Chapter 5 回溯算法

Backtracking Method

计算机科学与技术学院
网络工程系
黄 河

1

不断关注那些已被他人成功应用的新思路。你的原创思想只应该应用在那些你正在研究的问题上

托马斯.爱迪生(1847-1931)

2

基于搜索的算法设计技术

- 在现实世界中，很多问题没有(至少目前没有)有效的算法，这些问题的解只能通过漫无目的的穷举搜索来得到。只有满足约束条件的解才是可行解，只有满足目标函数的解才是最优解，这就有可能避免无效的搜索，提高搜索效率。
- 回溯法backtracking和分支限界法branch and bound均属于有组织的系统化搜索技术，可看作是穷举搜索的改进。
  - 蛮力法：生成问题的所有可能解，再去评估是否满足约束条件；
  - 回溯法：每次只构造可能解的一部分，然后评估这个部分解，如果有可能导致一个完整解，则对其进一步构造，否则不必继续构造这个部分解。这样可以避免搜索所有的可能解，适用于求解组合数量较大的问题。

2017/12/21 3

回溯法

- 回溯法有“通用解题法”之称。它在问题的解空间树中，按深度优先策略(DFS)，从根结点出发搜索解空间树。算法搜索至解空间树的任一节点时，先判断该节点是否包含问题的解，若肯定不包含，则跳过对该节点为根的子树搜索，逐层向其祖先节点回溯；否则，进入该子树，继续按深度优先搜索策略搜索。
- 回溯法在求问题的所有解时，要回溯到根，且根节点的所有子树都被搜索一遍才结束。
- 若只需求解一个解时，只要搜索到问题的一个解就结束。这种以深度优先方式系统搜索问题解的算法称为回溯法；
- 回溯法适用于：它适用于求解组合数较大的问题。

4

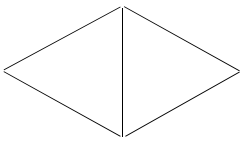
概述—问题的解空间

- 提出问题？
  - 复杂问题常常有很多的可能解，这些可能解构成了问题的解空间
  - 解空间？
    - 就是在穷举法中提到的所有可能解的搜索空间。一般而言，解空间中应该包括所有的可能解
    - 问题的解向量为 $X=(x_1, x_2, \ldots, x_n)$。$x_i$的取值范围为有穷集$S_i$。把 x_i 的所有可能取值的组合，称为问题的解空间。每一个组合是问题的一个可能解
    - 不正确的解空间会引发问题？可能会增加很多重复解，或者根本就搜索不到正确的解!
    - 例： 用桌面上的6根火柴棒为边搭建4个等边三角形

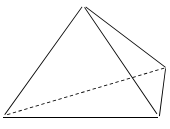
2017/12/21 5

概述—问题的解空间

- 示例：桌子上有6根火柴棒，要求以这6根火柴棒为边搭建4个等边三角形



(a) 二维搜索空间无解



(b) 三维搜索空间的解

错误的解空间将不能搜索到正确答案!

2017/12/21 6

概述—问题的解空间

❖例：*0/1*背包问题中，x_i 有*0/1* 两种取值，则解空间取值集合 $S=\{0, 1\}$，当$n=3$时，*0/1*背包问题的解空间是：

{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)}

即：当输入规模为n 时，有2^n 种可能的解。

❖例：货郎担问题，$S=\{1,2,\ldots,n\}$，当$n=3$时，$S=\{1,2,3\}$

货郎担*TSP*问题的解空间中的可能解有*27*个，是：

{(1,1,1),(1,1,2),(1,1,3),(1,2,1),(1,2,2),(1,2,3),…,(3,3,1),(3,3,2),(3,3,3)}

➢当输入规模为 n 时，它有 n^n 种可能的解。

➢考虑到约束方程 $x_i \neq x_j$。因此，货郎担问题的 解空间压缩为：

{(1,2,3),(1,3,2),(2,1,3),(2,3,1),(3,1,2),(3,2,1)}

➢当输入规模为n 时，它有 $n!$ 种可能的解。

概述—问题的解空间

■ 可能解的表示方式？(两种！物理意义不同)

如，对于n 个物品的 *0/1* 背包问题，其可能解有以下两种：

❖可能解由一个不等长向量组成，当物品 i $(1\le i\le n)$装入背包时，解向量中包含分量 i，否则，解向量中不包含分量 i，当$n=3$时，其解空间是：

