04	23.71	15.81	13.04	10.44
C			0	20.10	12.65	11.66	13.45
D				0	8.25	10.77	13.60
E					0	2.83	6.71
F						0	4.12
G							0

按各段弧节约值由大到小的顺序进行排列

序号	弧	节约值	序号	弧	节约值
1	(D,E)	34.70	9	(E,G)	25.53
2	(E,F)	33.44	10	(C,G)	24.25
3	(C,E)	31.29	11	(D,G)	23.11
4	(C,F)	30.07	12	(B,F)	18.13
5	(D,F)	29.97	13	(B,E)	17.57
6	(C,D)	28.31	14	(B,G)	16.70
7	(F,G)	25.91	15	(B,D)	14.14
8	(B,C)	25.80			

序号	弧	线路及说明	插入该弧的节约值
0		A-B-A,A-C-A,A-D-A,A-E-A,A-F-A,A-G-A	
1	(D,E)	A-B-A,A-C-A,A-D-E-A,A-F-A,A-G-A	34.70
2	(E,F)	A-B-A,A-C-A,A-D-E-F-A,A-G-A	33.44
3	(C,E)	E点与基点A不相邻，不插入	0
4	(C,F)	A-B-A,A-D-E-F-C-A,A-G-A	30.07
5，6	(D,F) (C,D)	这些点已在同一条线路上	0
7	(F,G)	F点与基点A不相邻，不插入	0
8	(B,C)	A-D-E-F-C-B-A,A-G-A	25.8
9，10	(E,G) (C,G)	E点、C点与基点A不相邻，不插入	0
11	(D,G)	A-G-D-E-F-C-B-A	23.11

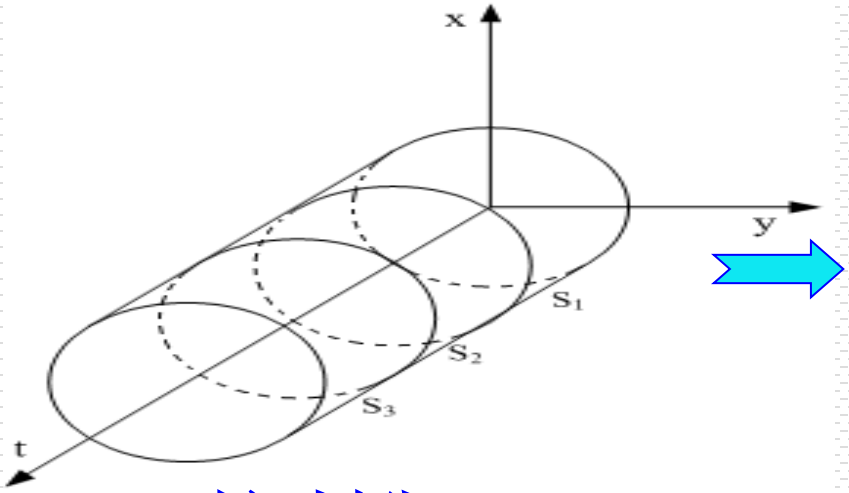


最后得到的线路为A-G-D-E-F-C-B-A，线路总长度为76.52

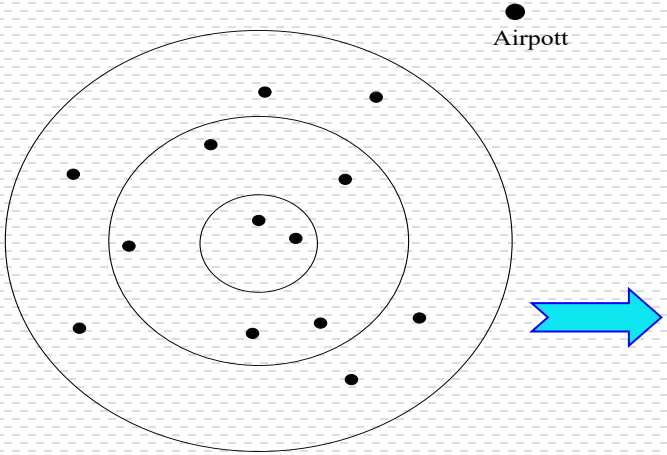
一般的求解方法

- 近似算法-扫描算法（Sweep Algorithm）

节约算法是用来解决运输车辆数目不限制的VRP问题的最有名的启发式算法。它的核心思想是：以起始点为原点建立极坐标系，然后从最小角度的两个客户开始建立一个组，按逆时针方向将客户逐个加入到组中，直到客户的需求总量超出了车辆的载重定额。然后建立一个新的组，继续该过程，直到将全部客户都加入到组中。



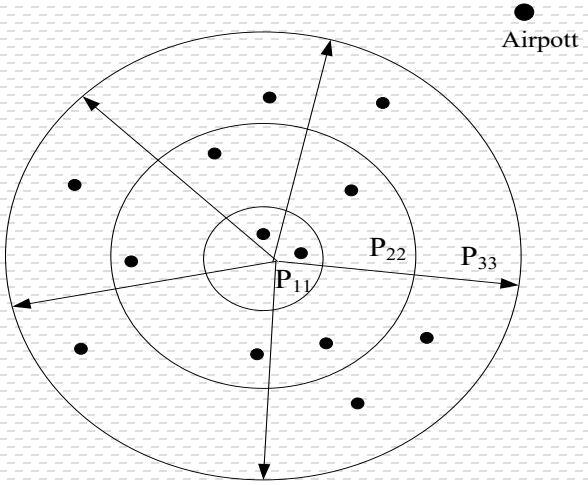
时间划分



地域划分

	地域维			
时间维	P_{11}	P_{12}	P_{13}	
	P_{21}	P_{22}	P_{23}	T1
	P_{31}	P_{32}	P_{33}	T2
				T3
				T4
				T5

时间地域划分



Sweep扫描

GA求解

- TSP已成为遗传算法学界的一个主要研究目标
- 对TSP的研究为遗传算法求解组合优化问题提供了丰富的经验和牢固的基础。
- 主要的努力在以下三个方面:
  - 采用适当的表达方法进行编码;
  - 设计可用的遗传算子，以保持父辈特性并避免不可行性;
  - 防止过早收敛。

GA求解

- 编码
  - 通常的二进制编码，不能较好地适用于 TSP 问题。
  - 人们为 TSP 提出了几种染色体表达方法，其中：
    - 随机键表达
    - 换位表达
  - 这两种表达方式不仅适用于 TSP，也适用于其它组合优化问题。

