算法设计与分析

第 6 章 分支限界法

金英 编写

计算机科学技术学院
软件学院

2021 年 4 月

目　录

第 6 章 分支限界法 (Branch-and-Bound)

分支限界法类似于回溯法，也是一种在问题的解空间树 T 上搜索解的算法。但分支限界法与回溯法有不同的求解目标。

回溯法的求解目标是找出 T 中满足约束条件的所有解，或任意一个解。

分支限界法的求解目标则是找出 T 中使得某一目标函数值达到极小或极大的解，即问题在某种意义下的最优解。

由于求解目标不同，导致分支限界法与回溯法对解空间树 T 的搜索方式有两点不同：

⑴ 回溯法以深度优先的方式搜索解空间树 T; 分支限界法以广度优先或以最小耗费（最大效益）优先的方式搜索解空间树 T。

⑵ 回溯法一般只通过约束条件，而分支限界法不仅通过约束条件，而且通过目标函数的限界来减少无效搜索，提高求解效率。

6.1 分支限界法概述

1. 基本思想

分支限界法的搜索策略是，在扩展结点处，先生成其所有的儿子结点（分支）（使其变为活结点），然后再从当前的活结点表中选择下一个扩展结点。为了有效地选择下一个扩展结点，以加速搜索的进程，在每一活结点处，计算一个函数值（限界），并根据这些已计算出的函数值（作为优先级），从当前活结点表中选择一个最有利的结点作为扩展结点，使搜索朝着解空间树上有最优解的分支推进，以便尽快地找出一个最优解。

2. 关键问题

采用分支限界法求解的 3 个关键问题如下：

⑴ 如何确定合适的限界函数。

⑵ 如何组织待处理结点的活结点表。

⑶ 如何确定解向量的各个分量。

3. 设计合适的限界函数

在搜索解空间树时，每个活结点可能有很多孩子结点（子树的根），其中有些子树搜索下去是不可能产生问题解或最优解的。可以设计好的限界函数在扩展时删除这些不

必要的孩子结点，从而提高搜索效率。

限界函数设计难以找出通用的方法，需根据具体问题来分析。一般地，先要确定问题解的特性：

⑴ 目标函数是求最大值：则设计上界限界函数 ub（根结点的 ub 值通常大于或等于最优解的 ub 值），若 s_i 是 s_j 的双亲结点，应满足 $ub(s_i) \geq ub(s_j)$，当找到一个可行解 $ub(s_k)$ 后，将所有小于 $ub(s_k)$的结点剪枝。

⑵ 目标函数是求最小值：则设计下界限界函数 lb（根结点的 lb 值一定要小于或等于最优解的 lb 值），若 s_i 是 s_j 的双亲结点，应满足 $lb(s_i) \leq lb(s_j)$，当找到一个可行解 $lb(s_k)$ 后，将所有大于 $lb(s_k)$的结点剪枝。

4. 组织活结点表

从活结点表选择下一个扩展结点的原则体现在对活结点表的组织方式（即，数据结构）之中。常见的活结点表的组织方式有以下二种：

① 队列式 (FIFO)