{ ( ), (1), (2), (3), (1, 2), (1, 3), (2, 3), (1, 2, 3) } 解向量中的分量表示物品

❖可能解由等长向量 $\{x_1, x_2, \ldots, x_n\}$ 组成，$x_i=1/0$ $(1\le i\le n)$ 分别表示物品 i 装入/不装入背包的情况；

当$n=3$时，其解空间是：

{(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0), (1, 1, 1) }

概述—问题的解空间

▪回溯法“可能解”及“解空间”的表达

▪ 可能解的表示：用等长向量 $X=(x_1, x_2, \ldots, x_n)$，其中分量 x_i $(1\le i\le n)$ 的取值范围是某个有限集合 $S_i=\{a_{i1}, a_{i2}, \ldots, a_{im}\}$，所有可能的解向量构成了问题的解空间。

▪ 解空间表达：用解空间树(Solution Space Trees)/也称状态空间树的方式组织：

- ▪第1层(根结点)：表示搜索的初始状态
- ▪第2层结点：表示对解向量的第一个分量做出选择后到达的状态
- ▪第1层到第2层间边上标出对第一个分量选择的结果
- ▪依此类推，从根结点→叶结点的路径构成了解空间的一个可能解
- ▪解向量长度为n，状态空间树的高度为？分量取值个数决定了树的哪个属性？

概述—问题的解空间

▪可行解：满足约束条件的解，解空间中的一个子集

▪最优解：

使目标函数取极值(极大或极小)的可行解，一个或少数几个

例：货郎担问题，有n^n种可能解。$n!$种可行解，只有一个或几个是最优解。

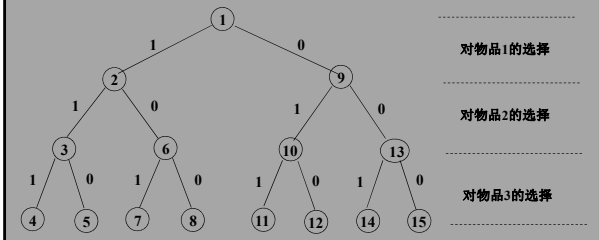
例：背包问题，有2^n种可能解，有些是可行解，只有一个或几个是最优解

有些问题，只要可行解，不需要最优解：

例：八皇后问题和图的着色问题

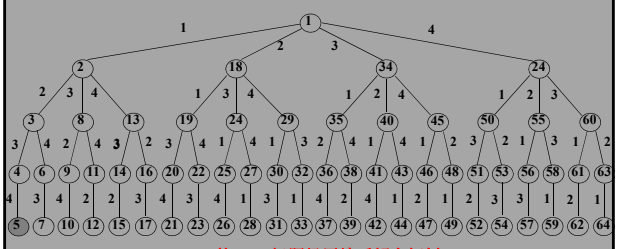
概述—问题的解空间

例：对于$n=3$ 的*0/1* 背包问题，其解空间树如下图所示，树中的*8*个叶子结点分别代表该问题的 *8* 个可能解。注意结点编号



概述—问题的解空间

例：对于$n=4$的*TSP*问题，下图是经压缩后的解空间树，树中的*24*个叶子结点分别代表该问题的*24*个可能解，例如结点*5*代表一个可能解，路径为*1→2→3→4→1*，长度为各边代价之和。



n=4 的 TSP 问题经压缩后解空间树

回溯法的设计思想

回溯法从根结点出发，按照深度优先策略遍历(DFS)解空间树，搜索满足约束条件的解。在搜索至树中任一结点时，先判断该结点对应的部分解是否满足约束条件/是否超出目标函数的界，也就是判断该结点是否包含问题的(最优)解，如果肯定不包含，则跳过对以该结点为根的子树的搜索，即所谓剪枝(*Pruning*)；否则，进入以该结点为根的子树，继续按照深度优先策略搜索。

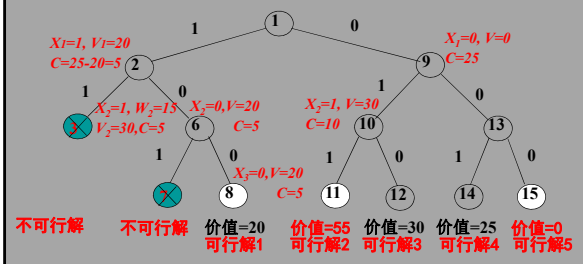
回溯法解题的步骤

回溯法解题通常包含以下三个步骤：

1) 针对所给问题，定义问题的解空间；
2) 确定易于搜索的解空间结构；
3) 以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

