智能优化算法

智能优化算法又称现代启发式算法，是一种具有全局优化性能、通用性强且适合于并行处理的算法。这种算法一般具有严密的理论依据，而不是单纯凭借专家经验，理论上可以在一定的时间内找到最优解或近似最优解。常用的智能优化算法有：遗传算法 、模拟退火算法、禁忌搜索算法、粒子群算法、蚁群算法。

1.粒子群算法

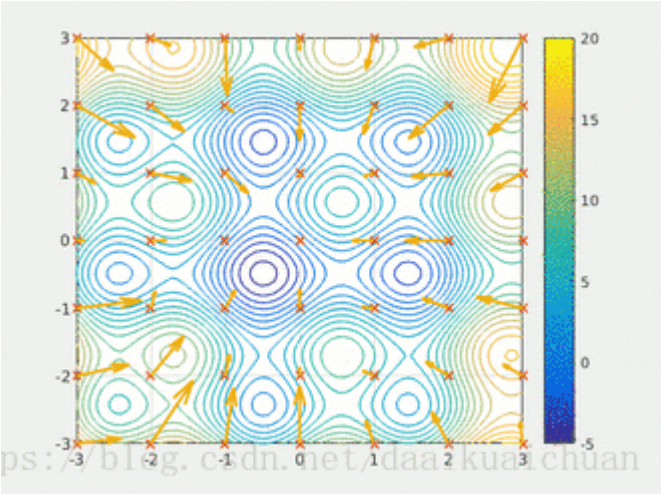
最优化算法之粒子群算法（PSO）_pso算法-CSDN博客

1.1 概念

粒子群优化算法(PSO：Particle swarm optimization) 是一种进化计算技术（evolutionary computation）。源于对鸟群捕食的行为研究。粒子群优化算法的基本思想：是通过群体中个体之间的协作和信息共享来寻找最优解．　　PSO的优势：在于简单容易实现并且没有许多参数的调节。目前已被广泛应用于函数优化、神经网络训练、模糊系统控制以及其他遗传算法的应用领域。

1.2 基本思想

粒子群算法通过设计一种无质量的粒子来模拟鸟群中的鸟，粒子仅具有两个属性：速度和位置，速度代表移动的快慢，位置代表移动的方向。每个粒子在搜索空间中单独的搜寻最优解，并将其记为当前个体极值，并将个体极值与整个粒子群里的其他粒子共享，找到最优的那个个体极值作为整个粒子群的当前全局最优解，粒子群中的所有粒子根据自己找到的当前个体极值和整个粒子群共享的当前全局最优解来调整自己的速度和位置。



1.3 更新规则

PSO初始化为一群随机粒子(随机解)。然后通过迭代找到最优解。在每一次的迭代中，粒子通过跟踪两个"极值"(pbest，gbest)来更新自己。在找到这两个最优值后，粒子通过下面的公式来更新自己的速度和位置。

公式（1）：

$$v_i = v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times rand() \times (gbest_i - x_i)$$

公式（2）：

$$x_i = x_i + v_i$$

在公式（1）、（2）中， i=1, 2, . . . , N，N是此群中粒子的总数。

v_i：是粒子的速度

$rand()$：介于(0, 1)之间的随机数

x_i：粒子的当前位置

c_1和c_2：是学习因子，通常$c_1 = c_2 = 2$

v_i的最大值为V_{max}(大于0)，如果v_i大于V_{max}，则$v_i = V_{max}$

公式（1）、（2）为*PSO*的标准形式。

公式(1)的第一部分称为【记忆项】，表示上次速度大小和方向的影响；公式(1)的第二部分称为【自身认知项】，是从当前点指向粒子自身最好点的一个矢量，表示粒子的动作来源于自己经验的部分；公式(1)的第三部分称为【群体认知项】，是一个从当前点指向种群最好点的矢量，反映了粒子间的协同合作和知识共享。粒子就是通过自己的经验和同伴中最好的经验来决定下一步的运动。以上面两个公式为基础，形成了PSO的标准形式。

公式（3）：

$$v_i = \omega \times v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times rand() \times (gbest_i - x_i)$$

