

模拟退火



引入

模拟退火是一种随机化算法。当一个问题的方案数量极大（甚至是无穷的）而且不是一个单峰函数时，我们常使用模拟退火求解。

解释

根据 [爬山算法](#) 的过程，我们发现：对于一个当前最优解附近的非最优解，爬山算法直接舍去了这个解。而很多情况下，我们需要去接受这个非最优解从而跳出这个局部最优解，即为模拟退火算法。

什么是退火？（选自 [百度百科](#)）

退火是一种金属热处理工艺，指的是将金属缓慢加热到一定温度，保持足够时间，然后以适宜速度冷却。目的是降低硬度，改善切削加工性；消除残余应力，稳定尺寸，减少变形与裂纹倾向；细化晶粒，调整组织，消除组织缺陷。准确的说，退火是一种对材料的热处理工艺，包括金属材料、非金属材料。而且新材料的退火目的也与传统金属退火存在异同。

由于退火的规律引入了更多随机因素，那么我们得到最优解的概率会大大增加。于是我们可以去模拟这个过程，将目标函数作为能量函数。

过程

先用一句话概括：如果新状态的解更优则修改答案，否则以一定概率接受新状态。

我们定义当前温度为 T ，新状态 S' 与已知状态 S （新状态由已知状态通过随机的方式得到）之间的能量（值）差为 ΔE ($\Delta E \geq 0$)，则发生状态转移（修改最优解）的概率为

$$P(\Delta E) = \begin{cases} 1, & S' \text{ is better than } S, \\ e^{-\frac{\Delta E}{T}}, & \text{otherwise.} \end{cases}$$

注意：我们有时为了使得到的解更有质量，会在模拟退火结束后，以当前温度在得到的解附近多次随机状态，尝试得到更优的解（其过程与模拟退火相似）。

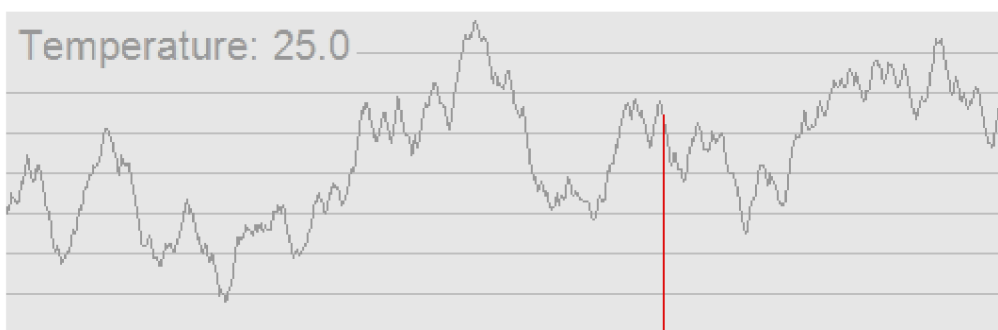
如何退火（降温）？

模拟退火时我们三个参数：初始温度 T_0 ，降温系数 d ，终止温度 T_k 。其中 T_0 是一个比较大的数， d 是一个非常接近 1 但是小于 1 的数， T_k 是一个接近 0 的正数。

首先让温度 $T = T_0$ ，然后按照上述步骤进行一次转移尝试，再让 $T = d \cdot T$ 。当 $T < T_k$ 时模拟退火过程结束，当前最优解即为最终的最优解。

注意为了使得解更为精确，我们通常不直接取当前解作为答案，而是在退火过程中维护遇到的所有解的最优值。

引用一张 [Simulated annealing - Wikipedia](#) 的图片（随着温度的降低，跳跃越来越不随机，最优解也越来越稳定）。



实现

此处代码以「BZOJ 3680」吊打 XXX（求 n 个点的带权类费马点）为例。

```
1  #include <cmath>
2  #include <cstdlib>
3  #include <ctime>
4  #include <iomanip>
5  #include <iostream>
6
7  constexpr int N = 10005;
8  int n, x[N], y[N], w[N];
9  double ansx, ansy, dis;
10
11 double Rand() { return (double)rand() / RAND_MAX; }
12
13 double calc(double xx, double yy) {
14     double res = 0;
15     for (int i = 1; i <= n; ++i) {
16         double dx = x[i] - xx, dy = y[i] - yy;
17         res += sqrt(dx * dx + dy * dy) * w[i];
18     }
19     if (res < dis) dis = res, ansx = xx, ansy = yy;
20     return res;
21 }
22
23 void simulateAnneal() {
24     double t = 100000;
25     double nowx = ansx, nowy = ansy;
26     while (t > 0.001) {
27         double nextx = nowx + t * (Rand() * 2 - 1);
```

```

28     double nxy = nowy + t * (Rand() * 2 - 1);
29     double delta = calc(nxtx, nxy) - calc(nowx, nowy);
30     if (exp(-delta / t) > Rand()) nowx = nxtx, nowy = nxy;
31     t *= 0.97;
32 }
33 for (int i = 1; i <= 1000; ++i) {
34     double nxtx = ansx + t * (Rand() * 2 - 1);
35     double nxy = ansy + t * (Rand() * 2 - 1);
36     calc(nxtx, nxy);
37 }
38 }
39
40 int main() {
41     std::cin.tie(nullptr)->sync_with_stdio(false);
42     srand(0); // 注意，在实际使用中，不应使用固定的随机种子。
43     std::cin >> n;
44     for (int i = 1; i <= n; ++i) {
45         std::cin >> x[i] >> y[i] >> w[i];
46         ansx += x[i], ansy += y[i];
47     }
48     ansx /= n, ansy /= n, dis = calc(ansx, ansy);
49     simulateAnneal();
50     std::cout << std::fixed << std::setprecision(3) << ansx << ' ' << ansy
51               << '\n';
52     return 0;
53 }

```

一些技巧

分块模拟退火

有时函数的峰很多，模拟退火难以跑出最优解。

此时可以把整个值域分成几段，每段跑一遍模拟退火，然后再取最优解。

卡时

有一个 `clock()` 函数，返回程序运行时间。

可以把主程序中的 `simulateAnneal();` 换成 `while ((double)clock()/CLOCKS_PER_SEC < MAX_TIME) simulateAnneal();`。这样子就会一直跑模拟退火，直到用时即将超过时间限制。


这里的 `MAX_TIME` 是一个自定义的略小于时限的数（单位：秒）。


习题

- 「BZOJ 3680」吊打 XXX

- [「JSOI 2016」炸弹攻击](#)
- [「HAOI 2006」均分数据](#)

 本页面最近更新：2025/8/29 18:05:34，[更新历史](#)

 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

 本页面贡献者： [Ir1d](#), [abc1763613206](#), [Mout-sea](#), [Siyuan](#), [sshwy](#), [Tiphereth-A](#), [7F88FF](#), [ChungZH](#), [Enter-tainer](#), [Ghastlcon](#), [Henry-ZHR](#), [HeRaNO](#), [hsfzLZH1](#), [iamtwz](#), [kenlig](#), [ouuan](#)

© 本页面的全部内容在 [CC BY-SA 4.0](#) 和 [SATA](#) 协议之条款下提供，附加条款亦可能应用