② 优先队列式 (PQ, Prioroty Queue)（最小优先队列和最大优先队列）

其中的优先队列可用堆实现。

所以，常见的分支限界法分为：

① 队列式（FIFO）分支限界法

② 优先队列式（PQ）分支限界法

⑴ 队列式分支限界法

队列式分支限界法将活结点表组织成一个队列，并按照队列先进先出（FIFO）原则选取下一个结点为扩展结点。步骤如下：

① 将根结点加入活结点队列。

② 从活结点队列中取出队头结点，作为当前扩展结点。

③ 对当前扩展结点，先从左到右地生成它的所有孩子结点，用约束条件或限界函数检查，把所有满足约束条件或限界函数的孩子结点加入活结点队列。

④ 重复步骤②和③，直到找到一个解或活结点队列为空为止。

⑵ 优先队列式分支限界法

优先队列式分支限界法的主要特点是将活结点表组组成一个优先队列（用堆实现），并选取优先级最高的活结点成为当前扩展结点。步骤如下：

① 计算起始结点（根结点）的优先级（与特定问题相关的信息的函数值决定优先级。一般，优先级以限界函数值为基础）并加入优先队列。

② 从优先队列中取出优先级最高的结点作为当前扩展结点，使搜索朝着解空间树上可能有最优解的分支推进，以便尽快地找出一个最优解。

③ 对当前扩展结点，先从左到右地生成它的所有孩子结点，然后用约束条件或限界函数检查，对所有满足约束条件或限界函数的孩子结点计算优先级并加入优先队列。

④ 重复步骤②和③，直到找到一个解或优先队列为空为止。

5. 确定最优解的解向量

分支限界法在搜索解空间树时，结点的处理是跳跃式的，回溯也不是单纯地沿着双亲结点一层一层地向上回溯，因此当搜索到某个叶子结点且该结点对应一个可行解时，如何得到对应的解向量呢？两种方法：

⑴ 对每个扩展结点保存从根结点到该结点的路径。每个结点带有一个可能的解向量。这种做法比较浪费空间，但实现起来简单，后面的示例均采用这种方式。

⑵ 在搜索过程中构建搜索经过的树结构。每个结点带有一个双亲结点指针，当找到最优解时，通过双亲指针找到对应的最优解向量。这种做法需保存搜索经过的树结构，每个结点增加一个指向双亲结点的指针。

6. 分支限界法的时间性能

一般情况下，在问题的解向量 $X=(x_1, x_2, \ldots, x_n)$ 中，分量 x_i（$1\le i\le n$）的取值范围为某个有限集合 $S_i=(s_{i_1}, s_{i_2}, ..., s_{i_r})$。问题的解空间由笛卡尔积 $S_1\times S_2\times\ldots\times S_n$ 构成，那么

⑴ 第 1 层根结点有$|S_1|$棵子树

⑵ 第 2 层有$|S_1|$个结点，第 2 层的每个结点有$|S_2|$棵子树，第 3 层有$|S_1|\times|S_2|$个结点

⑶ …

⑷ 第 n+1 层有$|S_1|\times|S_2|\times\ldots\times|S_n|$个结点，它们都是叶子结点，代表问题的所有可能解

在最坏情况下，时间复杂性是指数阶。

7. 分支限界法与回溯法的主要区别

分支限界法与回溯法的主要区别如表 6-1 所示。

6.2 单源最短路径问题

单源最短路径问题就是在给定的网络（要求边权是正数）中求源点到所有其他顶点

的最短路径。

表 6-1 分支限界法与回溯法的主要区别

方法	解空间搜索方式	存储活结点的数据结构	活结点存储特性	常用应用
回溯法	深度优先	栈	活结点的所有可行子结点被遍历后才从栈中出栈（每个活结点可能多次成为扩展结点）	找出满足条件的所有解
分枝限界法	广度优先	队列，优先队列	每个结点只有一次成为活结点的机会（每个活结点也只有一次成为扩展结点的机会）	找出满足条件的一个解或者特定意义的最优解