ω：叫做惯性因子，其值为非负。

其值较大，全局寻优能力强，局部寻优能力弱；

其值较小，全局寻优能力弱，局部寻优能力强。

动态ω能获得比固定值更好的寻优结果。动态ω可在PSO搜索过程中线性变化，也可以根据PSO性能的某个测度函数动态改变。

目前采用较多的是线性递减权值(Linearly Decreasing Weight, LDW)策略。

$$\omega^{(t)} = (\omega_{ini} - \omega_{end})(G_k - g) / G_k + \omega_{end}$$

G_k：最大迭代次数

ω_{ini}：初始惯性权值

ω_{end}：迭代至最大进化代数时的惯性权值

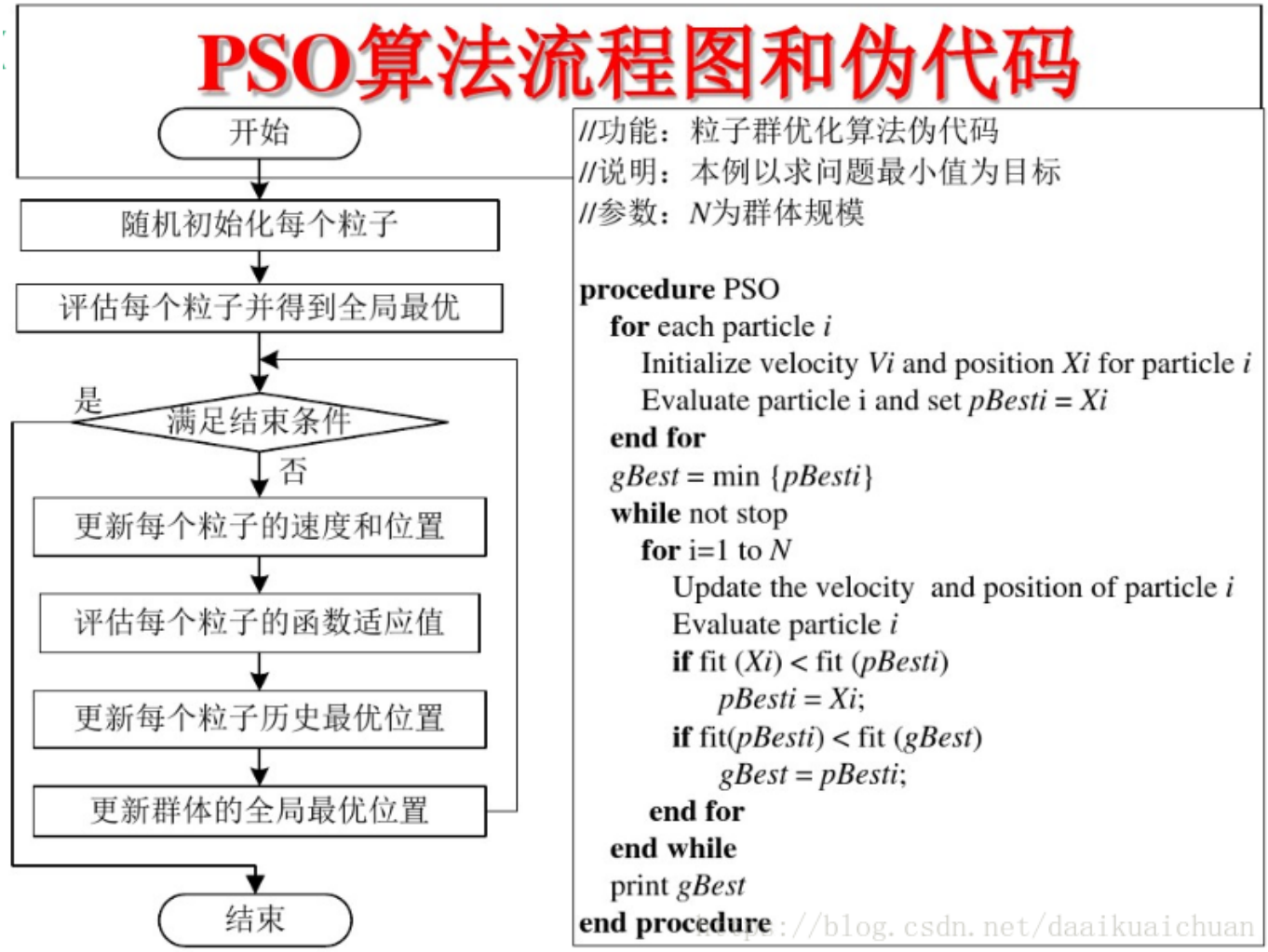
典型权值：

ω_{ini}=0.9，ω_{end}=0.4

ω的引入，使用PSO算法性能有了很大的提高，针对不同的搜索问题，可以调整全局和局部搜索能力，也使PSO算法有成功地应用于很多实际问题。

公式(2)和 公式(3)被视为标准PSO算法。

1.4 流程图和伪代码


PSO算法流程图和伪代码
开始
随机初始化每个粒子
评估每个粒子并得到全局最优
是
满足结束条件
否
更新每个粒子的速度和位置
评估每个粒子的函数适应值
更新每个粒子历史最优位置
更新群体的全局最优位置
结束
//功能：粒子群优化算法伪代码
//说明：本例以求问题最小值为目标
//参数：N为群体规模
procedure PSO
for each particle i
Initialize velocity Vi and position Xi for particle i
Evaluate particle i and set pBesti = Xi
end for
gBest = min {pBesti}
while not stop
for i=1 to N
Update the velocity and position of particle i
Evaluate particle i
if fit (Xi) < fit (pBesti)
pBesti = Xi;
if fit(pBesti) < fit (gBest)
gBest = pBesti;
end for
end while
print gBest
end procedure
https://blog.csdn.net/daaikuaichuan

1.5 简单的例子

符号	含义
n	粒子个数
c_1	粒子的个体学习因子，也称为个体加速因子
c_2	粒子的社会学习因子，也称为社会加速因子
w	速度的惯性权重
v_i^d	第d次迭代时，第i个粒子的速度
x_i^d	第d次迭代时，第i个粒子所在的位置
$f(x)$	在位置x时的适应度值(一般取目标函数值)
$pbest_i^d$	到第d次迭代为止，第i个粒子经过的最好的位置
$gbest^d$	到第d次迭代为止，所有粒子经过的最好的位置

例1 求f(x)

粒子群算法讲解以及MATLAB实现_粒子群matlab代码-CSDN博客

$\mathrm{f} = 21.5 + x_1 * \sin(4\pi * x1) + x_2 * \sin(20\pi * x_2)$ %目标函数

$\mathrm{x}_1^2 + x_2^2 < 20$ %条件

例2 求f(x)最大值

$\mathrm{f}(x_1, x_2) = 2\sin(2x_1^2) + 3\cos(x_2^2) + 2x_1 + \log(x_2^2)$ %目标函数

$-5 < x_1, x_2 < 5$ %条件

```
%%  粒子群优化算法求解
%   说明：
%       粒子群优化算法
%   输入：
%       目标函数和约束条件
%       种群参数
%   输出：
%       最优解
%% 初始化种群
f = @(x1, x2) 2*sin(2*x1.^2)+3*cos(x2.^2)+2*x1+log(x2.^2);
 % 函数表达式% 求这个函数的最大值
x = linspace(-5, 5, 100);
y = linspace(-5, 5, 100);
figure
[X, Y] = meshgrid(x, y);
Z = f(X, Y);
mesh(X, Y, Z);                  % 画出函数整体图

N = 1000;                        % 初始种群个数
d = 1;                          % 空间维数
ger = 100;                      % 最大迭代次数
limit = [-5, 5];                % 设置位置参数限制
vlimit = [-1, 1];               % 设置速度限制
w = 0.8;                        % 惯性权重
c1 = 0.5;                       % 自我学习因子
c2 = 0.5;                       % 群体学习因子
for i = 1:d
    x1 = limit(i, 1) + (limit(i, 2) - limit(i, 1)) * rand(N, d);
%初始种群的位置
    x2 = limit(i, 1) + (limit(i, 2) - limit(i, 1)) * rand(N, d);
%初始种群的位置
end
v1 = rand(N, d);                % 初始种群的速度
v2 = rand(N, d);                % 初始种群的速度
xm1 = x1;                       % 每个个体的历史最佳位置
xm2 = x2;                       % 每个个体的历史最佳位置
ym1 = zeros(1, d);              % 种群的历史最佳位置
ym2 = zeros(1, d);              % 种群的历史最佳位置
fxm = zeros(N, 1);              % 每个个体的历史最佳适应度
fym = -inf;                     % 种群历史最佳适应度
hold on
scatter3(xm1,xm2,f(xm1,xm2),'ro');title('初始状态图');
%% 群体更新
iter = 1;
record = zeros(ger, 1);         % 记录器
while iter <= ger
     fx = f(x1,x2) ; % 个体当前适应度
     for i = 1:N
            if fxm(i) < fx(i)
                fxm(i) = fx(i);      % 更新个体历史最佳适应度
```

```
                xm1(i,:) = x1(i,:);    % 更新个体历史最佳位置
                xm2(i,:) = x2(i,:);    % 更新个体历史最佳位置
            end
        end
if fym < max(fxm)
        [fym, nmax] = max(fxm);    % 更新群体历史最佳适应度
        ym1 = xm1(nmax, :);       % 更新群体历史最佳位置
        ym2 = xm2(nmax, :);       % 更新群体历史最佳位置
 end
    v1 = v1 * w + c1 * rand * (xm1 - x1) + c2 * rand * (repmat(ym1, N, 1) - x1);% 速度
更新
    v2 = v2 * w + c1 * rand * (xm2 - x2) + c2 * rand * (repmat(ym2, N, 1) - x2);% 速度
更新
    % 边界速度处理
    v1(v1 > vlimit(2)) = vlimit(2);
    v1(v1 < vlimit(1)) = vlimit(1);
    v2(v2 > vlimit(2)) = vlimit(2);
    v2(v2 < vlimit(1)) = vlimit(1);
    x1 = x1 + v1;% 位置更新
    x2 = x2 + v2;% 位置更新
    % 边界位置处理
    x1(x1 > limit(2)) = limit(2);
    x1(x1 < limit(1)) = limit(1);
    x2(x2 > limit(2)) = limit(2);
    x2(x2 < limit(1)) = limit(1);
    record(iter) = fym;%最大值记录
    mesh(X, Y, Z); % 画出函数整体图
    hold on
    scatter3(x1,x2,f(x1,x2), 'ro');title('状态位置变化')
    hold off
    pause(0.1)
    iter = iter+1;
end
%%
figure;plot(record);title('收敛过程')
x0 = 0 : 0.01 : 20;
figure;
mesh(X, Y, Z);
hold on
scatter3(x1,x2,f(x1,x2), 'ro');title('最终状态位置')
disp(['最大值：',num2str(fym)]);
disp(['变量取值：',num2str(ym1)]);
disp(['变量取值：',num2str(ym1)]);
```

2.遗传算法

遗传算法(Genetic Algorithm)_遗传算法csdn示意图-CSDN博客

2.1 定义

遗传算法（Genetic Algorithm, GA）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。

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；采用概率化的寻优方法，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。

遗传算法以一种群体中的所有个体为对象，并利用随机化技术指导对一个被编码的参数空间进行高效搜索。其中，选择、交叉和变异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。

