山东师范大学 SHANDONG NORMAL UNIVERSITY

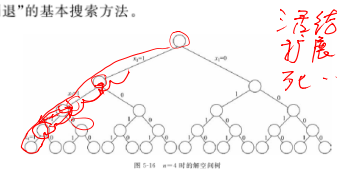
《计算复杂性理论》
第11讲 分支限界法(1)

山东师范大学信息科学与工程学院
段会川
2014年12月

目录

回溯法—概述

深度优先搜索是在明确给出了图中的各顶点及边(显式图)的情况下，按照深度优先搜索的思想对图中的每个顶点进行搜索，最终得出图的结构信息。回溯法是在仅给出初始结点、目标结点及产生子结点的条件(一般由问题题意隐含给出)的情况下，构造一个图(隐式图)，然后按照深度优先搜索的思想，在有关条件的约束下扩展到目标结点，从而找出问题的解。换言之，回溯法从初始状态出发，在隐式图中以深度优先的方式搜索问题的解。当发现不满足求解条件时，就回溯，尝试其他路径。通俗地讲，回溯法是一种"能进则进，进不了则换，换不了则退"的基本搜索方法。



王秋芬 5.3回溯法 P126

回溯法—算法框架及思想

回溯法是一种搜索方法。用回溯法解决问题时，首先应明确搜索范围，即问题所有可能解组成的范围。这个范围越小越好，且至少包含问题的一个(最优)解。为了定义搜索范围，需要明确以下几个方面：

(1) 问题解的形式：回溯法希望问题的解能够表示成一个 n 元组 $(x_1, x_2, \cdots, x_n)$ 的形式。

(2) 显约束：对分量 $x_i (i=1,2,\cdots,n)$ 的取值范围限定。

(3) 隐约束：为满足问题的解而对不同分量之间施加的约束。

(4) 解空间：对于问题的一个实例，解向量满足显约束的所有 n 元组构成了该实例的一个解空间。

注意：同一个问题的显约束可能有多种，相应解空间的大小就会不同，通常情况下，解空间越小，算法的搜索效率越高。

王秋芬 5.3回溯法 P126

回溯法—求解步骤

- 问题描述
- 问题分析
- 算法描述
- 步骤1：定义问题的解空间
- 步骤2：确定解空间的组织机构
- 步骤3：搜索解空间
  - 步骤3-1：设置约束条件
  - 步骤3-2：设置限界条件
  - 步骤3-3：执行搜索

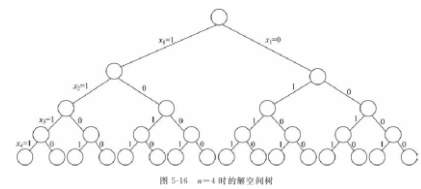
回溯法—结点类别

最后，搜索问题的解空间树。在搜索的过程中，需要了解几个名词：

(1) 扩展结点：一个正在生成孩子的结点称为扩展结点。

(2) 活结点：一个自身已生成但其孩子还没有全部生成的结点称为活结点。

(3) 死结点：一个所有孩子已经生成的结点称做死结点。



王秋芬 5.3回溯法 P128

回溯法—搜索思想

搜索思想：从根开始，以深度优先搜索的方式进行搜索。根结点是活结点并且是当前的扩展结点。在搜索过程中，当前的扩展结点沿纵深方向移向一个新结点，判断该新结点是否满足隐约束，如果满足，则新结点成为活结点，并且成为当前的扩展结点，继续深一层的搜索；如果不满足，则换到该新结点的兄弟结点（扩展结点的其他分支）继续搜索；如果新结点没有兄弟结点，或其兄弟结点已全部搜索完毕，则扩展结点成为死结点，搜索回溯到其父结点处继续进行。搜索过程直到找到问题的解或根结点变成死结点为止。