GA求解

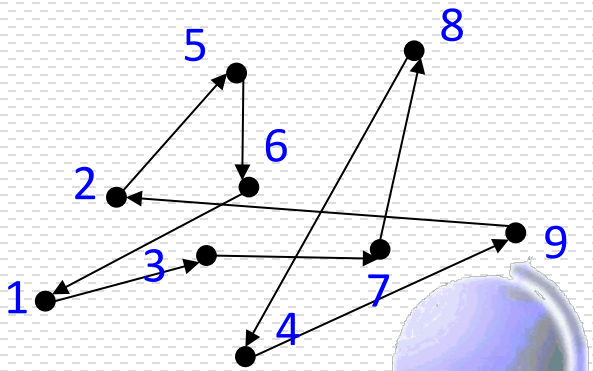
- 随机键表达
  - 首先是由 **Bean** 提出
  - 这种表达方法将解表达为一串 **(0,1)** 间的随机数
  - 这些随机数用作解码的分类键
  - 编码中位置 *i* 代表城市 *i*，位置 *i* 的随机数表示城市 *i* 在 **TSP** 巡回中的顺序
  - 这些随机键按升序排列，即可得到访问城市的顺序
  - 随机键消除了解的表达中的不可行性。
  - 这种表达方法*可用于各种顺序优化问题*，包括机器调度、资源分配、车辆线路和二次指派问题等。

染色体

1	2	3	4	5	6	7	8	9
0.23	0.82	0.45	0.74	0.87	0.11	0.56	0.69	0.78

访问顺序

6 → 1 → 3 → 7 → 8 → 4 → 9 → 2 → 5



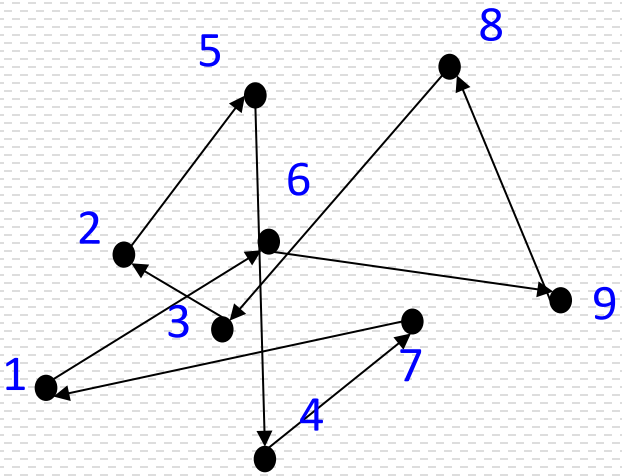
GA求解

- 换位表达
  - 也称为路径表达或顺序表达
  - 也许是TSP巡回的最自然的表达

染色体

1	2	3	4	5	6	7	8	9
3	2	5	4	7	1	6	9	8

访问顺序　3 → 2 → 5 → 4 → 7 → 1 → 6 → 9 → 8



  - 染色体中基因的值表示城市，基因的顺序表示访问城市的顺序
  - 这种表达的搜索空间是城市顺序“换位”的集合
  - 这种表达采用传统的单点交叉时，可能导致非法的巡回。为此，人们研究了多种交叉算子。

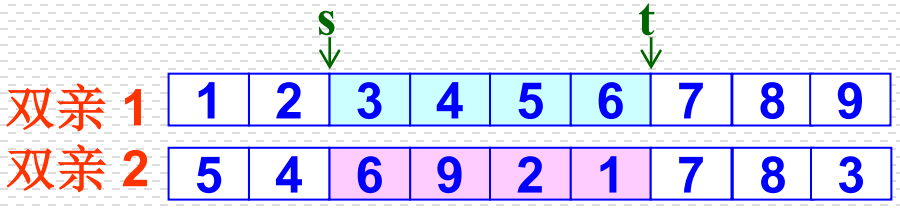
GA求解

- 交叉算子
  - 部分映射交叉(PMX)、
  - 顺序交叉(OX)、
  - 循环交叉(CX)、
  - 基于位置的交叉(PBX) 、
  - 基于顺序的交叉 (OBX)、
  - 子巡回交换交叉(SXX)、
  - 启发式交叉等
- 交叉算子的分类
  - 规范法
    - 可看作二进制串的两点或多点交叉在换位表达中的扩展
    - 后代中一些城市会丢失或重复，从而得到非法染色体。为此，需要嵌入修复程序来解决后代的非法性
  - 启发式法
    - 规范法的修复程序是盲目的，它不能保证交叉后的后代比双亲更好
    - 而启发式法的目的是为了产生改进的后代。

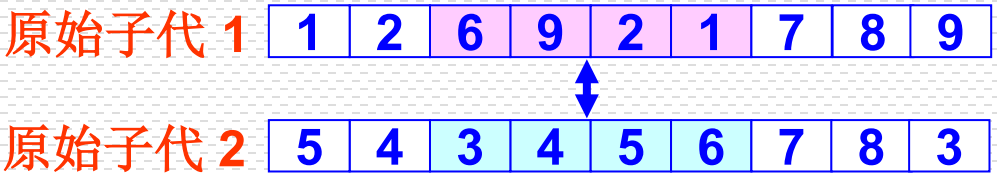
GA求解

- 部分映射交叉PMX(Partial-Mapped Crossover)
  - 由Goldberg和Lingle提出。可看作二进制串的两点或多点交叉在换位表达中的扩展。
  - 它用特别的修复程序来解决简单两点交叉引起的非法性
  - 所以PMX基本上是简单的两点交叉＋修复程序。
  - 步骤如下:

