DOI:10.13203/j.whugis1995.01.017
第20卷第1期 武汉测绘科技大学学报 Vol.20 No.1
1995年3月 Journal of Wuhan Technical University of Surveying and Mapping March 1995

0-1背包问题的多重分枝-限界算法

李鸣山 郑海虹
(武汉测绘科技大学计算机科学与工程系,武汉市珞喻路39号,430070)

摘 要 建立了0-1背包问题数学模型的一般形式,对通常的分枝-限界算法作了推广,给出了多重分枝-限界算法,有效地解决了具有多个背包的0-1背包问题;也可用于解决某些具有"多重"性质的0-1规划问题。

关键词 整数规划;0-1背包;分枝-限界算法
分类号 O211.65;TP312PA

1 0-1背包问题的多重分枝-限界算法

1.1 数学模型

多背包的0-1背包问题的数学模型如下:

设有l个背包,它们能容纳的重量分别为$M_1,M_2,\cdots,M_l$。有n个物品,称为物品1,2,$\cdots$,n,它们的重量分别为$w_1,w_2,\cdots,w_n$,将它们装入背包所取得的效益分别为$p_1,p_2,\cdots,p_n$。变量$x_{ij}=1$表示将物品i装入背包j,$x_{ij}=0$表示物品i不装入背包j。目标函数为:

$$\sum_{j=1}^{l}\sum_{i=1}^{n}x_{ij}p_i$$

约束条件为:

$$x_{ij}\in\{0,1\},\quad \sum_{j=1}^{l}x_{ij}\leqslant 1,\quad \sum_{i=1}^{n}x_{ij}w_i\leqslant M_j$$

$$1\leqslant i\leqslant n,\quad 1\leqslant j\leqslant l$$

据此确定使目标函数取最大值的x_{ij}值。

1.2 算法的基本思想

下面结合实例说明多重分枝-限界算法的基本思想。

设有两个背包,分别称为背包1和背包2,它们能容纳的重量分别为$M_1=35,M_2=37$。今有4个物品,其重量分别为$w_1=6,w_2=20,w_3=30,w_4=13$;各物品放入背包所获得的效益分别为$p_1=9,p_2=25,p_3=21,p_4=8$。需将物品装入两个背包(对每个物品不允许部分装入背包),使取得的效益最大。

对应每个背包,各有一棵状态空间树,树的结点代表物品装入背包的状况(与本实例对应的状态空间树见图1和图2),两棵树的高度都是n(n是物品个数),树中结点按宽度优先原则生成。将两棵树中任意一棵确定为主树,则另一棵称为副树。只有主树的叶结点才可能是解状态,而答案状态是解状态中使目标函数取最大值的叶结点。设对应背包1的状态树是主树,对应背包2的状态树是副树,并假定n个物品已按单位重量效益的非增次序排序:

$$p_i/w_i\geqslant p_{i+1}/w_{i+1},\qquad 1\leqslant i\leqslant n-1$$

(a)从主树的根结点开始,它是当前被扩展结点(称为E结点)。计算根结点处效益的下界

收稿日期:1994-09-23.李鸣山,男,58岁,副教授,现从事人工智能、算法理论研究。

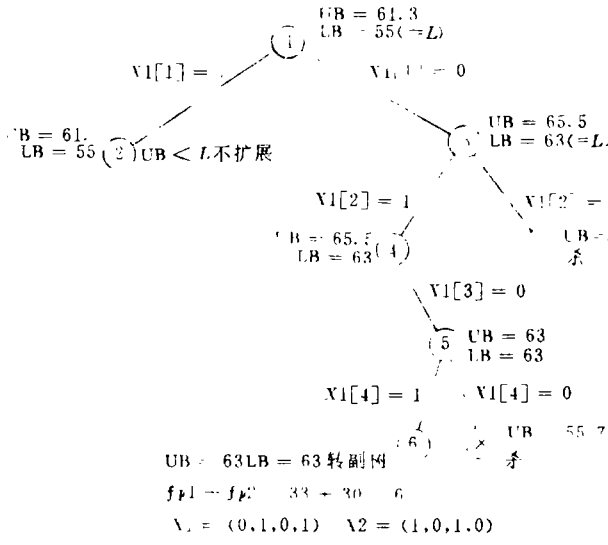

图 1 主树

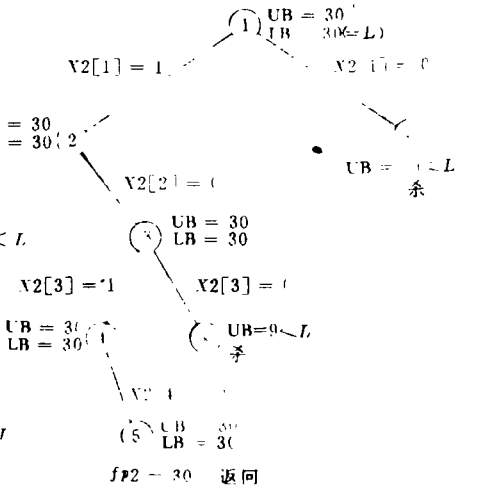

图 2 副树

LBB1 和上界 UBB1，计算的方法在(e)中说明。根结点处 LBB1 的值作为计算全过程的效益下界 L 的初值。

(b)生成根结点的两个子结点——左子结点和右子结点，它们分别表示将物品 1 放入背包 1 和不放入背包 1。若物品 1 可以放入背包 1(左子结点可行)，则实际生成左子结点，并将它加入活结点表，它的 LBB1 和 UBB1 值与父结点(即根结点)相同；否则不生成左子结点。对右子结点，它总是可行的。先计算它的 LBB1 和 UBB1 值，如果效益上界 UBB1$<L$，则扩展右子结点不可能导致最优解，它立即被杀死；否则将右子结点加入活结点表，并以 L 与 LBB1 中较大者作为新的 L 值。

(c)在活结点表中选取 UBB1 值最大的结点作为 E 结点，对它的两个子结点的处理与(b)相同。而刚才作为 E 结点的活结点则从活结点表中删除，因为它的子结点都已生成，对找最优解来说，它已没有存在的必要了。

(d)上述(c)反复进行。当扩展到一个叶结点，计算该叶结点处的 LBB1 和 UBB1 值。如果 UBB1$<L$，则杀死该叶结点；否则转到副树搜索，求得对于背包 2 的最优解。对副树的搜索与通常的分枝-限界算法完全一样，只需注意已装入背包 1 的物品不再考虑。主树的效益(即背包 1 的效益)与副树的效益(即背包 2 的效益)之和，即是一个完整的解对应的效益。

当活结点表为空，则效益最大的解就是最优解。

(e)关于 LBB1 和 UBB1 的计算。LBB1 由 LB1 和 LB2 两部分组成。LB1 是已装入背包 1 的物品所取得的效益与还可以整个装入背包 1 的物品装入背包 1 所能取得的效益之和，LB2 是剩余物品中可以整个装入背包 2 的物品装入背包 2 所能取得的效益之和。显然，LB1＋LB2 是物品装入两个背包所能取得的最大效益的一个下界。

UBB1 也由 UB1 和 UB2 两部分组成。UB1 是已装入背包 1 的物品所取得的效益和当前可以依次连续整个装入背包 1 的物品(设最后一个是物品 i)装入背包 1 所能取得的效益以及物品 $i+1$ 部分装入背包 1 所能取得的效益 c_1p_{i+1}/w_{i+1} 之和，其中 c_1 是背包 1 的剩余重量。UB2 是剩余物品中可以依次连续整个装入背包 2 的物品(设最后一个是物品 j)装入背包 2 所能取得的效益与物品 j 后面只能部分装入背包 2 的物品(设为物品 k)装入背包 2 所能取得的效益 c_2p_k/w_k 之和，其中 c_2 是背包 2 的剩余重量。在计算 UB1 和 UB2 时之所以要考虑物品部分装入

背包所能取得的效益，是因为它们都是效益的上界，并注意到物品已按单位重量的效益的非增次序排列。为使效益上界的计算较为准确，从而杀死尽可能多的结点而又不致于丢失最优解，还有一些细节需要考虑。例如当 $k=i+1$ 时，UB1 和 UB2 中都含有物品 $i+1$ 部分装入背包的效益。但若 $k=i+1=n-1$，此时物品 $n-1$ 只可能部分装入背包 1 或背包 2，则最好的情况是最后一个物品 n 可装入背包 1 或背包 2，因此计算 UB1 和 UB2 时只考虑 c_1p_{n-1}/w_{n-1} 和 c_2p_{n-1}/w_{n-1} 中较大者；若 $k=i+1=n$，则最后一个物品 n 可部分装入背包 1 或背包 2，其效益不计入 UB1 和 UB2。因为我们考虑的是 0-1 背包问题，其它一些只需作简单处理的细节不再赘述。