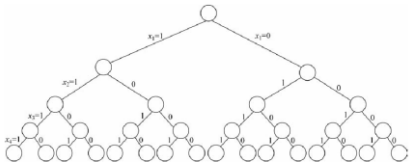


图 5-16　$n=4$ 时的解空间树

王秋芬 5.3回溯法 P128

回溯法—搜索思想

从回溯法的搜索思想可知，搜索开始之前必须确定问题的隐约束。隐约束一般是考察解空间结构中的结点是否有可能得到问题的可行解或最优解。如果不可能得到问题的可行解或最优解，就不用沿着该结点的分支继续搜索了，需要换到该结点的兄弟结点或回到上一层结点。也就是说，在深度优先搜索的过程中，不满足隐约束的分支被剪掉，只沿着满足隐约束的分支搜索问题的解，从而避免了无效搜索，加快了搜索速度。因此，隐约束又称为剪枝函数。隐约束（剪枝函数）一般有两种：一是判断是否能够得到可行解的隐约束，称之为约束条件（约束函数）；二是判断是否有可能得到最优解的隐约束，称之为限界条件（限界函数）。可见，回溯法是一种具有约束函数或限界函数的深度优先搜索方法。

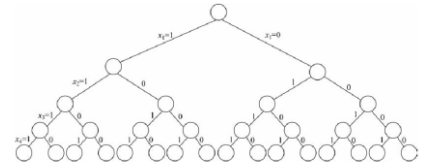


图 5-16　$n=4$ 时的解空间树

王秋芬 5.3回溯法 P128

回溯法—算法框架

总之，回溯法的算法框架主要包括三部分：

(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组织结构。

(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。

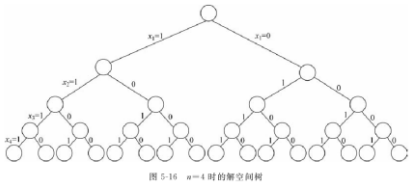


图 5-16　$n=4$ 时的解空间树

王秋芬 5.3回溯法 P128

目录

广度优先搜索—基本思想

给定图 $G=(V,E)$，它的初始状态是所有顶点均未被访问过，在图 G 中任选一个顶点 v 作为源点，则宽度优先搜索的思想为：先访问顶点 v，并将其标记为已访问过；然后从 v 出发，依次访问 v 的邻接点（孩子结点）$w_1,w_2,\cdots,w_t$，如果 $w_i(i=1,2,\cdots,t)$ 未访问过，则标记 w_i 为已访问过，将其插入到队列中；然后再依次从队列中取出 $w_1,w_2,\cdots,w_t$，访问它们的邻接点。依次类推，直到图中所有和源点 v 有路径相通的顶点均已访问过为止；若此时图 G 中仍然存在未被访问过的顶点，则另选一个尚未访问过的顶点作为新的源点，重复上述过程，直到图中所有顶点均已访问过为止。

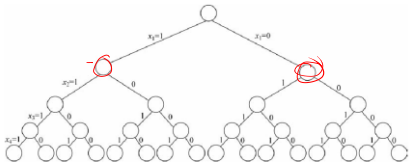


图 5-16　$n=4$ 时的解空间树

王秋芬 5.4 P169

广度优先搜索—示例

【例 5-13】　给定一个有向图，如图 5-55 所示，给出宽度优先搜索的一个序列。

(1) 问题分析。

根据宽度优先搜索的思想，初始状态是所有顶点均未被访问过，需要任选一个顶点作为源点，搜索过程中需要判断源点的邻接点是否被访问过，然后根据判断结果做不同的处理，目标状态为全部顶点已被访问过。为此，用一个标识数组 Visited[]标记图中的顶点是否被访问过，初始 Visited[]的值全部为 0。若某顶点已访问过，则将数组 Visited[]中的对应元素由 0 改为 1。

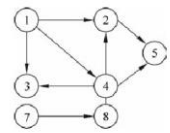


图 5-55　有向图

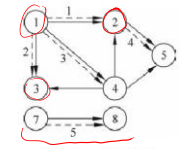


图 5-56　搜索顺序

王秋芬 5.4 P169

广度优先搜索—示例

（2）搜索过程。

假定选择顶点 1 为源点，Visited[1]=1，输出顶点 1，依次访问它的三个邻接点 2、3、4，它们均未被访问过，将其输出，标记 Visited[2～4]=1，并插入到队列中；从队列中取出顶点 2，它的邻接点 5 未被访问过，将其输出，标记 Visited[5]=1，并将顶点 5 插入到队列中；依次取出队列中的顶点，顶点 3 没有邻接点，顶点 4 的两个邻接点已访问过，顶点 5 没有邻接点。此时队列为空，图 5-55 中与顶点 1 相连通的顶点已访问完毕。再选择顶点 7 为源点，重复上述过程。搜索顺序如图 5-56 所示，图中虚线上的数据表示访问的次序。

