

《计算机模拟》



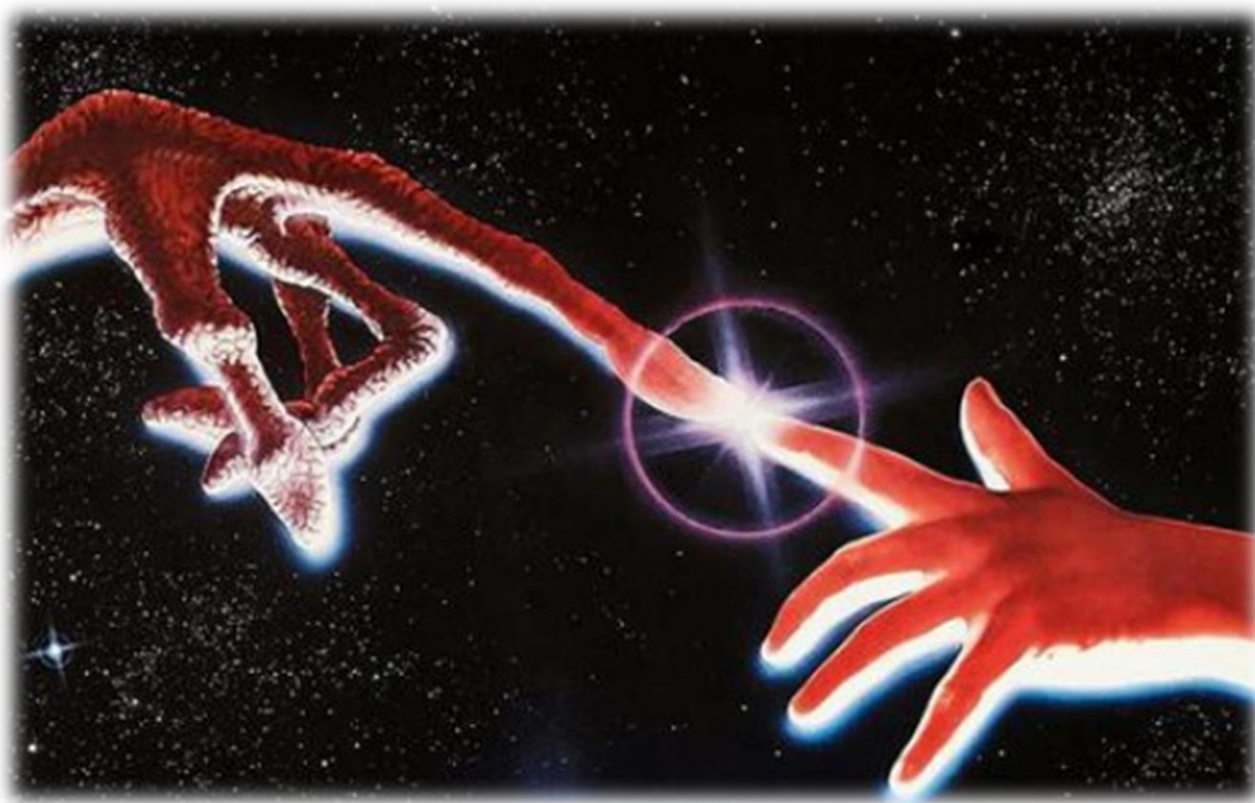
第10讲 – 仿生优化算法

胡贤良

浙江大学数学科学学院

本讲内容

1. 遗传算法
2. 蚁群算法
3. 一个应用案例



最优化问题是寻找一个给定目标函数 f 的最大值或者最小值的问题，其形式为：

$$\min_{x \in \Omega} f(x)$$

其中区域 Ω 被称为可行域或者约束集。

1. 遗传算法

- 一种模拟生物在自然环境中的遗传和进化的过程而形成的自适应全局优化概率搜索算法。
- 五个核心要素：参数编码、种群初始化、遗传算子设计、适应度函数设计、控制参数

使用示例：求shekel函数最小值

➤ 利用matlab的全局优化工具箱，计算处Shekel函数在区间 $2 \leq x \leq 9$ 的**最小值**：

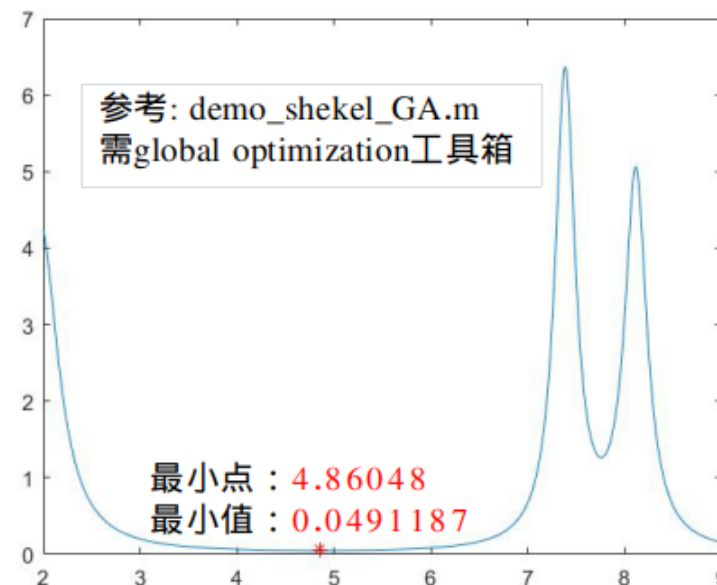
$$shekel = \frac{1}{10.295 * (x - 7.382)^2 + 0.1613} + \frac{1}{4.722 * (x - 1.972)^2 + 0.2327} + \frac{1}{10.928 * (x - 8.11)^2 + 0.2047}.$$

```
function y = shekel(x)
```

```
    y = 1 ./ (10.295*(x-7.382).^2 + 0.1613) + 1 ./ (4.722*(x-1.972).^2 + 0.2327) + 1 ./ (10.928*(x-8.111).^2 + 0.2047);  
end
```

- 算法的核心：**适应度函数**。它是对优化问题目标函数的一个简单变换。当群体越接近目标函数的全局最优则适应性函数值越大！
- 最大优点：对于优化问题，不需求导和函数连续性要求，具备全局寻优和并行能力。

```
[x,v] = ga(@shekel,1,[ ],[ ],[ ],[ ],2,9)
```



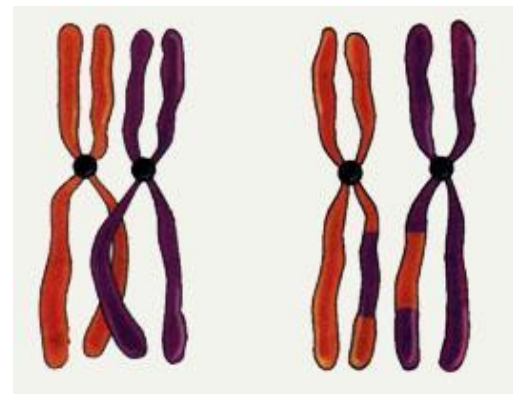
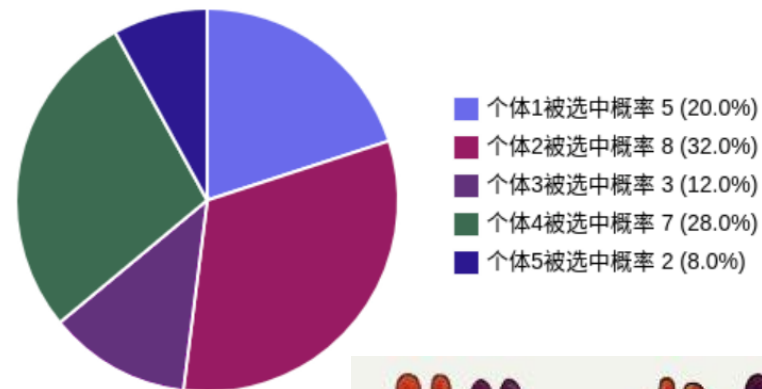
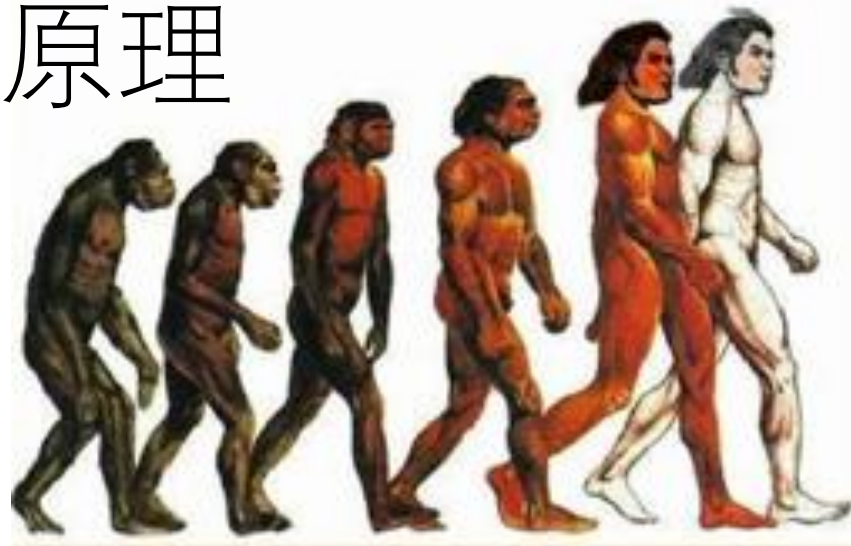
进化理论 - 遗传算法的仿生学原理

- **自然选择**：模仿生物进化自适应演化的一种优化方法，模拟生物的生存和繁殖均具有一定概率，即不确定性。同理于优化过程避免陷入局部最优或构造Markov链确保可遍历性。

- **适者生存**：算法的主要工作方式是将优化问题的所有可行解看做一群生物个体，对这群个体的基因进行**重组**、**变异**以及选择性**复制**以繁衍出下一代群体。

- 进化理论的一般特性

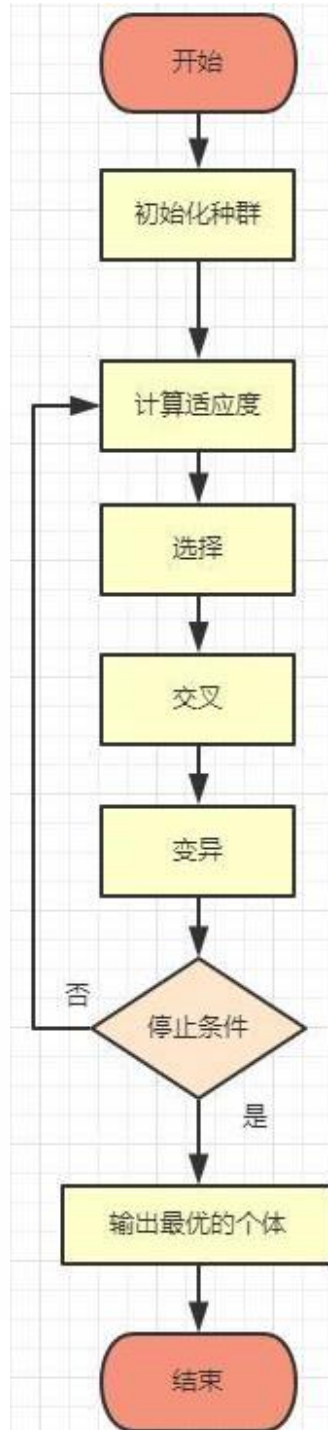
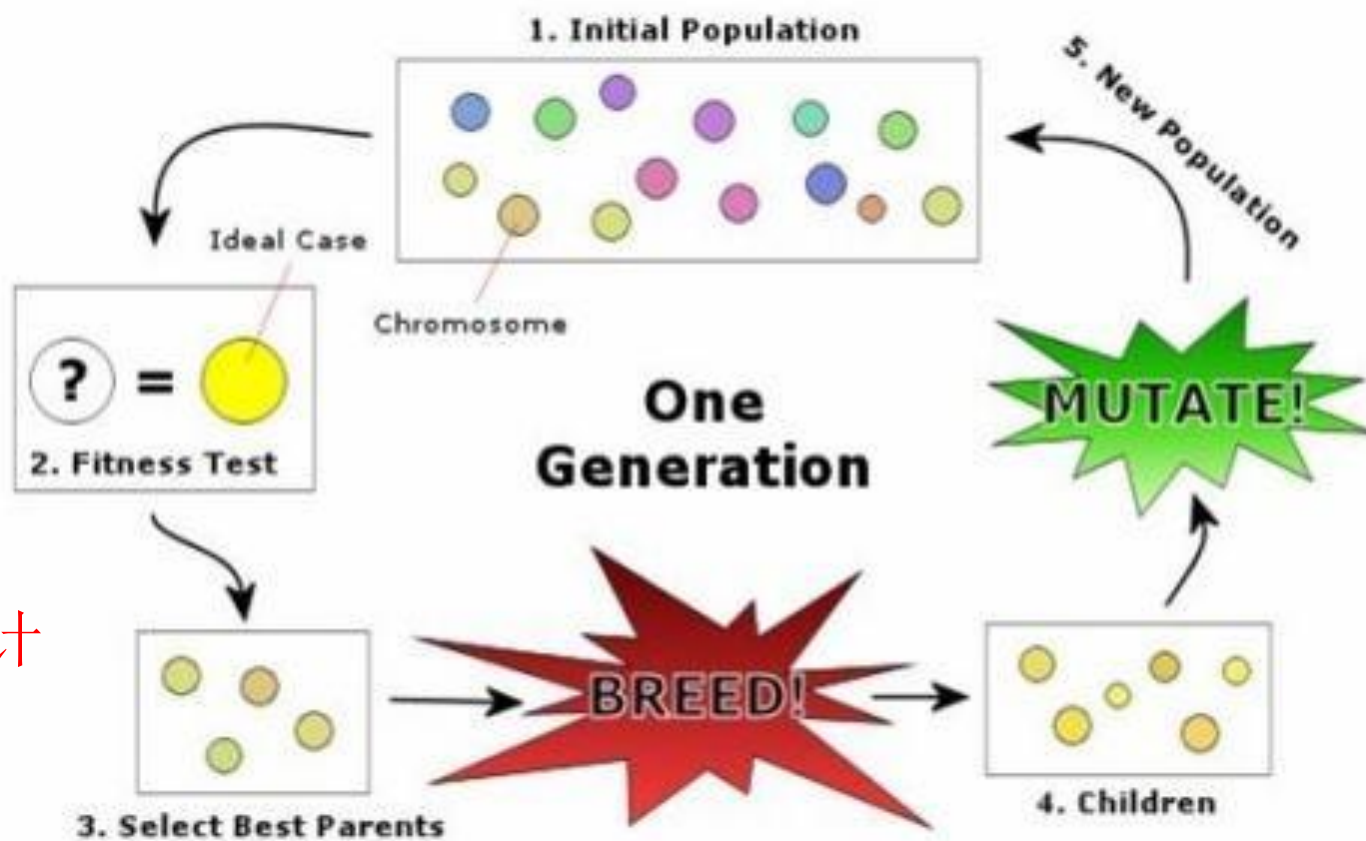
1. 进化过程发展在**染色体**上，而不是他们所**编码**的生物体上；
2. 自然选择导致了**适应性好**的个体的**染色体**具有**更多繁殖机会**；
3. **变异**可使子代染色体不同于父代，**重组**可导致子代**染色体**具有较大差异；
4. 生物进化**没有记忆**，生物体的**染色体**编码结构**直接导致**其与环境适应度；