例：设有一个无向连通网络如图 6-1 所示，并设源点为顶点 V_1，求源点到其他顶点的最短路径。

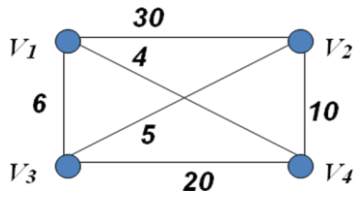

图 6-1 一个无向连通网络

1. 用队列式分支限界法求解

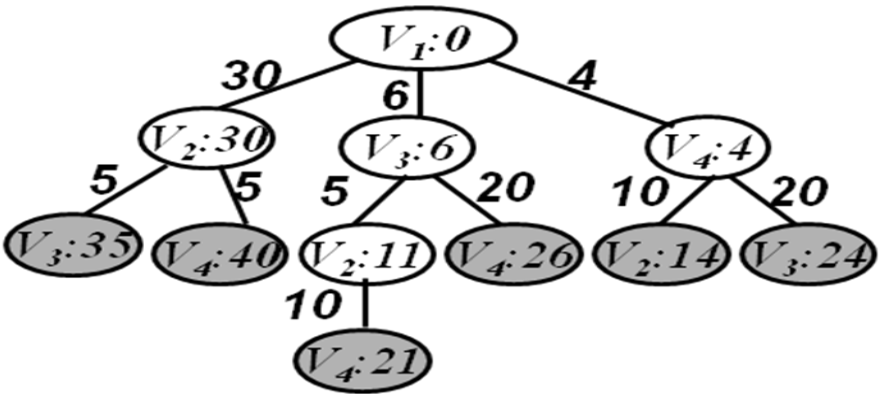

图 6-2 解空间树及其搜索停止状态

在图 6-2 所示的解空间树中，灰色结点是死结点。得到的结果：

最短路径	最短路径长度
v1—>v1	0
v1—>v3—>v2	11

v1—> v3　　6

v1—> v4　　4

按队列式分支限界法求解单源最短路径的算法可描述如下：

```
SSShortestPaths(int v)
{   E.i=v;            // 建立源点结点 E（根结点），E.i 为顶点编号，v 为源点
    E.length=0;    // E.length 记录根到扩展结点的路径长度
    dist[v]=0;      //dist[i]源点 v 到扩展结点 i 的当前最短路径长度（初值皆为 inf）
    INSERT(Q, E);          // Q 是一个队列，源点结点 E 进队
    while(!Empty(Q))       //队列不空循环
    {    E=DELETE(Q);  //出队列结点 E
        if(dist[E.i]<E.length)   continue;
         //若扩展结点对应顶点 E.i 的原来最短路径长度，
         //大于最近发现（对应的活结点后入队）的最短路径长度，
         //则剪枝
        for(j=1; j<=n; j++)
            if ((c[E.i][j]<inf)&&(E.length+c[E.i][j]<dist[j])) //图采用邻接矩阵 c 表示
                // 顶点 E.i 到顶点 j 有边并且新路径长度更短，则更新信息
                {   dist[j]=E.length+c[E.i][j];
                    prev[j]=E.i;          //前趋顶点
                    N.i=j;                //生成相邻顶点 j 的活结点 N
                    N.length=dist[j];
                    INSERT(Q,N);     //结点 N 进队
                }
    }
}
```

2. 用优先队列式分支限界法求解

解单源最短路径问题的优先队列式分支限界法用一小根堆（即为优先队列）来存储活结点表，其优先级是结点所对应的当前最短路长。在算法中，还利用结点间的控制关系进行剪枝。在一般情况下，如果解空间树中以结点 y 为根的子树中所包含的解优于以