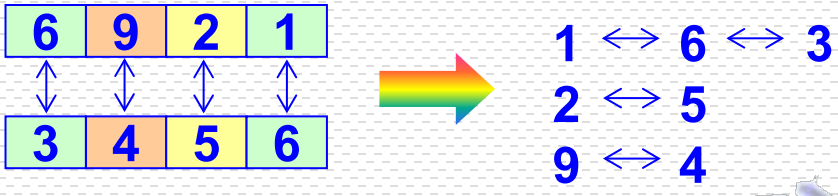
1：随机选取两个位置，由两点定义的子串称为映射段



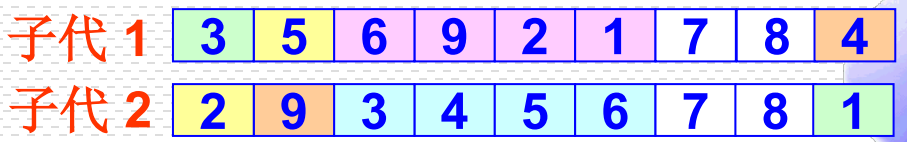
2：交换双亲的子串，形成原始后代



3：确定两映射段之间的映射关系



4：根据映射关系，将后代合法化



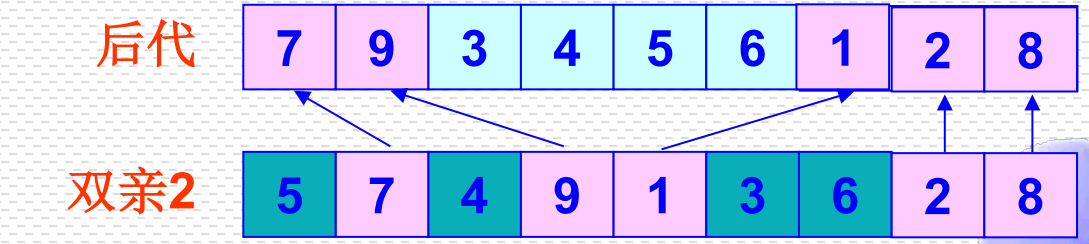
GA求解

- 顺序交叉OX(Order Crossover)
  - 由Davis提出
  - 可看作是一种带有不同修复程序的PMX的变型
  - 步骤如下：

1：从第一双亲随机选择一个子串

2：通过拷贝子串，产生一个原始后代

3：删去第二双亲中子串已有的城市，
得到原始后代需要的其它城市的顺序

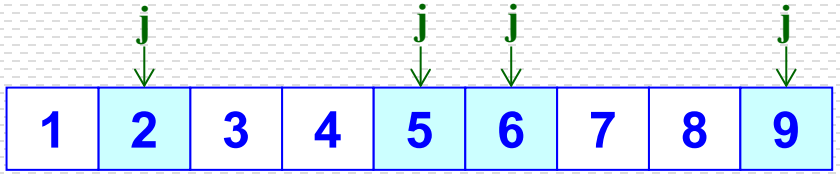


4：从左到右将这些城市定位到后代的空缺位置上

GA求解

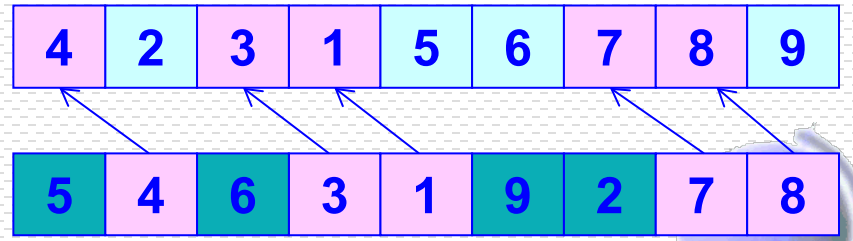
- 基于位置的交叉PBX（Position-Based Crossover）
  - 由 Syswerda 提出
  - 基本上是一种换位表达的均匀交叉＋一个修复程序。
  - 也可看作是 OX 的一种变型，其中城市的子串是不连续的。
  - 步骤如下：

1：从第一双亲上随机选取一组基因



2：复制这组基因到相应位置，产生一个原始后代

3：删去第二双亲中子串已有的城市，
得到原始后代需要的其它城市的顺序

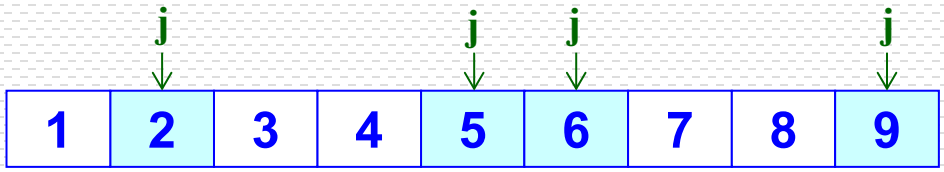


4：从左到右将这些城市定位到后代的空缺位置上

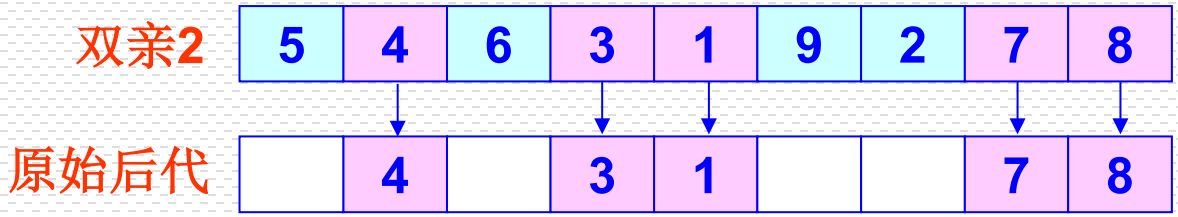
GA求解

- 基于顺序的交叉OBX（Order-Based Crossover）
  - 也是 Syswerda 提出的
  - 双亲2中“被选择基因”的顺序，被双亲1中相应基因的顺序所替代。
  - 实际上是基于位置的交叉的变型，步骤如下：

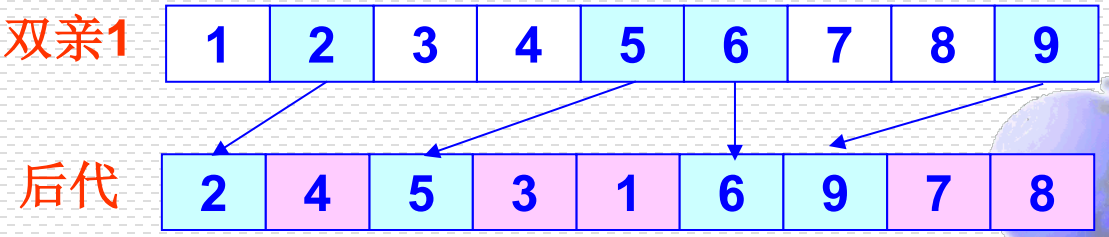
1：从双亲1随机选取一组基因



2: 从双亲2拷贝非选择的基因形成原始后代



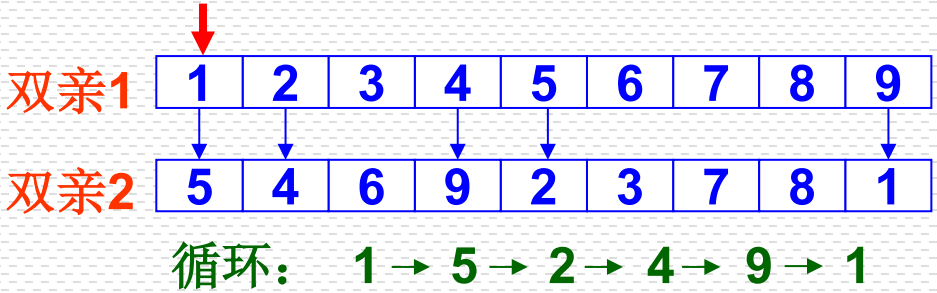
3：从双亲1复制选择的基因形成后代



GA求解

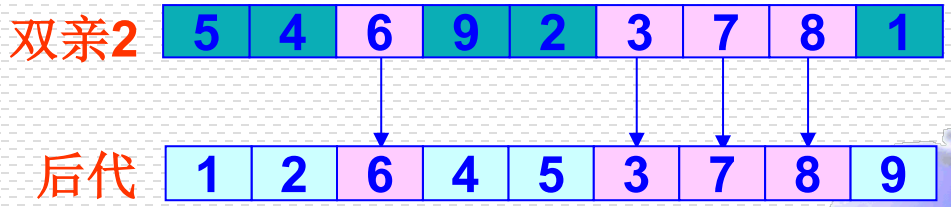
- 循环交叉CX（Cycle Crossover）
  - 由Oliver，Smith和Holland提出。和基于位置的交叉一样，该方法从一个双亲中取一些城市，而其他城市则取自另一个双亲。
  - 不同之处是来自第一个双亲的城市不是随机产生的，而是根据两个双亲相应位置城市构成的环确定的。
  - 步骤如下：