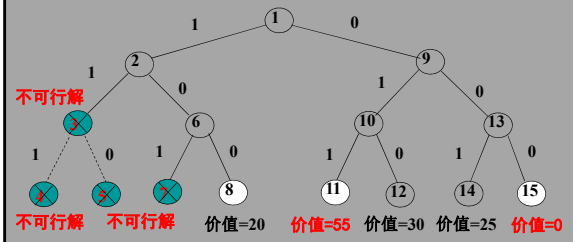
回溯法的设计思想

例：对于n=3 的0/1 背包问题，其参数如下：
重量W={20, 15,10}，价值V={20, 30, 25}，背包容量C=25
下图是解空间树结构结果，从根结点出发，其搜索过程如下：



回溯法的设计思想

在完全二叉树中，结点总数有15个



回溯法的设计思想

- 回溯法的搜索过程涉及的结点称为搜索空间，只是整个解空间树的一部分，在搜索过程中，通常采用两种策略避免无效搜索：
  - 用约束条件剪去得不到可行解的子树；
  - 用目标函数剪去得不到最优解的子树。
    这两类函数统称为剪枝函数(*Pruning Function*)。
- 需要注意的是：
  - 问题的解空间树是虚拟的，并不需要在算法运行时构造一棵真正的树结构，只需要存储从根结点到当前结点的路径
  - 如果解空间树中从根结点到叶结点的最长路径的长度为h(n)，则回溯法所需的计算空间通常为O(h(n))。而显式地存储整个解空间则需要$O(2^{h(n)})$ (0-1背包)或O(h(n)!) (n-queens)内存空间。

回溯法——递归回溯

- 回溯法对解空间作深度优先搜索，因此，在一般情况下用递归方法实现回溯法。

```
void backtrack (int t){
   if (t>n) output(x);
   else
      for (int i=f(n,t);i<=g(n,t);i++)
      {
         x[t]=h(i);
         if (constraint(t)&&bound(t)) backtrack(t+1);
      }
}
```

解空间树
活结点：如果已经生成一个node，而该node的所有儿子还未全部生成，则该结点称为~；
E-node 扩展结点:当前正在生成儿子的活结点称为~；
死结点：不再进一步扩展，或者儿子结点已经全部生成的node称为~；

- n用来控制递归的深度，当t>n表示已搜索到叶子, output输出可行解；
- f(n,t) & g(n,t)当前扩展结点出未搜索过的子树起始编号和终止编号；
- constraint(t)和bound(t)约束函数和限界函数

回溯法——迭代回溯

- 采用树的非递归深度优先算法，可将回溯法表示为非递归迭代过程。

```
void iterativeBacktrack ()
{
 int t=1;
 while (t>0) {
  if (f(n,t)<=g(n,t))
   for (int i=f(n,t);i<=g(n,t);i++) {
    x[t]=h(i);
    if (constraint(t)&&bound(t)) {
     if (solution(t)) output(x);
     else t++;}
    }
  else t--;
  }
}
```


Backtracking Method and Search Tree

1. **Thought of Backtracking Method**

   有一类问题，需要找出它的解集合，或要求找出满足某些约束条件下的最优解，最简单的方法是回溯法。

   所谓回溯就是走回头路，即在一定的约束条件下试探地搜索前进，若前进中受阻，则回头另择道路继续搜索(搜索路线是一棵树)

2. **n-Queens Puzzle (Gauss 1777-1855 德国数学家)**
   - 高斯8后问题 (1850提出)，即在8×8国际象棋棋盘上，安放8个皇后，要求彼此互不攻击，有多少个解，这些解的格局如何？

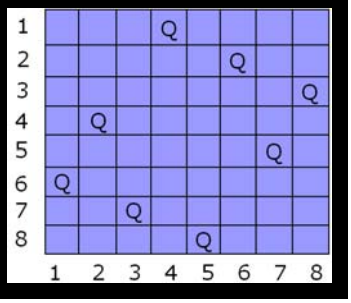
     他本人未解决此问题

     原因：$C_{64}^{8}=\frac{64!}{8!56!}=2^{32}$ 种格局，92个解(包括对称解)

回溯法——N皇后问题

- 在n × n格的棋盘上放置彼此不受攻击的n个皇后。按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。n后问题等价于在n × n格的棋盘上放置n个皇后，任何2个皇后不放在同一行或同一列或同一斜线上。