在图 1 和图 2 中，结点的编号是活结点表中结点生成的顺序号，⊗表示该结点不可行或因 UBB1$<L$ 而被杀死未加入活结点表，LB 和 UB 表示 LBB1 和 UBB1(对图 1)或 LBB2 和 UBB2(对图 2)。对图 1 的根结点 1，

$$UB1 = 9 + 25 + (35 - 6 - 20) \times 21/30 = 34 + 6.3$$

$$UB2 = 21 + (37 - 30) \times 8/13 = 21 + 4.31$$

因为 $6.3>4.31$，所以只取 6.3，于是，

$$UBB1=34+6.3+21=61.3,\quad LB1=9+25=34,\quad LB2=21,\quad LBB1=34+21=55$$

L 的初值也是 55。由结点 1 扩展结点 2 和 3，此时活结点表中有结点 2 和 3 两个活结点。结点 2 是结点 1 的左子结点，它的 UBB1 和 LBB1 值与父结点相同，分别为 61.3 和 55。结点 3 的 UBB1 值是 65.5，大于 61.3，所以扩展结点 3。结点 3 的 LBB1 值是 63，大于 L 的值 55，于是以 63 作为新的 L 值。结点 3 的右子结点的 UBB1 值是 55，小于 L 值，它立即被杀死，不加入活结点表。如此继续进行，当扩展到叶结点 6，此时得到关于背包 1 的效益 25+8=33。结点 6 的 UBB1 值是 63，不小于 L(此时 L 的值仍是 63)，于是转到副树按通常的分枝-限界算法搜索，得到关于背包 2 的效益 30。两个背包的效益之和是 63，这就是最优解对应的效益。虽然此时活结点表中还有结点 2，但它的 UBB1$=61.3<L$，不再扩展，于是搜索过程结束。图 1 和图 2 若是高为 4 的满二元树，共有 62 个结点，而我们只扩展了 11 个结点，所以算法的效率是很高的。

1.3　算法

我们用一种类 PASCAL 语言描述具体算法，其中符号的含义是明显的，主程序 KNAP1 用于搜索主树，过程 KNAP2 用于搜索副树，过程 LUBOUND1 和过程 LUBOUND2 分别用于计算主树和副树结点处的 LB1、UB1 和 LB2、UB2 值。算法中略去对有关数据的定义和说明及某些实现细节，尚有个别简单的过程只给出功能说明，未给出具体算法。

算法中各个变量的含义如下：

M_1、M_2 为背包 1 和背包 2 可容纳的重量，n 为物品数量，$w[i]$为物品 i 的重量，$p[i]$为物品 i 装入背包获得的效益。$k1$、$k2$ 为主树和副树的层数。$fp1$、$fp2$ 为背包 1 和背包 2 取得的效益，$cw1$、$cw2$ 为背包 1 和背包 2 剩余的重量。PARENT、LEVEL、TAG、RW、ALB、UBB 为每个结点的 6 个信息，依次是父结点、层次、标志(1 表示左子结点，0 表示右子结点)、剩余重量、已取得效益、效益上界。E1 为主树正在扩展的结点，ANS1 为主树的叶结点。

```
program KNAP1
1 初始化活结点表
2 E1←根结点
3 PARENT1(E1)←0;LEVEL(E1)←0;RW(E1)←M1;ALB1(E1)←0
4 call LUBOUND1 (p,w,M1,0,n,0,LB1,UB1)
```

```
5 call LUBOUND2 (p,w,M2,0,n,0,LB2,UB2)
6 LBB1←LB1+LB2;L←LBB1−e;
7 UBB1(E1)←UB1+UB2
8 loop
9   k1←LEVEL1(E1);fp1←ALB1(E1);cw1←RW1(E1)
10  case
11    :k1=n;//叶结点//
12    call KNAP2
13    if fp1+fp2≥L then L←fp1+fp1;ANS1←E1 endif
14    :else:;//E 有两个儿子//
15    if w[k1]≤cw1 then//左儿子可行//
16      call NEWNODE (E,k1+1,1,cw1−w(k1),fp1+p(k1),UBB1(E1) //左儿子加入活结点表//
17    endif
      //考察右儿子是否可能是活结点//
18    call LUBOUND1(p,w,cw1,fp1,n,k1+1,LB1,UB1)
19    call LUBOUND2(p,w,M2,0,n,0,LB2,UB2)
20    UBB1←UB1+UB2
21    if UBB1≥L then //左儿子会活//
22      call NEWNODE(E1,k1+1,0,cw1,fp1,UBB1)
          //右儿子加入活结点表//
23      L←max(L,LB1+LB2)
24    endif
25  endcase
26  if 活结点表为空 then exit endif
27  call LARGEST(E1)//取活结点表中 UBB1 值最大的结点作为下一个 E 结点//
28  until UBB1(E1)<L,repeat
29  call FINISH(L,ANS1,n)//输出结果//
30 end KNAP1
```