1：根据双亲相应的城市位置，找出一个循环



2：从双亲1复制循环中的基因到原始后代

3：删去双亲2中已在循环上的城市

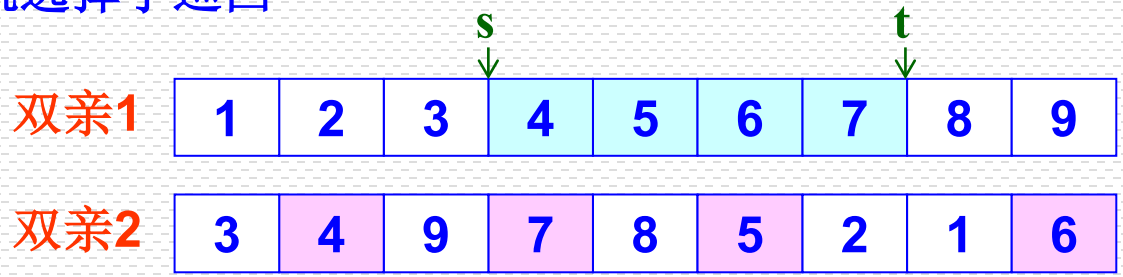


4：用剩余的城市填满后代剩余的位置

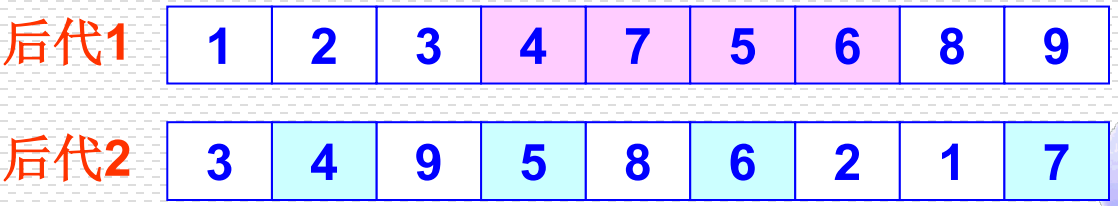
GA求解

- 子巡回交换交叉SXX (Subtour Exchange Crossover)
  - Yarnamura, Ono和Kobayashi提出
  - 从双亲中选择子巡回，子巡回中含有共同的城市
  - 后代由交换子巡回的方法产生。
  - 步骤如下：

1：在双亲中随机选择子巡回



2：交换子巡回



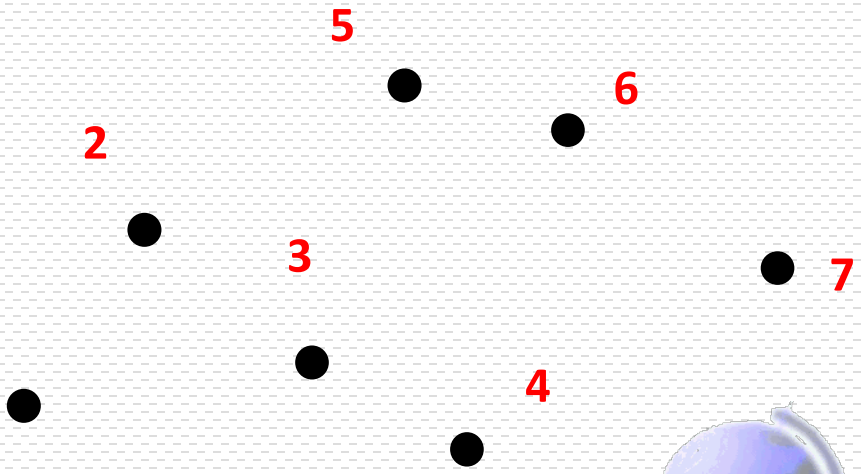
GA求解

- 启发式交叉
  - 由Grefenstette, Gopal, Rosrnaita和Gucht首先提出
  - 启发式交叉步骤（最近邻点法）：
    - 步骤1：从一对双亲中随机地选取一个城市作为开始城市，
    - 步骤2：由当前城市出发，选择一条不构成循环的最短边（由双亲表达的）。若两条边都构成循环，则随机选取一个能使巡回继续的城市；
    - 步骤3：如巡回完成，停机；否则转第二步。

双亲1	4	2	7	5	3	6	1
双亲2	3	2	5	1	7	4	6

随机选择开始城市为：3

后代1	3	2	5	1	4	6	7



  - 启发式交叉法明显优于其他方法

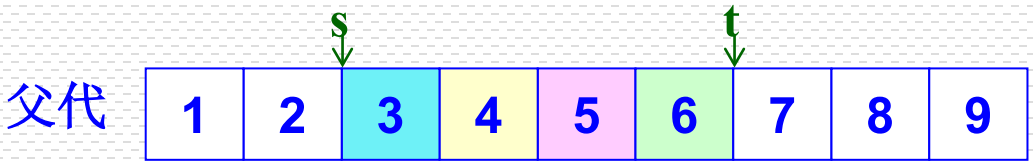
GA求解

- 变异算子
  - 反转变异 （Inversion）
  - 插入变异 （Insertion）
  - 移位变异 （Displacement）
  - 互换变异 （Swap mutation）
  - 启发式变异