结点 x 为根的子树中所包含的解，则结点 y 控制了结点 x，以被控制的结点 x 为根的子树可以剪去。

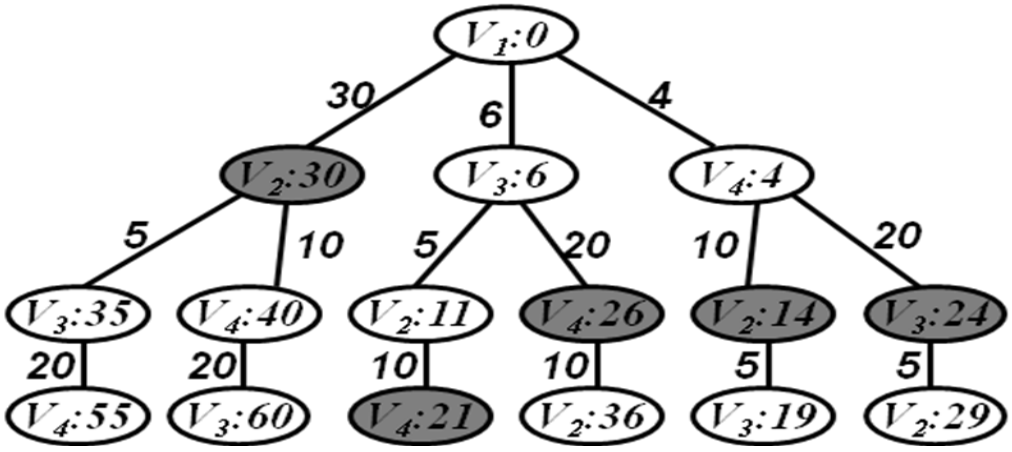

图 6-3 解空间树及其搜索停止状态

在图 6-3 所示的解空间树中，灰色结点是死结点（即被剪掉的结点）。

按优先队列式分支限界法求解单源最短路径问题的算法可描述如下：

```
template <class Type>
void graph <Type> :: ShortestPaths(int v)
{   MinHeap <MinHeapNode <Type> > H(1000);   //小根堆的容量为 1000
    MinHeapNode <Type> E;              //生成活结点
    E.i=v;   E.length=0;     dist[v]=0;   //定义源点为初始扩展结点
    //搜索问题的解空间
    while(true)
    { if(dist[E.i]>E.length)
            for(j=1; j<=n; j++)
                if ((c[E.i][j]<inf) &&(E.length+c[E.i][j]<dist[j]))
                {     //顶点 E.i 到顶点 j 可达，且满足控制约束
                    dist[j]=E.length+c[E.i][j];   prev[j]=E.i;
                    MinHeapNode <Type> N;          //生成活结点
                    N.i=j;     N.length=dist[j];
                    H.Insert(N);        //加入活结点优先队列
                }
```

```
            try { H.DeleteMin(E);}              //取下一个活结点
            catch (OutOfBounds) {break;}        //优先队列空
        }
    }
```

6.3 0-1 背包问题

0-1 背包问题是给定 n 种物品和一个背包。物品 i 的重量是 w_i，价值为 v_i，背包的容量为 c。如何选择物品装入背包，使得装入背包中的物品的总价值最大？对每种物品 i 只有两种选择，要么装入，要么不装入，不能将物品 i 装入背包多次，也不能只装入物品 i 的一部分，要求最终装包的总价值最高。

假设各物品依次按单位重量价值从大到小排序，相应的排序过程可在算法的预处理部分完成。在解 0-1 背包问题的优先队列式分支限界算法中，活结点优先队列中结点的优先级由该结点的上界函数 Bound 计算出的值 uprofit 给出。可用大根堆实现活结点优先队列。

在下面给出的解 0-1 背包问题的优先队列式分支限界算法 MaxKnapsack( )中用到的计算当前扩展结点的右子树中可能获得的最优值上界的 Bound()函数，在其执行之前需要进行预处理，将物品依其单位重量价值从大到小排序。然后，Bound()函数是按照单位价值递减的次序依次将物品装包，直到装不下时，再按照将当前物品（即剩余物品中单位价值最高的物品）的一部分装入背包使得背包装满的假设来计算获得的价值的，由此得到的价值是当前扩展结点右孩子对应分支可能获得价值的上界。