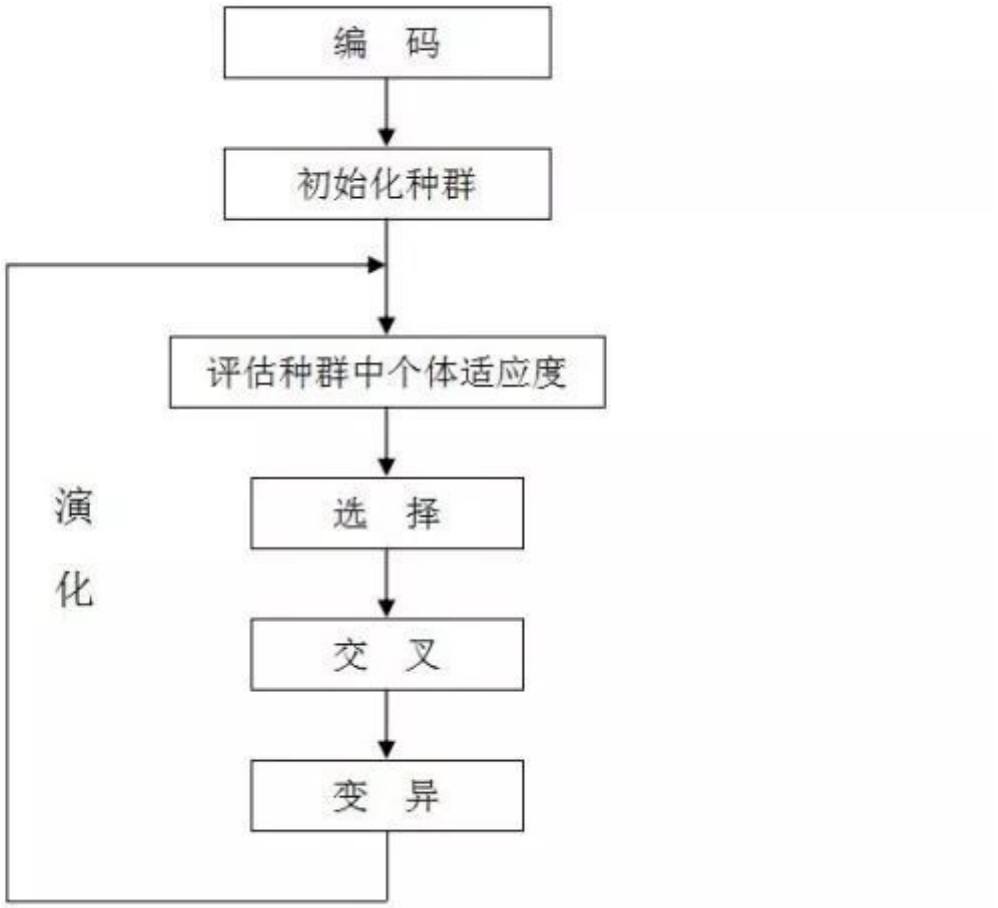
2.1.2 遗传算法的执行过程(参照百度百科)

遗传算法是从代表问题可能潜在的解集的一个种群（population）开始的，**而一个种群则由经过基因（gene）编码的一定数目的个体(individual)组成。每个个体实际上是染色体(chromosome)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（generation）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（fitness）大小选择（selection）个体，并借助于自然遗传学的遗传算子（genetic operators）进行组合交叉（crossover）和变异（mutation），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（decoding），可以作为问题近似最优解。**



2.2 生物学术语

基因型(genotype)：性状染色体的内部表现。

表现型(phenotype)：染色体决定的性状的外部表现，或者说，根据基因型形成的个体的外部表现。

进化(evolution)：种群逐渐适应生存环境，品质不断得到改良。生物的进化是以种群的形式进行的。

适应度(fitness)：度量某个物种对于生存环境的适应程度。

选择(selection)：以一定的概率从种群中选择若干个个体。一般，选择过程是一种基于适应度的优胜劣汰的过程。

复制(reproduction)：细胞分裂时，遗传物质DNA通过复制而转移到新产生的细胞中，新细胞就继承了旧细胞的基因。

交叉(crossover)：两个染色体的某一相同位置处DNA被切断，前后两串分别交叉组合形成两个新的染色体。也称基因重组或杂交。

变异(mutation)：复制时可能（很小的概率）产生某些复制差错，变异产生新的染色体，表现出新的性状。

编码(coding)：DNA中遗传信息在一个长链上按一定的模式排列。遗传编码可看作从表现型到基因型的映射。

解码(decoding)：基因型到表现型的映射。

个体(individual):指染色体带有特征的实体。

种群(population:个体的集合，该集合内个体数称为种群。

2.3 步骤

1. 首先寻找一种对问题潜在解进行“数字化”编码的方案。（建立表现型和基因型的映射关系）
2. 随机初始化一个种群，种群里面的个体就是这些数字化的编码。
3. 接下来，通过适当的解码过程之后。
4. 用适应性函数对每一个基因个体作一次适应度评估。
5. 用选择函数按照某种规定择优选择。
6. 让个体基因变异。
7. 然后产生子代。

由交叉和变异产生新一代种群，返回步骤4，直到最优解产生

2.3.1 编码

编码是应用遗传算法时要解决的首要问题，也是设计遗传算法时的一个关键步骤。编码方法影响到交叉算子、变异算子等遗传算子的运算方法，很大程度上决定了遗传进化的效率。

迄今为止人们已经提出了许多种不同的编码方法。总的来说，这些编码方法可以分为三大类：**二进制编码法、浮点编码法、符号编码法。**下面分别进行介绍。

(1) 二进制编码法

就像人类的基因有AGCT 4种碱基序列一样。不过在这里我们只用了0和1两种碱基,然后将它们串成一条链形成染色体。**一个位能表示出2种状态的信息量，因此足够长的二进制染色体便能表示所有的特征。这便是二进制编码**，如下：

1110001010111

它由二进制符号0和1所组成的二值符号集。它有以下一些优点：

1. 编码、解码操作简单易行
2. 交叉、变异等遗传操作便于实现
3. 合最小字符集编码原则
4. 利用模式定理对算法进行理论分析。

二进制编码的缺点是：**对于一些连续函数的优化问题，由于其随机性使得其局部搜索能力较差**，如对于一些高精度的问题，当解迫近于最优解后，由于其变异后表现型变化很大，不连续，所以会远离最优解，达不到稳定。

（2）浮点编码法

二进制编码虽然简单直观，一目了然。但是存在着连续函数离散化时的映射误差。个体长度较短时，可能达不到精度要求，而个体编码长度较长时，虽然能提高精度，但增加了解码的难度，使遗传算法的搜索空间急剧扩大。

所谓浮点法，是指个体的每个基因值用某一范围内的一个浮点数来表示。在浮点数编码方法中，必须保证基因值在给定的区间限制范围内，遗传算法中所使用的交叉、变异等遗传算子也必须保证其运算结果所产生的新个体的基因值也在这个区间限制范围内。如下所示：

1.2-3.2-5.3-7.2-1.4-9.7

浮点数编码方法有下面几个优点：

1. 适用于在遗传算法中表示范围较大的数。
2. 适用于精度要求较高的遗传算法。
3. 便于较大空间的遗传搜索。
4. 改善了遗传算法的计算复杂性，提高了运算交率。
5. 便于遗传算法与经典优化方法的混合使用。
6. 便于设计针对问题的专门知识的知识型遗传算子。
7. 便于处理复杂的决策变量约束条件。

（3）符号编码法

符号编码法是指个体染色体编码串中的基因值取自一个无数值含义、而只有代码含义的符号集如｛A,B,C…｝。

符号编码的主要优点是：

1. 符合有意义积术块编码原则。
2. 便于在遗传算法中利用所求解问题的专门知识。
3. 便于遗传算法与相关近似算法之间的混合使用。

给出一个例子解释编码原理：

假定位置坐标可以比喻为区间[-1, 2]的某一个x坐标（有了x坐标，再通过函数表达式可以算出函数值 <==> 得到染色体编码，解码得到位置坐标，在查询位置坐标算出高度也就是函数值）