GA求解

- 反转变异 (Inversion Mutation)
  - 在染色体上随机地选择两点，将两点间的子串反转。
  - 例如：

step 1: 随机选择子巡回

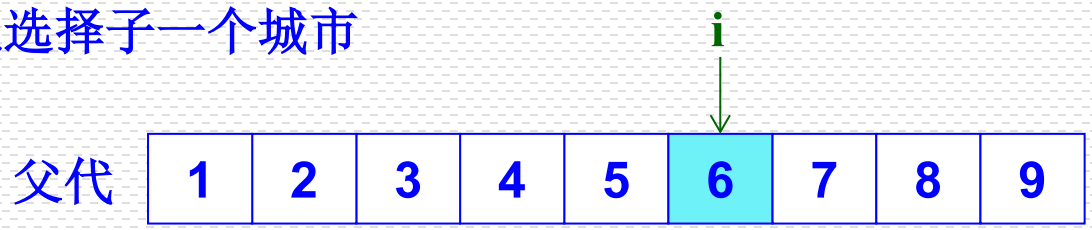


step 2 : 反转子巡回，形成后代

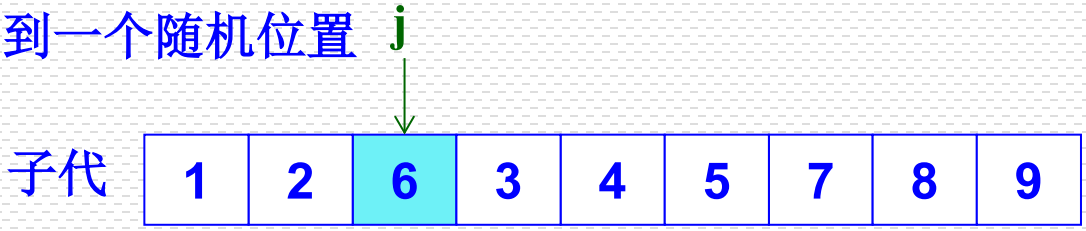
GA求解

- 插入变异 (Insertion Mutation)
  - 随机地选择一个城市，并将它插入到一个随机的位置中。
  - 例如：

step 1: 随机选择子一个城市



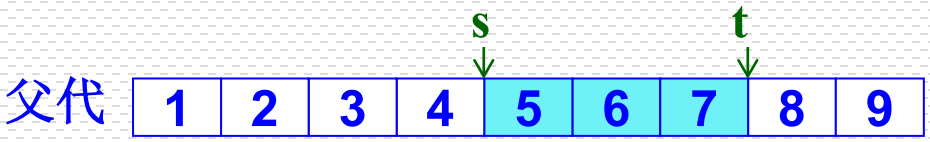
step 2 : 插入到一个随机位置



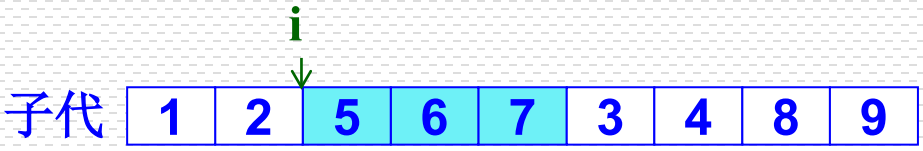
GA求解

- 移位变异 (Displacement Mutation)
  - 随机地选择一个子巡回，并将其插入到一个随机的位置中。插入变异可以看作是子巡回只含有一个城市的移位变异。
  - 例如：

step 1: 随机选择一个子巡回

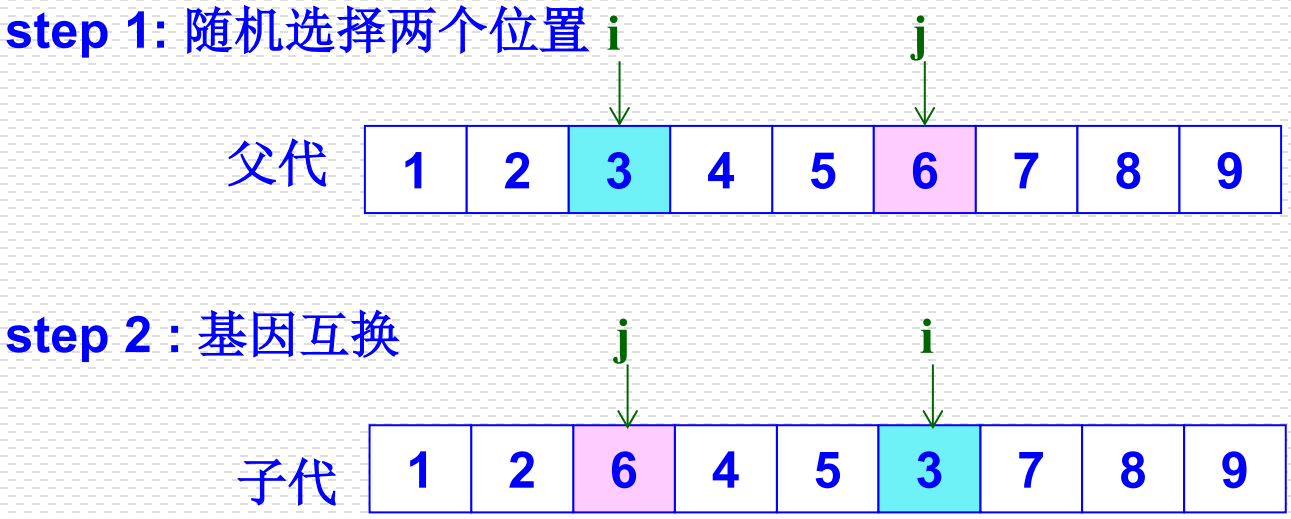


step 2 : 插入到一个随机位置



GA求解

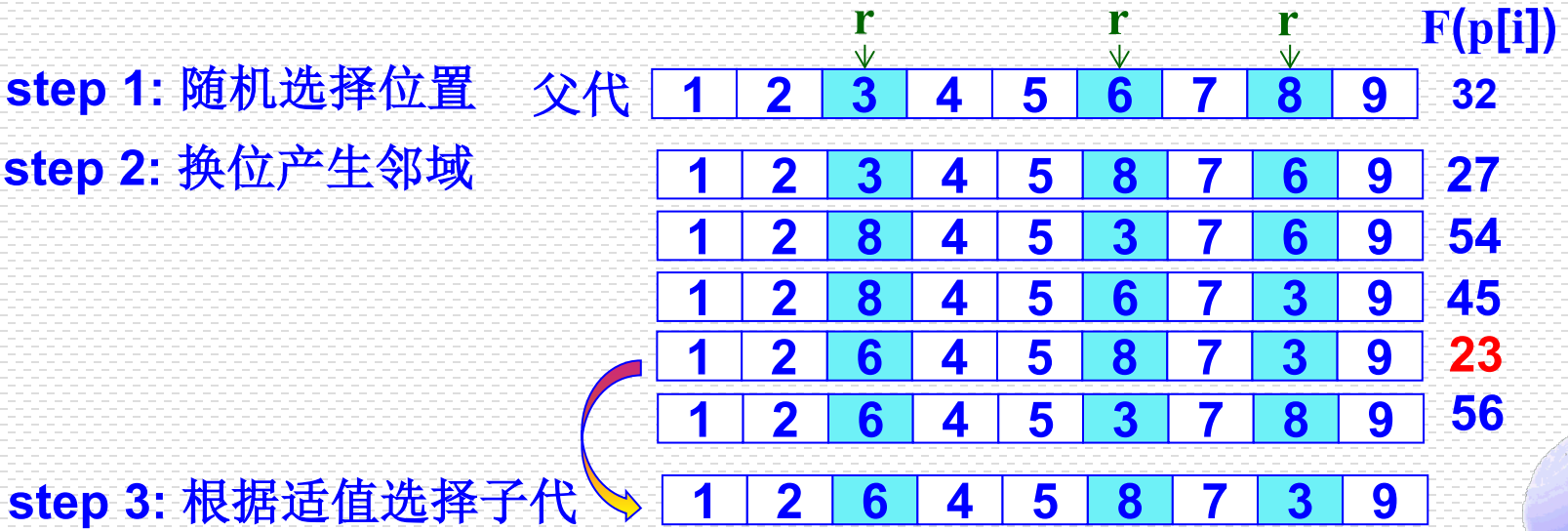
- 互换变异 (Swap Mutation)
  - 随机地选择两个位置，并将这两个位置上的城市相互交换。
  - 这种变异实质上是TSP问题的2优启发式
  - 例如：



GA求解

- 启发式变异 (Heuristic Mutation)
  - 由Cheng和Gen提出。采用邻域技术，以获得后代的改进。
  - 启发式变异过程如下：

步骤 1：随机地选出 λ 个基因；
步骤 2：按所有选出基因的可能的换位产生邻域；
步骤 3：评估所有邻域点，
选出最好的作为变异产生的后代。



End

经典运筹学问题与模型

(Models for Classical Operations Research Problems)

