有序组合树法求解 0-1 背包问题初探

安 晨，付永军

（兰州交通大学交通运输学院，甘肃兰州 730070）

摘 要：以 0-1 背包问题为研究对象，建立数学模型，采用有序组合树法对中小规模的背包问题进行求解．与传统的贪婪算法相比，该算法更容易找到最优解．并通过实例说明该算法对解决中小规模的 0-1 背包问题是行之有效的．

关键词：背包问题；有序组合树；算法

中图分类号：O221.4 **文献标识码**：A **文章编号**：1006-0375(2008)01-0010-05

背包问题是整数规划中一类较为特殊的问题，是一个经典的 NP-完全问题，也是一个典型的优化难题，其计算的复杂性为 $O(2^n)$ [1]．背包问题在现实生活中具有广泛的应用背景，如物流公司的货物发配问题、集装箱的装运问题以及工厂的下料问题等都可以用背包问题进行求解．解决背包问题的算法有最优算法和启发式算法，最优算法包括穷举法、动态规划法、隐数法以及有序组合树法[2]等，启发式算法包括贪婪算法、遗传算法、模拟退火算法以及禁忌搜索算法等一些智能算法．这些算法各有优缺点，有序组合树法原理简单，操作简便．对许多算例，用有序组合树法处理，更容易找到最优解，甚至树根即为最优解．

1 0-1 背包问题

给定 n 件物品和一背包，物品 i 的价值为 c_i，体积为 v_i，背包的容积为 V，如何选择装入背包的物品，才能使得装入背包中物品的总价值最大．引入 x_i 为一 0-1 变量，各变量的约束如下：

$$x_i=\begin{cases}1, & \text{第 } i \text{ 件物品装入背包}\\ 0, & \text{第 } i \text{ 件物品不装入背包}\end{cases} \qquad i=1,2,\cdots,n$$

背包容量约束：$\sum_{i=1}^{n} x_i v_i \le V$

0-1 背包问题的数学模型如下：

$$\max Z=\sum_{i=1}^{n} x_i c_i \qquad x_i=0 \text{ 或} 1,\ i=1,2,\cdots,n$$
$$s.t.\quad \sum_{i=1}^{n} x_i v_i \le V$$

收稿日期：2007-07-09

作者简介：安晨（1983-），女，甘肃庆阳人，硕士研究生，研究方向：交通运输规划与管理

2 有序组合树法

2.1 有序组合树法的标准型

有序组合树的特点是充分利用目标函数中的优化信息，对变量组合方案排定优劣次序，构造一棵树，按照一定步骤在构造树上顺次搜索，一旦发现某个方案满足全部约束条件，则该方案即为最优解．为使整个运算过程程式化，要求 0-1 背包问题具有以下标准型：

$$\min Z=\sum_{i=1}^{n}c_i x_j \quad ,\quad i=1,2,\cdots,m\ ,\quad x_j=0 \text{ 或}1,\quad j=1,2,\cdots,n$$
$$s.t.\ \sum a_{ij}x_j \ge b_i$$

其中，a_{ij} 和 b_j 都是整数，c_j 是非负整数，且 $c_1 \le c_2 \le \cdots \le c_n$.

2.2 有序组合树法的基本思想

在求解过程中，有序组合树法暂不考虑问题的约束条件，利用目标函数所提供的优化信息对变量组合方案排定优劣次序，然后，顺次对组合方案用约束条件进行检查．一旦发现某个组合方案满足了全部约束条件，则该方案既为最优解；若没有一个方案满足全部约束条件，则该问题无解．由此可见，此算法的思路不是在满足约束的可行解集中找最优，而是在无约束最优解集中找可行，与单纯形法[3]的思路恰好相反．