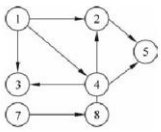


图 5-55　有向图

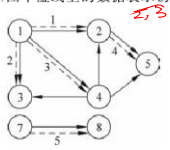


图 5-56　搜索顺序

王秋芬 5.4 P169

广度优先搜索—示例

【例 5-14】 给定一个无向图，如图 5-57 所示，给出宽度优先搜索的一个序列。

搜索过程如下：假定选择顶点 1 为源点，Visited[1]=1，输出顶点 1，依次访问它的两个邻接点 2、4，它们均未被访问过，将其输出，标记 Visited[2]=1，Visited[4]=1，并插入到队列中；从队列中取出顶点 2，它的邻接点 5 未被访问过，将其输出，标记 Visited[5]=1，并将 5 插入到队列中；从队列中取出顶点 4，它的邻接点 3 和 7 未被访问过，将其输出，标记 Visited[3]=1，Visited[7]=1，并将顶点 3 和 7 插入到队列中；依次取出队列中的顶点 5 和顶点 3，顶点 5 的邻接点已访问过，顶点 7 的邻接点 6 未被访问过，标记 Visited[6]=1，并将顶点 6 插入到队列中。取出队列中的顶点 6，它的邻接点已访问过，此时，队列为空，搜索结束。搜索顺序如图 5-58 所示。（注：图中虚线上的数据表示访问的次序。）

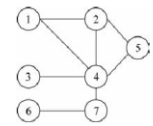


图 5-57　无向图

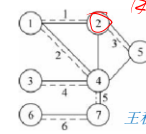


图 5-58　搜索顺序

王秋芬 5.4 P169-70

广度优先搜索—算法实现

```
bool Visited[n+1];                    //标记图中顶点有未
for(int i=1;i<=n;i++)
  Visited[i]=0;                       //用 0 表示顶点未被
void BFSV0 (int v0)                   //从 v0 开始广度优
{ int w;
  visit(v0); Visited[v0]=1;
  InitQueue(&Q);                      //初始化空队
  InsertQueue(&Q,v0);                 // v0 进队
  while (!Empty(Q))
  {
    DeleteQueue(&Q, &v);              //队头元素出队
    for(int i=1;i<=n;i++)             //依次访问 v 的邻接
    {
      if(g[v][i]!=0)  w=i;            //g 表示图的邻接矩
      if (!Visited(w))
      { visit(w);Visited[w]=1;InsertQueue(&Q, w);}
    }
  }
}
BFS()
{
  for(int i=1;i<=n;i++)
    if(Visited[i]==0)
      BFSV0(i);
}
```

王秋芬 5.4 P170

目录

- 回溯法回顾
- 广度优先搜索
- 分支限界法基本思想
- 0/1背包问题队列式分支限界算法
- 0/1背包问题优先队列式分支限界算法

分支限界法—基本思想

分支限界法类似于回溯法，也是一种在问题的解空间树中搜索问题解的算法，它常以宽度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。分支限界法首先将根结点加入活结点表（用于存放活结点的数据结构），接着从活结点表中取出根结点，使其成为当前扩展结点，一次性生成其所有孩子结点，判断孩子结点是舍弃还是保留，舍弃那些导致不可行解或导致非最优解的孩子结点，其余的被保留在活结点表中。再从活结点表中取出一个活结点作为当前扩展结点，重复上述扩展过程，一直持续到找到所需的解或活结点表为空时为止。由此可见，每一个活结点最多只有一次机会成为扩展结点。

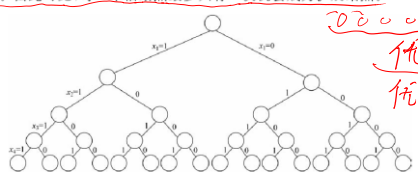


图 5-16　$n=4$ 时的解空间树

王秋芬 5.5 P171

分支限界法—基本思想

可见，分支限界法搜索过程的关键在于判断孩子结点是舍弃还是保留。因此，在搜索之前要设定孩子结点是舍弃还是保留的判断标准，这个判断标准与回溯法搜索过程中用到的约束条件和限界条件含义相同。活结点表的实现通常有两种方法：一是先进先出队列，二是优先级队列，它们对应的分支限界法分别称为队列式(FIFO)分支限界法和优先队列式分支限界法。