遗传算法： 解决之道

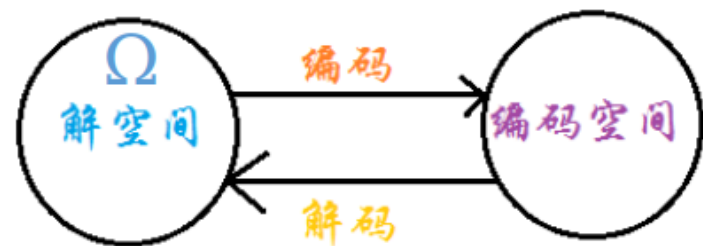
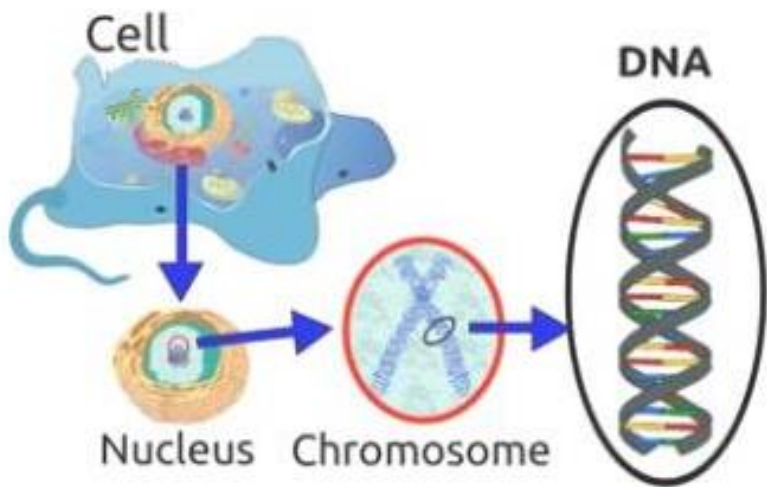
五个核心要素：

1. 参数编码
2. 种群初始化
3. 遗传算子设计
4. 适应度函数设计
5. 控制参数



① 参数编码 - 解的表示

- 目的：将**优化问题**转化为**组合问题**
- 常用编码方案：二进制编码、整数编码、实数编码
- 以函数优化问题为例：
 - 二进制：稳定性高、种群多样；存储空间大
 - 实数：容易理解、不需要解码；容易早收敛



➤ 二进制编码

- 编码、解码操作简单
- 交叉、变异等遗传操作简单
- 符合最小字符集编码原则
- 模式定理适用
- 编码长度过长存储开销大
- 编码长度过短不足以表达特征
- 不能直接反映问题本身结构

编码方案寻优 - 格雷码

- 便于提高GA局部搜索能力

例：表示相邻整数 $X_1 = 175$, $X_2 = 176$

➤ 10位(考虑区间[0, 1023]) 二进制编码：

$X_1 = 0010101111, X_2 = 0010110000$

➤ 10位格雷码：

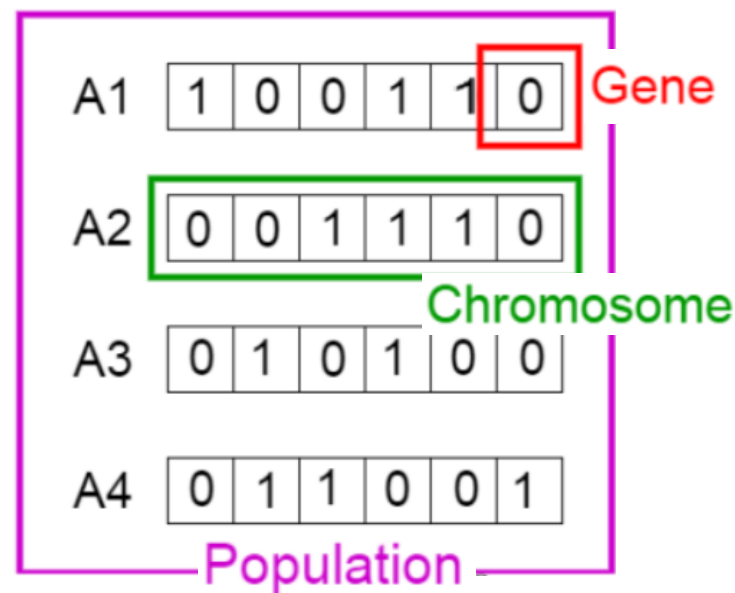
$X_1 = 0010100100, X_2 = 0010101100$

- 交叉、变异等遗传操作简单
- 符合最小字符集编码原则
- 模式定理分析适用

十进制	二进制码	格雷码
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

② 初始化种群

- 遗传算法中最小的分类单元是基因(Gene)
- 随机产生n条染色体(Chromosome)或个体(Individual)
 - 策略1: 根据固有的知识构造初始群体的分布空间
 - 策略2: 冷启动, 即完全随机生成后迭代一定步数
- 上述n条染色体/个体合在一起, 称为群体(Population)
- 群体的规模设置:
 - 过小很可能会导致过早收敛
 - 过大会影响计算时间
- 若具有某些先验知识, 可在满足要求的集合中随机抽取, 可以加快收敛速度!



③ 遗传算子

- 定义三种随机算子:

选择、重组和变异

- 生成状态空间 Ω 上的Markov链 X_t
- 可以证明:
通过合适地定义上述三种随机算子,
生成的链 $\{X_t\}$ 总是不可约的和非周期的,
且收敛于一个稳定分布!

Initial 群体 C_0

$B \leftarrow$ 在群体 C_0 中最优的适应度

For $t = 1$ to N do

采用随机选择方式来创建群体 C_1

交叉染色体1与2, 3与4, ...

以后代取代它们的父母

for 遍历 C_1 中的每个染色体 c do

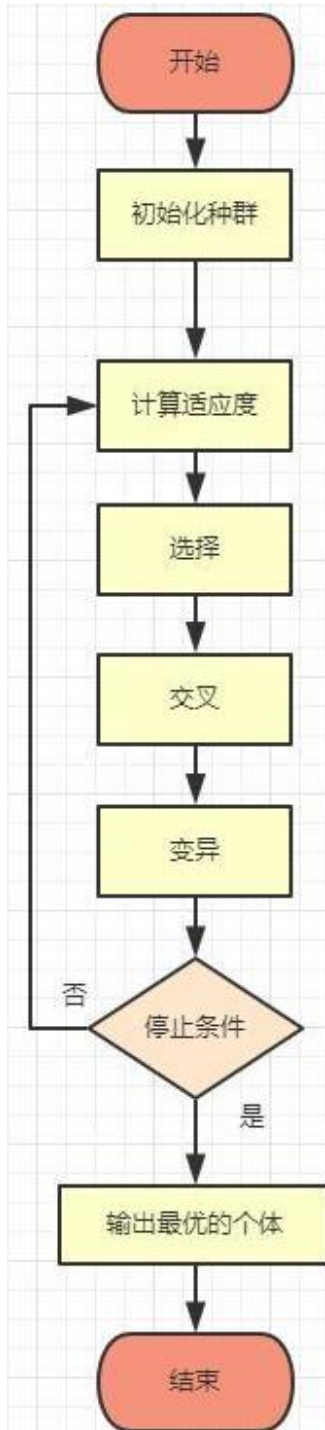
以概率 p_m 变异 c

end for

群体 C_0 复制为群体 C_1

更新最优的 B

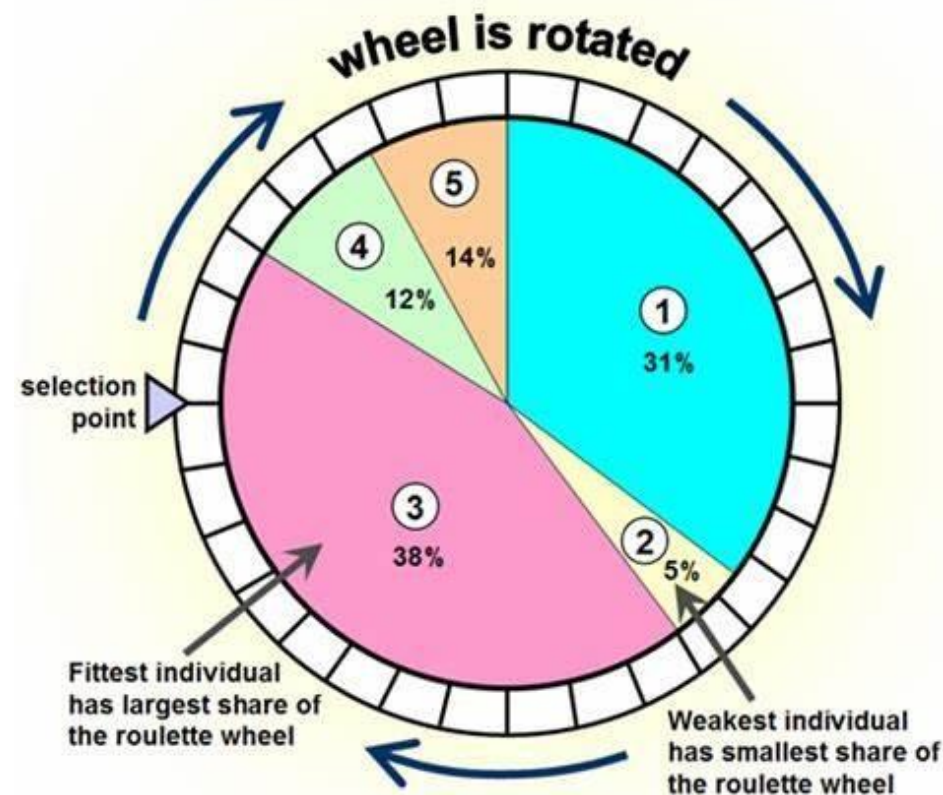
End



基本算子 1：选择 (Selection)

以适应度为权重概率选出一群个体的染色体，并繁衍下一代。构造选择方式灵活：

1. **轮盘赌**选择策略: 一种回放式随机采样方法。每个个体进入下一代的概率等于它的适应度值与整个种群中个体适应度值和的比例。选择误差较大。
2. 随机竞争选择: 按轮盘赌选择一对个体，然后让这两个个体进行竞争，适应度高的被选中。如此反复直到选满为止。机制类似于淘汰制。
3. 最佳保留选择:
 - ① 按轮盘赌选择方法执行遗传算法的选择操作;
 - ② 将当前群体中适应度最高的个体结构完整地复制到下一代群体中。

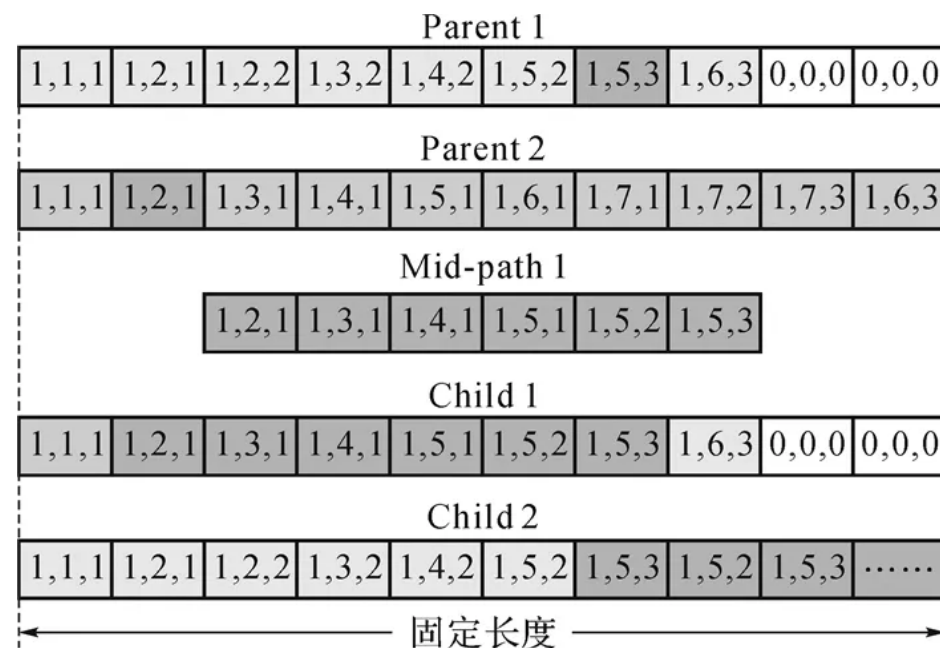


基本算子 2：交叉 (crossover)