图 1 是 4 个变量的有序组合树，树根节点标号为 0，称为树的第 0 层．树根节点向右衍生 4 个子节点，标号分别为 1，2，3 和 4，称为树的第一层；1 号节点向右衍生 3 个子节点，标号分别为 2，3 和 4，第一层的 2 号节点向右衍生两个子节点，标号分别为 3 和 4，第一层的 3 号节点向右衍生一个子节点，标号为 4，4 号节点不再有子节点，这些子节点称为树的第二层．依此类推，构造下面的 3-4 层，各层的节点数分别为：$c_4^0=1$，$c_4^1=4$，$c_4^2=6$，$c_4^3=4$和$c_4^4=1$，构造树的全部节点数为 16，0-1 背包问题的解实质上是变量的某种组合方式，如问题有 4 个 0-1 变量，则变量的全部组合方案数为 $c_4^0+c_4^1+c_4^2+c_4^3+c_4^4=16$，即该问题在无约束的情况下共有 16 个解．4 个变量的有序组合树的全部节点数恰好等于 4 个 0-1 变量背包问题的全体无约束解个数．

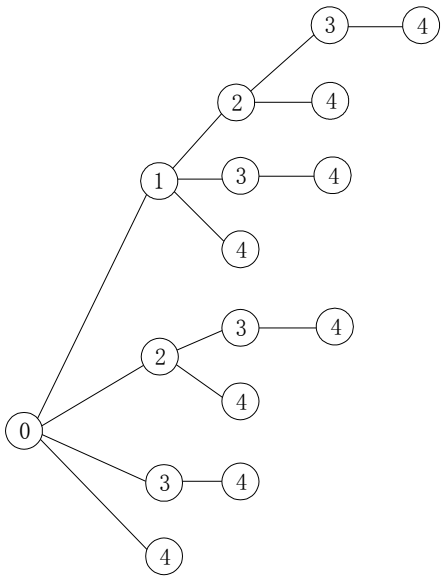

图 1　四个变量的有序组合树

把上述构造方法推至一般，当变量数为 n 时，组合树共有 $n+1$ 层，全部节点数为 2^n .

2.3 关于有序组合树的几点说明

（1）树的节点号就是变量的下标号，第 j 号节点对应变量 x_j .

（2）自树根到某一节点 j 的一条路表示一个解：路中节点（不含树根）对应的变量取值为 1，其余变量取值为 0．树根也代表一个解，即所有变量取值均为 0.

（3）同一条路上的相邻节点是父子关系，左边的父节点号小于右边的子节点号，同一父节点左边的同一层节点之间是兄弟关系，上边的兄弟节点号小于下边的弟节点号．因此，靠上、靠

左的路代表的解优于靠下、靠右的路所代表的解，这一特点是设计算法的基本依据[4].

3 算法设计

3.1 将原问题模型转换成标准型

在进行具体实际操作之前，先将原问题数学模型转换成有序组合树所规定的标准型，具体转换按如下约定进行：

（1）令$W=-Z$，使目标函数由极大化转换成极小化；

（2）令$x_i=1-x_i^{'}$，代入目标函数中，消去x_i，用$x_i^{'}$代替之，使目标函数中各变量的系数为正．求解结果若$x_i^{'}$=1，则$x_i=0$；若$x_i^{'}=0$，则$x_i=1$．常熟项不影响模型求解，故可省略[5].

（3）对不等号为“≤”的约束，两端同时乘以-1，使“≤”变成“≥”.

3.2 算法设计步骤

将原问题转化成标准型后，设计算法步骤如下：

S_1：以树根作为初始解，将目标函数值“0”标记在节点旁；

S_2：用约束条件检验树根是否可行．若是，找到最优解，输出结果，算法结束；否则划“×”勾销，然后向右上衍生，到达其“长子”1号节点，得到新的解向量；

S_3：计算新的解的目标值，并标记在节点旁；

S_4：在已计算目标值而未划“×”勾销的解集中找出目标值最小的解；

S_5：用约束条件检验该解是否可行．若是，找到最优解，输出结果，算法结束；否则划“×”勾销，执行下一步；

S_6：检查该节点有无“次弟”和“长子”节点．若有，向正下和右上个衍生一步，到达两个新的节点形成新的解向量，转S_3；若无，执行下一步；

S_7：检查有无已计算目标值而未划“×”勾销的解．若有，转S_4；若无，算法结束，原问题无解.