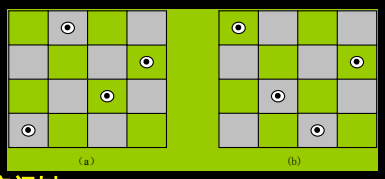
解向量如何描述？
解向量取值集合？
约束条件？



回溯法——N皇后问题

- **四皇后问题的求解过程**
  - 四皇后问题的解空间
    - 向量$X=(x_1, x_2, x_3, x_4)$表示皇后的布局
    - 分量x_i表示第i行皇后的列位置
    - x_i的取值范围$S_i=\{1,2,3,4\}$，有n^n个可能解，即其解空间树是一个完全4叉树
  - 例：向量$x_1=(2,4,3,1)$ 和 $x_2=(1,4,2,3)$ 对应于图(a)和(b)的皇后的两种布局。
    - 显然，这两种布局都不满足问题的要求

回溯法——N皇后问题



- **状态空间树**
  - 4叉完全树
  - 约束方程：

针对列的约束

$$x_i \neq x_j \qquad 1 \le i \le 4, 1 \le j \le 4, i \neq j$$

针对斜线约束

$$|i-j| \neq |x_i-x_j| \qquad 1 \le i \le 4, 1 \le j \le 4, i \neq j$$

回溯法——N皇后问题

- 可见，棋盘的每一行上可以而且必须摆放一个皇后，所以，N皇后问题的可能解用一个n元向量$X=(x_1, x_2, \ldots, x_n)$表示，其中$1 \le i \le n$并且$1 \le x_i \le n$，即第i个皇后放在第i行第$x_i$列上。
- 由于两个皇后不能位于同一列上，所以，解向量X必须满足约束条件：

$$x_i \neq x_j$$

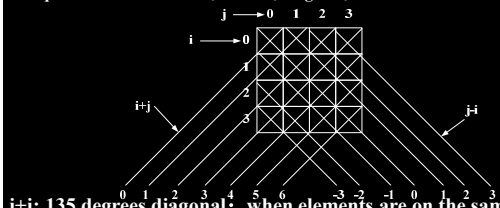
- 若两个皇后摆放的位置分别是(i, x_i)和(j, x_j)，在棋盘上斜率为-1的斜线上，满足条件$i-j=x_i-x_j$，在棋盘上斜率为1的斜线上，满足条件$i+j=x_i+x_j$。综合两种情况，由于两个皇后不能位于同一斜线上，所以解向量X必须满足约束条件：

$$|i-x_i| \neq |j-x_j|$$

Backtracking Method and Search Tree

- Attack (Constraint Condition)

two queens on the same row, column, diagonal, will attack each other



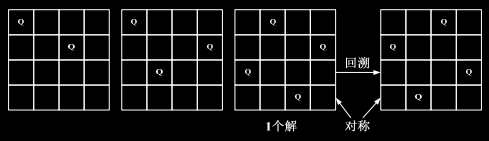
i+j: 135 degrees diagonal: when elements are on the same diagonal, the sums of column numbers and row numbers are the same (2n-1)
0～2n-2

j-i: 45 degrees diagonal: when elements are on the same diagonal, the differences between column numbers and row numbers are the same (2n-1)
–(n-1),…,0,…,n-1

Backtracking Method and Search Tree

- Backtracking Method

put queens one by one from the first line, in every line, from the first column, try whether current position is safe, put a queen if current position is safe, if all positions in one line are not safe, then backtrack to last line and reset



将上述求解过程中棋盘状态的每一步变化用树来表示，则可用4叉树表示,该树反映了状态空间中搜索过程，不满足约束条件的结点不再生长(即被剪枝)。 先序遍历该树

Backtracking Method and Search Tree

- n-Queen Algorithm

设try[0..n-1]存放解，下标为行，值为列，即：设try[i]=j，则(i, j)表示棋盘上第i行，第j列存一皇后，逐行放置，每行只有一皇后故可不考虑行冲突,只要考虑列和2条对角线冲突。

```
void Queens(i, col, diag45, diag135){ //i, col, diag45, diag135为值参
  // 全局量try进入此处时，部分解try[0..i]已求出，col, diag45, diag135
  // 是集合，初始调用为Queens(-1, Φ,Φ, Φ)
  if (i == n-1) // 搜索到叶子
     print try[0..n-1]; // 输出一个解
  else // 试求部分解try[0..i+1]，即在(i+1)行上放皇后
     for (j = 0; j < n; j++) // 试探在第(i+1)行上放皇后
        if (j ∉ col && j-(i+1) ∉ diag45 && (i+1)+j ∉ diag135) {
           // (i+1, j)位置安全
           try[i+1] = j;
           Queens(i+1, col ∪ {j}, diag45 ∪ {j-i-1}, diag135 ∪ {i+j+1});
        } // endif
}
```


