AI实验一 "N皇后问题"说明文档

PB15111662 李双利

- ### 实验说明

  我选用的两个算法分别为“爬山算法”和“模拟退火算法”。

  - 文件结构

    在NQueens目录下共有两个cpp源代码文件、两个linux下的可执行文件以及该说明文档。

    输入文件为input.txt，运行程序时需要将input.txt文件放在该目录下，并且输入m和n。

    输出文件有两个分别为output_simulated_annealing.txt和output_hill_climbing.txt。

  - 实验要求

    棋盘中必须存在 M 对皇后会互相攻击，同时，其余的皇后之间依然 按“N 皇后”问题所描述的要求不能互相攻击。

    从实验要求分析可知，除了要解决大规模的N皇后在较短时间内计算出结果的问题之外，还需要考虑M，这也是本次实验相对困难的部分，对于M的处理我主要是集中在随机初始化状态和启发函数上。

  - 运行环境

    OS：Ubuntu 16.04 lts 64位

    CPU：i5-7200U

    Memory：8GB

  - 编译执行说明

    ```
    # 编译源代码
    g++ simulated_annealing.cpp -o sa
    g++ hill_climbing.cpp -o hc
    # 执行
    ./sa
    ./hc
    ```

- ### 算法思想

  首先分析一下本实验的不同之处，即输入的参数m，与传统的“N皇后”问题相比要求有且仅有m对皇后冲突，所以对于m要特殊处理。

  一种简单的思路是将启发函数（棋盘上冲突皇后的对数）直接设为m，然后在开始执行爬山或者模拟退火时完全随机初始化。这样理论上是可以找到解，但是随机初始化后的初始冲突数往往很大，从而收敛到m会很慢，还有一个问题是初始化的冲突数可能小于m，而采用的爬山/模拟退火策略是从较大的值向m靠近，这样就永远无法找到解。为了克服这一问题，我在随机初始化时做了以下处理：

  1. 首先就是将第i行的皇后放在第i列，这样初始化之后冲突数最大，为$\frac{n*(n-1)}{2}$

2. 求解整数方程$\frac{n^{'}*(n^{'}-1)}{2}=m$，解得$n^{'}=\frac{1+\sqrt{1+8m}}{2}$，所以固定后$n^{'}$个皇后的位置不变，这样已经保证了冲突数至少存在$t=\frac{[n^{'}]*([n^{'}]-1)}{2}$对冲突，这已经很接近于m了，求出其差距值为$ex_m=m-t$。
3. 然后对于前$n-n^{'}$而言，如果ex_m取值较小，那么可以设定一个`conflict()`函数，直接指定其随机过程的目标冲突数，如果ex_m较大（大于`conflict()`返回值），为了保证大于m，我将冲突上限设定为了$ex_m+\sqrt{ex_m}$。
4. 然后根据冲突上限值进行反复随机初始化，直到接近设定值（经过实践，取合适的冲突限值，不很小的情况下总是可以实现花费很小代价的随机化）。

   经过以上处理，初始化后的冲突数高于输入m值且很接近于m（一般情况下差值小于几百），这为爬山算法或者模拟退火算法起到了很好的优化，即爬山开始就处于靠近最优值的位置。

- 爬山算法

  爬山算法属于局部搜索算法，因此是一种解决最优化问题的启发式算法。 在实际运用中，爬山算法不会前瞻与当前状态不直接相邻的状态，只会选择比当前状态 价值更好的相邻状态，所以简单来说，爬山算法就是向价值增长方向持续移动的循环过程。 初始状态下，每行每列都是一个皇后，之后的求解就是对存在冲突的列，寻找其他的冲 突列来进行交换。因为寻找是随机的，所以每次程序的运行结果可能也存在一定的偶然误差。 求解过程中，当得到了局部最优值时，如果不是全局最优解（即冲突数为m），则随机生成初始状态，重新求解，直到得到全局最优解。

- 模拟退火算法

  模拟退火算法，本质上也是一种随机算法，但与随机重启爬山算法不同的是，它能在一定概率下转移到较差的状态。在爬山算法的基础上，设定退火状态参数，在搜索遇到局部最优解非全 局最优解的时候，根据温度和 value 变化值设定概率接收“走一条较坏的路”,调整 参数值(初始温度，退火速度，每温度运行步数)可以实现 1 概率求解，并且得到较好性能。障碍处理等同爬山算法。