二进制编码：通常方式是1步交叉，即将父母一方的k位初始序列取代第二个亲本的初始k个位置形成新的数位串，以产生后代，其中位置k被随机选择。

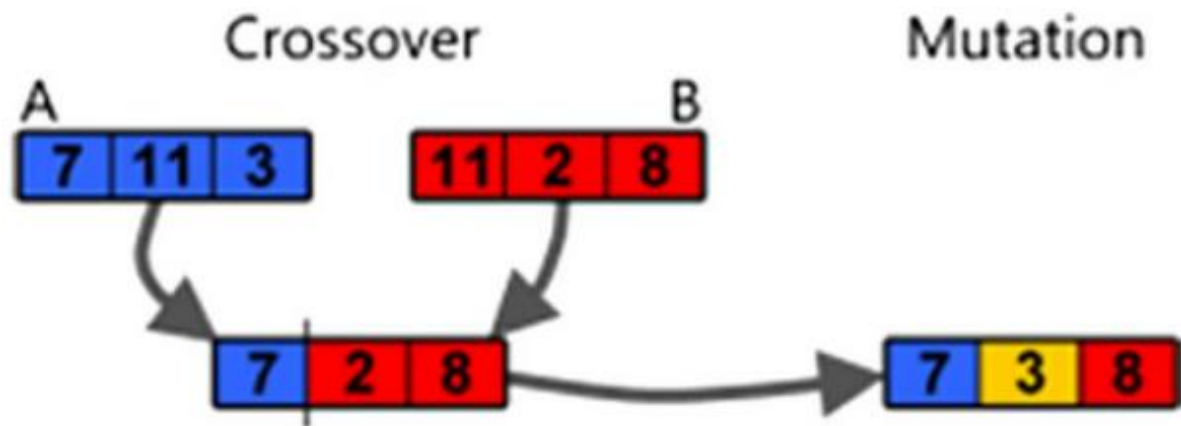
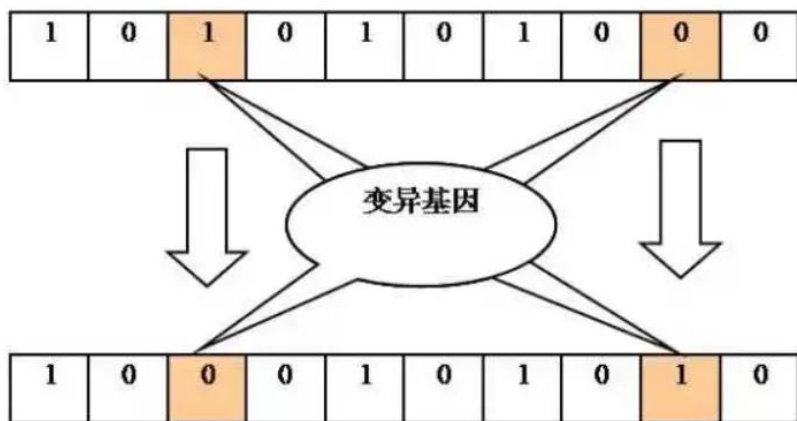


实值编码：先随机选择2个parent，然后选择部分进行交换：比如群体的size是N，先选择最好的M个，就需要在产生N-M个才能维持好种群的大小。假设每个染色体的长度是K，那么先把M个种群中的染色体的第一个基因，按照倍数复制成长度为N-M，接着随机再给这个N-M排序下，组成了N-M个后代的第一个基因，以此类推，就完成了任意2个parent的基因交换和重组。其实方式多样：



基本算子 3：变异 (mutation)

- 二进制编码：随机选择二进制串的一位或多位进行翻转或逆序：



- 实数值编码：遍历N个染色体上的K个基因，然后产生随机数，如果这个随机数小于X，就给这个基因做mutation，如果大于则不更改，具体的实现的时候可以灵活处理。

遗传算法 - 灵活的流程

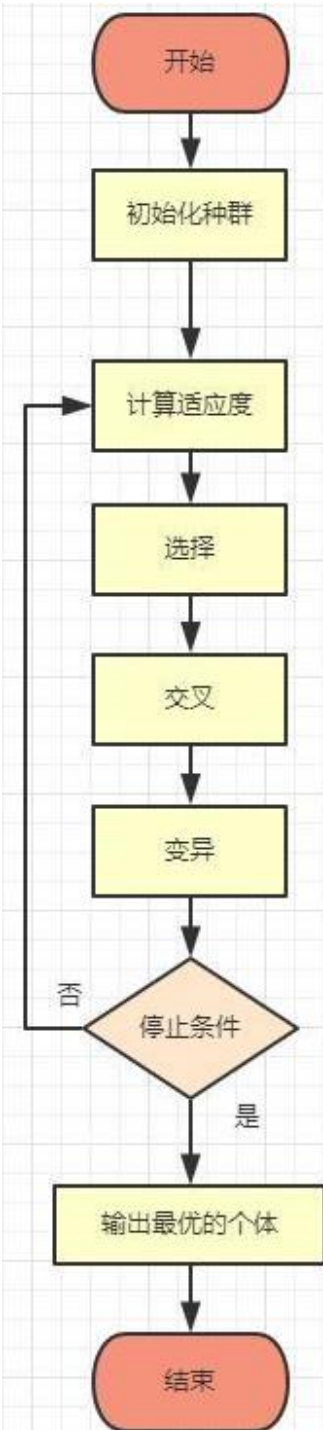
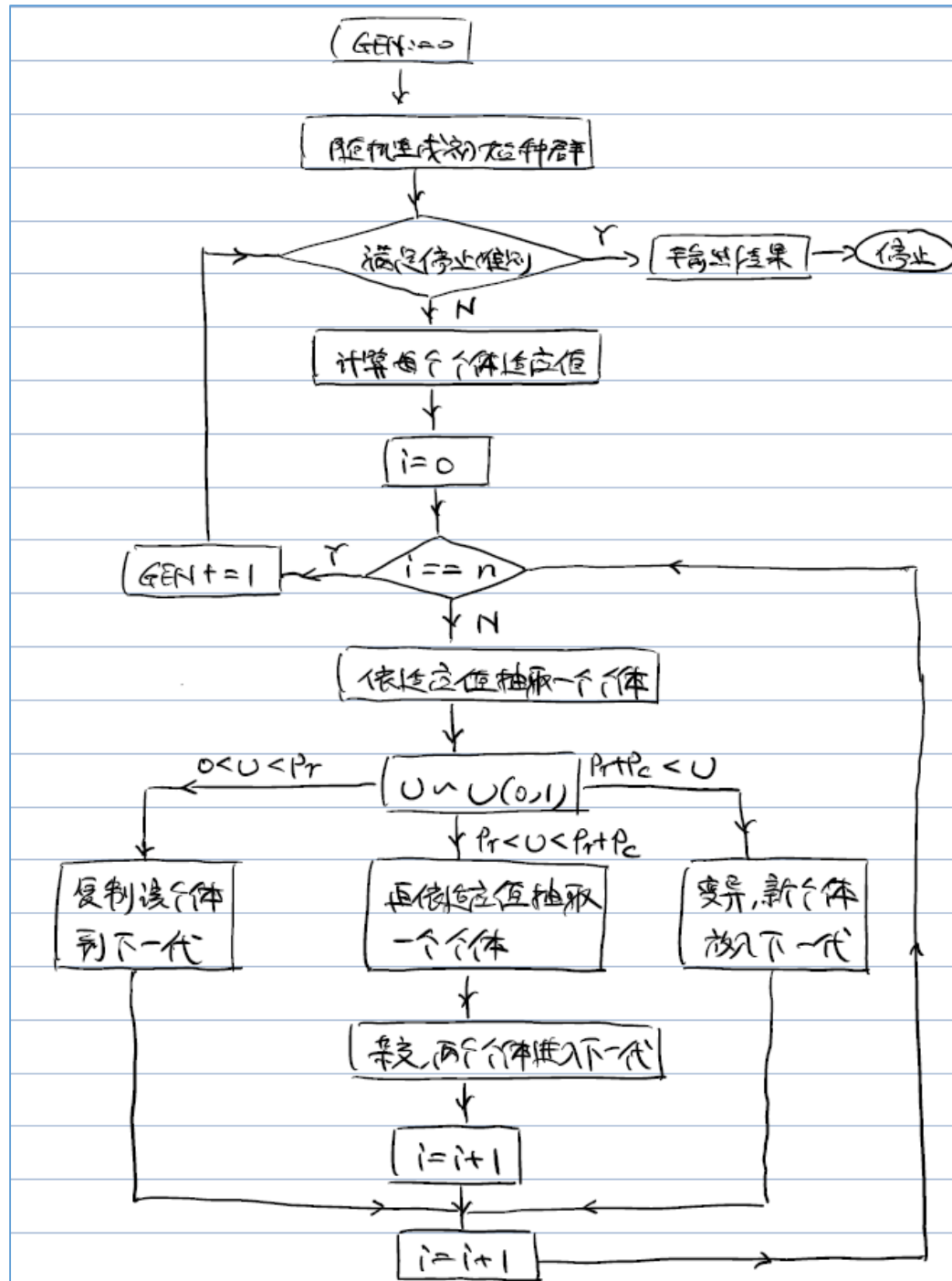
- 依如下概率抽取个体 x_i , 并记 f_i 为该个体的表现值:

$$p_i = \frac{f_i}{\sum_{i=1}^n f_i}$$

- 对抽取的个体 x_i , 分别以概率 p_r, p_c, p_m 进行复制、交叉和变异, 这里要求:

$$\begin{cases} p_r, p_c, p_m \geq 0, \\ p_r + p_c + p_m = 1. \end{cases}$$

- 终止运行的条件:
 - 传代次数/运行时间 限制, 如1万代/1周
 - 适应度无显著变化



例1: n 阶方阵 $C = (c_{ij})$ 的积和式

➤ 行列式

$$\det(C) = \sum_{\sigma} \text{sign}(\sigma) \prod_{i=1}^n c_{i,\sigma(i)},$$

其中, σ 是 $\{1, 2, \dots, n\}$ 的一个置换, $\sigma(i)$ 是置换 σ 中的第 i 个数.

➤ 积和式 - 运算量为 $O(n!)$

$$\text{perm}(C) = \sum_{\sigma} \prod_{i=1}^n c_{i,\sigma(i)}.$$

1. 若 C 中有一个0元素, 则积和式中有 $(n-1)!$ 项为0
2. 若 C 的某一行或某一列元素为0, 则积和式为0
3. 交换矩阵的任意两行 (列) 不改变积和式的值

$n:m$ 积和式 (组合) 优化问题

在具有 m 个元素1的 n 阶0-1矩阵的集合中, 最大的积和式是多少?

例. 对于3: 5积和式问题, 最大积和式是2。其矩阵如下:

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

矩阵 C 不唯一, 因为:

$$\text{perm} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} = \text{perm} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = 2.$$

n:m积和式优化问题 - 遗传算法

1. 定义染色体（编码方案）：

用m个1的n阶0-1矩阵作为染色体；

回顾“编码原则”：

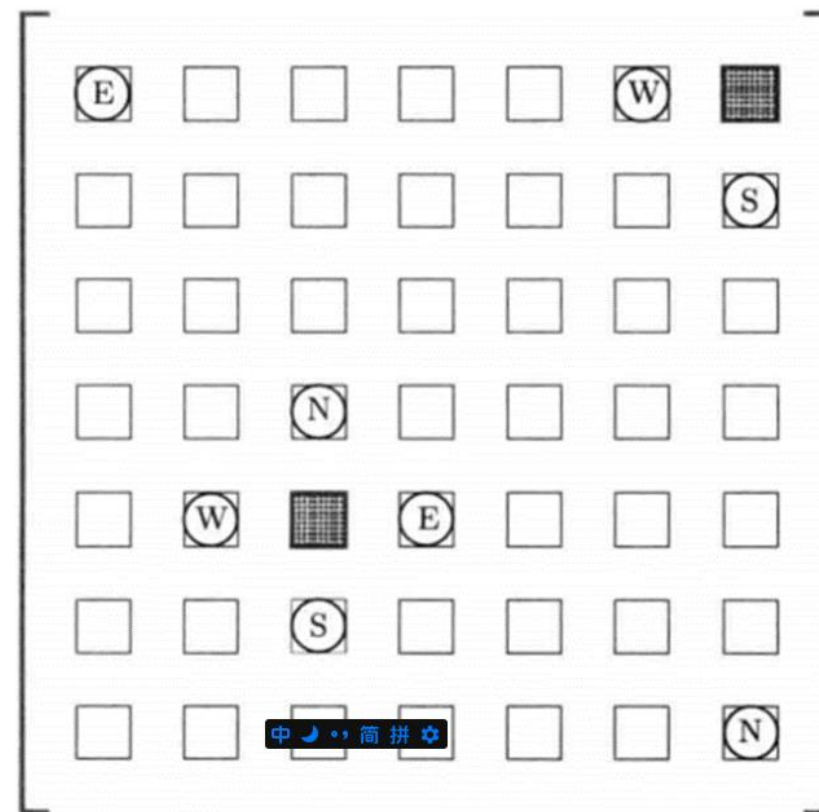
- 有意义：应使用能易于与所求问题相关
- 最小字符集

2. 定义适应度函数：

矩阵的积和式 $\text{perm}()$ ；

3. 定义遗传算子：(变异和交叉)

