类似筛法的 N 皇后问题求解算法①

程元斌

(江汉大学 数学与计算机科学学院, 武汉 430056)

摘 要: 迄今为止, 已有多种基于不同理论的八皇后问题算法. 本文提出一种类似筛法的新算法: 在棋盘某一格放上一个皇后的同时划去经过这一格的纵、横、及正负 45 度线上的所有格位, 或者说筛去这些格位; 后来的皇后只能放在未被占据或划去的格位上; 若所有的皇后都能放入一个格位, 则得到了一个布局或一个解. 依据这种思路容易制定一个 N 皇后问题的简洁算法. 实验结果表明, 筛法算法的效率大大高于经典的回溯法.

关键词: N 皇后问题; 筛法; 回溯法

Sieve Algorithm to N Queens Problem

CHENG Yuan-Bin

(School of Mathematics & Computer Science, Jianghan University, Wuhan 430056, China)

Abstract: So far, there have been many different algorithm of eight queens problem. A similar sieve algorithm is presented in this paper: Using a algorithm similar to the sieve method in number theory: place the first queen in the first grid of the first row, and mark out the row, the column, the downward diagonal and the upward diagonal through the first queen; then place the second queen on the most left grid of the unmarked grid of the second row, and mark out the row, the column, the downward diagonal and the upward diagonal through the second queen; like this, queens are added, one by one. if all the queens can be placed on the board, a layout or a solution received. Traversing all grids of all row, we would get all solutions of N-queens problem. Based on the simple idea, we can develop a N-queens program easily. The experimental results show that the efficiency of the sieve program is much higher than the classic backtracking program.

Key words: N queens problem; sieve method; backtracking

1 引言

八皇后问题最早是由国际象棋棋手马克斯·贝瑟尔于 1848 年提出. 之后陆续有数学家对其进行研究, 其中包括高斯. 并且高斯将其推广为更一般的 n 皇后摆放问题. 八皇后问题的第一个解是在 1850 年由弗朗兹·诺克给出的. 电子计算机问世后, 八皇后问题成为数据结构及相关课程的典型问题. 例如, 著名计算机科学家 E. W. Dijkstra 就在其著作《Structured Programming》中讨论了八皇后问题的算法[1]. 迄今为止, 已有多种基于不同理论的八皇后问题算法. 其中包括经典的回溯算法[2,3], 以及遗传算法[4,5]、神经网络算法[6]、模拟 DNA 算法[7]、位运算算法[8]、基于 Q−矩阵的搜索算法[9]等. 这些算法的共同特点是从 N 皇后问题的一个解必须满足的两个约束出发来设计求解的算法.

本文从一个新的视角, 提出一种类似于数论中筛法的算法. 筛法算法直接从 N 皇后问题的定义出发, 使 N 皇后问题得到一个简洁明快的解法.

2 基本思路与基本算法

N 皇后问题的一个解, 或称一个布局, 可以用一个一维向量 A 表示. A 的第 i 个分量 a_i 表示第 i 行的皇后所在的列号. 根据 N 皇后问题的规则, A 的任意两个分量 a_i 与 a_j 须满足两个约束: 1)$a_i \neq a_j$; 2)$|a_i - a_j| \neq |i - j|$. 这两个约束也成为回溯算法及上述其他算法的基石.

① 收稿时间:2013-04-03;收到修改稿时间:2013-05-13

然而, 实际上, 我们完全不必理会这两个约束, 而只须从问题的定义出发就可以给出完美的解答. 这就是筛法. 面向 N 皇后问题的筛法的基本思路是: 首先, 在棋盘第 1 行选一格放上第一个皇后, 同时划去其所在的行、列及正负 45 度线上的所有格位; 然后, 在第 2 行没有被划掉的某个格子中选一格放上第 2 个皇后, 并划去其所在的行、列及正负 45 度线上的所有格子; 这次所划去的格子中可能有些已经在上一次被划掉, 不过这无关紧要; 放在第 2 行的皇后显然不会与第 1 行的皇后冲突, 因为产生冲突的所有格子已事先被划掉了. 然后再放第 3 行的皇后, ……; 如此继续下去, 如果到第 N 行时, 仍有没有划掉的格子, 就获得了一个布局. 这样, 只要按上述规则遍历第 1 行的所有格及其余各行未被划去的所有格, 就得到了 N 皇后问题的所有布局.