过程 KNAP2 与通常的 0-1 背包问题分枝-限界算法基本相同，仅需注意在判断 E 结点的左子结点是否可行时，除了判断背包 2 的剩余重量 $cw2$ 是否大于等于第 $k2$ 个物品的重量 $w[k2]$外，还要判断物品 $k2$ 是否已装入背包 1。因为树中每个结点的信息中含有父结点信息及它是左或右子结点的标志，因此当由主树的叶结点调用 KNAP2(算法第 12 行)时，由叶结点的信息容易确定哪些物品已装入背包 1。

```
    procedure LUBOUND1(p,w,cw,cp,n,k,LB1,UB1)//cw 是背包 1 剩余重量,cp 是背包 1 已取得的效
益。考虑物品 k,…,n。LB1 和 UB1 是本过程计算得到的效益的下界和上界//
1 LB1←cp;c←cw
2   for i←k to n do
3     if c<w[i] then UB1←LB1+c * p[i]/w[i]
4       for j←i+1 to n do
5         if c≥w[j] then c←c−w[j],LB1←LB1+p[j]
6         endif
7       repeat
```

```
8       return
9     endif
10  c←c−w[i];LB1←LB1+p[i]
11  repeat
12  UB1←LB1
13 end LUBOUND1
```

过程 LUBOUND2 与 LUBOUND1 是类似的，主要不同之处是第 5、10 两行。第 5 行中判断物品 j 是否可装入背包(对 LUBOUND2 是背包 2)，除判断背包 2 的剩余重量 c 是否大于等于物品 j 的重量 $w[j]$外，还需判断物品 j 的效益是否已计入 LB1 中。对第 10 行也需作类似考虑。

算法的正确性是不难理解的，因为给出的算法实质上是在穷尽搜索过程中利用 UBB1、LBB1 等启发函数限制生成那些不可能导致最优解的结点，故当算法结束时得到的必定是最优解。经用随机产生的 10 组不同的背包重量、物品数目、物品重量及效益数据验算，本算法所实际生成的结点数约为穷举法所生成结点数的 13.0%。

参 考 文 献

1 Ellis Horowitz, Sartaj Sahni. Fundamentals of Computer Algorithms. Maryland: Computer Science Press, Inc., 1978. 370～403

2 Mitten L. Branch-and-bound Methods: General Formulation and Properties. Oper. Res., 1970(18): 24～34

3 Ibaraki T. The Power of Dominance Relations in Branch-and-bound Algorithms. J. ACM, 1977, 24(2): 264～279

A Multi-branch-and-bound Algorithm for 0-1 Knapsack Problems

Li Mingshan　Zheng Haihong

(Dept. of Computer Science and Engineering, WTUSM, 39 Luoyu Road, Wuhan, China, 430070)

Abstract　This paper has extended the general branch-and-bound algorithm and given multi-branch-and-bound algorithm. The multi-branch-and-bound algorithm can solve 0-1 knapsack problems with more than one knapsack efficiently.

Key words　integer-programming; 0-1 knapsack; multi-branch-and-bound algorithm