主讲人：朱晗

东北财经大学 管理科学与工程学院

Email：hanzhu@dufe.edu.cn

感谢东北大学系统工程研究所课程组

本节参考资料：https://mp.weixin.qq.com/s/pwCZd2zzUP_O-DgpiKgYOQ

禁忌搜索算法求解TSP问题

禁忌搜索算法（Tabu Search, TS）

- 禁忌搜索算法（Tabu Search, TS）是局部搜索算法（Local Search, LS）的推广，其核心思想是通过一种“记忆”方法，即禁忌表，对已经进行的搜索过程进行记录和选择，指导下一步的搜索方向，进而避免陷入局部最优解。

- 禁忌搜索算法的基本理论
- 禁忌搜索算法的算法流程
- 基于TSP问题的禁忌搜索算法Python代码

禁忌搜索算法的原理介绍

◆ 原理介绍

- 禁忌搜索算法是一种全局逐步寻优的全局性邻域搜索算法，主要涉及以下10个关键技术：编码、初始解、评价函数、邻域结构、候选集、禁忌表、选择策略、破禁策略、停止准则以及解码。

● 1.1 编码

- 编码是将优化问题的决策变量从其解空间转换到算法所能处理的搜索空间的表示方法。常用的编码方式有两种：二进制编码和序列编码。以TSP问题为例对这两种编码方式进行说明。

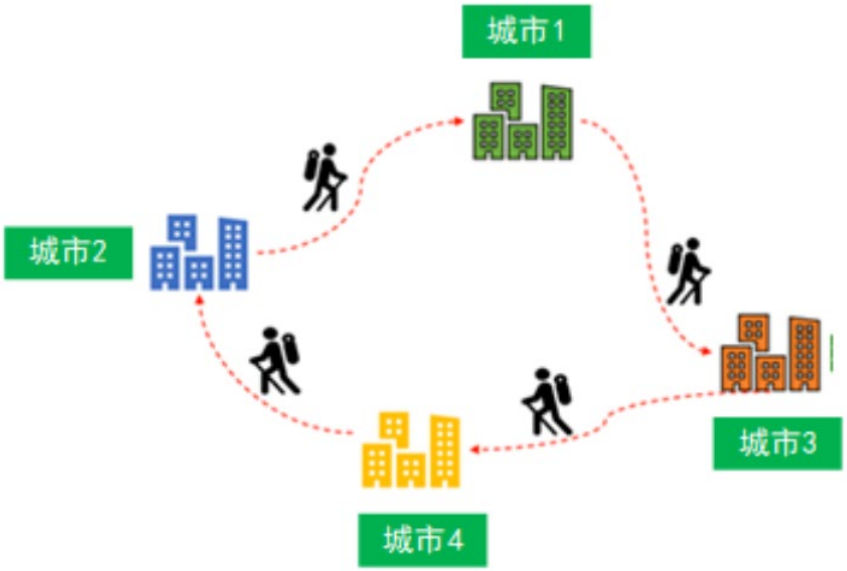
禁忌搜索算法的原理介绍

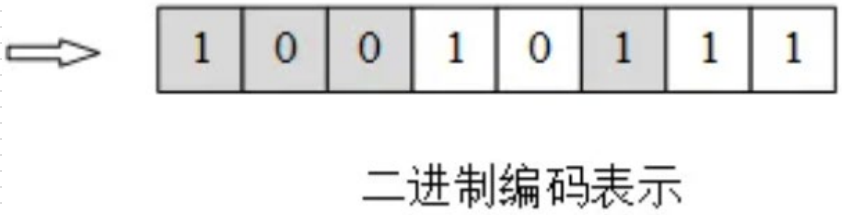
（1）二进制编码

- 二进制编码方法是使用0，1两个基本字符将决策变量表示成二进制编码符号串。

（2）序列编码

- 序列编码是一种根据问题决策变量的形式，进行特殊编码的方法。





禁忌搜索算法的原理介绍

- **1.2 初始解**
  - 在寻求最优解的过程中，禁忌搜索算法是以初始解为基础进行全局搜索的。初始解可以随机产生，也可以根据经验或由其他算法产生。
- **1.3 评价函数**
  - 通常可以将目标函数作为评价函数。评价函数也可以有其他形式，可根据具体问题对评价函数进行设计。
- **1.4 领域结构**
  - TSP问题三种常见的邻域结构：元素交换，元素嵌入和元素反转。

禁忌搜索算法的原理介绍

（1）元素交换

- 元素交换指的是将部分元素的位置进行交换。

（2）元素嵌入

- 元素嵌入指的是将某部分元素序列嵌入到其他位置。

（3）元素反转

- 元素反转指的是当前解的序列进行反转。

禁忌搜索算法的原理介绍

- **1.5 候选集**
  - 一般地，禁忌搜索算法不需要在每次迭代中搜索邻域中所有的解，可以只搜索邻域的一个子集，即候选集。
  - 通过这种方式，可以只对当前解邻域内部分解进行评价，有利于加快算法的搜索速度。

$$N(x)=\{(1,2,3,4),\ (2,1,3,4),(3,2,1,4),(4,2,3,1),(1,3,2,4),(1,2,4,3)\}$$

- **1.6 禁忌表**

- 禁忌表是用来存放禁忌对象的一个容器，禁忌表中的禁忌对象在解禁之前不能被再次搜索。
- 引入禁忌表是为了防止搜索出现循环，陷入局部最优解的情况出现。
- 禁忌表包括禁忌对象和禁忌长度两个主要指标。

禁忌搜索算法的原理介绍

禁忌对象

- 禁忌对象指的是禁忌表中被禁止变化的元素（即禁掉谁）。

（1）以状态本身或者状态的变化作为禁忌对象。
（2）以状态分量以及分量的变化作为禁忌对象。
（3）以状态分量以及目标值变化作为禁忌对象。

禁忌长度

- 禁忌长度指的是搜索过程中，禁忌对象被禁忌的步长（即禁多久）。
- 禁忌长度可以设置为固定值，也可以是动态变化的，可根据具体问题设计某种规则进行变化。
- 禁忌长度的选择会对算法的有效性产生重要影响。

禁忌搜索算法的原理介绍

- **1.7 选择策略**
  - 在禁忌搜索算法中，选择的目的是从候选解集合中选择合适的解，以进行下一次迭代运算。
  - 一种常用的方法是选择目标函数值最小的解，从而使算法趋向于最优解。
- **1.8 破禁策略**
  - 破禁策略也叫藐视准则，是对禁忌表的适度放松。
  - 其作用是为了防止由于禁忌表的存在，而错过优异解的现象出现，同时也是为了避免候选集中所有的解均被禁忌掉，算法无法继续进行的情况出现。
  - 从候选集中选择适应值最好的候选解，将其与当前最好解进行比较，若其是被禁忌的解，但是优于当前最好解，则可将其解禁，用来作为下一迭代的当前解。

