**模拟退火算法**

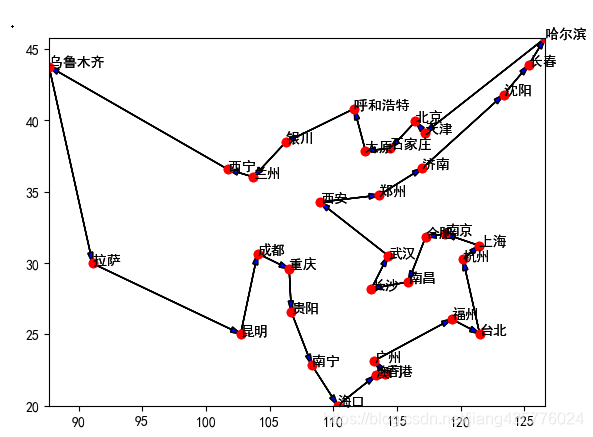
## 一、情景引入与算法介绍

**1.1 情景引入**

试想，现在有一个旅行商人要拜访n个城市，他必须选择要走的路径。

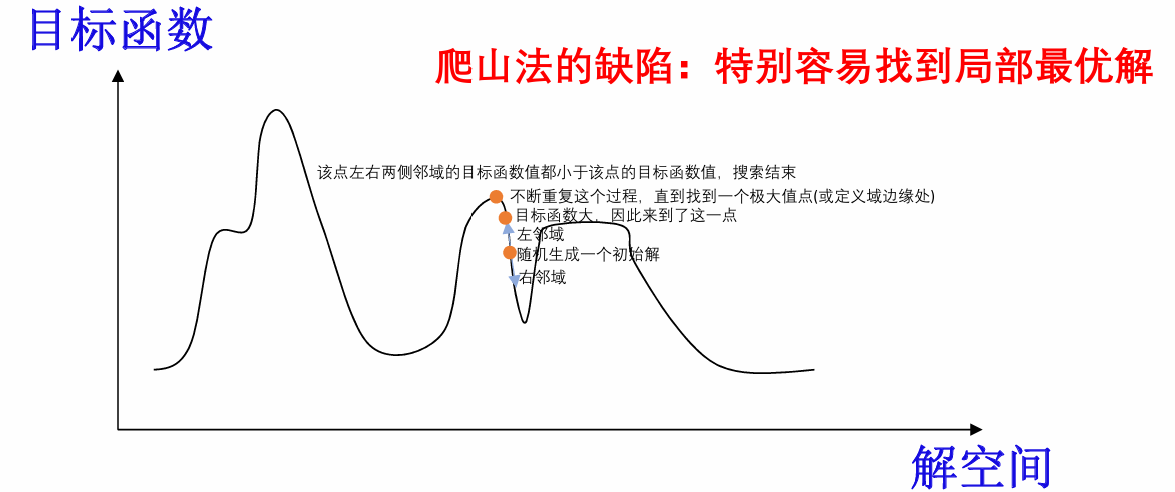
**路径的限制**是每个城市只能拜访一次，且最后要回到原来出发的城市。

**路径的选择目标**是要求路径路程为所有路径中的最小值。



这是图论中非常著名的旅行商问题。此外，还有书店买书问题、运货问题等等，其本质都是**搜索某个目标函数的最值**，模拟退火就介绍了一种搜索最值的思想。

在正式介绍模拟退火前，我们先来看一个例子，介绍一下**爬山法**



以使用爬山法寻找连续一元函数的最大值为例：

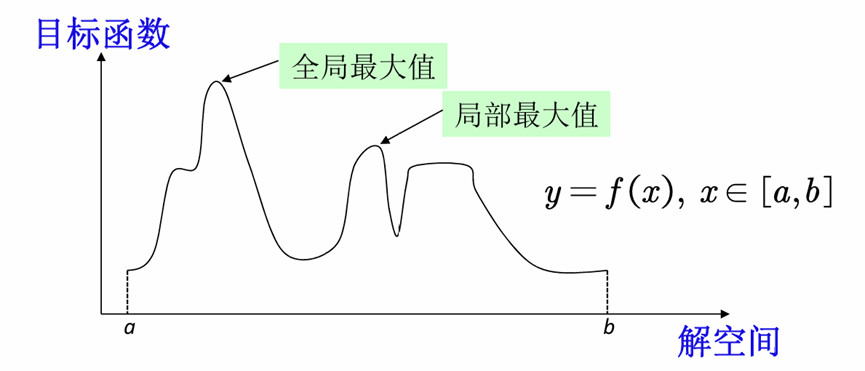
(1) 首先，在解空间中随机生成一个初始解（如上图）。

(2) 根据初始解的位置，我们在下一步有两种走法：向左邻域走一步或者向右邻域走一步（走的步长越小越好，但会加大计算量）。

(3) 比较不同走法下目标函数的大小，并决定下一步往哪个方向走。若向左走目标函数较大，则往左走一小步。

(4) 不断重复这个步骤，直到找到一个极大值点(或定义域边缘处)，此时我们结束搜索。

那么我们发现，搜索极易搜索到局部最大值即停止，在该情景中，若要继续搜索到全局最大值，需要再向左走一段。



因此，我们认为贪心算法实质上是一种“目光狭隘”的算法，也即**贪心算法。**而模拟退火算法可以克服贪心算法的问题，它在搜索到局部最优解后，仍有一定概率接收一个比当前解更差的解，也就有了找到全局最大值的可能性。