  随着时间的推移，转移到较差状态的概率逐渐变小。可以证明如果让T下降得足够慢，这个算法找到全局最优解的概率逼近于1。在我的参数和初始化方法下，经过多次测试，实际中找到解的概率为85%，并且n越小可能越容易失败，所以对于模拟退火算法也进行了随机重启，在找不到解时，再次进行初始化。

- 优化策略&复杂度分析

  - 空间性能优化&空间复杂度

    一般情况下"N皇后问题"需要一个$n*n$的二维数组来保存皇后状态，其空间复杂度为$O(n^2)$，如果输入n的规模很大时（比如百万级），那么所占的空间将会很大，所以为了优化空间，我采用的是用一个一维数组来保存皇后，第i个元素表示第i行皇后的列位置，这样空间复杂度为$O(n)$。爬山算法和模拟退火算法均是一维的皇后数组，所以空间复杂度都为$O(n)$

  - 时间性能优化&时间复杂度

    - 优化**1**：随机初始化，保证以接近于**m**的冲突数开始执行算法

      具体优化策略在算法思想部分已经说明，这样就避免了从一个完全随机的状态（冲突数可能为$O(n^2)$）到达冲突数m，理论上只需要从$O(\sqrt{m})$次爬山过程（通过调参还可以降的更低）。

      在实验中我还发现了一个很严重的问题就是，在爬山失败重新找一个随机的初始状态进行爬山时，由于伪随机数的存在，经过验证一秒内的随机结果相同，为了避免伪随机数的出现对算法的性能影响，我把随机数的产生改成了毫秒级别。

```
struct timeb timeSeed;
ftime(&timeSeed);
srand(timeSeed.time * 1000 + timeSeed.millitm);
```

- 优化2：优化计算冲突值的时间代价为O(1)

  计算冲突值的函数原本需要$O(n^2)$，每次传入一个棋盘状态，得到冲突的皇后数目。可以将这个过程降为$O(1)$，维护两个$2*n-1$的数组，记录两个方向的所有对角线上的皇后数目，在初始化时计算原先状态的冲突数，空间复杂度仍然是$O(n)$。计算新的冲突数时，只需要加上两皇后交 换前后的冲突数差值即可。

- 优化3：采用首选爬山法

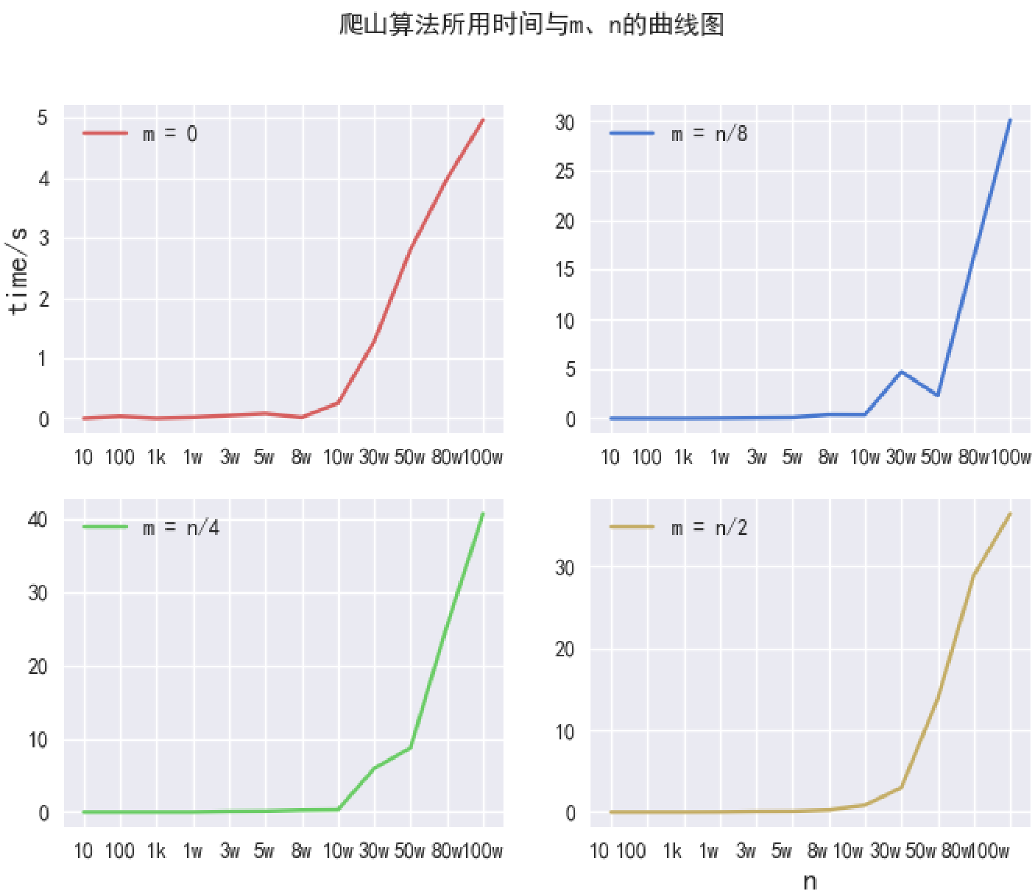
  在爬山时，每次采取的策略是尝试交换两个皇后的列坐标值，每次遇到更优的状态就更新，而不是找到最优状态，这样收敛速度没有最陡爬山法快，但是在有上千个后继结点的情况下，经过验证其时间要远远少于最陡的爬山策略。