Backtracking Method and Search Tree

- n-Queen Algorithm

the implementation process of the algorithm above is the state tree (preorder traversal)

- Improvement

we can set the Boolean array (all initial values are false) to test security in practical algorithm

② Put Queens (i+1, j)

令：col[j] = true, 表示第j列已有皇后

diag45[(n-1)+j-(i+1)]=true,表示该对角线上已有皇后

// 移位n-1

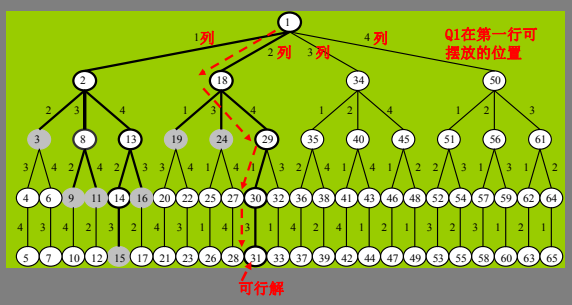
diag135[(i+1)+j]=true,表示该对角线上已有皇后

② Security Test

if (!col[j] && !diag45[n-i+j-2] && !diag135[i+1+j])

Note: 用数组要注意它们不是值参，因此可将其说明为结构，内含数组

N皇后问题



八皇后问题

■时间复杂度计算

- 运行时间，取决于所访问结点个数*c* 。每访问一个结点，就计算若干次约束方程。约束方程的执行次数，最少*1*次，最多*n-1*次。因此，总次数为*O(cn)*。
- 结点个数 *c* 是动态生成的，对不同实例，具有不确定性。一般可由 *n* 的多项式确定。
- 在*4*叉完全树中，结点总数有*1＋4＋16＋64＋256＝341*个。
- 回溯法处理时，需要搜索的结点*远小于解空间树结点*。约为前者的*8%*
- 实际模拟表明：*当n=8时*，被访问的结点数与结点总数之比约为*1.5%*。
- 在最坏情况下的花费是$O(n^n)$。
- 空间复杂性：显然，这个算法需要使用一个具有 *n* 个分量的向量来存放解向量，所以，算法所需要的工作空间为 $\Theta(n)$ 。

回溯法求解0-1背包问题

- 构造出问题的状态空间树以后，就可以从其根结点出发搜索解空间，即决定每个物品的取舍。
- 为了使目标函数的值增加最快，可以优先选择价值最大的物品装入背包，然后是价值量次之的物品……直至背包装不下为止。但是，如果所选择的物品重量很大，使得背包载重量消耗速度太快，以至后续能装入背包的物品迅速减少，使得继续装入背包的物品在满足了约束方程的要求以后，无法达到目标函数的要求。
- 因此，最好优先选择那些既能使目标函数的值增加最快，又能使背包载重量消耗速度较慢的物品装入背包。为了达到这个目的，首先把所有物品按价值重量比的非增顺序排列，然后按照这个顺序进行搜索。

31

回溯法求解0-1背包问题

- 在装包过程中，要尽量优先选择价值重量比较高的物品装入背包。表现在搜索过程中，就是要尽量沿着左子树结点前进。当不能继续前进时（假设该结点为T），就得到问题的一个部分解，并把搜索转移到右子树。估计由该部分解所能得到的最大价值，即结点的上限。
- 可以用贪婪算法处理剩余物品：将按照价值重量比非增顺序排列的剩余物品依次装入背包，至无法完全装入下一个物品时，就将该物品的一部分装满背包。这样就可以得到一个上限。如果该值大于当前最优值：继续由右子树向下搜索，扩大部分解，直至找到可行解；保存可行解，并把可行解的值作为当前最优值，向上回溯，寻找其他可行解；若该值小于当前最优值：丢弃当前正在搜索的部分解，向上回溯。反复使用此方法，直至搜索完整个解空间。

32

回溯法求解0-1背包问题

```
BOUND( v,w,k,W )
1 b ← v
2 c ← w
3 for i ← k+1 to n
4       do c ← c + w[i]
5          if c < W
6             then b ← b + v[i]
7             else return (b+(1-(c-W)/w[i])*v[i])
8 return b
```

可分割情况下最大价值

贪心算法

33