**1.2 算法介绍**

举例说明，求解函数y = 11sin(x) + 7cos(5x)在[-3,3]内的最大值，首先随机生成，计算出对应的函数值后，在附近随机找一个新解。

1. 若,则接受新解
2. 若，**定义一个接受概率**，且 。

（这种想法很直观，新解与旧解越接近，我们就越愿意接受新解。同时，这个概率应满足处于0到1之间，且随着时间的增加而减小，即在搜索前期“广撒网”，后期“精准打击”。）

为保证,我们可以定义，随时间递增。

1. 当时，生成一个0到1之间的随机数，如果,我们就接收新解,然后在的基础上寻找下一个新解，重复上面的步骤。反之，如果，则不接受新解，在的基础上寻找下一个新解，重复上面的步骤。
2. 搜索结束，获得最优解。

下面，我们重点注意一下搜索过程中**的设置**以及**新解的生成：**

的设置：模拟退火

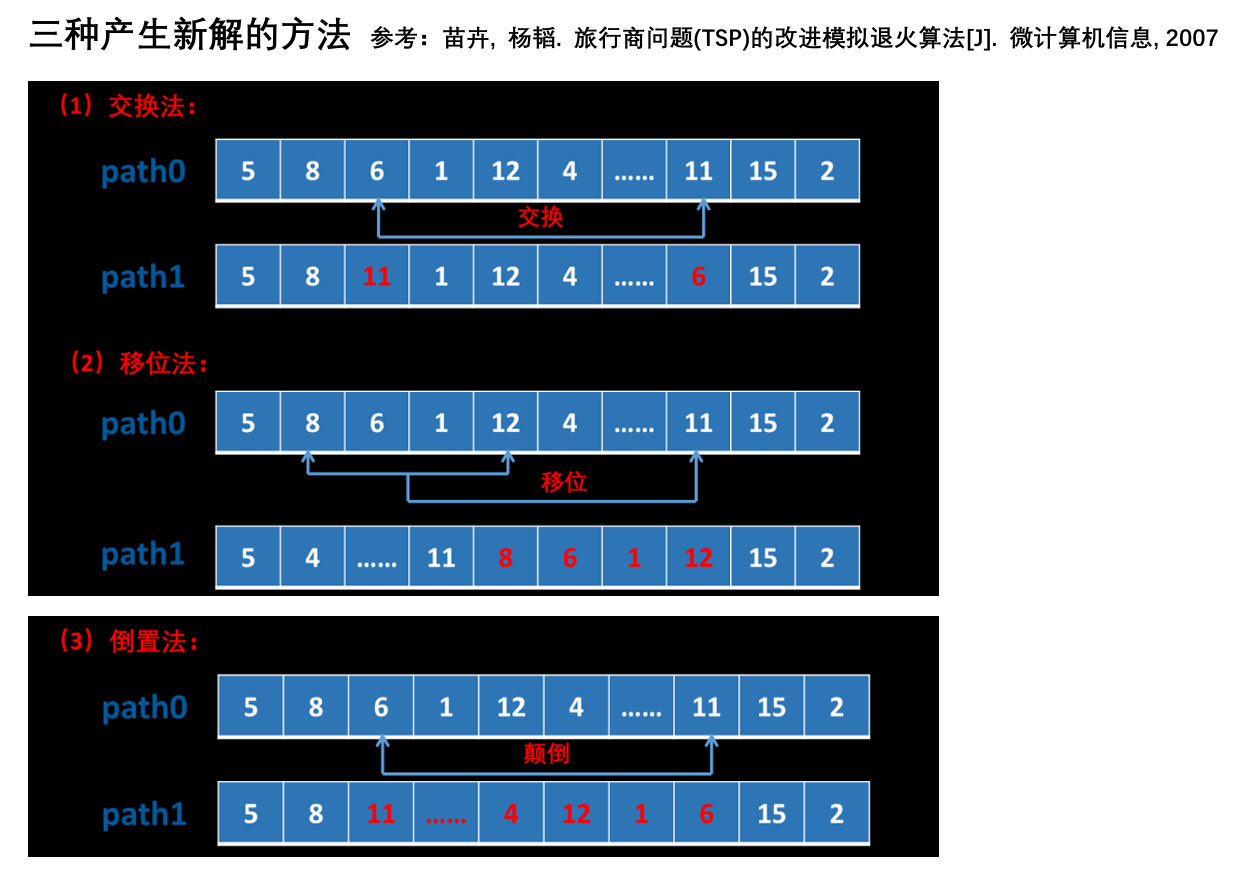
退火是一种金属热处理工艺，指的是将金属加热到一定温度，保持一定时间，然后以适宜速度冷却。退火过程中，温度随时间变化，我们通常定义初始温度为，温度下降过程表示为=，常取0.95。（参数和初始温度可根据需求自定义）