在这里假如我们要求解精确到六位小数，由于区间长度为2 - (-1) = 3 ,为了保证精度要求，至少把区间[-1,2]分为3 × 10^6等份。又因为

2^21 = 2097152 < 3*10^6 < 2^22 = 4194304

所以编码的二进制串至少需要22位。

把一个二进制串(b0,b1,....bn)转化为区间里面对应的实数值可以通过下面两个步骤：

1) 将一个二进制串代表的二进制数转化为10进制数：

$$(b_0 \cdots b_{20} b_{21})_2 = (\sum_{i=0}^{21} b_i \cdot 2^i)_{10} = x'$$

2) 对应区间内的实数：

$$x = -1 + x' \frac{(2-(-1))}{2^{22}-1}$$

例如一个二进制串(1000101110110101000111)通过上面换算以后，表示实数值0.637197。

好了，上面的编码方式只是举个例子让大家更好理解而已，**编码的方式千奇百怪，层出不穷，每个问题可能采用的编码方式都不一样。**在这一点上大家要注意。

2.3.2 评价个体的适应度

适应度函数也称评价函数，是根据目标函数确定的用于区分群体中个体好坏的标准。适应度函数总是非负的，而目标函数可能有正有负，故需要在目标函数与适应度函数之间进行变换。

评价个体适应度的一般过程为：

1. 对个体编码串进行解码处理后，可得到个体的表现型。

2. 由个体的表现型可计算出对应个体的目标函数值。

3. 根据最优化问题的类型，由目标函数值按一定的转换规则求出个体的适应度。

2.3.3选择

遗传算法中的选择操作就是用来确定如何从父代群体中按某种方法选取那些个体，以便遗传到下一代群体。选择操作用来确定重组或交叉个体，以及被选个体将产生多少个子代个体。

绍几种常用的选择算子：

1. 轮盘赌选择（Roulette Wheel Selection）：是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（Stochastic Tournament）：每次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。

3. 最佳保留选择：首先按轮盘赌选择方法执行遗传算法的选择操作，然后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。

4. 无回放随机选择（也叫期望值选择Excepted Value Selection）：根据每个个体在下一代群体中的生存期望来进行随机选择运算。方法如下:

（1） 计算群体中每个个体在下一代群体中的生存期望数目N。

（2） 若某一个体被选中参与交叉运算，则它在下一代中的生存期望数目减去 0.5，若某一个体未被选中参与交叉运算，则它在下一代中的生存期望数目减去1.0。

（3） 随着选择过程的进行，若某一个体的生存期望数目小于0时，则该个体就不再有机会被选中。

5. 确定式选择：按照一种确定的方式来进行选择操作。具体操作过程如下：

（1） 计算群体中各个个体在下一代群体中的期望生存数目N。

（2） 用N的整数部分确定各个对应个体在下一代群体中的生存数目。

（3） 用N的小数部分对个体进行降序排列，顺序取前M个个体加入到下一代群体中。至此可完全确定出下一代群体中M个个体。

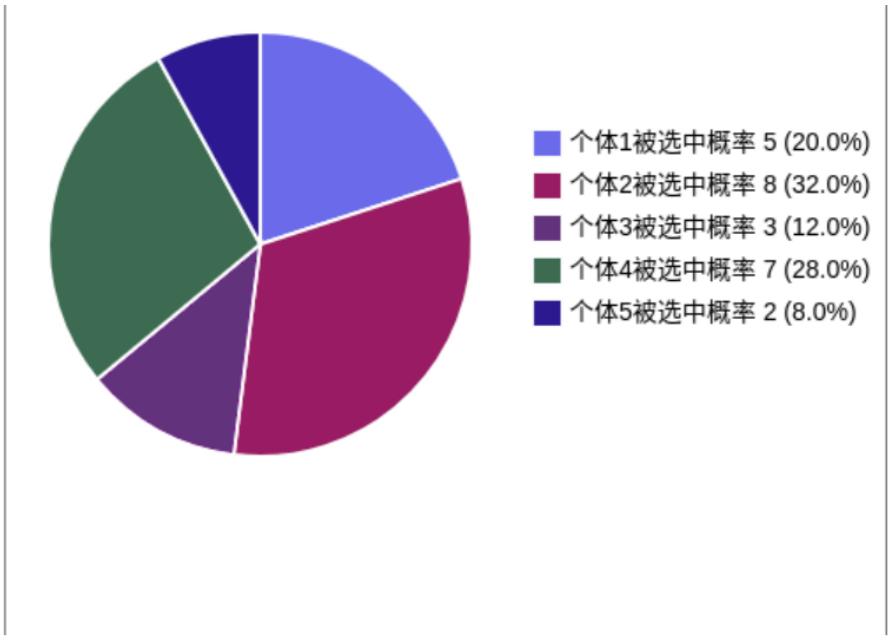
6. 无回放余数随机选择：可确保适应度比平均适应度大的一些个体能够被遗传到下一代群体中，因而选择误差比较小。

7. 均匀排序：对群体中的所有个体按期适应度大小进行排序，基于这个排序来分配各个个体被选中的概率。

8. 最佳保存策略：当前群体中适应度最高的个体不参与交叉运算和变异运算，而是用它来代替掉本代群体中经过交叉、变异等操作后所产生的适应度最低的个体。

9. 随机联赛选择：每次选取几个个体中适应度最高的一个个体遗传到下一代群体中。

10. 排挤选择：新生成的子代将代替或排挤相似的旧父代个体，提高群体的多样性。



当指针在这个转盘上转动，停止下来时指向的个体就是天选之人啦。可以看出，适应性越高的个体被选中的概率就越大。

2.3.4 遗传--染色体交叉(crossover)

遗传算法的交叉操作，是指对两个相互配对的染色体按某种方式相互交换其部分基因，从而形成两个新的个体。

适用于二进制编码个体或浮点数编码个体的交叉算子：

1. 单点交叉（One-point Crossover）：指在个体编码串中只随机设置一个交叉点，然后再该点相互交换两个配对个体的部分染色体。

2. 两点交叉与多点交叉：

(1) 两点交叉（Two-point Crossover）：在个体编码串中随机设置了两个交叉点，然后再进行部分基因交换。

(2) 多点交叉（Multi-point Crossover）

3. 均匀交叉（也称一致交叉，Uniform Crossover）：两个配对个体的每个基因座上的基因都以相同的交叉概率进行交换，从而形成两个新个体。

4. 算术交叉（Arithmetic Crossover）：由两个个体的线性组合而产生出两个新的个体。该操作对象一般是由浮点数编码表示的个体。

二进制编码的染色体交叉过程非常类似高中生物中所讲的同源染色体的联会过程—随机把其中几个位于同

2.3.5 变异--基因突变(Mutation)

