# **[模拟退火](https://www.cnblogs.com/rvalue/p/8678318.html)**

爬山算法就是一只兔子看到一座山峰，然后跳来跳去最后跳上山顶

模拟退火就是一只喝醉的兔子，一开始乱跳，过一会酒醒了，然后再跳上山顶

## **模拟退火算法**

模拟退火是一种随机化算法，用于求函数的极值。

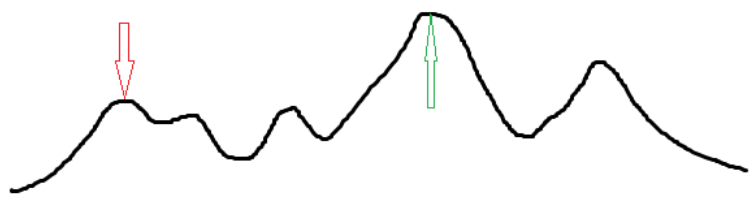
比如给出一个问题，我们要求最优解的值，但是可能的方案数量极大，直接搜索会T飞(或者方案是连续的总数无穷根本没法搜)，这种时候我们一般会有两种选择:

### **爬山算法**

爬山算法每次在当前找到的方案附近寻找一个新的方案(常见方式是随机一个差值)，然后如果这个解更优那么直接转移.

对于单峰函数来说这显然可以直接找到最优解(不过你都知道它是单峰函数了为啥不三分呢?)

但是对于多数我们求解的函数来说，它并不一定会长成这个样子...于是就极其有可能钻进一个局部的最优解出不来了

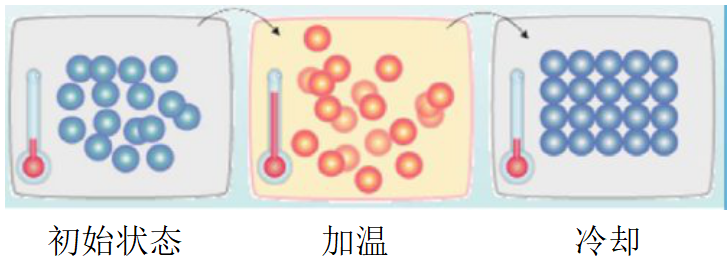


算法得出的最优解与初始解的位置以及搜寻的附近解的区域大小有关.

当然如果你寻找新方案的区间很大的话有概率跳出去，但是太大的话又可能跳来跳去跳乱了从而找不到最优解...

### **退火的理论部分**

退火其实本来是冶金工业里的术语...大概过程是先把晶体加热到极高的温度再缓慢降温，在这个过程中减少晶体中的缺陷(达到能量最低的最稳定状态)



具体理论是这样的:

设E[{xi}]表示某一物质体系在微观状态{xi}下的内能，对于给定温度T，若体系处于热平衡态时,E[{xi}]服从Boltzmann分布:



其中k为 Boltzmann 常数.

T下降,E随之下降.若T下降的速度足够慢,则体系总可以保持热平衡态. 当T→0时E趋近于最小. 这样的物质降温过程被称为退火过程.

然后机智的我们发现这个过程最终和我们的最优化过程类似!

于是我们去模拟这个过程，按照退火的规律引入更多随机因素，这样我们得到最优解的概率就会增加。

可以证明，模拟退火得到的最终解依概率收敛于最优解.

等等模拟这个过程?

根据热力学规律并结合计算机对离散数据的处理，我们定义: 如果当前温度为T，当前状态与新状态之间的能量差为ΔE，则发生状态转移的概率为:



显然如果ΔE为正的话转移是一定会成功的，但是对于ΔE<0我们则以上式中计算得到的概率接受这个新解.

然后我们维护温度T即可. 这里我们有三个参数: 初温T0，降温系数d，终温Tk

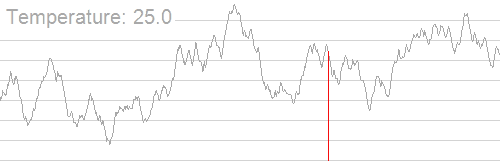
一般T0是个比较大的数,d是个接近1但是小于1的值,Tk是个接近0的正值.

首先让温度T=T0,然后进行一次转移尝试,然后让T=dT.

当T<Tk时模拟退火过程结束,当前解作为最优解.

看起来好像并不是很难理解?