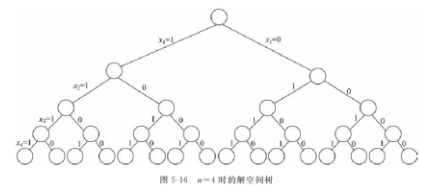


图 5-16　$n=4$ 时的解空间树

王秋芬 5.5 P171

分支限界法—基本思想

队列式分支限界法按照队列先进先出(FIFO)的原则选取下一个结点作为当前扩展结点。优先队列式分支限界法按照规定的优先级选取队列中优先级最高的结点作为当前扩展结点。优先队列一般用二叉堆来实现，最大堆实现最大优先队列，体现最大效益优先；最小堆实现最小优先队列，体现最小费用优先。

分支限界法的一般解题步骤为：

(1) 定义问题的解空间。

(2) 确定问题的解空间组织结构(树或图)。

(3) 搜索解空间。搜索前要定义判断标准(约束函数或限界函数)，如果选用优先队列式分支限界法，则必须确定优先级。

王秋芬 5.5 P171

目录

0/1背包问题的队列式分支限界法示例

定义约束条件为：$\sum_{i=1}^{n} w_i x_i \leqslant C$，限界条件为：cp+rp>bestp。其中，cp 表示当前已装入背包的物品总价值，初始值为 0；rp 表示剩余物品的总价值，初始值为所有物品的价值之和；bestp 表示当前最优解，初始值为 0，当 cp>bestp 时，更新 bestp 为 cp。

采用队列式分支限界法对该实例的搜索过程如图 5-59～图 5-62 所示。(注：图中深色结点表示死结点，已不在活结点表中；结点旁括号内的数据表示背包的剩余容量、已装入背包的物品价值。)

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P172

0/1背包问题的队列式分支限界法示例

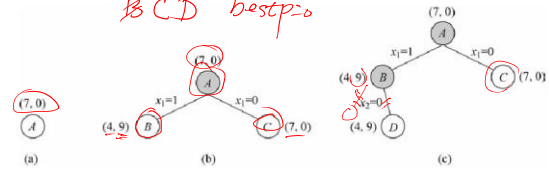


图 5-59 搜索过程 1

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P172

0/1背包问题的队列式分支限界法示例

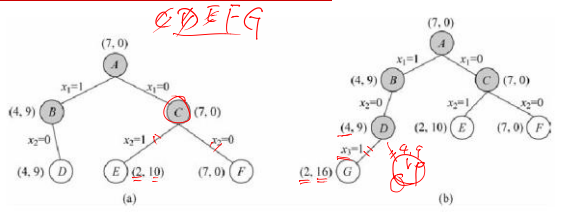


图 5-60 搜索过程 2

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P173

0/1背包问题的队列式分支限界法示例

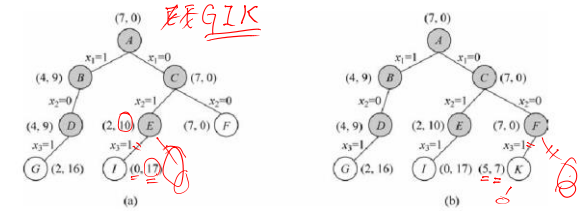


图 5-61 搜索过程 3

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P173

0/1背包问题的队列式分支限界法示例

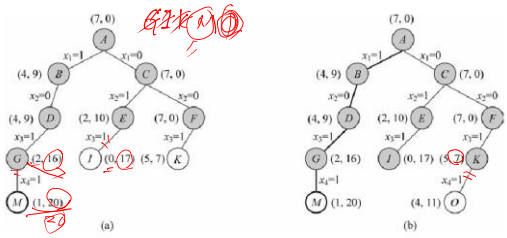


图 5-62　搜索过程 4

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P174

目录

0/1背包问题的优先队列式分支限界法示例

优先级定义为：活结点代表的部分解所描述的装入背包的物品价值上界，该价值上界越大，优先级越高。活结点的价值上界 up=活结点的 cp+剩余物品装满背包剩余容量的最大价值 r′p。

