

# 中山大学数据科学与计算机学院

## 人工智能本科生实验报告

(2018-2019) 学年秋季学期

教学班级	16级计科二班	专业（方向）	计算机科学与技术
学号	16337334	姓名	周启恒

## 约束满足问题CSP

### (1) 原理

#### N皇后问题规则：

对于最终棋盘

- 每一行有且仅有一个皇后
- 每一列有且仅有一个皇后
- 每一条对角线有且仅有一个皇后

#### 回溯

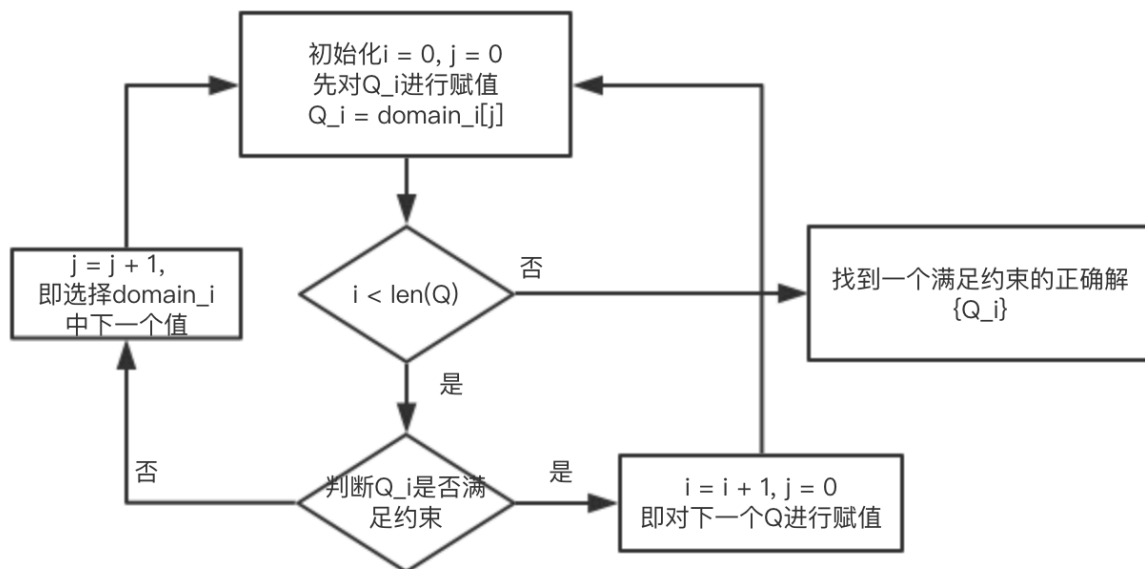
- 思想：暴力搜索的方法，遍历所有情况，不断舍弃不满足约束的解，最终找到所有满足约束的解。
- 基本步骤：
  - 对于某个状态，搜索当前对象的值域，取其中一个值进行赋值，得到新的状态。
    - 若新的状态满足约束条件则继续。
    - 否则将其视为无效解，回溯到上一个步骤重新选择值域中的其他值。
    - 当新的状态到达最终期望的目标状态也返回，进行之后的搜索。

#### 前项检测

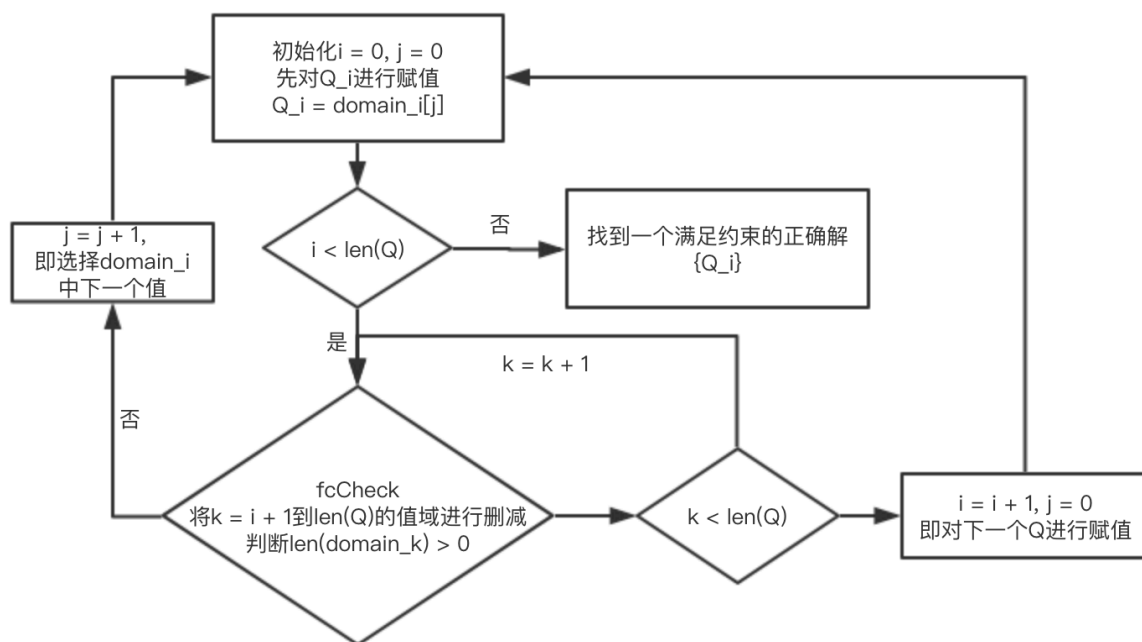
- 思想：本质上与回溯算法相同，但在搜索过程中，根据当前对象的赋值改变还未赋值的对象的值域，从而达到更好的剪枝效果，提高搜索效率。
- 基本步骤
  - 对于某个状态，搜索当前对象的值域，取其中一个值进行赋值。
  - 对于还未赋值的对象，根据当前的选择和约束条件，对自己的值域进行调整，从而得到新的状态。
  - 其他步骤与回溯算法相同