按优先队列式分支限界法求解 0-1 背包问题的算法可描述如下：

```
typedef    struct bbnode
{
    struct bbnode *parent;    // 记解空间树中的双亲
    int lchild;               // 记当前结点是否为其双亲的左孩子，1:左孩子；0：右孩子
} bbnode; // 解空间树中结点格式
typedef    struct
{   double uprofit;       // 当前结点对应分支的可能获得最高价值的上界
    double profit;        // 到当前结点的累计装包价值
    double weight;        // 到当前结点的累计装包重量
    int level;            // 当前结点在解空间树中所处的层号
    bbnode *ptr;          // 当前结点地址
}   HeapNode;            // 堆（优先队列）中结点类型

#define MAXSIZE 1000
```

```
#define MaxHeap PQNode data[MAXSIZE]   // 定义大根堆类型，用于存在活结点
bestx = new int[n+1];     // bestx 数组表示当前最优解，算法结束时记的是真正最优解
double bestp = 0;          // bestp 表示当前最优值，其初值为 0，算法结束时记的是最优值
double Bound (int i)     // 计算第 i 层右孩子结点对应分支可能获得价值的上界
{
    cleft = c-cw;           // 背包剩余容量
    b = cp;                 // 当前装包总价值
    // 以物品单位重量价值递减序将物品装入背包
    for(j=i；j<=n && w[i] <= cleft；j++)
    {
        cleft - = w[i];
        b+ = p[i];
        i++;
    }
    // 当不能将第 i 个物品全部装入背包时，按照装满背包计算可能获得价值的上界
    if（i <= n） b += p[i] / w[i] * cleft;
    return b;
}
AddLiveNode(MaxHeap H,double up,double cp,double cw, int child, int level, bbnode *E)
{    //构造解空间树中结点*b, 将来用于构造最优解；并构造活结点 N 插入大根堆 H 中
     bbnode *b = new bbnode;
     b->parent = E;
     b->lchild = child;
     HeapNode    N;
     N.upprofit = up;
     N.profit = cp;
     N.weight = cw;
     N.level = level;
     N.ptr = b;
     InsMaxHeap(H, N); //插入大根堆
}
double MaxKnapsack( )   // 0-1 背包问题的优先队列式分支限界算法，它也是 A*算法
{
    MaxHeap H;       // 定义一个大根堆 H
    double cp=0;     // cw 表示到当前扩展结点累计装包重量，初值为 0
    double cw=0;     // cp 表示到当前扩展结点累计装包价值，初值为 0
    InitMaxHeap(H); // 初始化大根堆 H
    up = Bound(1);   // 计算最优值上界
    bbnode *E = NULL;     // 记当前结点地址，用于构造最优解
    int i =1;          //   从解空间树第 1 层的根开始搜索
    while (i != n+1)        // 判断是否搜索到解空间树的叶子层
    {
        if (cw + w[i]<=c) // 左孩子为可行结点
```

```
        {
            if (cp + p[i] > bestp)
                bestp = cp+p[i] ;
            AddLiveNode(H, up, cp+p[i], cw += w[i], 1, i+1, E);    // 将活结点放入堆 H 中
        }
        up = Boud(i+1); //从当前扩展结点进右孩子之前，计算右分支可能获得价值上界
        if (up > bestp)    // 右子树可能含有最优解, 所以让右孩子成活结点，放入堆 H 中
            AddLiveNode(H,up, cp, cw, 0, i+1, E);    // 将活结点放入堆 H 中
        HeapNode N;
        DeleteMaxHeap(H, N); //删除堆中首元素，将删除的元素赋给 N
        E = N. ptr;
        up = N. uprofit;
        cw = N. weight;
        cp = N. profit;
        i = N.level;
    }
    // 构造最优解
    for(j=n；j > 0;    j--)
    {
        bestx[j] = E->lchild;
        E = E->parent;
    }
    return cp;    // 返回最优值
}
```