- 变异：“东南西北环绕式的领域交换”
- 交叉：模板化全局交换



1. 交叉算子的实现 - 模板化全局交换

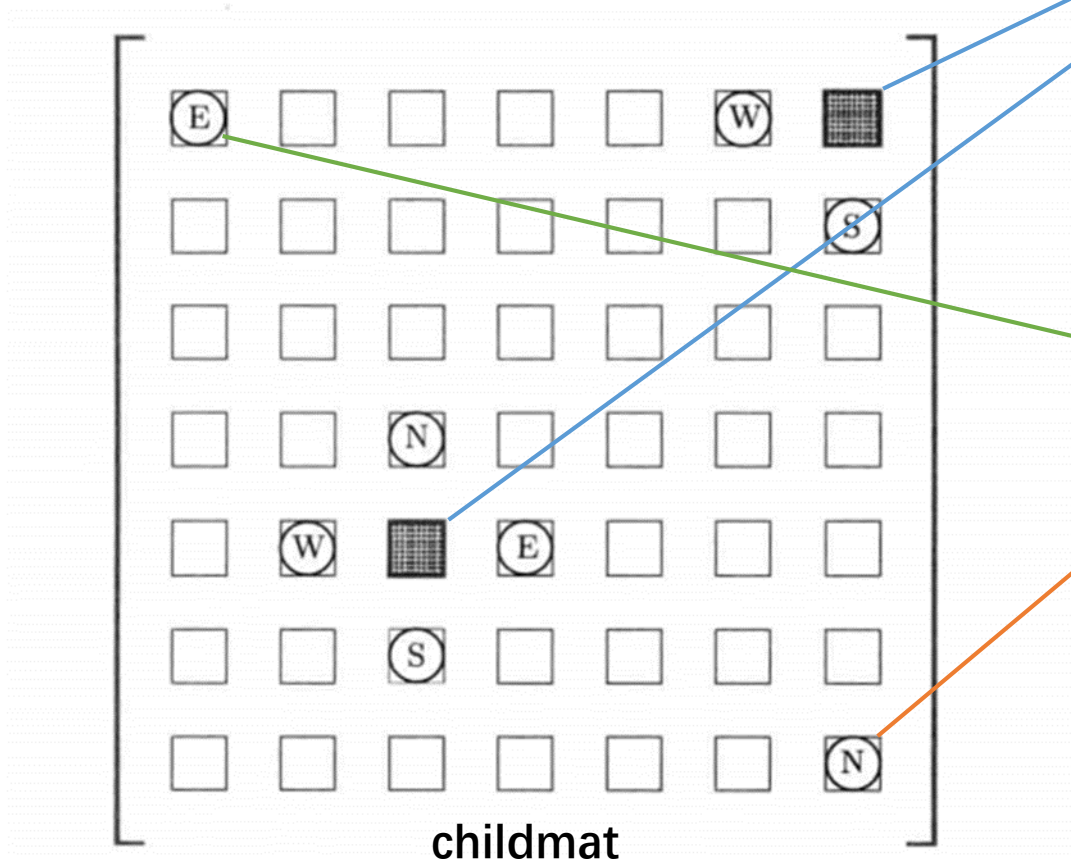
设矩阵A和B是被选出来进行交换的两个矩阵。首先，将矩阵的元素按行的顺序依次头为相接地排成一维数组，并将数组地头尾连成环绕地形式。然后，随机选出一个共同的起点位置，沿着两个数组的元素向后移动直到第一次出现两个不同的元素，则交换这两个元素，继续沿着数组的元素移动，不断地及进行类似的交换元素，这个过程直至遇到第一次出现不同元素的位置时结束。

```
function newmat = crossover(mat1, mat2)
[m, n] = size(mat1);

% 一种实现方式，可向量化优化之
for i = 1:m
    for j = 1:n
        if mat1(i, j) ~= mat2(i, j)
            mat1(i, j) = 1 - mat1(i, j);
            mat2(i, j) = 1 - mat2(i, j);
        end
    end
end

newmat = mat1;
end
```

2. 变异算子的实现



```
function childmat = mutate(childmat)
[tx,ty] = size(childmat);
x = randi(tx);
y = randi(ty);
```

% “东南西北环绕式的领域交换”

```
ewns = [x+1, x-1, x, x; ...
        y, y, y-1, y+1]';
```

for i = 1:8 % 超出边界的情况周期化，如左图

```
if ewns(i) == 0
```

```
    ewns(i) = tx;
```

```
elseif ewns(i) == tx+1
```

```
    ewns(i) = 1;
```

```
end
```

```
end
```

```
for i = 1:4
```

```
    coord = ewns(i,:);
```

```
    if childmat(coord(1), coord(2)) ~= childmat(x,y)
```

```
        childmat(coord(1), coord(2)) = ...
```

```
            1 - childmat(coord(1), coord(2));
```

```
        childmat(x,y) = 1 - childmat(x,y);
```

```
    end
```

```
end
```


- 注意要保证1的数量不变!

算例主程序

1. 小规模 3:5:

```
popnum = 12;  
n = 3; % 阶数  
m = 5; % 1的个数  
maxGen = 1000;  
  
perm_GA(popnum,n,m,maxGen);
```

最大值2，此时矩阵为：



0	1	1
0	1	1
1	0	0

2. 中等规模

```
perm_GA(12,10,40,1000);
```

最大值为:1074,矩阵为:

1	0	1	0	0	1	0	0	1	0
0	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	0	0	1	0
1	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	0	1
1	1	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	1	1	0	1
0	0	0	1	1	1	0	0	1	0
1	1	0	1	1	0	0	0	1	0

3. 更大点

```
perm_GA(12,14,40,5000);
```

% 轮盘赌选择

tournamentSize = 4; %设置大小

for k=1:popnum

% 选择父代

tourPopDistances = zeros(tournamentSize, 1);

for i = 1:tournamentSize

randomRow = randi(popnum);

tourPopDistances(i,1) = B(randomRow);

end

% 选择最好的A

parent1 = max(tourPopDistances);

parent1mat = mat(:, :, find(B==parent1, 1));

for i = 1:tournamentSize

randomRow = randi(popnum);

tourPopDistances(i,1) = B(randomRow);

end

% 选择最好的B

parent2 = max(tourPopDistances);

parent2mat = mat(:, :, find(B==parent2, 1));

% 执行交叉

submat = crossover(parent1mat, parent2mat);

% 执行变异

submat = mutate(submat);

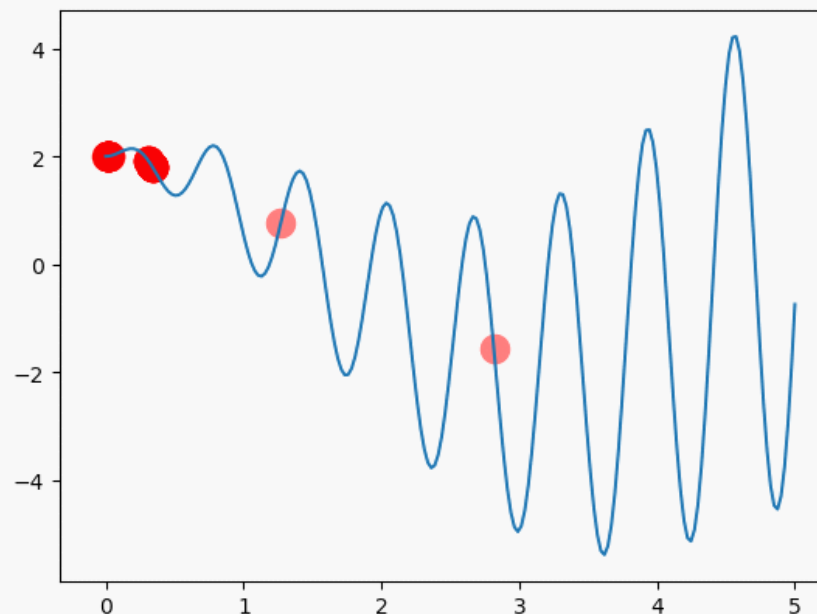
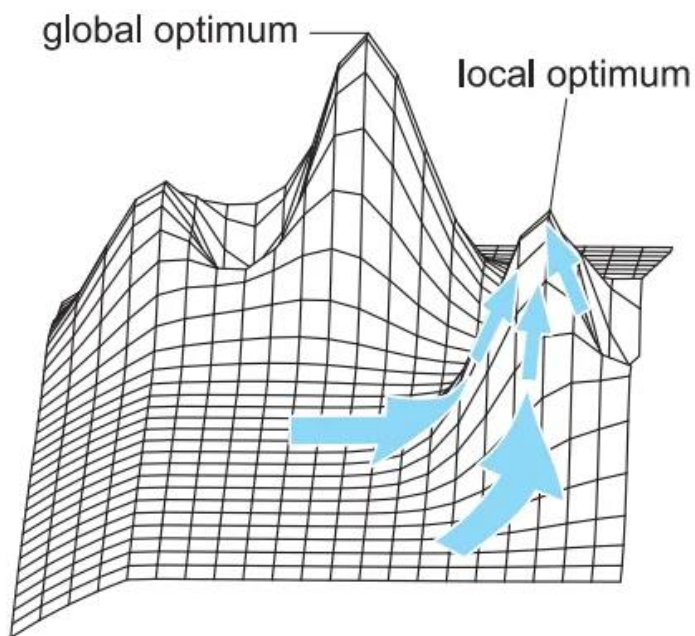
% 获得新一代

offspring(:, :, k) = submat;

end

粘滞现象及修正

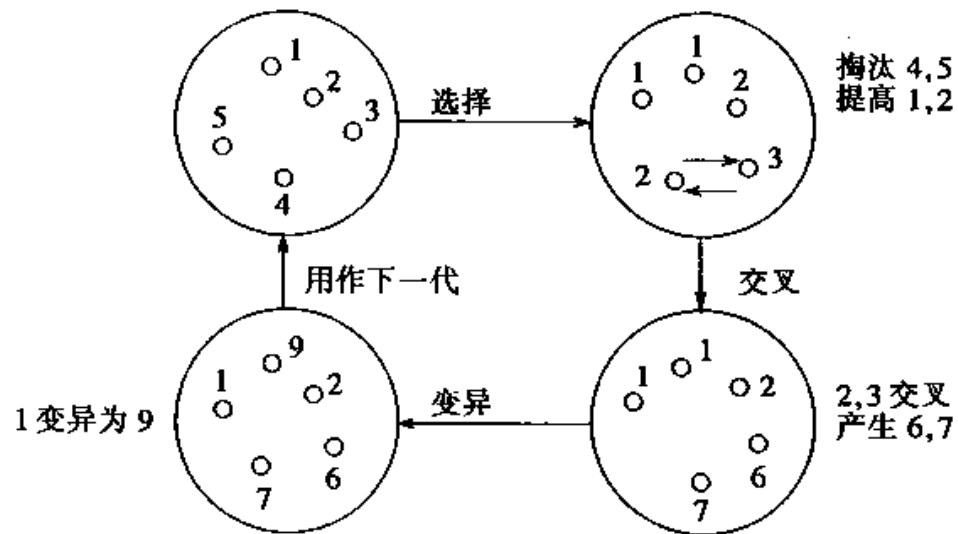
- 粘滞/早熟现象：遗传算法有时适应度的改善变得越来越慢，似乎该算法已基本粘滞或“陷入”局部最优处。



- 修正算法：修改当前交叉操作的群体范围
 - 具体地，可将当前状态与可选状态空间中的任意/随机状态进行交叉。

模式定理 (Schema Theorem)

在遗传算子选择、交叉、变异的作用下，确定位数少、定义长度短以及平均适应度高于种群平均适应度的模式在子代中呈指数级增长。



记 $m(H, t)$ 为时间代 t 特定模式 H 有 m 个代表串在种群 $A(t)$ 中。模式

$A_i(t)$ 再生概率 $p_i = \frac{f_i}{\sum_{i=1}^n f_i}$, 整个种群的适应度: $\bar{f} = \frac{\sum_{i=1}^n f_i}{n}$. 那么,

- 当采用非重叠的 n 个串的新种群代替 $A(t)$ 时, 有:

$$m(H, t+1) = m(H, t) \times n \times \frac{f(H)}{\sum_{i=1}^n f_i} = m(H, t) \times \frac{f(H)}{\bar{f}}$$

- 假设某个特定模式增长的速度为 $c\bar{f}$, 即 $f(H) = (1 + c)\bar{f}$ 则有

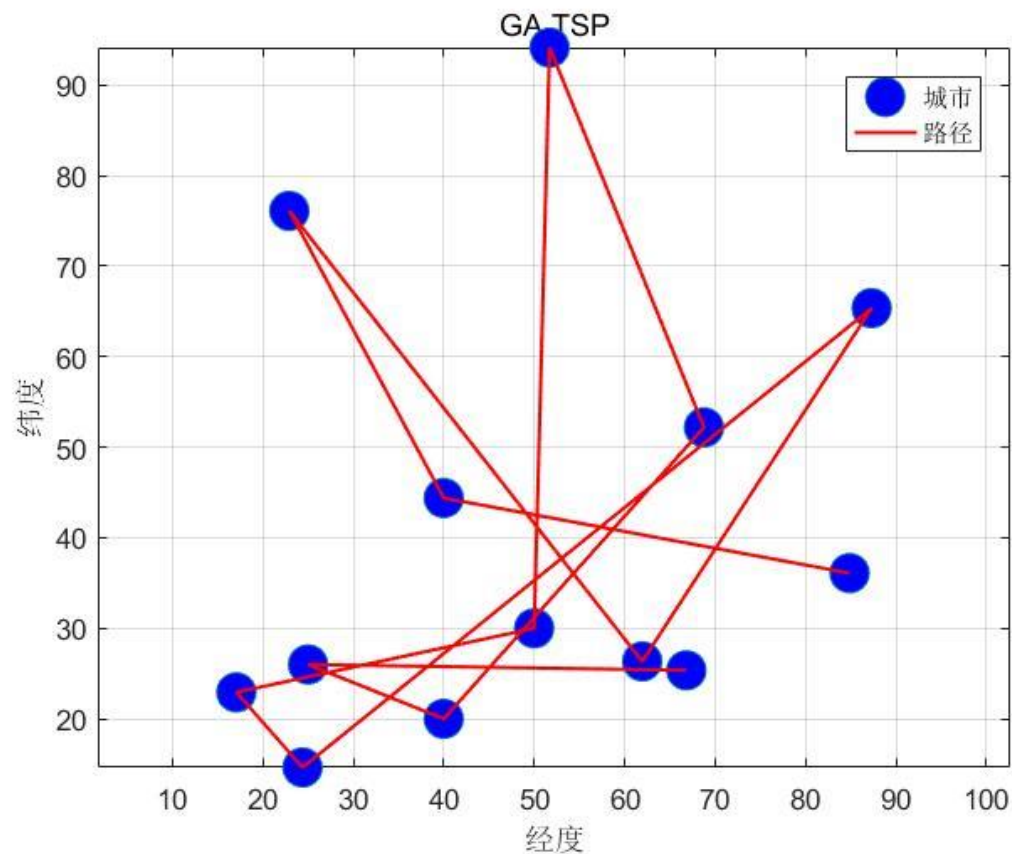
$$m(H, t+1) = m(H, t) \times \frac{(1 + c)\bar{f}}{\bar{f}} = \dots = (1 + c)^t m(H, 0)$$