为了适应一般的 N 皇后问题, 并且算法简洁, 用递归算法比较合适. 下面是直接运用上述思想的基本算法的类 C++描述. 其中, Q 是 N X N 矩阵, 表示棋盘 Q 初始值为全 0; A 是布局向量, 存放一个布局; i 表示第 i 行. 从主程序执行 Queens(Q,0)开始求解过程.

```
Queens( Q,i)
{
 for (j = 0; j < N; j++)
  if (Q[i][j] == 0)
  {
    A[i] = j;   //将 i 行的皇后置于 j 列
    if (i == N-1)   //已到最后一行?
    {
      putout( A); //输出一个解
      return;
    }
    //拷贝 Q 到 t, 对本循环所有的 j, Q 不变.
    copyq(t, Q);
    //划去 i 行所有列, 注 1
    for (c = 0; c < N; c++) t[i][c] = 1;
    //划去 j 列所有行
    for (r = i + 1; r < N; r++) t[r][j] = 1;
    for (r = i + 1, c = j - 1; r < N && c >0; r++, c--)
t[r][c] = 1;       //注 2
    for (r = i + 1, c = j + 1; r < N && c < N; r++, c++)
t[r, c] = 1; //注 3
    //恢复 t[i][j]表示皇后占位, 注 4
    t[i][j] = 0;
    //将目前的棋局交给下一行(轮)次继续处理
    Queen(t, i + 1);
  }
}
```

注 1: 可略去此行语句, 因为处理 i 之后的行时不需要 i 行信息

注 2, 注 3: 分别划去 t[i][j]正负 45 度线上所有单元格(小于 i 的行中的已在之前的计算中划去, 且处理 i 之后的行时不需要)

注 4: 可略去此行语句, 因为处理 i 之后的行时不需要 i 行信息

3　算法的具体实现与优化

上述基本算法虽然简洁明了, 但与经典的回溯法相比, 效率并不高. 要提高效率, 还必须进行优化. 算法优化主要考虑如何减小空间复杂度与时间复杂度. 具体说来, 主要是减小数据空间和遍历次数.

首先, 将棋盘的每一格用一个比特位表示. 这样, 不仅可大大减小 Q 矩阵与 t 矩阵占用的空间, 而且对于 VC 一类的编程语言, 当 N 不大于 64 时, 可方便地用一维整型数组表示二维矩阵, 矩阵的复制速度大大提高, 位的填充与识别也十分方便. 这是发挥本算法优势的关键. 采用比特位表示后, 筛法所用时间已减至回溯法的 30%以下.

其次, N 皇后问题的任何一个布局, 经过转置或水平翻转或垂直翻转或 nπ/2 旋转后, 均依然是 N 皇后问题的一个布局. 大多数情况下, 通过上述操作, 一个布局可以派生出包括它自身的 8 个互不相同的布局; 但是对对称形式的布局, 通过上述操作, 一个布局只派生出 4 个甚至 2 个互不相同的布局. 因此, 原则上只要求出第 1 行所有 j<=(N+1)/2 的所有布局, 再通过上述变换, 就可得到 N 皇后问题的所有布局. 这一方法基本上与算法类别无关, 文献[2][3][7][8]都对此做了或繁或简的论述. 本文不再赘述. 不过, 应当指出, 因为上述变换可能产生重复的布局, 所以上述变换不能无条件全部使用. 不会产生重复布局的布局称为独立布局. 显然, 只有独立布局变换生成的布局不会与另一独立布局生成的布局重复, 而第 1 行所有 j<=(N+1)/2 的所有布局集合中的布局并非独立布局.