约束条件：同(1)中队列式分支限界法相同。限界条件：up=cp+r′p>bestp。

采用优先队列式分支限界法对上述实例的搜索过程如图 5-63、图 5-64 所示。

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P173

0/1背包问题的优先队列式分支限界法示例

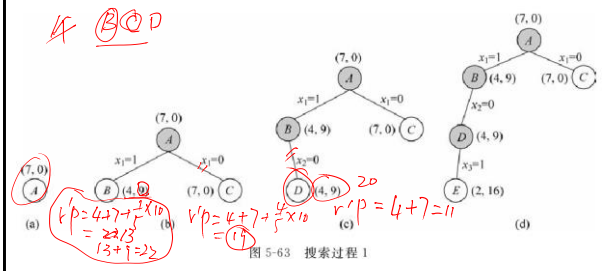


图 5-63　搜索过程 1

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P174

0/1背包问题的优先队列式分支限界法示例

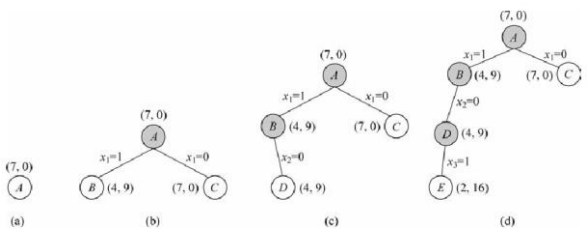


图 5-63　搜索过程 1

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P174

0/1背包问题的优先队列式分支限界法示例

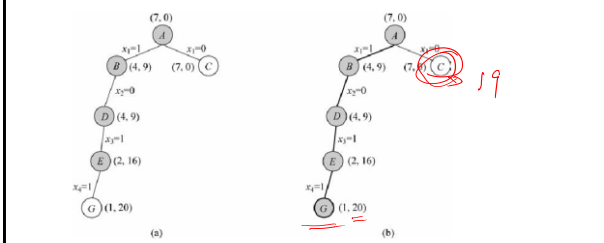


图 5-64　搜索过程 2

i	1	2	3	4
w_i	3	5	2	1
v_i	9	10	7	4

$W = 7$
$X^* = (1, 0, 1, 1)$
$W^* = 3 + 2 + 1 = 6$
$V^* = 9 + 7 + 4 = 20$

王秋芬 5.5 P175

0/1背包问题优先队列式分支限界算法实现

```
H = initH(1000); bestx = new int[n + 1];
int i = 1; E = 0; cw = cp = 0; int bestp = 0;          //当前最优值
int up = Bound(1);                                      //价值上界
while(i!= n + 1)                                        //搜索子集空间树
  {
    if(cw + w[i]<= c)                                   //检查当前扩展结点的左孩子结点
    {                                                   //左孩子结点为可行结点
      if(cp + p[i]> bestp)   bestp = cp + p[i];
      AddLiveNode(up, cp + p[i], cw + w[i], true, i + 1);
    }
  up = Bound(i + 1);
  if(up> bestp)  AddLiveNode(up, cp, cw, false, i + 1); //检查当前扩展结点的右孩子结点
  //取下一扩展结点
    HeapNode N;
    N = DeleteMaxH(H);
    E = N.ptr; cw = N.weight; cp = N.profit; up = N.uprofit;
    i = N.level;
  }                                                     //end while
for(i = n;i > 0;i -- )
  { bestx[i] = E -> lChild; E = E -> parent; }
return cp;
```

王秋芬 5.5 P178

0/1背包问题优先队列式分支限界算法实现

```
int Knap::Bound( int i)                                 //计算上界
{
    int cleft = c - cw;                                 //剩余容量
    int b = cp;
    //以物品单位重量价值递减序装入物品
    while(i <= n && w[i]<= cleft)
      {
        cleft -  = w[i];
        b +  = p[i];i++;}
    //装满背包
    if(i <= n)
      b +  = p[i] * w[i] * cleft;
    return b;
}
```

王秋芬 5.3.2 P141

目录

- 回溯法回顾
- 广度优先搜索
- 分支限界法基本思想
- 0/1背包问题队列式分支限界算法
- 0/1背包问题优先队列式分支限界算法