

模拟退火

Simulate Annealing Arithmetic

(参考自 oi-wiki)

Idea: 一句话概括: 如果新状态的解更优, 则更改当前答案; 否则以一定概率更改答案。

定义当前温度为 T , 新状态与当前状态之间的能量差为 $|\Delta E|$, 则发生转移的概率为:

$$P(\Delta E) = \begin{cases} 1 & \text{新状态更优} \\ e^{-\frac{\Delta E}{T}} & \text{新状态更劣} \end{cases}$$

设置初始温度 T_0 , 降温系数 d , 终止温度 T_k , 则退火过程如下:

1. 使温度 $T = T_0$;
2. 进行转移尝试, 随后降温至 $T := dT$;
3. 重复过程 2 直至 $T < T_k$.

注意, 为了使解更精确, 取整个退火过程中的最优解, 而非最终解。

Trick:

- 重复退火直到快超时: `while((double)clock()/CLOCKS_PER_SEC < MAX_TIME) simulateAnneal();`
- 为了让解更好, 可以在退火结束后以当前温度在当前最优解附近随机多次, 尝试得到更优的解。

Code (以 [JSOI2004]平衡点为例) :

```
1  #define T0 3000
2  #define Tk 1e-15
3  #define cool 0.996
4  #define MAX_TIME 0.75
5  inline double Rand(){ return (double)rand() / RAND_MAX; }
6  inline double calc(double xx, double yy){
7      double res = 0;
8      for(int i = 1; i <= n; i++){
9          res += sqrt((xx-x[i])*(xx-x[i])+(yy-y[i])*(yy-y[i])) * w[i];
10         if(res < ans) ans = res, ansx = xx, ansy = yy;
11     }
12     return res;
13 }
14 void simulateAnneal(){
15     double T = T0;
16     double nowx = ansx, nowy = ansy;
17     while(T > Tk){
18         double nextx = nowx + T * (rand()*2-RAND_MAX);
19         double nexty = nowy + T * (rand()*2-RAND_MAX);
20         double deltaE = calc(nextx, nexty) - calc(nowx, nowy);
21         if(Rand() < exp(-deltaE / T)) nowx = nextx, nowy = nexty;
22         T *= cool;
23     }
24     for(int i = 1; i <= 1000; i++){
25         double nextx = ansx + T * (2*Rand()-1);
26         double nexty = ansy + T * (2*Rand()-1);
27         calc(nextx, nexty);
28     }
29 }
30 int main(){
31     // ...
32     while((double)clock()/CLOCKS_PER_SEC < MAX_TIME)
33         simulateAnneal();
```