遗传算法中的变异运算，是指将个体染色体编码串中的某些基因座上的基因值用该基因座上的其它等位基

例如下面这串二进制编码：

```
101101001011001
```

经过基因突变后，可能变成以下这串新的编码：

```
001101011011001
```

以下变异算子适用于二进制编码和浮点数编码的个体：

1. 基本位变异（Simple Mutation）：对个体编码串中以变异概率、随机指定的某一位或某几位仅因座

2. 均匀变异（Uniform Mutation）：分别用符合某一范围内均匀分布的随机数，以某一较小的概率来替

3. 边界变异（Boundary Mutation）：随机的取基因座上的两个对应边界基因值之一去替代原有基因值

4. 非均匀变异：对原有的基因值做一随机扰动，以扰动后的结果作为变异后的新基因值。对每个基因座

5. 高斯近似变异：进行变异操作时用符号均值为 P 的平均值，方差为P**2的正态分布的一个随机数来替

2.4 简单的例子

遗传算法原理及其matlab程序实现_遗传算法实例matlab-CSDN博客

目标函数： $\max \mathrm{f}(x, y)=\sin x+\cos y+0.1 x+0.1 y$

条件：$-10<x, y<10$

```
clc;
clear;
%% 基础参数
N = 100;  %种群内个体数目
N_chrom = 2; %染色体节点数，也就是每个个体有多少条染色体，其实说白了就是看适应函数里有几个自变量。
iter = 1000; %迭代次数，也就是一共有多少代
mut = 0.2;  %突变概率
acr = 0.2; %交叉概率
best = 1;
chrom_range = [-10 -10;10 10];%每个节点的值的区间
chrom = zeros(N, N_chrom);%存放染色体的矩阵
fitness = zeros(N, 1);%存放染色体的适应度
fitness_ave = zeros(1, iter);%存放每一代的平均适应度
fitness_best = zeros(1, iter);%存放每一代的最优适应度
chrom_best = zeros(1, N_chrom+1);%存放当前代的最优染色体与适应度
%% 初始化，这只是用于生成第一代个体，并计算其适应度函数
chrom = Initialize(N, N_chrom, chrom_range); %初始化染色体
fitness = CalFitness(chrom, N, N_chrom); %计算适应度
chrom_best = FindBest(chrom, fitness, N_chrom); %寻找最优染色体
fitness_best(1) = chrom_best(end); %将当前最优存入矩阵当中
fitness_ave(1) = CalAveFitness(fitness); %将当前平均适应度存入矩阵当中
%% 用于生成以下其余各代，一共迭代多少步就一共有多少代
for t = 2:iter
    chrom = MutChrom(chrom, mut, N, N_chrom, chrom_range, t, iter); %变异
    chrom = AcrChrom(chrom, acr, N, N_chrom); %交叉
    fitness = CalFitness(chrom, N, N_chrom); %计算适应度
    chrom_best_temp = FindBest(chrom, fitness, N_chrom); %寻找最优染色体
    if chrom_best_temp(end)>chrom_best(end) %替换掉当前储存的最优
        chrom_best = chrom_best_temp;
    end
    %%替换掉最劣
    [chrom, fitness] = ReplaceWorse(chrom, chrom_best, fitness);
    fitness_best(t) = chrom_best(end); %将当前最优存入矩阵当中
    fitness_ave(t) = CalAveFitness(fitness); %将当前平均适应度存入矩阵当中
end
%% 作图
figure(1)
plot(1:iter, fitness_ave, 'r', 1:iter, fitness_best, 'b')
grid on
legend('平均适应度', '最优适应度')
e=PlotModel(chrom_best);
%% 输出结果
disp(['最优染色体为', num2str(chrom_best(1:end-1))])
disp(['最优适应度为', num2str(chrom_best(end))])

%种群初始化
function chrom_new = Initialize(N, N_chrom, chrom_range)
chrom_new = rand(N, N_chrom);%rand函数产生由在(0, 1)之间均匀分布的随机数组成的数组（m,n）
for i = 1:N_chrom %每一列乘上范围
```

```
    chrom_new(:, i) = chrom_new(:, i)*(chrom_range(2, i)-chrom_range(1,
i))+chrom_range(1, i);
end

%适应度计算
function fitness = CalFitness(chrom, N, N_chrom)
fitness = zeros(N, 1);
%开始计算适应度
for i = 1:N
    x = chrom(i, 1);
    y = chrom(i, 2);
    fitness(i) = sin(x)+cos(y)+0.1*x+0.1*y;%%该函数是定义的适应度函数，也可称为代价函数，
用于以后筛选个体的评价指标
end

%选择淘汰
function chrom_best = FindBest(chrom, fitness, N_chrom)
chrom_best = zeros(1, N_chrom+1);
[maxNum, maxCorr] = max(fitness);%因为所有个体对应的适应度大小都被存放在fitness矩阵中
chrom_best(1:N_chrom) =chrom(maxCorr, :);
chrom_best(end) = maxNum;

%变异处理
%% 用于对每代共100个个体进行变异处理
function chrom_new = MutChrom(chrom, mut, N, N_chrom, chrom_range, t, iter)
for i = 1:N %%N是个体总数，也就是每一代有多少头袋鼠
    for j = 1:N_chrom  %N_chrom是染色体节点数，就是有几条染色体
        mut_rand = rand; %随机生成一个数，代表自然里的基因突变，然后用改值来决定是否产生突
变。
        if mut_rand <=mut  %mut代表突变概率，即产生突变的阈值，如果小于0.2的基因突变概率阈
值才进行基因突变处理，否者不进行突变处理
            mut_pm = rand; %增加还是减少
            mut_num = rand*(1-t/iter)^2;
            if mut_pm<=0.5
                chrom(i, j)= chrom(i, j)*(1-mut_num);
            else
                chrom(i, j)= chrom(i, j)*(1+mut_num);
            end
            chrom(i, j) = IfOut(chrom(i, j), chrom_range(:, j)); %检验是否越界
        end
    end
end
chrom_new = chrom;%%把变异处理完后的结果存在新矩阵里

%变异是否超出自变量范围判断
function c_new = IfOut(c, range)
if c<range(1) || c>range(2)
    if abs(c-range(1))<abs(c-range(2))
        c_new = range(1);
    else
        c_new = range(2);
```

```
    end
else
    c_new = c;
end

%交叉处理
function chrom_new = AcrChrom(chrom, acr, N, N_chrom)
for i = 1:N
    acr_rand = rand;%生成一个代表该个体是否产生交叉的概率大小，用于判别是否进行交叉处理
    if acr_rand<acr %如果该个体的交叉概率值大于产生交叉处理的阈值，则对该个体的染色体（两条，因为此案例中有两个自变量）进行交叉处理
        acr_chrom = floor((N-1)*rand+1); %要交叉的染色体
        acr_node = floor((N_chrom-1)*rand+1); %要交叉的节点
        %交叉开始
        temp = chrom(i, acr_node);
        chrom(i, acr_node) = chrom(acr_chrom, acr_node);
        chrom(acr_chrom, acr_node) = temp;
    end
end
chrom_new = chrom;

%最差个体统计
function [chrom_new, fitness_new] = ReplaceWorse(chrom, chrom_best, fitness)

max_num = max(fitness);
min_num = min(fitness);
limit = (max_num-min_num)*0.2+min_num;

replace_corr = fitness<limit;

replace_num = sum(replace_corr);
chrom(replace_corr, :) = ones(replace_num, 1)*chrom_best(1:end-1);
fitness(replace_corr) = ones(replace_num, 1)*chrom_best(end);
chrom_new = chrom;
fitness_new = fitness;

end

%计算平均适应度值
function fitness_ave = CalAveFitness(fitness)
[N ,~] = size(fitness);
fitness_ave = sum(fitness)/N;

%绘制结果
function y = PlotModel(chrom)
x = chrom(1);
y = chrom(2);
z = chrom(3);
figure(2)
scatter3(x, y, z, 'ko')
hold on
```

```
[X, Y] = meshgrid(-10:0.1:10);
Z =sin(X)+cos(Y)+0.1*X+0.1*Y;
mesh(X, Y, Z)
y=1;
```

3.模拟退火算法

模拟退火算法（SA）-CSDN博客

3.1 概念

模拟退火算法(Simulated Annealing，SA)是基于Monte-Carlo迭代求解策略的一种随机寻优算法，其出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。 从某一较高初温出发，伴随温度参数的不断下降,结合概率突跳特性在求解空间中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。

优点：

模拟退火算法是一种通用的优化算法，理论上算法具有概率的全局优化性能。 赋予搜索过程一种时变且最终趋于零的概率突跳性，从而可有效避免陷入局部极小并最终趋于全局最优的串行结构的优化算法。 应用：VLSI(超大规模集成电路)最优设计、图像处理、组合优化问题、生产调度、控制工程、机器学习、神经网络、信号处理等领域。

3.2 算法思想

模拟退火算法的思想借鉴于固体的退火过程，当固体的温度很高时，内能比较大，固体内的粒子处于快速无序运动状态，当温度慢慢降低，固体的内能减小，粒子逐渐趋于有序，最终固体处于常温状态，内能达到最小，此时粒子最为稳定。