- ✓ 一些特定的（优秀的）个体，在代代遗传的过程中，会向着全局最优解更快地收敛！
- ✓ “具有低阶、短定义距以及平均适应度高于种群平均适应度的模式”被定义为**积木块**。
- ✓ **积木块** (building block) 假设：“积木块”在遗传操作下相互结合，最终会接近全局最优解！

例2：旅行商问题(TSP)

1. 对旅行商问题进行编码，编码方式如下：
每个城市都有一个编号，以5个城市为例，
如：旅程(1-3-2-4-5),则编码为(1 3 2 4 5)
2. 产生初始种群，采用完全随机产生方式。

```
pop = zeros(popSize, cityNum);  
for i=1:popSize  
    pop(i,:) = randperm(cityNum);  
end
```



初始地图

TSP的适应度函数

TSP目标是路径总长度最短，因此，适应度函数可以为：

$$f(w_1, \dots, w_n) = \frac{1}{\sum_{j=1}^n d(w_j, w_{j+1})}$$

```
function [ fitnessvar, sumDistances,minPath, maxPath ] = ...  
                                fitness( distances, pop )  
  
[popSize, col] = size(pop);  
sumDistances = zeros(popSize,1);  
fitnessvar = zeros(popSize,1);  
for i=1:popSize  
    for j=1:col-1  
        sumDistances(i) = sumDistances(i) + ...  
                                distances(pop(i,j),pop(i,j+1));  
    end  
end  
minPath = min(sumDistances);  
maxPath = max(sumDistances);  
for i=1:length(sumDistances)  
    fitnessvar(i,1) = (maxPath - sumDistances(i,1) +  
        0.000001) / (maxPath - minPath + 0.00000001);  
end  
end
```

TSP问题的遗传算子

采用**赌轮选择**机制

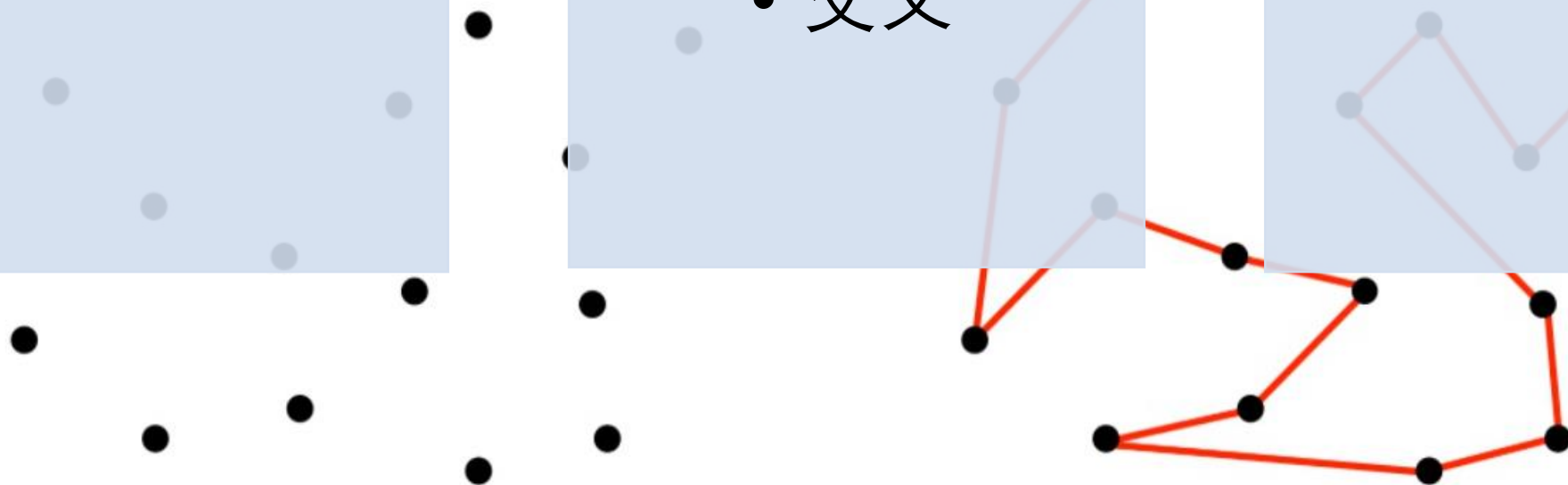
- 选择

采用**部分随机匹配交叉**,即随机选取两个交叉点,定义这两点间的区域为匹配区域,并交换两个父代的匹配区域。

- 交叉

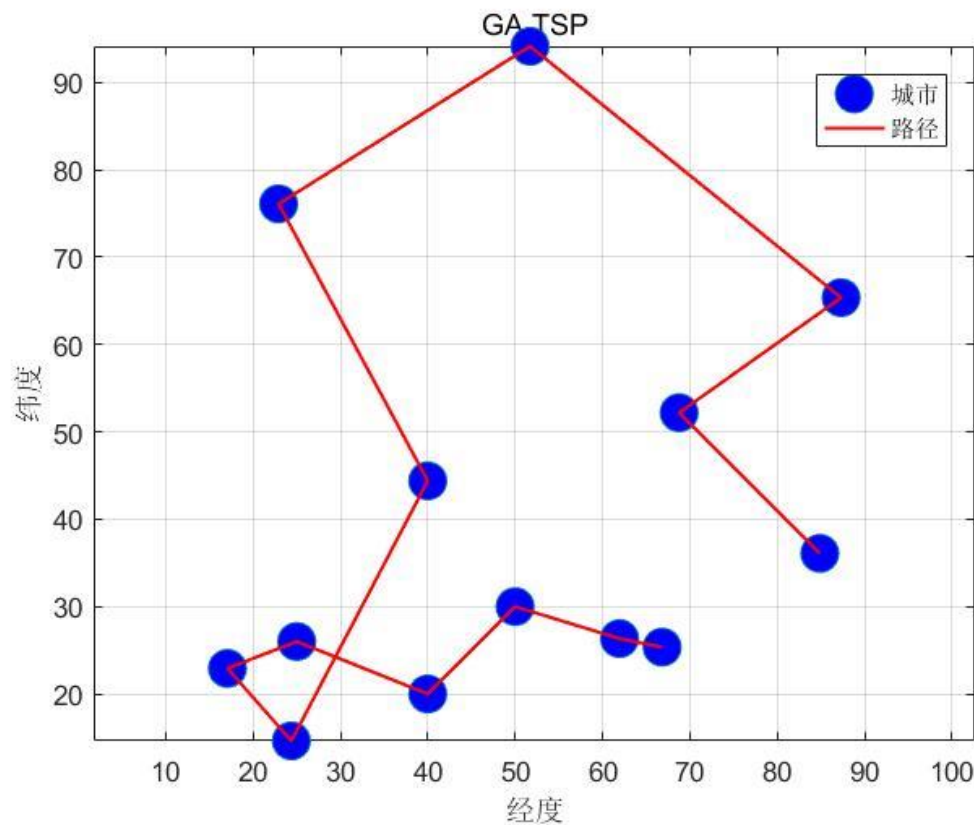
随机在序列中抽取两个城市,然后**交换**它们的**位置**

- 变异

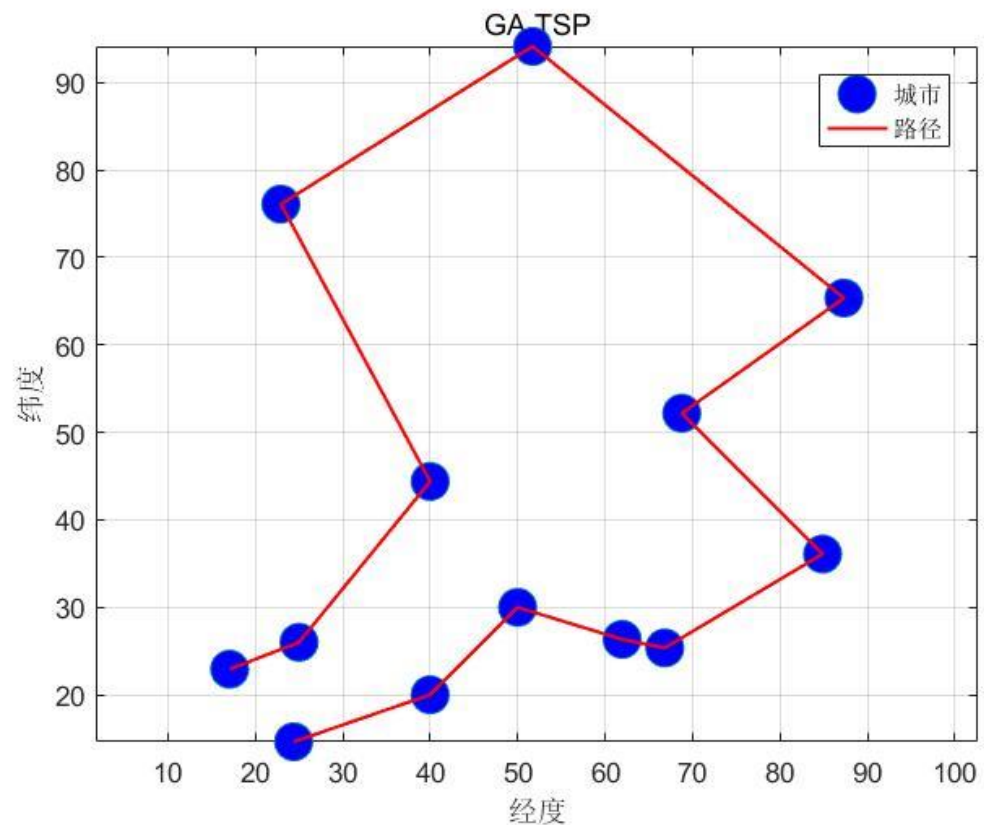


TSP实例分析

cities= [66.83, 61.95, 40, 24.39, 17.07, 22.93, 51.71, 87.32, 68.78, 84.88, 50, 40, 25; ...
25.36, 26.34, 44.39, 14.63, 22.93, 76.1, 94.14, 65.36, 52.19, 36.09, 30, 20, 26];



早熟现象(局部极值)



最短路径

遗传算法 - 总结

与传统优化算法不同之处：

- 针对参变量进行编码
- 从一个点的群体开始搜索
- 利用适应值信息，无需导数
- 采用概率转移规则推动演化

➤优越性：

1. 全局（可能达到）优化
2. 隐含的并行性：编码的不同部分以及不同的操作算子可同时进行
3. 以编码、适应度函数为导向，所需问题背景少、“过程”解释统一

➤劣势：

1. 过多依赖于适应度函数，但需要使用者自行提供，且无固定标准
2. 群体大小选择不当可能会过早收敛，错过全局最优解
3. 计算参数（群体大小、交换频率、突变频率等）稍有不同可能会得到不同结果
4. 若染色体过长(基因数过多)，群体规模、计算时间急剧增加

遗传算法(Genetic Algorithm)

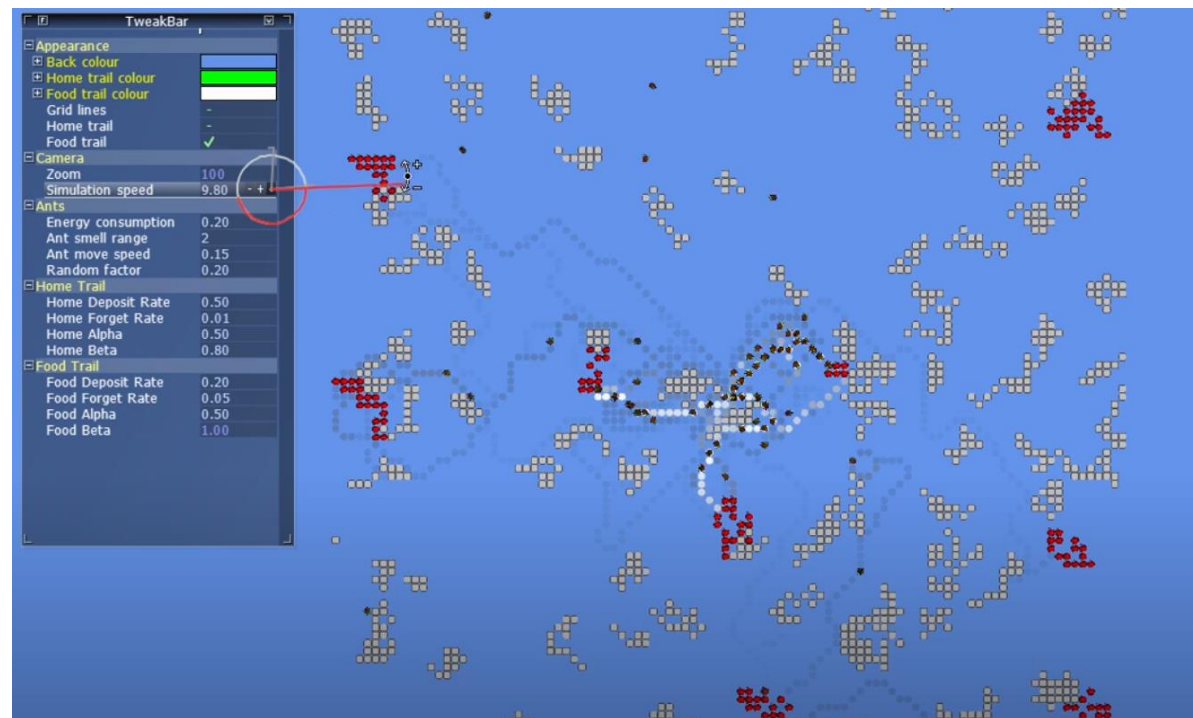
- 1967, J. D. Bagley最早（在其博士论文中）提出遗传算法的概念
- 1975, John. H. Holland 开展系统性研究,出版专著《自然系统和人工系统的自适应》（**Adaptation in Natural and Artificial Systems**）标志着遗传算法的诞生
- 1989年, D.E.Goldberg 总结出基本遗传算法(Simple Genetic Algorithms, SGA), 出版专著《**Genetic Algorithms in Search , Optimization, and Machine Learning**》
- 1991年, L.Davis编辑出版《遗传算法手册》（**Handbook of Genetic Algorithms**）
- 1990s, 逐步发展成为**进化计算**研究的重要分支, 应用领域广泛: 模式识别、神经网络、图像处理、机器学习、优化控制、生物科学、社会科学等
- 21世纪, 以不确定性、非线性、时间不可逆为内涵的复杂性科学成为一个研究热点。遗传算法与元胞自动机、混沌理论、人工智能一样, 对今后**计算技术**有重大影响



J.H. Holland (1929-2015) , 遗传算法之父,复杂理论和非线性科学的先驱,约翰·霍普金斯大学心理学和电气工程与计算机科学教授.