根据的性质，我们将其设置为金属冷却过程中温度的倒数，即。

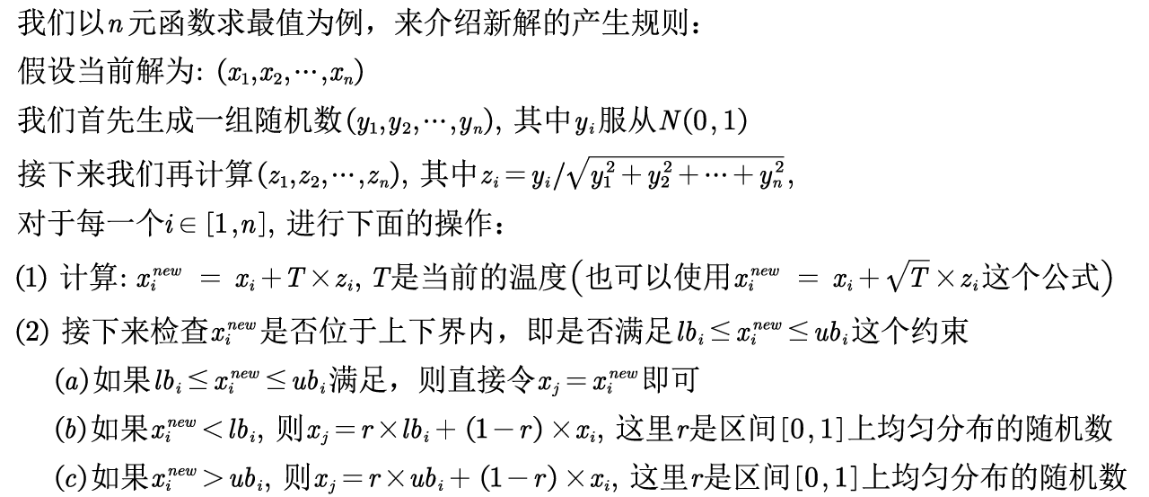
**在程序设计中，为了保证搜索足够彻底，我们需要借助循环，搜索降温过程中每一个温度对应的最优解，进行比较获得最终的最优解。**

新解的生成：

新解的生成方法需要具体问题具体分析，例如我们开头进行举例的旅行商问题（TSP）,产生新解的本质是将路径中的节点顺序（即经过的城市，这里用数字简单表示）进行调换。



此外，例如求一元函数或多元函数最值时，产生最优解的方法如下：（这里参考的是MATLAB优化工具箱中的思路）



各位可以通过代码进一步感受和分析模拟退火算法的巡行逻辑。

#### 二、代码说明

**2.1模拟退火算法寻找函数最优解**

1）含单个自变量的函数

fx1.m文件主要实现了用模拟退火算法求解函数 y = 11sin(x) + 7cos(5x)在[-3,3]内的最大值，经程序运行后所获最佳位置为1.2750，对应最优质为17.4928

该程序基本思路包括函数定义；初始化参数；生成初始解；模拟退火过程进行外内循环，内循环中生成随机解并判断是否接受，更新最优解，外循环控制温度降低；最后进行结果输出。

2）含两个自变量的函数

fx2.m文件求解含有两个未知变量函数y = x1^2+x2^2-x1\*x2-10\*x1-4\*x2+60

在[-15,15]内的最小值。

该程序与上述思路基本一致，这里不再赘述。

**2.2 模拟退火算法解决TSP问题**

TSP.m文件解决旅行商(TSP)问题，找出一条拜访n个城市且路径最短的最优化方案，经程序运行后所获得的最短路径为6659.4。需要用到的函数有：calculate\_tsp\_d.m(计算路径距离)，gen\_new\_path.m(改变路径)

该程序首先将各个城市间的距离记录在矩阵中，初始化参数后随机生成初始解，经退火算法后找出最优解，最后按照城市的坐标画出了最佳路径的路线图像。

**2.3 模拟退火算法解决书店买书问题**

BookStore.m文件解决书店买书问题，寻找花费最少的买书策略，经程序运行后得到的最优值为467,。其中需要用到的函数有：calculate\_money.m(计算方案所需花费)，gen\_new\_way.m(改变方案)。每本书在每家店的价格及每家店的运费矩阵保存在book\_data.mat文件中，打开查看步骤如下：

在命令行窗口中输入：

```

A=load('book\_data.mat') %加载数据集并储存在变量A中

```

```

A.M %查看M矩阵，即每本书在每家店的价格

A.freight %查看freight矩阵，即表示每家店的运费

```

该程序首先从数据中提取出书店数量与购书数量保存到矩阵中，初始化参数后随机生成初始解，经退火算法后找出最优解，输出最佳方案。