经过以上优化后，两个算法的时间复杂度均为$O(\sqrt{m}*n^2)$。

- 结果分析

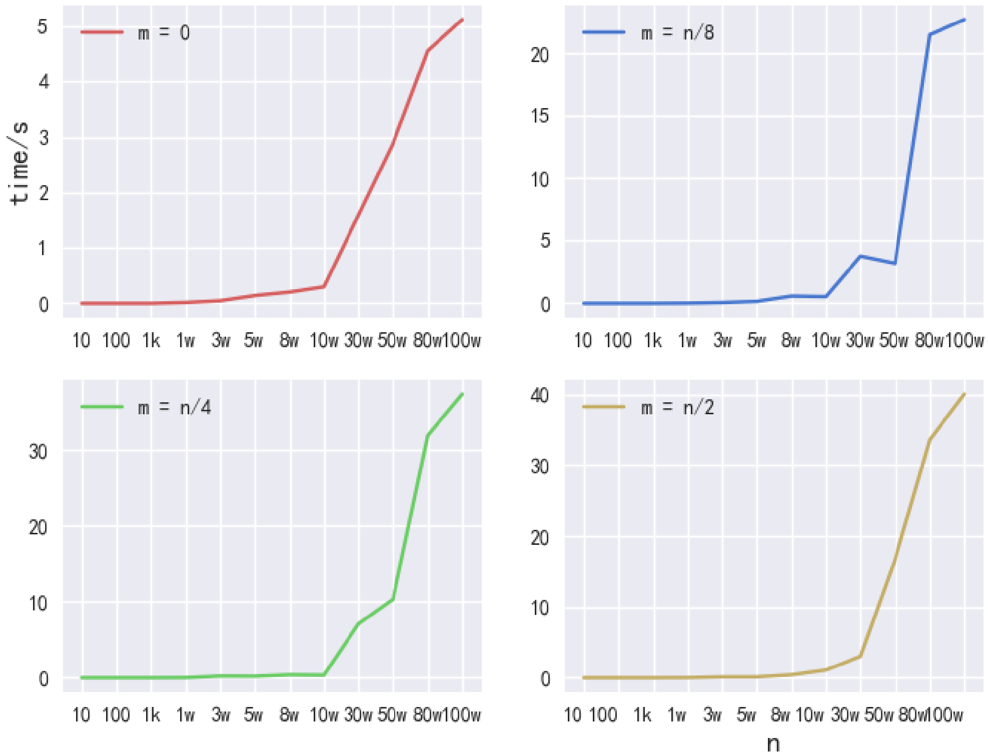
经过我用脚本测试大量mn取值的样例，n取10~1000000，m分别取了0、$\frac{n}{8}$、$\frac{n}{4}$、$\frac{n}{2}$。结果可视化如下：

  - 爬山算法实验结果



- 模拟退火算法实验结果

模拟退火算法所用时间与m、n的曲线图



从实验曲线分析可知，在我的实现算法下，在要求冲突数$m = 0$时，百万级别的皇后问题也可以在几秒之内接的，随着m的增加，找到皇后解的总体用时趋势是增加，但是由于 随机因素的存在，可能多次重启爬山，有些用时结果会偏大或偏小。