2. 蚁群算法简介

一种基于**种群**的**启发式**随机搜索算法



一个参考程序： Ant Colony Simulation

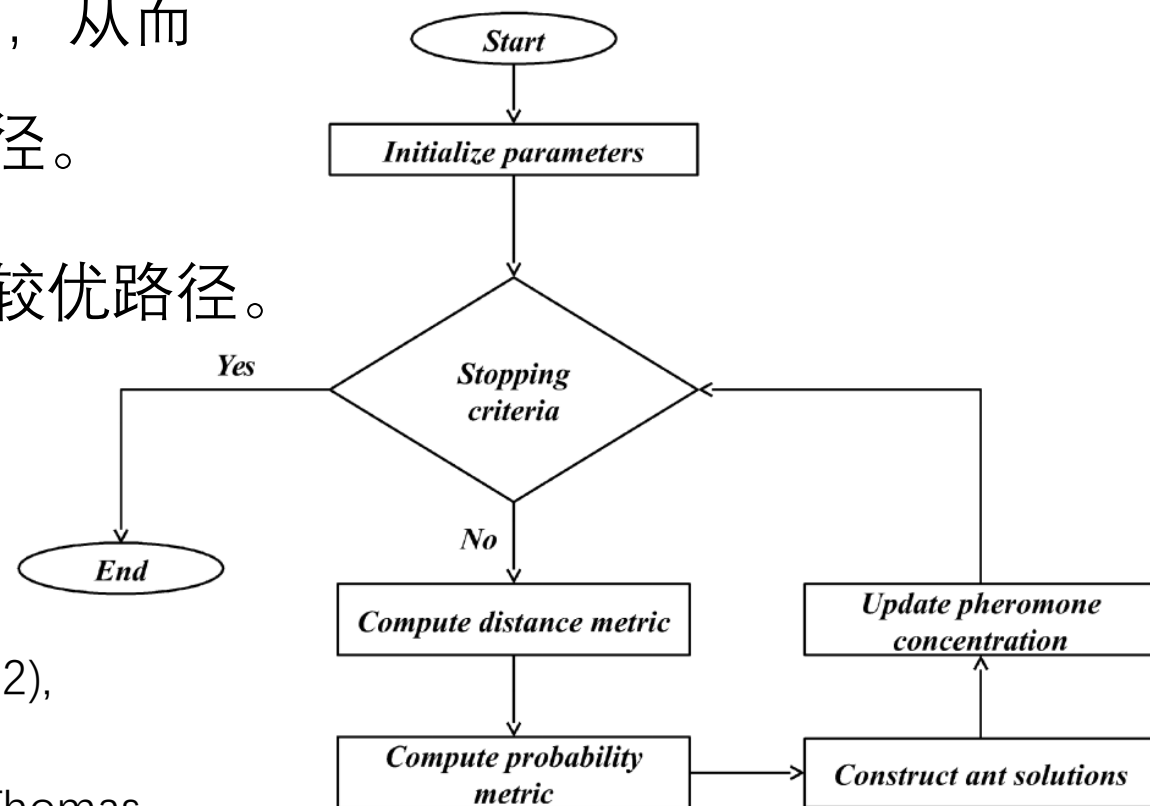
<http://alexbelezjaks.com/works/ant-colony-simulation/>

蚁群算法(Ant Colony Optimization)

- 蚁群算法是受自然界中蚂蚁觅食的行为启发的，蚁群通过“信息素”(pheromone)来实现信息的传递，从而得以在寻找食物的过程中选择一个较短的路径。
- 蚁群算法核心是通过信息素的更新实现选择较优路径。
 - 不同信息素的定义会影响整体算法
 - 不同问题信息素的定义也完全不同

■ An investigation of some properties of an “ant algorithm”
A. Coloni, M. Dorigo, V. Maniezzo *et al.* PPSN, vol. 92 (1992), pp. 509-520

■ Parsons S . Ant Colony Optimization by Marco Dorigo and Thomas Stützle, MIT Press, 305 pp. ISBN 0-262-04219-3[M]. Cambridge University Press, 2005.



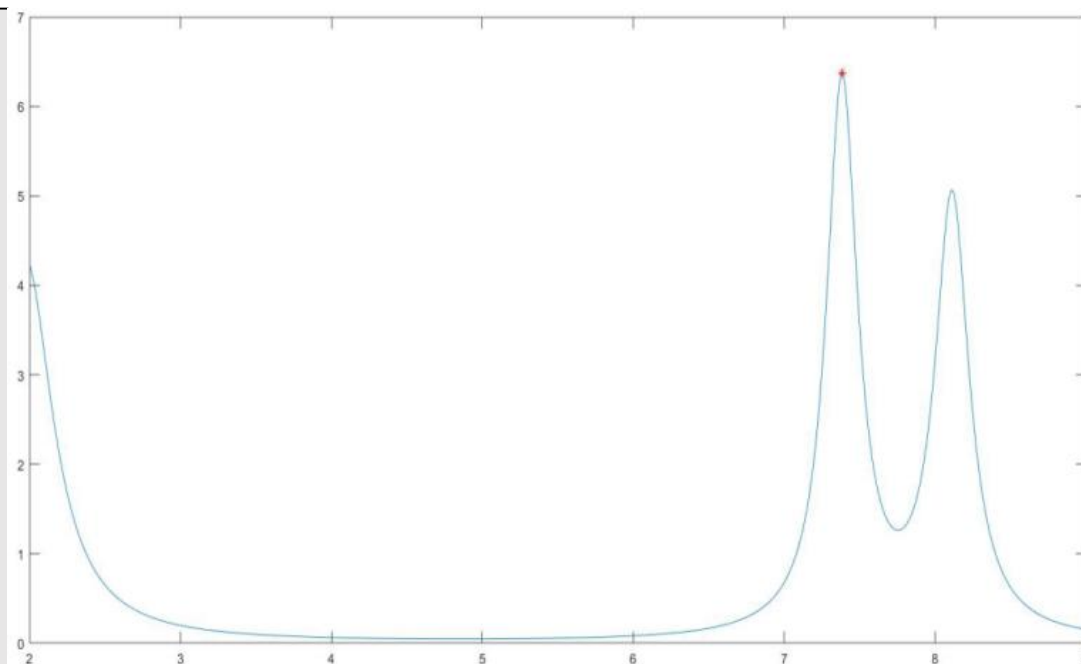
例1： 计算Shekel函数在区间 $2 \leq x \leq 9$ 的最大值

- 基本思想： 以shekel函数值作为信息素(为了让蚂蚁向较大处行进)
- 核心算法： 根据信息素来确定蚂蚁移动方向的概率大小， 信息素越大的方向， 蚂蚁移动的概率越大：

```
tau(i) = shekel(x(i)); %信息素

% 转移概率
P(T,i)=(tau(BestIndex)-tau(i)) / tau(BestIndex);

if P(T,i)<P0
    temp1=x(i)+(2*rand-1)*lamda;
else
    temp1=x(i)+(Upper_1-Lower_1)*(rand-0.5);
end
```



例2：旅行商问题(TSP)

与遗传算法类似，编码方式如下：每个城市都有一个编号，以5个城市为例：

旅程(1-3-2-4-5),则编码为(1 3 2 4 5)

蚁群算法参数包括蚂蚁数量，信息素因子，挥发因子等等

<code>m = 50;</code>	<code>% 蚂蚁数量</code>
<code>%% 影响城市的转移概率</code>	
<code>alpha = 1;</code>	<code>% 信息素因子</code>
<code>beta = 5;</code>	<code>% 启发函数因子</code>
<code>%% 影响信息素的更新</code>	
<code>rho = 0.1;</code>	<code>% 信息素挥发因子</code>
<code>Q = 1;</code>	<code>% 信息因子常系数</code>
<code>iter_max = 500;</code>	<code>% 最大迭代次数</code>

主要流程

城市间转移概率综合了信息素和启发函数的信息，常见定义为

$$P = \text{信息素}^{\alpha} + \text{启发函数值}^{\beta}$$

然后利用轮盘赌来确定当前蚂蚁的下一个访问城市

```
for j = 2:n
    tabu = Table(i,1:(j - 1)); % 已访问的城市集合(禁忌表)
    allow_index = ~ismember(citys_index,tabu);
    allow = citys_index(allow_index); % 待访问的城市集合
    P = allow;
    for k = 1:length(allow) % 计算城市间转移概率
        P(k) = Tau(tabu(end),allow(k))^alpha * Eta(tabu(end),allow(k))^beta;
    end
    P = P/sum(P); % 归一化概率，方便之后与rand产生的随机数比较
    Pc = cumsum(P); % 想一想：cumsum的作用是什么？
    target_index = find(Pc >= rand); % 轮盘赌法选择下一个访问城市
    target = allow(target_index(1)); % 想一想：为什么要下标（1）？
    Table(i,j) = target; % 填坑
end
```

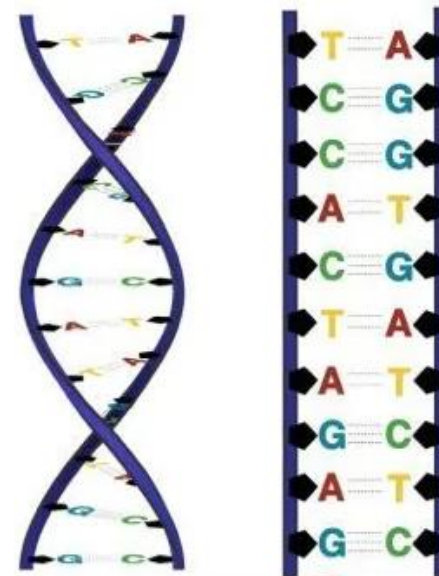

核心步骤：更新信息素表

根据蚂蚁的路径长度来更新信息素表，使得信息素和路径长度负相关

```
% 更新信息素
Delta_Tau = zeros(n, n);
for i = 1:m    % 逐个蚂蚁计算
    for j = 1:(n - 1)    % 逐个城市计算
        Delta_Tau(Table(i,j),Table(i,j+1)) = Delta_Tau(Table(i,j),Table(i,j+1)) + Q/Length(i);
    end
    Delta_Tau(Table(i,n),Table(i,1)) = Delta_Tau(Table(i,n),Table(i,1)) + Q/Length(i);
end
Tau = (1-rho) * Tau + Delta_Tau;
```

仿生智能算法

- 1858年，[伦敦林奈学会](#)(1788)发表了[达尔文](#)(1844)与[华莱士](#)(1858)的论文
- 1865年，[孟德尔](#)（Gregor Johann Mendel）发现遗传定律
- 1950s，生物学家揭示了基因在自然演化过程中的作用
- 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
 - **遗传算法**（Genetic Algorithm）是仿真生物遗传学和自然选择机理
 - 进化(Evolutionary)发展了遗传算法，进化策略（选择、交叉、变异等）有很多变化
 - 一群（弱）智能个体([Agent](#))通过相互协作而表现出基于群体协作的随机搜索算法
 - **蚁群算法**（Ant Colony Optimization, ACO）受到自然界中蚂蚁觅食行为的启发
 - 人工蜂群算法（Artificial Bee Colony, ABC）模仿蜜蜂行为
 - 粒子群优化算法（Particle Swarm optimization, PSO）模拟鸟群觅食行为
 - 人工神经网络算法（Artificial Neural Network, ANN）模仿生物神经网络



越来越多的群体智能算法有必要吗？

1 动物篇

猫群优化算法、鱼群优化算法、黑猩猩优化算法、海象优化算法、受精优化算法（我没有开车确实有这个算法）、耳廓狐优化算法（耳廓狐是什么鬼？）、宽吻海豚算法（是不是每种海豚都可以来一个算法呢？）、白骨顶鸡优化算法、科莫多巨蜥算法、北方苍鹰优化算法、猫和老鼠优化器、侏儒猫鼬优化算法、缎蓝园丁鸟优化器、哈里斯鹰优化算法、混沌灰狼算法（我是在玩游戏王卡牌吗？）、灰太狼优化算法（喜羊羊算法那肯定也得安排上了）、胡桃夹子鸟优化算法

2 昆虫篇

蟑螂优化算法、天牛群优化算法、蜘蛛优化算法、蜻蜓优化算法、蝴蝶优化算法、蚱蜢优化算法、飞蛾扑火优化算法、蝗虫优化算法、蜉蝣优化算法、蚁狮优化算法、黑寡妇算法（我还以为是复仇者联盟呢）、人工蝴蝶优化算法、蝗虫搜索算法、群居蜘蛛算法

3 植物篇

树木生长算法、入侵杂草优化算法、花朵授粉算法、树种优化算法、向日葵算法、树木生理算法、植物孢群优化算法、蒲公英优化器

4 人类社会篇

贫富优化算法、登山队优化算法、循环系统优化算法（这也能是一个算法）、儿童绘画发展优化算法（名字略微有点长）、阿基米德优化算法（那是不是可以搞一个爱因斯坦算法）、战争策略算法、战争艺术优化算法、厨师优化算法（是不是可以搞一个理发员优化算法，快递员优化算法呢）、阿里巴巴和四十大盗算法（看名字我根本不知道这是一个算法）、法医调查优化算法、社交滑雪驱动算法（每一项体育运动都可以折腾一个算法出来）、政治优化算法、未来搜索算法、随机油漆算法、吉萨金字塔建造算法、学校算法、最有价值球员算法、大逃杀算法、饥饿游戏搜索算法、拔河优化算法、多元宇宙优化器、光学显微镜算法