Wikipedia上的动图:



一般来讲模拟退火在参数合适的情况下效果拔群，TSP随便跑(x

### **看起来好像很难理解?**

#### **[请看例题](http://www.cnblogs.com/Le-mon/p/9043130.html)**

### **解释一下此例题中一些细节：**

#### **1.安利一个模拟退火的模板:**

void fire()//模拟退火{

先定义一个温度t（看心情咯）。

while(t>1e-14)（视题目而定）

{

t\*=delta//降温(delta的值看心情，一般是0.99左右的数)

x=rand()\*t(随机出新的解用，t控制变动的范围)

newans=getans(x)（得到新解的ans）

if(判断是否更新) then 更新答案

else if(随机判断是否更新) then 更新目标

（有一定概率更新以确保不会卡在区域最优解里面）

}

}

#### **2.T在程序中的作用：**

1.可以说T是整个函数的核心部分，T的处理通常决定了退火的成功与否。

2.首先T一个类似cnt的数，表示温度是否降到0（答案是否趋于稳定）

3.在随机新解时T决定了新解的波动范围，温度越低波动越小，这样可以更大的概率更精确地命中最优解（形象来说就是在温度越低时原子波动越小）。

4.T还影响了是否更新一个不是优于当前解的目标，例如刚刚的题目中：

if(exp(-DE/t)\*RAND\_MAX>rand())

### **注意事项：**

1.关于t和delta值的确定，如果太大会导致降温太快而错过最优解，但太小又会导致时间超时，所以建议用极限数据测试一下时间。希望大家多做几个类似题目感受，才能更好把握。

2.是否更新不是最更解的目标，需要通过热力学公式P(ΔE)=eΔE/kT来灵活运用。

## **总结**

其实在上面的例子中也可以体现出来, 这个算法的要点在于新解的选取以及参数的调整...实际上利用退火过程的性质大胆随机再配合调参经验一般效果拔群OwO，用熟了之后绝对是骗分利器，但是作为一个随机化算法并不一定能找到最优解。

## **模拟退火的实际使用**

实际使用里这函数可不一定是个单元函数...而且寻找新解好像是个很模糊的东西，毕竟很多时候我们会发现我们要求解的问题的所有可能解并不是离散的...

先拿道题说说...

## **3680: 吊打XXX**

Time Limit: 10 Sec  Memory Limit: 128 MBSec  Special Judge

Submit: 4247  Solved: 1566

[[Submit](https://www.lydsy.com/JudgeOnline/submitpage.php?id=3680)][[Status](https://www.lydsy.com/JudgeOnline/problemstatus.php?id=3680)][[Discuss](https://www.lydsy.com/JudgeOnline/bbs.php?id=3680)]

## **Description**

gty又虐了一场比赛，被虐的蒟蒻们决定吊打gty。gty见大势不好机智的分出了n个分身，但还是被人多势众的蒟蒻抓住了。蒟蒻们将n个gty吊在n根绳子上，每根绳子穿过天台的一个洞。这n根绳子有一个公共的绳结x。吊好gty后蒟蒻们发现由于每个gty重力不同，绳结x在移动。蒟蒻wangxz脑洞大开的决定计算出x最后停留处的坐标，由于他太弱了决定向你求助。

不计摩擦，不计能量损失，由于gty足够矮所以不会掉到地上。

## **Input**

输入第一行为一个正整数n(1<=n<=10000)，表示gty的数目。

接下来n行,每行三个整数xi，yi，wi，表示第i个gty的横坐标，纵坐标和重力。

对于20%的数据，gty排列成一条直线。

对于50%的数据，1<=n<=1000。

对于100%的数据，1<=n<=10000,-100000<=xi,yi<=100000

## **Output**

输出1行两个浮点数（保留到小数点后3位），表示最终x的横、纵坐标。

## **Sample Input**

3

0 0 1

0 2 1

1 1 1

## **Sample Output**

0.577 1.000

HINT

## **Source**

在这个题中我们为了到达稳定状态要让整个体系的总重力势能最低。

重力势能怎么求呢? 别忘了这绳子的总长度是不会变的。。。于是某个质点的重力势能和到绳结的水平距离成一次函数关系。

我们为了简化问题，可以将某个质点对最终答案产生的贡献计算为dis(o，x)×mx。 然后我们要让这个值最小化。

这个时候我们可以考虑模拟退火。 首先随机一个点作为初始解(为了加速收敛，我们可以直接取各个点坐标的平均值所在的店)。 然后随机两个值作为差值加到这个点的坐标上作为下一个解。

然后模拟退火直接往上套就可以了233

具体实现就是一个 while 循环，循环内有4步:

1. 根据当前解找到下一个解
2. 计算下一个解的 "能量" (也就是价值)
3. 决定是否要接受这个新解
4. 降温

找下一个解的时候有一个提高精度的小技巧: 根据当前温度决定差值的范围。 这样在降温即将结束接近最优解的时候可以有更大的概率更精确地命中最优解。

当然如果是解是离散的就不能这样搞了。 以及生成下一个解的时候万万不能全部重新随机生成，那就和瞎随没区别了。。。要随机作出一些相对小的修改。

具体做法就是使用一个产生[0，1]随机实数的函数，将随机区间转为[−1，1]后乘上T作为差值。 (也就是**生成一个[−T，T]的随机值作为差值**)