禁忌搜索算法的原理介绍

- **1.9 停止准则**

（1）设置最大迭代次数，例如：将算法的最大迭代次数设置为500。

（2）设置最好的解不再更新的次数，例如：将最好的解不再更新的次数设置为100，达到该条件后停止迭代。

（3）设置某个禁忌对象禁忌的次数，例如：假设上述TSP问题中，禁忌对象为解(1, 2, 3, 4)，当其禁忌次数达到50次时，停止迭代。

- **1.10 解码**

  - 解码是编码的反变换过程。
  - 算法终止运行并输出最优结果后，可以进一步将最优结果解译为更为直观、易懂的表现形式。

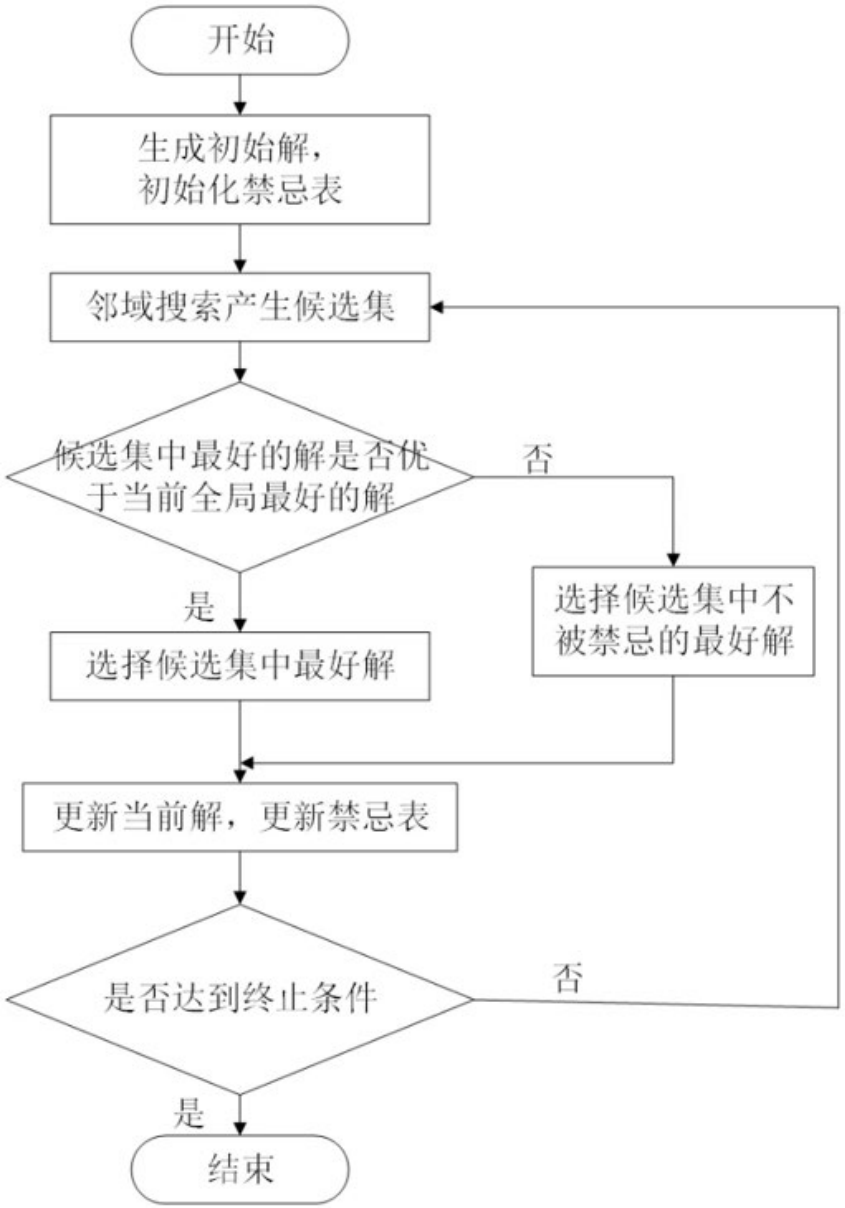
禁忌搜索算法的算法流程

◆ 算法流程

（1）初始化，产生初始解，清空禁忌表；
（2）判断是否满足终止条件，若满足，则输出当前最优解并进一步执行解码操作；否则执行步骤（3）；
（3）在当前解的邻域内产生候选解集合，并计算候选解的适应值；
（4）将适应值最好的解与当前搜索得到的最好的解进行比较：如果优于当前最好解，更新其为最好的解，并作为下次迭代的当前解；否则从候选集中选择未被禁忌的最好解作为下次迭代的当前解；】
（5）更新禁忌表；
（6）返回步骤（2）。

禁忌搜索算法的算法流程

- 算法流程



禁忌搜索算法的代码实现

◆ Python代码实现

➢ 以禁忌搜索算法求解50个城市TSP问题为背景，从环境声明、主函数以及子函数定义几个方面，详细介绍Python代码实现过程。

● 3.1 环境声明

```
import numpy as np  #关于数值运算的函数库
import random  #关于随机数的函数库
import xlrd #读取excel的函数库
import copy  #复制变量的函数
import matplotlib.pyplot as plt #绘图
from matplotlib import rcParams #绘图
```

禁忌搜索算法的代码实现

```
############### 类声明 ###################################
class Node:    #将节点的所有信息打包为Node类变量
    def __init__(self):
        self.node_id = 0 #节点编号
        self.x = 0.0      #节点横坐标
        self.y = 0.0      #节点纵坐标
class Solution: #将求解打包为Solution类变量
    def __init__(self):
        self.cost = 0
        self.route = []
    def copy(self):
        solution = Solution()
        solution.cost = self.cost
        solution.route = copy.deepcopy(self.route)
        return solution
```

禁忌搜索算法的代码实现

```
############### 数据导入 ################################
#将"input_node.xlsx"中的节点信息汇总为集合N
N={}
book = xlrd.open_workbook("input_node.xlsx")
sh = book.sheet_by_index(0)
for l in range(1, sh.nrows):  # read each lines
    node_id = str(int(sh.cell_value(l, 0)))
    node = Node()
    node.node_id = int(sh.cell_value(l, 0))
     node.x = float(sh.cell_value(l, 1))
     node.y = float(sh.cell_value(l, 2))
     N[l-1]=node
```

禁忌搜索算法的代码实现

```
#根据节点信息计算距离矩阵distance，distance[i][j]代表从i 节点到j 节点的出行成本
distance=[[0 for j in range(sh.nrows-1)] for i in range(sh.nrows-1)]
for i in range(sh.nrows-1):
    for j in range(sh.nrows-1):
        distance[i][j]=((N[i].x-N[j].x)**2+(N[i].y-N[j].y)**2)**0.5
```

```
############### 参数设置 ###############################
Iter=500       #迭代次数
City_number = sh.nrows-1  # 城市数量
tabu_length = City_number*0.2  #禁忌长度
candidate_length = 200 #候选集的个数
```

```
history_best = [] #存放历史最优目标值
history_route=[[]]#存放历史最优路径
best_solution.cost = 9999#初始最优值
```