原创 王源 科研式学习 2023-09-11 18:03 发表于广东

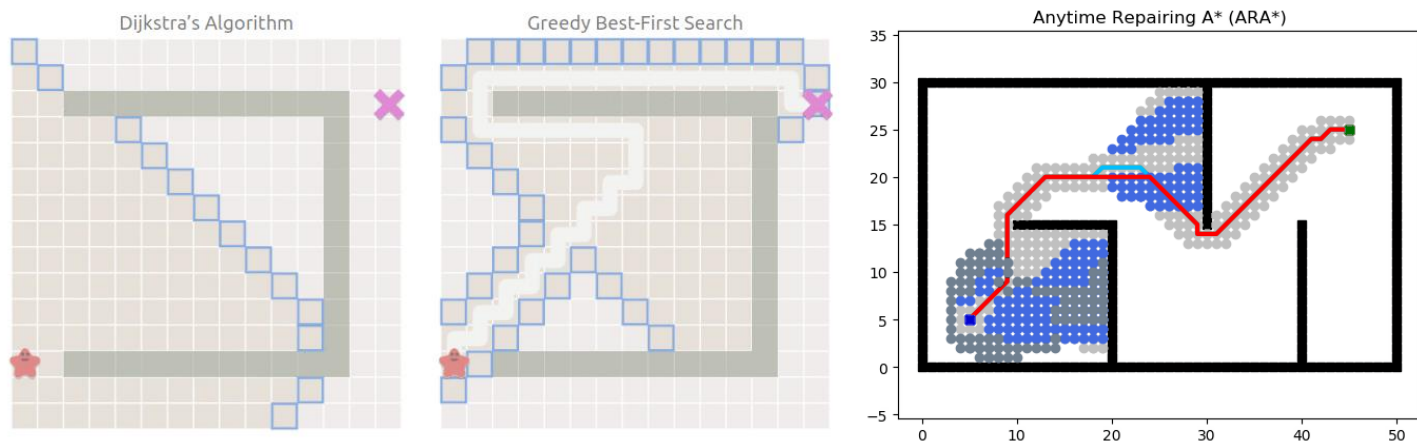
存在的问题：

- **旧瓶装新酒：**随意改参数，加算子，在测试算法上跑结果，挑结果，然后发表。这种现象在别的领域也确实存在，但是在群体智能优化算法领域由于其门槛低，导致这种现象尤为突出。
- **借生物学的概念来伪装：**通过这些伪装就是作者想要突出自己方法的创新性，然而本质都类似。如果刨去这些生物学伪装概念去看这些方法没有什么创新性。

前沿话题：A* 算法



- 与 Dijkstra 相比，A* 算法能找到最短路径，且搜索空间更小；
- 与 GBFS 相比，A* 算法能找到最短路径，但搜索空间更大。



内部人担忧“威胁人类生存”！OpenAI的神秘重大突破“Q*算法”究竟是什么？

常嘉帅 11-24 14:07

摘要：

据报道，Q*可能具备GPT-4所不具备的基础数学能力，或意味着与人类智能相媲美的推理能力，网友推测，这可能代表OpenAI朝着其设定的AGI目标迈出了一大步。

- 深度Q网络 (DQN) :

将Q-learning与深度神经网络结合，DQN可以处理高维状态空间，使其更适合复杂任务。

- 迁移学习:

使Q-learning模型在一个领域受过训练后能够将其知识应用于不同但相关的领域的技术，可能是通向AGI所需泛化的一步。

- 元学习:

在Q-learning框架中实现元学习可以使人工智能学会如何学习，动态地调整其学习策略，这对于AGI至关重要。

Q-learning在人工智能领域，尤其是在强化学习中，代表了一种重要的方法论。

毫不奇怪，OpenAI正在使用Q-learning RLHF来尝试实现神秘的AGI。

A*算法+Q-learning

一位斯坦福博士Silas Alberti表示，OpenAI的Q*可能与Q-learning有关，表示贝尔曼方程的最优解。

又或者，Q*指的是A*算法和Q学习的结合。



Silas Alberti
@SilasAlberti

What could OpenAI's breakthrough Q* be about?

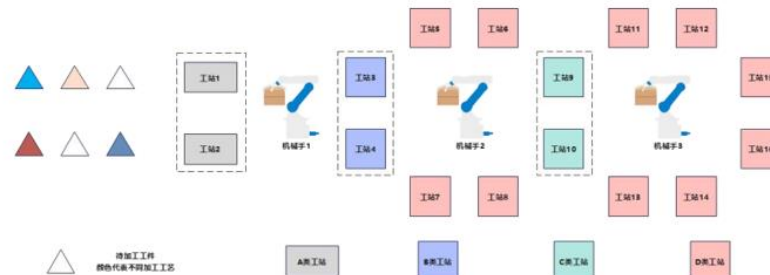
3. 一个应用案例

一类时空约束装配线平衡问题的随机模拟方法

难题：一种复杂组合设备调度优化问题

技术背景

□ 工厂生产设备调度是一类常见的调度优化问题，目前工厂中有一种复杂组合设备。该设备具有真空密闭、空间狭小、无缓存空间、成本极高等特征。该设备有多个加工区域，每个加工区域有多个工站和一个在各工站间搬运工件的机械手。设备的典型结构如图一所示。为节约成本，充分利用设备的生产能力，业界对此类设备的调度优化诉求越来越高。该调度问题的并行、重入、滞留时间、插单等场景给设计高效可靠的调度优化算法增加了极大的挑战。该问题在仅考虑单加工区域及滞留时间的情况下，已被证明为NP-hard问题。由于问题的复杂性，目前文献研究中考虑的因素相对较少，难以在工业界应用。



图一：典型设备结构

需求描述

该调度问题旨在满足各项约束的情况下，为机械手寻找最佳搬运序列。针对三个机械手、三个加工区域、多工站的组合设备场景（如图一），该调度问题以最小化加工时间为优化目标，并需要满足以下场景：

- A类工站：工站1和工站2，此类工站是工件加工的出入口，可分别放置25个工件；
- B类工站：工站3和工站4，是工件加工环境的切换工站，至多可同时处理2个工件，此类工站不能同时作业；

目前解决方案

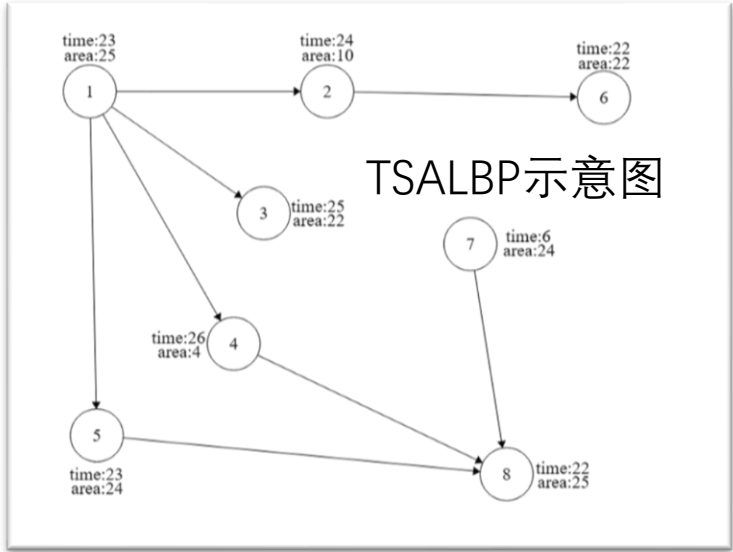
- **遗传算法**：约束条件考虑不全，可能产生不可行解。且当调度问题规模稍大时（工站点超过10个，调度75个工件），算法耗时过长（> 5min），远超业务可接受范围；
- **贪心策略**：每个机械手高频地寻找下一步可搬运的工件，存在算法短视和多机械手不能协同运作而导致死锁现象

技术诉求

- **数学模型诉求**：统一泛化的数学规划模型，可分段/分场景建模（但要处理好模型/场景之间的边界和协同问题），揭示复杂组合设备生产运行的机理和规律，建立适用于产业界迫切需要的可靠且高效的系统方法；建立应急调度策略，考虑临时插单、加工时间波动、加工设备故障等情况；
- **算法设计和实现诉求**：可采用包括但不限于数学规划类算法、（元/混合）启发式算法、机器学习类算法等；
- **求解效果诉求**：
 - ① **求解时间**：在图一设备的场景下，计算资源：i7-8700 CPU、3.2GHz、32G DDR4 2933/3200，调度75个工件，求解时间不超过1s；

时空装配线平衡问题(TSALBP)

- 装配线平衡问题：在装配线生产过程中，如何分配和安排工作任务，使得整个生产线的工作量能够尽量平均，并达到最高效率。
- 简单装配线平衡问题(SALBP)：任务分配仅受优先关系和循环时间的限制
- 时空装配线平衡问题更实用：
 - 任务顺序满足所有优先约束
 - 最大工作时间不大于生产节拍c
 - 最大工作面积不大于可用面积A
- TSALBP问题的类型则取决于任务站数量m，生产节拍c和可用面积A是作为固定值还是待优化变量



名称	m	c	A	类型
TSALBP-F	已知	已知	已知	F
TSALBP-1	最小化	已知	已知	OP
TSALBP-2	已知	最小化	已知	OP
TSALBP-3	已知	已知	最小化	OP
TSALBP-1/2	最小化	最小化	已知	MOP
TSALBP-1/3	最小化	已知	最小化	MOP
TSALBP-2/3	已知	最小化	最小化	MOP
TSALBP-1/2/3	最小化	最小化	最小化	MOP

TSALBP的类型。后缀1, 2和3分别指的是最小化 m , c 和 A 。如果问题本身是可行解之一，则类型为F；如果它是一个单目标优化问题，则类型为OP；如果它是一个多目标优化问题，则类型为MOP

优化模型

- 用数学公式表达TSALBP的1/3变体:

$$\min f^0(x) = m = \sum_{k \in K} \max_{j=1, \dots, n} x_{jk} \quad (1)$$

$$f^1(x) = A = \max_{k=1, \dots, UB_m} \sum_{j=1}^n a_j x_{jk} \quad (2)$$

$$\text{s.t. } \sum_{k=E_j}^{L_j} x_{jk} = 1, j = 1, 2, \dots, n \quad (3)$$

$$\sum_{k=1}^{UB_m} \max_{j=1, 2, \dots, n} x_{jk} \leq m \quad (4)$$

$$\sum_{j=1}^n t_j x_{jk} \leq c, k = 1, 2, \dots, UB_m \quad (5)$$

$$\sum_{j=1}^n a_j x_{jk} \leq A, k = 1, 2, \dots, UB_m \quad (6)$$

$$\sum_{k=E_i}^{L_i} k x_{ik} \leq \sum_{k=E_j}^{L_j} k x_{jk}, j = 1, 2, \dots, n; \forall i \in P_j \quad (7)$$

$$x_{jk} \in \{0, 1\}, j = 1, 2, \dots, n; k = 1, 2, \dots, UB_m \quad (8)$$

其中:

- n 是任务数,
- x_{jk} 是决策变量, 如果任务 j 分配给站 k 则取值1, 否则取0,
- a_j 是任务 j 的区域信息,
- UB_m 是站数 m 的上限,
- E_j 是任务 j 可能被分配到的最早站点,
- L_j 是任务 j 可能被分配到的最晚站点,
- UB_m 是站数的上限。在这个例子中, 它等于任务的数量, 从而
- 方程式(3)的约束将每个任务的分配限制在一个站点, (4)将决策变量限制为站点总数, (5), (6)与时间和区域上限有关, (7)表示任务中的优先关系, (8)表示变量 x_{jk} 的二元性质。

1. 贪婪随机自适应搜索算法

- 第一步：随机贪婪解的生成

- (1) 计算每个任务可以被分配到的最早和最晚工作站；
- (2) 依据启发值按顺序分配任务到任务站，并在每次分配后更新待分配任务的启发值；
- (3) 随机选择进行工作面积或工作站数量的局部搜索；
- (4) 将输出的结果计入最终解集中；
- (5) 重复进行 (2) (3) (4) 步，直到循环次数达到规定上限；
- (6) 计算解集的近似Pareto前沿。

- 第二步：多目标(对A和m分别使用两种不同的)局部搜索：

目标A的局部搜索方法：

- (1) 找到当前面积最大的工作站；
- (2) 将分配到该工作站的某个任务分配到其他可能的工作站；
- (3) 如果该改动可以减少最大面积，则保留该改动；
- (4) 重复第 (2) (3) 步，直至找到局部最优解

目标m的局部搜索方法：

- (1) 找到当前面积最小的工作站；
- (2) 将该工作站的所有任务
分配到其他可能的工作站；
- (3) 找到分配到其他站的最佳方案。

2. 多蚁群系统(Multiple Ant Colony System, MACS)

- (1) 计算每个任务可以被分配到的最早和最晚工作站;
- (2) 计算初始信息素矩阵;
- (3) 基于信息素矩阵的分布随机产生解决方案;
- (4) 根据解决方案的有效程度, 更新信息素矩阵或者更新近似Pareto集;
- (5) 重复进行 (3) (4) 步, 直到循环次数达到规定上限, 然后输出最终的近似Pareto集。

➤ 在每一步的所有候选任务中进行选择的转移规则如下:

$$j = \begin{cases} \arg \max_{j \in \Omega} (\tau_{kj} \cdot [\eta_j^0]^{\lambda\beta} \cdot [\eta_j^1]^{(1-\lambda)\beta}), & q \leq q_0, \\ \hat{i}, & \text{其他} \end{cases}$$

其中, \hat{i} 是通过以下概率分布选择的节点:

$$p(j) = \begin{cases} \frac{\tau_{kj} \cdot [\eta_j^0]^{\lambda\beta} \cdot [\eta_j^1]^{(1-\lambda)\beta}}{\sum_{u \in \Omega} \tau_{ku} \cdot [\eta_u^0]^{\lambda\beta} \cdot [\eta_u^1]^{(1-\lambda)\beta}}, & j \in \Omega \\ 0, & \text{其他} \end{cases}$$