而且, 目前还没有见到直接或的所有独立布局的快速算法. 所以, 目前比较可行的方法是根据各变换间的关系, 进行部分遍历加部分变换. 而上述文献中, 除了文献[3]采用树结构讨论了上述部分变换的实现外, 其余的均未对变换的具体实现进行讨论. 本文根据筛法的特点, 使用了水平翻转、垂直翻转与 180 度旋转. 具体讨论如下:

不难证明, 对第 1 行所有 j<=N/2 的所有布局, 经过水平翻转, 当 N 为奇数时再加上第 1 行 j=(N+1)/2 的所有布局, 便得到 N 皇后问题的无重复的所有布局.

进一步, 对第 1 行(i=0)所有 j=0 的布局, 经过垂直翻转与 180 度旋转, 分别得到相应的最后一行(i=N-1)的所有 j=0 与 j=N-1 的布局; 而当 i=0,j=1 时, 因为此前相应的最后一行(i=N-1)的所有 j=0 与 j=N-1 的布局已经有了, 所以此轮对最后一行只须遍历[1, N-2]区间. 但实际上只须遍历[2, N-2]区间, 因为列 1(j=1)的所有格已被筛去了. 以此类推, 又可减少约 1/2 的遍历次数. 不难证明, 这样的处理所得依然是无重复的所有布局.

实验结果表明, 这一改进使得在改进 1 的基础上又将效率提高了约 65%.

第三, 因为处理 i 之后的行时不需要 i 行信息, 所以, 可以减去相应部分, 这包括基本算法中注 1 至注 4 所述, 还有 copyq()也只需要复制 i+1 行开始的数据.

第四, 在基本算法中, 每设置一个皇后占位, 都要分配一个局部变量 t,以存放对应的未完成布局, 直到以该未完成布局为基础的所有布局均给出后才释放该 t, 即处理完从此点出发的其后各行后才释放该 t. 虽然任意时刻未被释放的 t 的数量不超过 N, 但需要频繁的进行内存的分配与回收, 这也增加了时间复杂度. 对基本算法略加考察可知, 任一时刻任一行至多有一个未完成布局. 所以, 可以用全局变量 Q[N]的 Q[i+1]取代 t, Q[i]取代 Q,并取消 Queen()的参数 Q.即将 Queen(Q, i)改为 Queen(i), 以及其他涉及到 t 与 Q 的更改.

但也许是操作系统的缘故, 改进 3 与 4 产生的效益并不明显, 远没有预期的大.

下面给出上述改进后的程序. 其中, bj[j] = 1<<j, 是全局数组元素. 主函数执行 queen(0, 0, (N+1)/2).

```
void queen(int i, int jstart, int jend)
{
  for (j = jstart; j <= jend; j++)
  if ((Q[i][i] & bj[j]) == 0)
  {
    A[i] = j;
    if (i == 1) //已到最后一行?
    {
      //输出这个布局及其派生布局
      putout(A, N);
      return;
    }
    if (i == 0)
    {
      for (k = i + 1; k < N; k++)
       Q[N-1][k] = Q[i][k];//复制布局拷贝
      for (r = i + 1; r < N; r++)
       Q[N-1][r] |= bj[j];
      for(r=i+1,c=j-1;r<N && c>=0;r++,c--)
       Q[N-1][r] |= bj[c];
      for(r=i+1,c=j+1;r<N && c<N;r++,c++)
       Q[N-1][r] |= bj[c];
     //接下来从行 N-1 开始处理
     queen( N - 1, j + 1, N - 1 - j);
    }
    else
    {
        for (k = i - 1; k > 0; k--)
         Q[i-1][k]=q[i][k];//复制布局拷贝
        for (r = i - 1; r > 0; r--)
         Q[i-1][r] |= bj[j];
        for(r=i-1,c=j-1;r>0 && c>=0;r--,c--)
         Q[i-1][r] |= bj[c];
        for(r=i-1,c=j+1;r>0 && c<N;r--,c++)
         Q[i-1][r] |= bj[c];
         //继续处理其余的行
         queen(i - 1, 0, N - 1);
        }
    }
}
```