禁忌搜索算法的代码实现

- 3.2 主函数

```
############## 主函数 ############################
for k in range(Iter):
    candidate,candidate_distance,swap_position = get_candidate(current_solution.route,distance,City_number)
    min_index = np.argmin(candidate_distance)
    #判断邻域动作是否在禁忌表中，若不在禁忌表中则判断候选集中最小值与全局最优值的大小
    if swap_position[min_index] not in tabu_list:#若邻域动作不在禁忌表中
        if candidate_distance[min_index] < best_solution.cost:#若候选集中最小值小于最优值，则更新当前解与最优解
            best_solution.cost = candidate_distance[min_index]
            best_solution.route = candidate[min_index].copy()
            current_solution.route = candidate[min_index].copy()
            current_solution.cost = candidate_distance[min_index]
            tabu_list.append(swap_position[min_index])#更新禁忌表
            history_best.append(best_solution.cost)
            history_route.append(best_solution.route)
        else:#若候选集中最小值大于最优值，则只更新当前解
            current_solution.route = candidate[min_index].copy()
            current_solution.cost = candidate_distance[min_index]
            tabu_list.append(swap_position[min_index])
            history_best.append(best_solution.cost)
            history_route.append(best_solution.route)
    else:#若邻域动作在禁忌表中，判断候选集中最小值与全局最优值的关系
        if candidate_distance[min_index] < best_solution.cost:#若候选集中最小值小于最优值，则此邻域动作解禁，并更新全局最最优解与当前解
            del_list.append(tabu_list[tabu_list.index(swap_position[min_index])])
            del tabu_list[tabu_list.index(swap_position[min_index])]
            best_solution.cost = candidate_distance[min_index]
            best_solution.route = candidate[min_index].copy()
            current_solution.route = candidate[min_index].copy()
            current_solution.cost = candidate_distance[min_index]
            history_best.append(best_solution.cost)
        else:#若候选集中最小值大于最优值，则将此解设置为最大，不再搜索，更新当前解
            candidate_distance[min_index] = 99999
            current_solution.route = candidate[min_index].copy()
            current_solution.cost = candidate_distance[min_index]
            tabu_list.append(swap_position[min_index])#更新禁忌表
            history_best.append(best_solution.cost)
            history_route.append(best_solution.route)
    if len(tabu_list) > tabu_length:#禁忌表移动
        del tabu_list[0]
```

禁忌搜索算法的代码实现

- 3.3 子函数

（1）关键技术1：计算目标函数值。

```
############### 计算路径成本消耗 ##############################
def route_cost(route,distance,City_number):
    cost = 0
    for i in range(City_number - 1):
        cost += distance[int(route[i])][int(route[i + 1])]
    cost += distance[int(route[City_number - 1])][int(route[0])]  # 加上返回起点的距离
    return cost
```

禁忌搜索算法的代码实现

- 3.3 子函数

（2）关键技术2：实现两点交换的邻域结构，即：2-opt。

```
############## 邻域操作 ##############################
def swap(route,route_i,route_j):
    new_route=copy.deepcopy(route)
    while route_i<route_j:
        temp=new_route[route_i]
        new_route[route_i]=new_route[route_j]
        new_route[route_j]=temp
        route_i+=1
        route_j-=1
    return new_route
```

禁忌搜索算法的代码实现

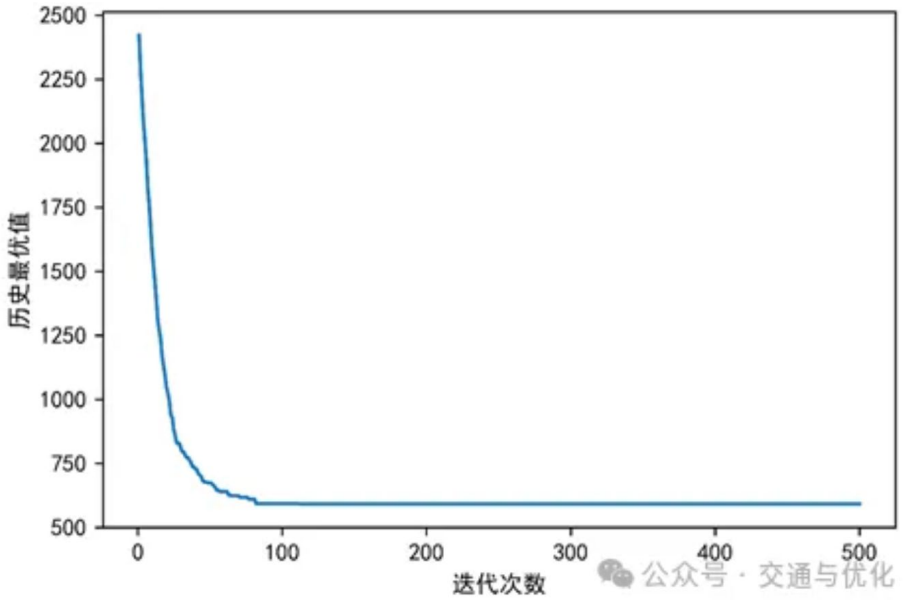
- 3.3 子函数

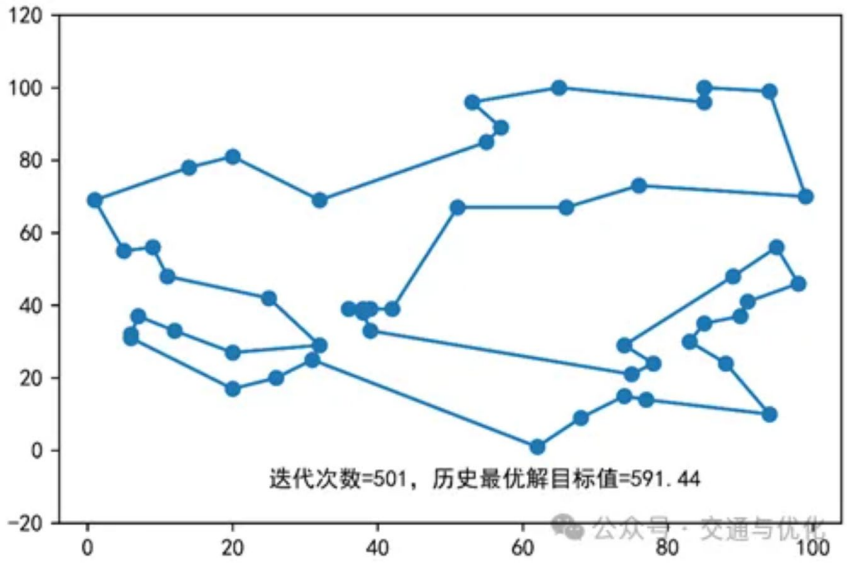
（3）关键技术3：根据邻域结构随机生成候选解集合。

```
############### 产生邻域候选解集合#############################
def get_candidate(route,distance,City_number):
    swap_position = []
    i = 0
    while i < candidate_length:
        current = random.sample(range(0,City_number),2)
        if current not in swap_position:
            swap_position.append(current)
            candidate[i] = swap(route,current[0],current[1])
            candidate_distance[i] = route_cost(candidate[i],distance,City_number)
            i += 1
    return candidate,candidate_distance,swap_position
```

禁忌搜索算法的代码实现

- 3.4 求解结果





End