## (2) 流程图

### 回溯



### 前向检测



## (3) 关键代码 (c++带注释)

### 回溯法

#### check

```
/*  
检测方法
```

判断当前状态是否合法，即是否符合N皇后问题的规则。

1. 每一行有且仅有一个皇后（这里不可能出现）
2. 每一列有且仅有一个皇后
3. 每一条对角线有且仅有一个皇后

```
*/  
bool check(int puzzle[len], int num){  
    for(int i = 0; i < num - 1; i++){  
        if(puzzle[i] == puzzle[num - 1] || abs(puzzle[i] - puzzle[num - 1])  
== abs(i - num + 1))  
            return false;  
    }  
    return true;  
}
```

## 搜索

```
/*  
puzzle: 一个一位数组，每一位对应一行，其中的值代表对应行的皇后的列序号  
num: 代表当前搜索的行序号  
*/  
void search(int puzzle[len], int num){  
    //判断当前状态是否合法  
    if(!check(puzzle, num)) return;  
    //搜索到最底端，结束  
    if(num == len){  
        ++ total;  
        // if(total == 1) printQueen(puzzle);  
        return;  
    }  
    //对于当前行，搜索所有摆放位置  
    for(int i = 0; i < len; i++){  
        //直接赋值，之后判断  
        puzzle[num] = i;  
        search(puzzle, num + 1);  
    }  
}
```

## 前向检测

### 结构体Queen

```

/*
domain:该行的能摆放的位置的值域，一开始全部置为-1.
num:该行能摆放位置的数量
*/
struct Queen{
    int domain[len];
    int num;
    Queen():num(len){
        for(int i = 0; i < len; i ++){
            domain[i] = -1;
        }
    }
};

```

## fcCheck

```

/*
row, col:当前皇后的行列位置
对row之后的所有行进行值域缩减。
*/
bool fcCheck(int row, int col, vector<Queen>& CurDom){
    //若该行已经被放置皇后，则不继续搜索
    if(CurDom[row].domain[col] != -1) return false;
    //对当前行之后的所有行的值域进行改变
    for(int i = 1; i < len - row; i ++){
        //与当前皇后同一列的位置都删除
        if(CurDom[row + i].domain[col] == -1){
            CurDom[row + i].domain[col] = row;
            -- CurDom[row + i].num;
        }
        //与当前皇后同一对角线的位置都删除
        if(col - i >= 0 && CurDom[row + i].domain[col - i] == -1){
            CurDom[row + i].domain[col - i] = row;
            -- CurDom[row + i].num;
        }
        if(col + i < len && CurDom[row + i].domain[col + i] == -1){
            CurDom[row + i].domain[col + i] = row;
            -- CurDom[row + i].num;
        }
        //如果这一行的值域为空，则说明该解不正确，停止搜索
        if(!CurDom[row + i].num)
            return false;
    }
    return true;
}

```

## FC搜索

```

void FC(vector<Queen >& CurDom, int num){
    //到达底端，结束搜索
    if(num == len){
        ++ total;
        return;
    }
    //对当前行所有的位置进行搜索
    for(int i = 0; i < len; i ++){
        //fcCheck中对值域进行改变，如果check返回真则可以继续下一行的搜索
        if(fcCheck(num, i, CurDom)) FC(CurDom, num + 1);

        //回溯，对fcCheck中的改变进行恢复
        for(int j = 1; j < len - num; j ++){
            if(CurDom[num + j].domain[i] == num){
                CurDom[num + j].domain[i] = -1;
                ++ CurDom[num + j].num;
            }
            if(i - j >= 0 && CurDom[num + j].domain[i - j] == num){
                CurDom[num + j].domain[i - j] = -1;
                ++ CurDom[num + j].num;
            }
            if(i + j < len && CurDom[num + j].domain[i + j] == num){
                CurDom[num + j].domain[i + j] = -1;
                ++ CurDom[num + j].num;
            }
        }
    }
}

```

## (4) 优化

具体都是代码方面的优化。

- fcCheck的变化与回溯时对指定位置进行操作。
- 不进行矩阵的复制（提速效果明显）

## (5) 结果展示

- N=8, N=10的结果：

N: 8

Q \* \* \* \* \* \* \* \*

\* \* \* \* Q \* \* \*

\* \* \* \* \* \* \* Q

\* \* \* \* \* Q \* \*

\* \* Q \* \* \* \* \*

\* \* \* \* \* \* Q \*

\* Q \* \* \* \* \* \*

\* \* \* Q \* \* \* \*

num of solutions: 92

回溯: 0.000257

num of solutions: 92

前向检测: 0.000234

N: 10

Q \* \* \* \* \* \* \* \*

\* \* Q \* \* \* \* \* \*

\* \* \* \* \* Q \* \* \*

\* \* \* \* \* \* Q \* \*

\* \* \* \* \* \* \* Q

\* \* \* \* Q \* \* \* \*

\* \* \* \* \* \* \* Q \*

\* Q \* \* \* \* \* \* \*

\* \* \* Q \* \* \* \* \*

\* \* \* \* \* Q \* \* \*

num of solutions: 724

回溯: 0.004012

num of solutions: 724

前向检测: 0.00383

算法	N = 8(92)	N = 10(724)	N = 13(73712)	N = 16(14772512)
回溯	0.000244s	0.004454s	0.62406s	190.653s
前向检测	0.000218s	0.003979s	0.534973s	137.869s