**不过实际操作的时候我们较少直接输出最终解，而是选择在模拟退火的过程中单独维护一个解，只在遇到更优解的时候将其更新，增加正确率。**

**另一个提高正确率的方法是: 多次进行模拟退火过程(或者说"重新烧热再退火一遍")，每次取最优解。**

**还有就是最后烧完之后可以再在全局最优解的基础上进行爬山。**

本题的参考实现:

#include <bits/stdc++.h>

const int MAXN=1e4+10;

struct Point{

double x;

double y;

Point(double x=0,double y=0){

this->x=x;

this->y=y;

}

};

Point P[MAXN];

int n;

Point ans;

int g[MAXN];

double minAns=DBL\_MAX;

double Rand();

double Sqr(double);

double Calc(const Point&);

bool Accept(double,double);

double EucDis(const Point&,const Point&);

Point SimulatedAnnealing(Point,double,double,double);

int main(){

scanf("%d",&n);

Point init;

for(int i=0;i<n;i++){

scanf("%lf%lf%d",&P[i].x,&P[i].y,g+i);

init.x+=P[i].x;

init.y+=P[i].y;

}

init.x/=n;

init.y/=n;

SimulatedAnnealing(init,1e5,1-7e-3,1e-3);//initAns:初始解,初始温度,dec:降温速度,终止温度。

printf("%.3f %.3f\n",ans.x,ans.y);

return 0;

}

inline double Calc(const Point& origin){

double ans=0;

for(int i=0;i<n;i++){

ans+=EucDis(origin,P[i])\*g[i];

}

if(ans<minAns){

::ans=origin;

minAns=ans;

}

return ans;

}

bool Accept(double delta,double tmp){//降温或接受一定概率的升温

 return delta<0||Rand()<exp(-delta/tmp);

}

Point SimulatedAnnealing(Point initAns,double initT,double dec,double end){

//initAns:初始解，initT:初始温度,dec:降温速度，end:终止温度。

double tmp=initT;

Point now=initAns;

double nowAns=Calc(now);

while(tmp>end){

//tmp\*(Rand()\*2-1)，生成一个[−T，T]的随机值作为差值

Point next=Point(now.x+tmp\*(Rand()\*2-1),now.y+tmp\*(Rand()\*2-1));

double ans=Calc(next);//计算重力势能

if(Accept(ans-nowAns,tmp)){

now=next;

nowAns=ans;

}

tmp\*=dec;//降温

}

for(int i=0;i<1000;i++){

Point rnd=Point(ans.x+tmp\*(Rand()\*2-1),ans.y+tmp\*(Rand()\*2-1));

Calc(rnd);

}

return now;

}

inline double Rand(){

return double(rand())/double(RAND\_MAX);

}

inline double Sqr(double x){

return x\*x;

}

inline double EucDis(const Point& a,const Point& b){

return sqrt(Sqr(a.x-b.x)+Sqr(a.y-b.y));

}

但是调参的过程还是比较看脸的...究极非洲大酋长慎用(x

一般情况下我们会在时间允许的情况下尽量多地尝试新的解. 一般降温系数d与1的差减少一个数量级，时间消耗大约多10倍,T0和Tk变化一个数量级，时间消耗不会变化很大。

这种时候我们可以试着先本机跑跑自造数据看看精度怎么样. 如果发现经常陷入局部最优解的话考虑增大T0和d，如果发现最终精度不够的话考虑减小Tk.

至于模拟退火的正确率计算么...好像只有实验是最方便的了吧(x

今天上午考试的时候手调一波参数然后极限数据下测试100次发现精度达标率有60%就交了...然后A了...

于是借此在高二那边水了个rk4 (逃

### **模拟退火解旅行商问题**

刚刚说模拟退火TSP随便跑...那么我们就说说TSP怎么跑...

有人可能会问了这个东西怎么求下一个解?

其实还是随机...

对于TSP，我们的一个方案其实就是一个遍历顺序(也就是一个排列)

这时我们在生产新解的时候可以选择随机选取两个结点，然后将它们在排列中的位置交换一下(~~好暴力啊~~)

然而事实证明效果拔群...

## **总结**

模拟退火在OI中是一种在最优化问题中骗分的好方法

对于一些奇奇怪怪的多元函数也可以用这个方法来求解

其实在上面的例子中也可以体现出来，这个算法的要点在于新解的选取以及参数的调整...

实际上利用退火过程的性质大胆随机再配合调参经验一般效果拔群OwO

但是作为一个随机化算法并不一定能找到最优解qwq...IOI赛制/ACM赛制的话可能骗分更容易些?(~~毕竟可以多次~~提交~~233~~)