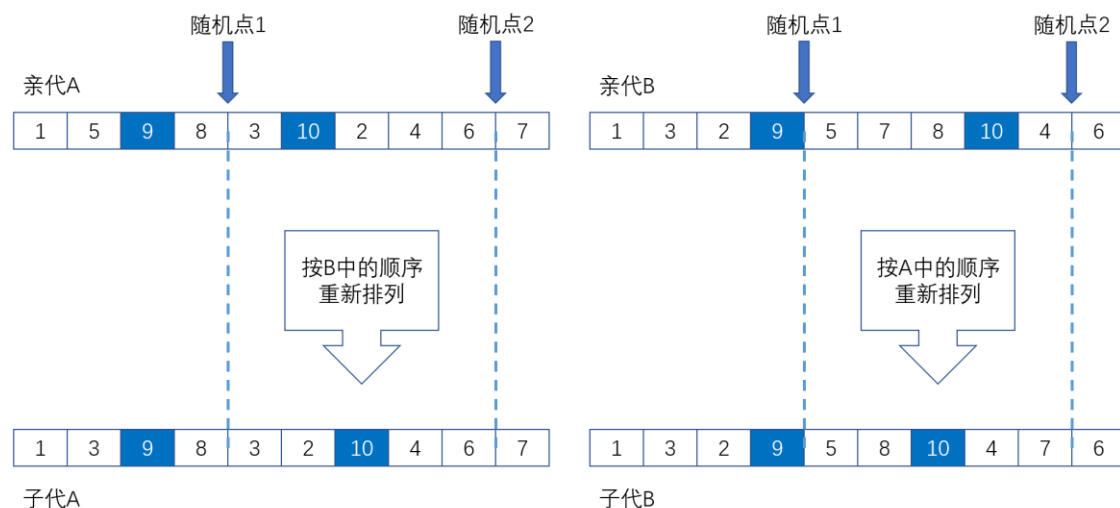
3. 改进遗传算法

- (1) 计算每个任务可以被分配到的最早和最晚工作站;
- (2) 按随机贪婪方式建立初始种群;
- (3) 按概率判定是否进行杂交, 然后随机选择两个亲代进行杂交;
- (4) 按概率判定是否进行变异, 然后随机选择一个亲代进行突变;
- (5) 比较子代和亲代优化效果函数的结果, 如果子代的结果比亲代中的最大值小, 则用子代替换该亲代;
- (6) 重复进行 (3) (4) (5) 步, 直到循环次数达到规定上限;
- (7) 以最终的种群作为输出结果, 并计算其近似Pareto前沿。

1	3	2	9	5	7	8	10	4	6
---	---	---	---	---	---	---	----	---	---

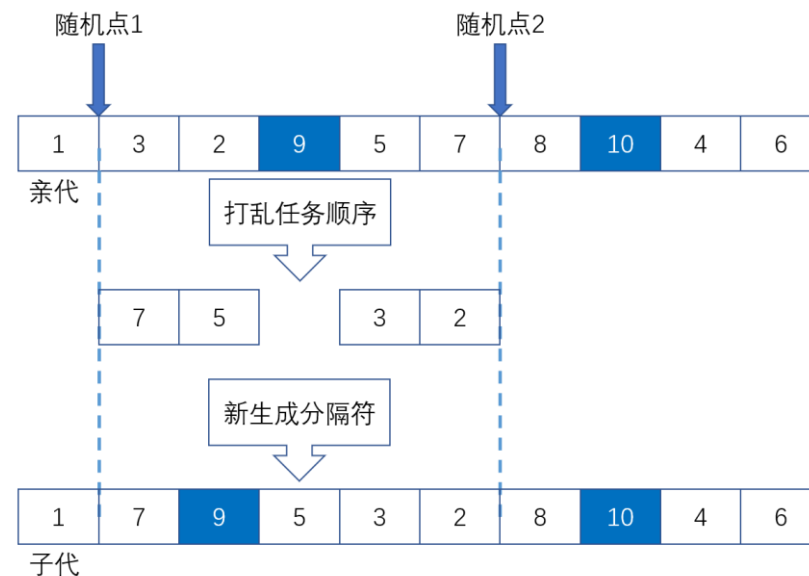
表示方案: 分隔符是不代表任何特定任务的虚拟基因, 它们被插入到代表任务的基因列表中。通过这种方式, 定义了分配给特定工作站的任务组。分隔符的最大可能数量是 $n-1$ (n 是任务的数量), 因为它对应于具有 n 个站的装配线配置, 每个站都相当于一个任务。任务使用 $\{1, \dots, n\}$ 中的数字进行编码, 而分隔符采用 $\{n+1, \dots, 2 \cdot n-1\}$ 。因此, 基因型又是一种基于顺序的表示。下图显示了新编码方案的示例, 代表三个站点的基因型, 分别具有3个、3个和2个任务。染色的基因被作为分隔符。

改进遗传算法 - 算子

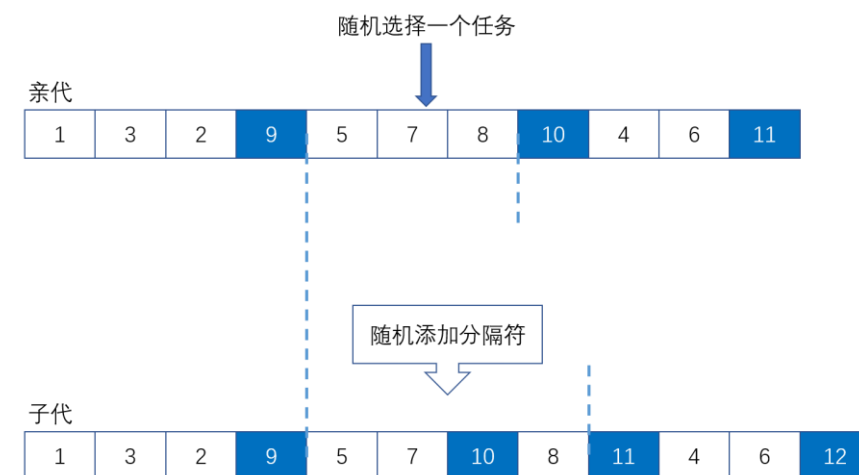


- 交叉算子:

- ✓ 选择两个随机切割点
- ✓ 对于第一个后代, 随机点之外的基因直接从第一个父母那里复制
- ✓ 两个切割点内的基因被复制, 但按照它们在第二个父代中出现的顺序进行复制。第二个后代遵循相同的机制, 但其父母相反。



- 变异算子: 扰乱突变(上)与分隔突变(下)



两个评价指标:

超体积比 (Hypervolume Ratio, HVR) : 假设存在一个 m 维的多目标优化问题, 其中目标函数

$$f(x) = (f_1(x), f_2(x), \dots, f_m(x)), \quad x \in X.$$

设 $R = (r_1, r_2, \dots, r_m)$ 是一个参考点可以是最大值或最小值点, 其中, r_i 表示第 i 个目标函数的参考值。则定义参考点 R 与一个 Pareto 最优解集合 S 所构成的超体积:

$$V(S, R)$$

$$= \int_S \max\{f_1(x) - r_1, 0\} \times \dots \times \max\{f_m(x) - r_m, 0\} dx$$

- 即Pareto最优解集合的超体积与全局最优解集合的超体积之比, 取值范围 $[0, 1]$
- 越接近1表示算法的性能越好。此时获得的近似解集合与全局最优之间的距离越近

集合覆盖率 (Coverage, Cov) 用于衡量一个多目标优化问题的 Pareto 最优解集合与参考点之间的距离和多样性:

假设存在一个 m 维的多目标优化问题, 其中目标函数为 $f(x) = (f_1(x), f_2(x), \dots, f_m(x))$, $x \in X$, X 是定义域。设 S 是一个大小为 n 的 Pareto 最优解集合。令目标函数的集合

$$F = \{f_1, f_2, \dots, f_m\}$$

对于任意一个目标函数 $f \in F$, 定义其覆盖率为 S 对于 f 的覆盖率 $Cov(S, f)$ 为:

$$Cov(S, f) = |\{x \in X \mid f(x) \leq f(S^*)\}| / |X|$$

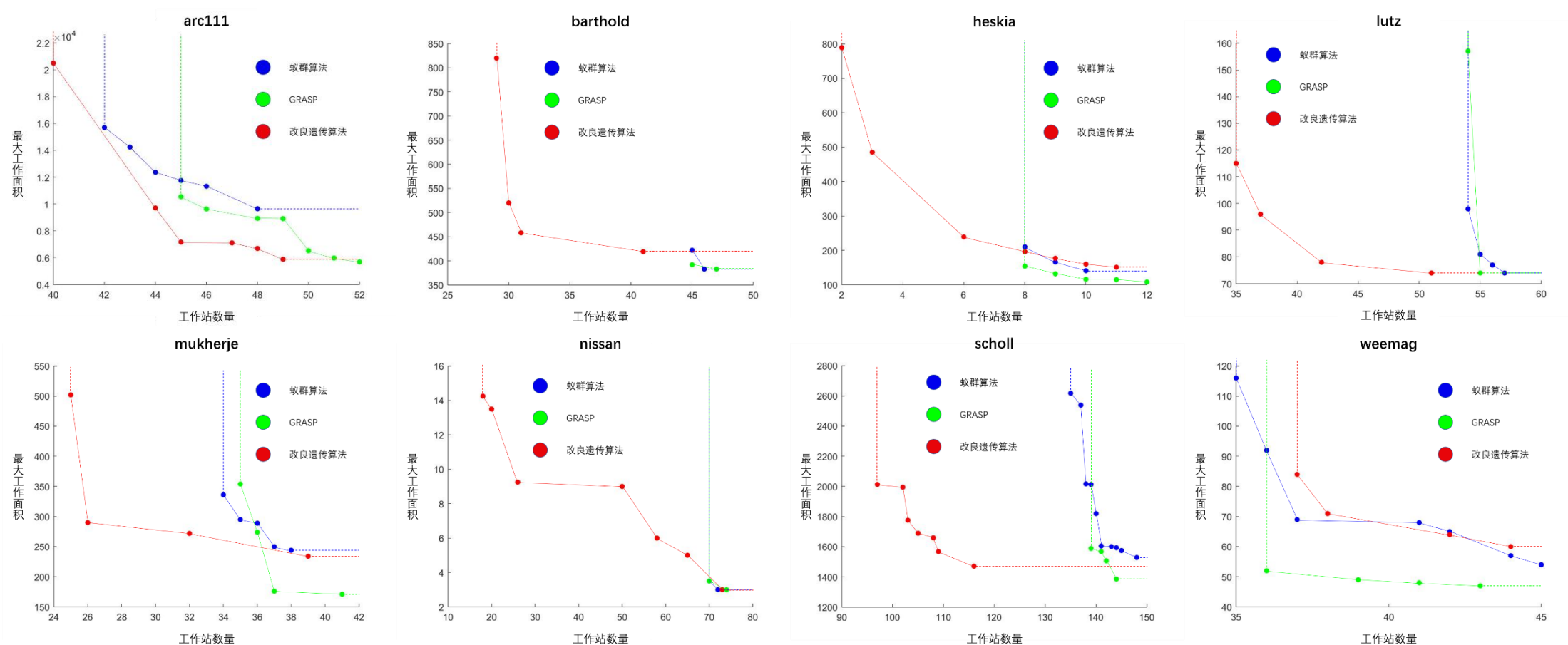
其中, S^* 是全局最优解集合, $f(S^*)$ 表示 f 在全局最优解集合上的最小值。集合覆盖率 $Cov(S, F)$ 定义为所有目标函数的覆盖率的平均值, 即:

$$Cov(S, F) = (Cov(S, f_1) + Cov(S, f_2) + \dots + Cov(S, f_m)) / m$$

- 集合覆盖率指标的取值范围在 $[0, 1]$ 之间
- Cov 值越接近 1 表示算法的性能越好, 此时 Pareto 最优解集合对于目标函数的多样性越好。

结果展示

- 使用三种算法解决八个问题实例所得出的近似Pareto前沿如下图所示。
- 近似Pareto前沿越靠近原点，代表该算法的优化效果越好！



结果展示 – 超体积比

算例名称	蚁群算法	GRASP	改良遗传算法
arc111	0.7646	0.7997	0.9482
barthold	0.2944	0.2968	0.9798
heskia	0.5803	0.6088	0.9657
lutz	0.0895	0.0653	1.0000
mukherje	0.5298	0.5818	0.9072
nissan	0.2511	0.2490	0.9989
scholl	0.2959	0.3669	0.9704
weemag	0.7837	0.9873	0.6885

使用超体积比评估三种算法的优化效果：

- 在前七个算例中，改良遗传算法拥有明显优势；
- 在最后一个算例weemag中，GRASP拥有明显优势。

结果展示 - 集合覆盖率

C(A,B)	蚁群算法, GRASP	蚁群算法, 遗传算法	GRASP, 蚁群算法	GRASP, 遗传算法	遗传算法, 蚁群算法	遗传算法, GRASP
arc111	0.280	0.460	0.962	0.770	0.990	0.550
barthold	0.990	0.902	1.000	0.902	0.999	0.960
heskia	0.550	0.870	1.000	0.900	0.996	0.470
lutz	0.840	0.440	0.994	0.460	1.000	1.000
mukherje	0.690	0.740	0.988	0.720	0.994	0.690
nissan	1.000	0.530	0.994	0.520	0.999	1.000
scholl	0.490	0.470	0.984	0.450	1.000	0.580
weemag	0.470	0.970	1.000	0.998	0.991	0.480

- 1 is better!

Homework 10

1. 课后作业题1:请将GA算法的运算结果与上周SA算法的结果对比, 看是否能得到相同的最优解? 若不同, 请解释原因。
2. 课后作业题5(只需用遗传算法)。
3. 课后作业题6(找A,r,K三个参数最优值)。
4. 整理求解TSP的GA算法和ACO算法, 并与SA算法比较, 求解10、30、50城市TSP问题的最优解。你还有没有其他仿生/启发式算法结果? 请尝试对比。