模拟退火算法包含两个部分即Metropolis算法和退火过程。Metropolis算法就是如何在局部最优解的情况下让其跳出来，是退火的基础。

Metropolis准则 1953年Metropolis提出重要性采样方法，即以概率来接受新状态，而不是使用完全确定的规则，称为Metropolis准则，可以显著减小计算量。

假设前一状态为 x(n)，系统受到一定扰动，状态变为 x(n+1)，相应地，系统能量由 E(n) 变为 E(n+1)。 定义系统由 x(n)) 变为 x(n+1) 的接收概率为 p（probability of acceptance）：

$$p = \begin{cases} 1, & E(n+1) < E(n) \\ \exp(-\frac{E(n+1)-E(n)}{T}), & E(n+1) \geq E(n) \end{cases}$$

公式解释：当状态转移之后，如果能量减小了，那么这种转移就被接受了（以概率 1 发生）。如果能量增大了，就说明系统偏离全局最优位置（能量最低点，模拟退火算法所要寻找的就是密度最高能量最低的位置）更远了，此时算法不会立即将其抛弃，而是进行概率判断：首先在区间 [0,1]产生一个均匀分布的随机数 ε ，如果 ε<p 这种转移也将被接受，否则拒绝转移，进入下一步，如此循环。

其核心思想是当能量增加时以一定概率接收，而不是一味的拒绝，即陷入局部最优时有一定概率能跳出局部最优，其中以能量的变化量和 T 进行决定概率 p 的大小，所以这个 p 值是动态的。

3.3 模拟算法

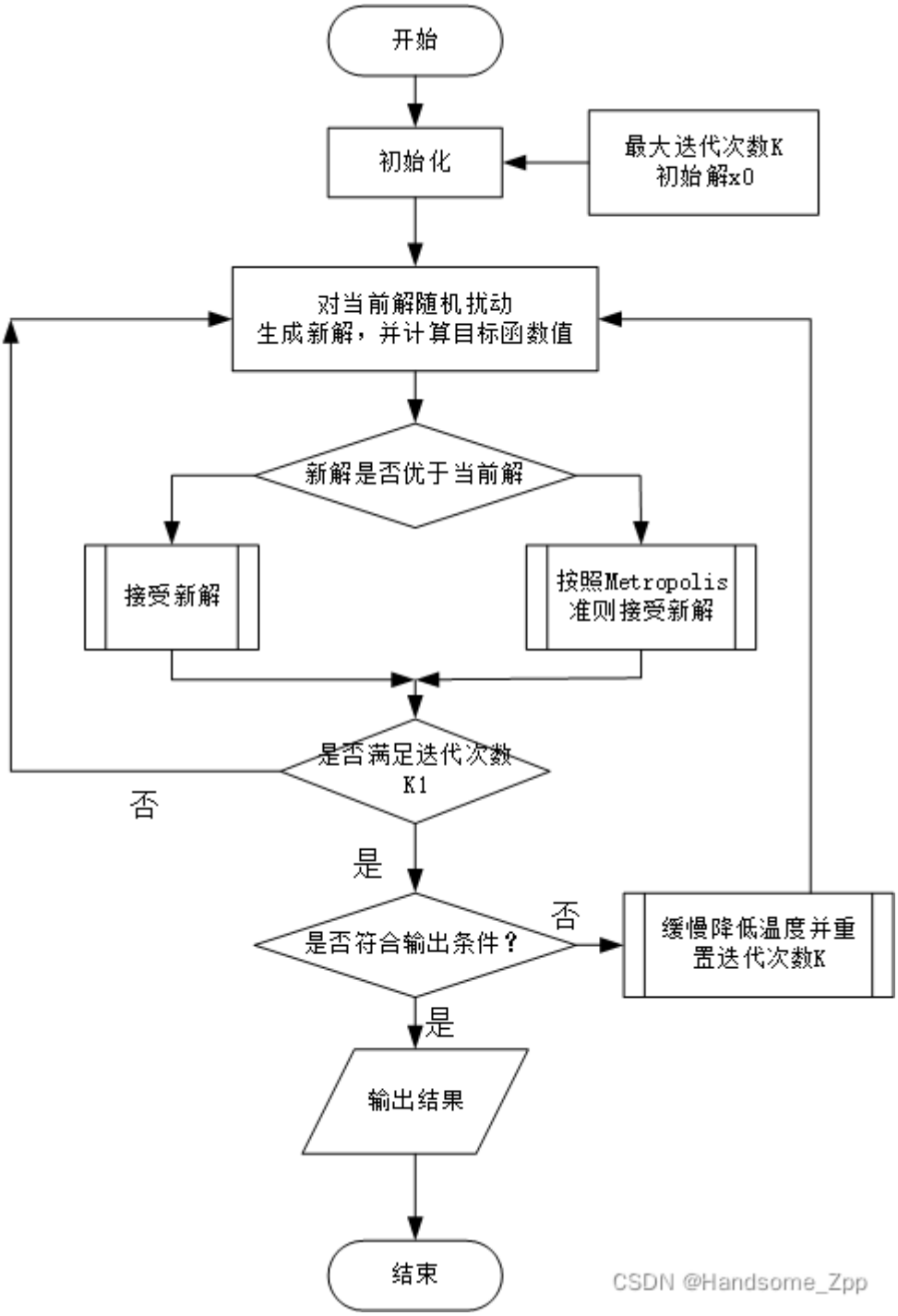
3.3.1 退火算法参数控制

Metropolis算法可以调整 T 的大小，控制算法收敛速度。T 如果过大，就会导致退火太快，达到局部最优值就会结束迭代，如果取值较小，则计算时间会增加。实际应用中采用退火温度表，在退火初期采用较大的T值，随着退火的进行，逐步降低，具体如下：

- 初始的温度T(0)应选的足够高，使的所有转移状态都被接受。初始温度越高，获得高质量的解的概率越大，耗费的时间越长。
- 退火速率： 最简单的下降方式是指数式下降。$\mathrm{T} = \lambda T$,其中λ是小于1的正数，一般取值为0.8到0.99之间，使得对每一温度，有足够的转移尝试，指数式下降的收敛速度比较慢。
- 终止温度： 如果在若干次迭代的情况下没有可以更新的新状态或者达到用户设定的阈值，则退火完成。

固体退火	模拟退火算法
粒子状态	可行解
粒子能量最低状态	最优解
设置初始温度	初始解
能量	目标函数
冷却	控制参数的下降
等温过程	Metropolis抽样过程

3.3.2 流程



算法实质分两层循环，在任一温度水平下，随机扰动产生新解，并计算目标函数值的变化，决定是否被接受。由于算法初始温度比较高，这样，使 E 增大的新解在初始时也可能被接受，因而能跳出局部极小值，然后通过缓慢地降低温度，算法就最终可能收敛到全局最优解，具体流程为：

1.初始温度T0， 令 T = T0，随机产生一个初始解 X0 ，并计算对应的目标函数值 E(x0) 。

2.令 $T = \lambda T$,其中λ取取值0到1之间，为温度下降速。

3.对当前解x_t施加随机扰动，在其邻域内产生一个新解x_{t+1} ，并计算对应的目标函数值 $E(x_{t+1})$，计算 $\Delta E = E(x_{t+1}) - E(x_t)$