例： n=3 时的 0-1 背包问题，w={16，15，15}，p={45，25，25}，c=30。在其解空间树上搜索最优解的过程如图 6-4 所示。

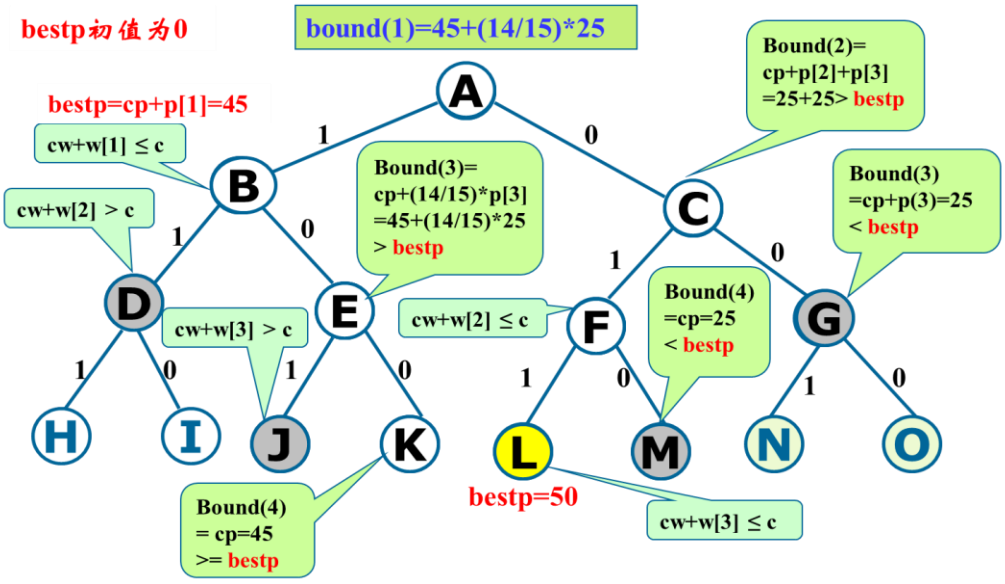

图 6-4　0-1 背包问题的解空间树及其搜索过程

在图 6-4 所示的解空间树中，灰色结点是死结点（即被剪掉的结点），从解空间树的根结点到叶子结点 L 的路径对应最优解{0, 1, 1}，即本例的最优解是将物品 2 和物品 3 装包。在图 6-4 中凡是在搜索过程中计算过 Bound(i)、判断过 cw +w[i] ≤ c、修改过 bestp 的地方都做了标注。

上述求解 0-1 背包问题的基于优先队列的分支限界算法 MaxKnapsack( )，使用 Best-first 策略求解优化问题（使用优先队列存放活结点），并且使用最优值的上界函数 Bound(i)作为代价估计函数，所以它事实上也是一个 A*算法，因而该算法一旦得到一个解（在解空间树中搜索到叶子），那一定就是最优解！所以 MaxKnapsack( ) 算法第一次搜索到叶子后就结束了。

算法复杂性分析：无论采用队列式分支限界法还是优先队列式分支限界法求解 0-1 背包问题，最坏情况下要搜索整个解空间树，所以最坏时间和空间复杂度均为 $O(2^n)$，其中 n 为物品个数。

本章习题

1. 设有 3 件工作分配给 3 个人，将工作 i 分配给第 j 个人所花的费用为 C_{ij}，现将为每一个人都分配 1 件不同的工作，并使总费用达到最小。设：

$$C=\begin{pmatrix}2 & 3 & 1\\ 1 & 2 & 3\\ 3 & 2 & 1\end{pmatrix}$$

要求：

（1）画出该问题的解空间树；

（2）写出该问题的剪枝策略(即限界条件)，要求只保留第一个最优解；

（3） 按优先队列式分支限界法搜索解空间树，并用剪枝策略对解空间树中该剪枝的位置打×，并给出最优解及最优值。

2. 给定背包容量 C = 20，6 个物品的重量分别为(5, 3, 2, 10, 4, 2)，价值分别为(11,8,15,18,12,6)。画出分治限界法求解 0/1 背包问题的搜索空间。

3. 画出按优先队列式分支限界法求解 8 迷问题。要求：

（1）确定代价函数；

（2）针对下面的例子，画出搜索空间。

8 迷问题描述：具有 8 个编号小方块的魔方（九宫格中有 8 个数字），要求从初始状态经过一系列变换，变换成指定的目标状态。规定每步只能移动一个数字到相邻空格中（不能走斜线）。例：设具有 8 个编号小方块的魔方初始状态如图 6-5 所示。要求经过一系列移动，从初始状态变换到的指定的目标状态（如图 6-6 所示）。

2	3	
5	1	4
6	8	7

图 6.5 初始状态

1	2	3
8		4
7	6	5

图 6-6 目标状态

4. 设计算法实现优先队列（用堆实现）的删除操作。
5. 设计算法实现优先队列（用堆实现）的插入操作。

参考书

1. 王晓东. 计算机算法设计与分析 (第 5 版)[M]. 北京:电子工业出版社，2018.
2. Thomas H.Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms (Second Edition)[M]. The MIT Press, 2013.