其中, putout(A, N)程序如下

```
void putout(int t[], int n)
{
    print(t);//执行输出
    for(j=0;j<n;j++)
```

```
    w[j] = n - 1 - t[j];//水平翻转
  print(w);
  if(t[0]+t[n-1-0] == n-1) //对称布局?
    return;
  for(j=0;j<n;j++)
    w[j] = t[n-1-j];//垂直翻转
  print(w);
  for (j=0;j< n;j++)
    v[j] = n - 1 - w[j];//180 度旋转
  print(v);
}
```

4 实验结果

根据上述改进后的算法(程序)及文献[2]所示的回溯算法, 笔者用 VC++分别编写了完整的 N 皇后问题的回溯算法实验程序与筛法算法实验程序, 在 CPU 为 pentium2.7GHz, 2G 内存, win7 操作系统的 PC 机上运行, 得到表 1 所示实验数据.

表 1 N 皇后问题实验数据

N	布局数	运行时间(s)	
		回溯算法	筛法算法
<9		0.000	0.000
9	352	0.002	0.000
10	724	0.008	0.002
11	2680	0.043	0.007
12	14200	0.206	0.037
13	73712	1.338	0.141
14	365596	8.519	0.806
15	2279184	57.935	5.189
16	14772512	421.077	36.465
17	95815104	3375.823	261.955
18	666090624	26001.601	2022.539

两种算法得到的布局数都是一样的, 如表 1 所示. 这些布局数也与维基百科中公布的布局数一致. 笔者亦对 N=9 以下两种算法的详细结果排序后进行了比对, 结果亦完全一致. 可以认定两种算法都是完全正确的. 从实验结果可见, 筛法算法的效率大大高于一般的回溯算法.

5 结束语

显然, 将筛法应用于 N 皇后问题是成功的. 虽然本文给出的程序仅能计算不大于 64 的 N 皇后问题, 但对于 N 大于 64 的情况, 采用比特位表示的程序也是不难实现的.

另外, 能否将筛法应用于其他的布局与博弈问题, 也许是值得探讨的. 若本文能起到抛砖引玉的作用, 将是作者倍感荣幸之事.

参考文献

1 Dahl OJ, Dijkstra EW, Hoare CAR. Structured Programming, Academic Press, London,1972 ISBN 0-12-200550-3.

2 张万军.N 皇后问题回溯算法探讨.宜宾学院学报,2006,06:64–66.

3 刘寒冰,李福荣,叶茂功.N 皇后问题的回溯算法改进.软件导刊,2010,(7):63–65.

4 王振义.遗传算法求解 N 皇后问题的优化.山西大同大学学报(自然科学版),2010,26(2):13–14,17.

5 刘娟,欧阳建权,陈良军.用混合遗传算法求解 N 皇后问题.湘潭大学自然科学学报,2007(10102):37–41.

6 白艳萍,杨明.一类求解八皇后问题的神经网络模型.山西大学学报(自然科学版),2001,(1):22–25.

7 周康,魏传佳,刘朔,卢军.八皇后问题所有解的模拟 DNA 算法.华中科技大学学报(自然科学版),2009,(6):24–27,39.

8 熊金平,唐郑熠.基于位运算的N皇后问题的解法.计算机与数字工程,2011,39(1):42–44,82.

9 王哲,栾英姿.N 皇后问题的快速搜索算法.计算机技术与发展,2009,19(6):72–75.

10 韩宇南,吕英华,黄小红.并行改进回溯算法实现 N 皇后问题的快速计数.计算机工程与应用,2006,36:1–3.