4.按照 Metropolis准则 判断是否接受新解， 若$\Delta < 0$，接受新解作为当前解，否则按照概率判断是否接受。 5.在温度 T 下，重复 L 次扰动和接受过程，即执行步骤 3 和 4 。 6.判断温度是否达到终止温度水平，若是则终止算法，否则返回步骤 2。

```
% 模拟退火算法SA 解01背包问题
clc;clear;close all
weight = [2,5,18,3,2,5,10,4,11,7,14,6];     % 每个货物重量
price = [5,10,13,4,3,11,13,10,8,16,7,4];    % 每个货物单价
weight_cons = 46;    % 背包重量约束
%初始化
alpha = 0.95;        % 退火系数
t_begin = 200;       % 最高温度
t_end = 0.1;         % 结束温度
t = t_begin;         % 当前退火温度
solution_new = ones(1,length(weight));     % 新解
solution_current = zeros(1,length(weight));  % 当前解
value_current = 0;           % 当前解的价值
value_best =0;               % 最大的价值
solution_best = solution_new;
counter = 0;         % 记录迭代次数
while t>t_end
    counter = counter+1;
    for i = 1:100     % 马尔科夫长度
        %生成新的解
        index = randi([1,length(weight)],1,1);  %随机生成1~10的一个整数
        solution_new(1,index) =~solution_new(1,index);
        % 判断是否满足约束
        while sum(solution_new.*weight) > weight_cons
            index = randi([1,length(weight)],1,1);
            solution_new(1,index) =~solution_new(1,index);
        end
        % 判断是否接受当前解
        value_new = sum(solution_new.*price);             % 计算最新价值
        probability = exp((value_new-value_current)/t);  % 当前概率
        if (value_new>value_current)||probability>rand    % 新解更好或者劣解满足概率
            value_current = value_new;
            solution_current = solution_new;
        else
            solution_new = solution_current;
        end
        % 与最优解对比
        if value_current>value_best
            value_best = value_current;
            solution_best = solution_current;
        end
    end
    %保存每个过程的最优解
    value_list(counter,:)= value_best;
    solution_list(counter,:) = solution_best;
    t = t*alpha;
end
%显示当前最优解
figure(3);
plot(value_list)
```

```
xlabel('迭代次数')
ylabel('目标函数值')
title('适应度进化曲线')
fprintf('最大价值：%f，货物重量：%d\n',value_best,sum(solution_best.*weight));
disp(['解：',num2str(solution_best)]);
```

4.禁忌搜索算法

智能优化算法学习笔记（3）–禁忌搜索算法（TS）_禁ji搜索算法是智能优化算法吗-CSDN博客

4.1 概念

禁忌搜索(Tabu Search,TS)是Glover于1986年提出的一种全局搜索算法。禁忌搜索属于模拟人类智能的一种优化算法，它模仿了人类的记忆功能，在求解问题的过程中，采用了禁忌技术，对已经搜索过的局部最优解进行标记，并且在迭代中尽量避免重复相同的搜索（但不是完全隔绝），从而获得更广的搜索区间，有利于寻找到全局最优解。

禁忌搜索算法已经在组合优化、生产调度、机器学习、电路设计和神经网络等领域取得了很大的成功，近年来又在函数全局优化方面得到较多的研究，并大有发展的趋势。

混合其他现代计算方法，如遗传算法、蚁群算法等技术对传统的禁忌搜索算法进行性能的改进与提高。

4.2 禁忌搜索算法基本概念

定义1 **禁忌表**是用来存放（记忆）禁忌对象的表。它是禁忌搜索得以进行的基本前提。禁忌表本身是有容量限制的，它的大小对存放禁忌对象的个数有影响，会影响算法的性能。

定义2 **禁忌对象**是指禁忌表中被禁的那些变化元素。禁忌对象的选择可以根据具体问题而制定。例如在旅行商问题中可以将交换的城市对作为禁忌对象，也可以将总路径长度作为禁忌对象。

定义3 **禁忌期限也叫禁忌长度**，指的是禁忌对象不能被选取的周期。禁忌期限过短容易出现循环，跳不出局部最优，长度过长会造成计算时间过长。

定义4 **渴望准则也称特赦准则。**当所有的对象都被禁忌之后，可以让其中性能最好的被禁忌对象解禁，或者当某个对象解禁会带来目标值的很大改进时，也可以使用特赦规则。

定义5 **领域**即是给定点附近其他点的集合。领域就是指对当前解进行一个操作（这个操作可以称之为领域动作）可以得到的所有解的集合。

定义6 **领域动作是一个函数**，通过这个函数，对当前解s，产生其相应的邻居解集合。

例如：对于一个bool型问题，其当前解为：s = 1001，当前领域动作定义为翻转其中一个bit时，得到的邻居解的集合N(s)={0001，1101，1011，1000}，其中N(s)属于S。

定义7 **候选集合**由领域中的邻居组成。禁忌搜索算法中的候选集合指的是搜索过程中待考虑的解的集合，每个解都可以看作是一个候选解。在禁忌搜索算法中，通过对候选解进行评估和选择，来进行搜索过程，同时还需要考虑候选解的禁忌属性，即哪些解不能被选择，以保证搜索的效率和质量。候选集合的大小和组成方式会影响禁忌搜索算法的搜索效果和效率。

定义8 **评价函数**（目标函数评价函数是候选集合元素选取的一个评价公式，候选集合的元素通过评价函数值来选取。

4.3 禁忌搜索算法

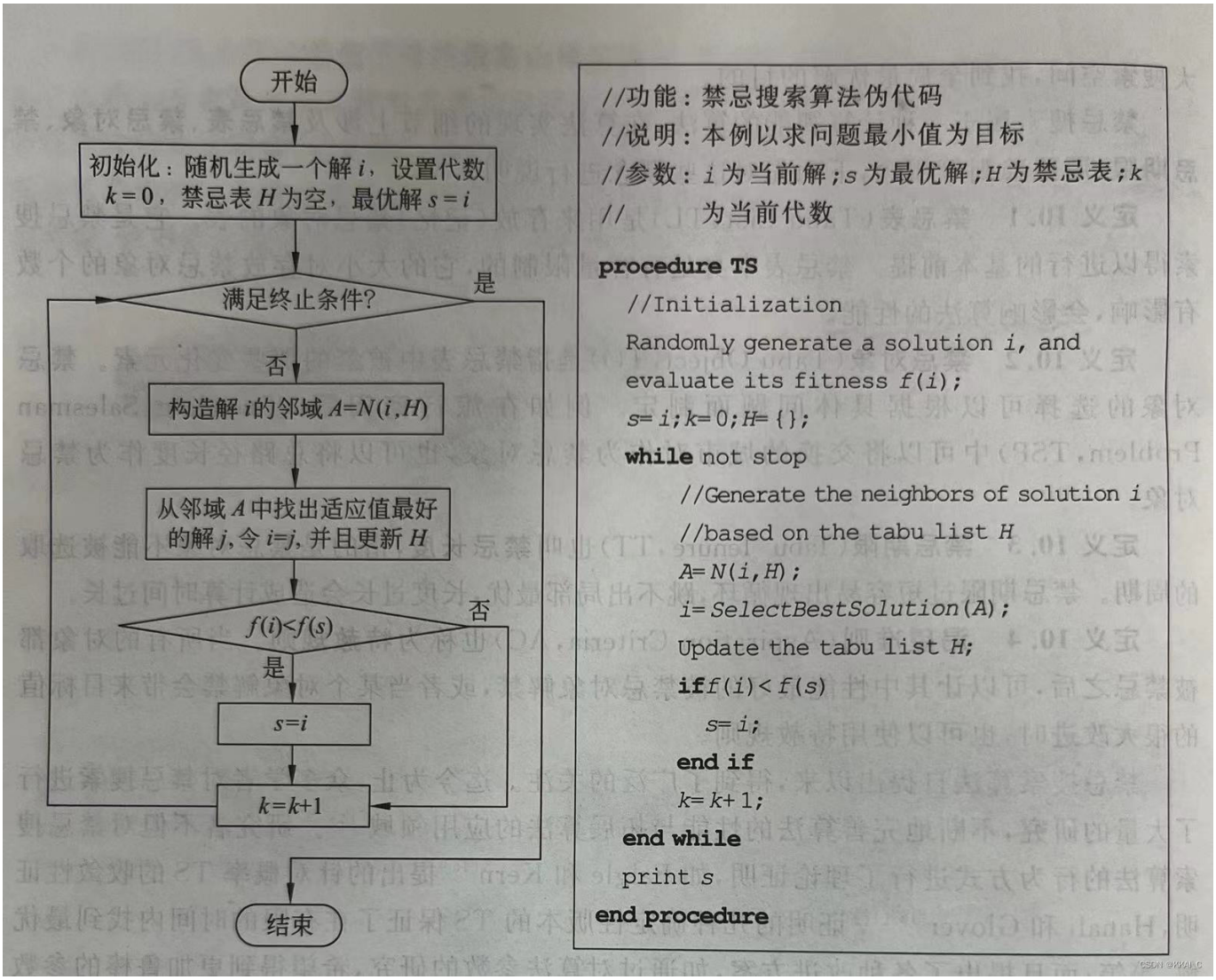
4.3.1步骤

step1.禁忌表H = 空集，并选定一个初始解x_i;

step2.在x_i的领域$N(x_i)$中选择候选集$CanN(x_i)$；在$CanN(x_i)$中选一个评价值最佳的解x_{i+1}

- 不在禁忌表中，选为当前解
- 在禁忌表中，满足破禁条件，为当前解，否则从不在禁忌表解中，选择一个最优解

step3.重复step2，直到满足停止条件



4.3.2 例子

目标函数：max $f(x,y)=(\cos(x^2+y^2)-0.1)/(1+0.3(x^2+y^2)^2)+3$

条件：$-5<x,y<5$

```
%%%%%%f(x, y)=(cos(x^2+y^2)-0.1)/(1+0.3*(x^2+y^2)^2)+3%%%%%%
clear all;                  %清除所有变量
close all;                  %清图
clc;                        %清屏
x=-5:0.01:5;
y=-5:0.01:5;
N=size(x,2);
for i=1:N
    for j=1:N
        z(i,j)=(cos(x(i)^2+y(j)^2)-0.1)/(1+0.3*(x(i)^2+y(j)^2)^2)+3;
    end
end
mesh(x,y,z)
xlabel('x')
ylabel('y')

%%%%%%%%%%%%%%%禁忌搜索算法求函数极值问题%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%初始化%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%清屏
xu=5;                                   %上界
xl=-5;                                  %下界
L=randi([5 11],1,1);                   %禁忌长度取5,11之间的随机数
Ca=5;                                   %邻域解个数
Gmax=200;                               %禁忌算法的最大迭代次数;
w=1;                                    %自适应权重系数
tabu=[];                                %禁忌表
x0=rand(1,2)*(xu-xl)+xl;                %随机产生初始解%Y = rand(m,n) 或 Y = rand([m n]) 返回
一个m x n的随机矩阵。
bestsofar.key=x0;                       %最优解
xnow(1).key=x0;                         %当前解
%%%%%%%%%%%%%%%最优解函数值，当前解函数值%%%%%%%%%%%%%%%%%
bestsofar.value=func2(bestsofar.key);
xnow(1).value=func2(xnow(1).key);
g=1;
while g<Gmax
    x_near=[];                          %邻域解
    w=w*0.998;
    for i=1:Ca
        %%%%%%%%%%%%%%%%%%%%%%产生邻域解%%%%%%%%%%%%%%%%%%%%%%
        x_temp=xnow(g).key;
        x1=x_temp(1);%注意1和l
        x2=x_temp(2);
        x_near(i,1)=x1+(2*rand-1)*w*(xu-xl);
           %%%%%%%%%%%%%%%%%%边界条件处理%%%%%%%%%%%%%%%%%%%
           %%%%%%%%%%%%%%%%%%%%边界吸收%%%%%%%%%%%%%%%%%%%%%%
        if x_near(i,1)<xl
            x_near(i,1)=xl;
        end
        if x_near(i,1)>xu
            x_near(i,1)=xu;
```

```
        end
        x_near(i,2)=x2+(2*rand-1)*w*(xu-xl);
            %%%%%%%%%%%%%%%%%边界条件处理%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%边界吸收%%%%%%%%%%%%%%%%%%%%
        if x_near(i,2)<xl
            x_near(i,2)=xl;
        end
        if x_near(i,2)>xu
            x_near(i,2)=xu;
        end
        %%%%%%%%%%%%%%%计算邻域解点的函数值%%%%%%%%%%%%%%%%%%
        fitvalue_near(i)=func2(x_near(i,:));
    end
    %%%%%%%%%%%%%%%%%%%%最优邻域解为候选解%%%%%%%%%%%%%%%%%%%%
    temp=find(fitvalue_near==max(fitvalue_near));
    candidate(g).key=x_near(temp,:);
    candidate(g).value=func2(candidate(g).key);
    %%%%%%%%%%%%%%%候选解和当前解的评价函数差%%%%%%%%%%%%%%%%%%
    delta1=candidate(g).value-xnow(g).value;
    %%%%%%%%%%%%%%%候选解和目前最优解的评价函数差%%%%%%%%%%%%%%%%
    delta2=candidate(g).value-bestsofar.value;
    %%%%%候选解并没有改进解，把候选解赋给下一次迭代的当前解%%%%%%
    if delta1<=0
        xnow(g+1).key=candidate(g).key;
        xnow(g+1).value=func2(xnow(g).key);
        %%%%%%%%%%%%%%%%%%%%%%更新禁忌表%%%%%%%%%%%%%%%%%%%%%%%%
        tabu=[tabu;xnow(g+1).key];
        if size(tabu,1)>L
            tabu(1,:)=[];
        end
        g=g+1;                      %更新禁忌表后，迭代次数自增1
    %%%%%%%如果相对于当前解有改进，则应与目前最优解比较%%%%%%%%%%
    else
        if delta2>0                 %候选解比目前最优解优
            %%%%%%%%%%把改进解赋给下一次迭代的当前解%%%%%%%%%%%%
            xnow(g+1).key=candidate(g).key;
            xnow(g+1).value=func2(xnow(g+1).key);
            %%%%%%%%%%%%%%%%%%%%%%更新禁忌表%%%%%%%%%%%%%%%%%%%%%%%%
            tabu=[tabu;xnow(g+1).key];
            if size(tabu,1)>L
                tabu(1,:)=[];
            end
            %%%%%%%%把改进解赋给下一次迭代的目前最优解%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%包含藐视准则%%%%%%%%%%%%%%%%%%%%%%%%
            bestsofar.key=candidate(g).key;
            bestsofar.value=func2(bestsofar.key);
            g=g+1;                  %更新禁忌表后，迭代次数自增1
        else
            %%%%%%%%%%%%%%%判断改进解时候在禁忌表里%%%%%%%%%%%%%%%
            [M,N]=size(tabu);
            r=0;
```

```
            for m=1:M
                if candidate(g).key(1)==tabu(m,1) && candidate(g).key(2) == tabu(m,1)
                    r=1;
                end
            end
            if  r==0
                %%改进解不在禁忌表里，把改进解赋给下一次迭代的当前解
                xnow(g+1).key=candidate(g).key;
                xnow(g+1).value=func2(xnow(g+1).key);
                %%%%%%%%%%%%%%%%%%%%更新禁忌表%%%%%%%%%%%%%%%%%%
                tabu=[tabu;xnow(g).key];
                if size(tabu,1)>L
                    tabu(1,:)=[];
                end
                g=g+1;                %更新禁忌表后，迭代次数自增1
            else
                %%%如果改进解在禁忌表里，用当前解重新产生邻域解%%%%%
                xnow(g).key=xnow(g).key;
                xnow(g).value=func2(xnow(g).key);
            end
        end
    end
    trace(g)=func2(bestsofar.key);
end
bestsofar ;          %最优变量及最优值
figure
plot(trace);
xlabel('迭代次数')
ylabel('目标函数值')
title('搜索过程最优值曲线')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%适配值函数%%%%%%%%%%%%%%%%%%%%%%%%%%
function y=func2(x)
y=(cos(x(1)^2+x(2)^2)-0.1)/(1+0.3*(x(1)^2+x(2)^2)^2)+3;
```