4 实例计算

给定一背包的容积为20，有4件待装物品的体积分别为6，5，9，7，其对应的价值分别为9，11，13，15，应该如何进行装包，使得包内所装物品的总价值最大，同时又能使所给背包的容量得到充分的利用.

在进行计算前，先对所给物品按其价值的大小进行升序排列，以满足算法的要求.

根据上面所给数据建立问题的数学模型：

$$\begin{aligned}&\max\ Z=9x_1+11x_2+13x_3+15x_4\\&s.t.\quad 6x_1+5x_2+9x_3+7x_4\le 20\end{aligned}\qquad x_i=0\text{或}1,\ i=1,2,3,4$$

将上面模型转换成有序组合树法规定的标准型：

$$\begin{aligned}&\min W=9x_1^{'}+11x_2^{'}+13x_3^{'}+15x_4^{'}\\&s.t.\quad 6x_1^{'}+5x_2^{'}+9x_3^{'}+7x_4^{'}\ge 8\end{aligned}\qquad x_i^{'}=1,2,3,4.$$

按照算法步骤进行计算，得到图 2 所示的有序组合树，图中粗实线所示的一条路即为最优解，$x_3^{'}$=1，$x_1^{'}=x_2^{'}=x_4^{'}$=0，则得到$x_1=x_2=x_4$=1，x_3=0．即 4 件物品中，装入 3 件，分别是第一件、第二件和第四件，第三件不装入背包，总共占用背包的体积为 19（小于背包容量 20），得到的总价值为 37，是最优装包方式．

下面讨论一下树根即为最优解的情况：

就上面的实例，只把背包的容积改为 28，其他的数值都不变，可建立下面模型：

$$\max Z=9x_1+11x_2+13x_3+15x_4$$

$$s.t.\quad 6x_1+5x_2+9x_3+7x_4\le 28$$

$$x_i=0 \text{ 或 } 1,\ i=1,2,3,4$$

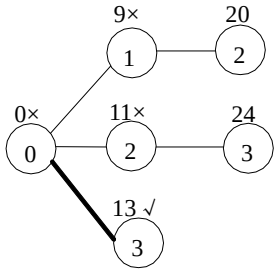

图 2　有序组合树生长结果

把模型转换为标准型：

$$\min W=9x_1^{'}+11x_2^{'}+13x_3^{'}+15x_4^{'}$$

$$s.t.\quad 6x_1^{'}+5x_2^{'}+9x_3^{'}+7x_4^{'}\ge -1$$

$$x_i^{'}=1,2,3,4$$

按照算法步骤进行计算，很容易得到树根即为最优解，结果如图 3 所示的一棵生长树结果．得到的解为：$x_1^{'}=x_2^{'}=x_3^{'}=x_4^{'}$=0，则$x_1=x_2=x_3=x_4$=1，4 件物品全部装入背包，总体积是 27，总价值是 48．

图 3　有序组合树生长结果

从上面实例计算可以看出，对于存在可行解的 0-1 背包问题，一般不需要把组合树的全部构造出来．

解 0-1 背包问题有好几种贪婪策略，每个贪婪策略都要采用多步来完成背包的装入．在每一步过程中利用贪婪准则选择一个物品装入背包．一种贪婪准则为：从剩余的物品中，选出可以装入背包的价值最大的物品，利用这种规则，价值最大的物品首先被装入（假设有足够容量），然后是下一个价值最大的物品，如此继续下去．这种策略不能保证得到最优解．例如，考虑 n = 2，w = [100,10,10]，p = [20,15,15]，c = 105．当利用价值贪婪准则时，获得的解为 x, = [1,0,0]，这种方案的总价值为 20．而最优解为[0,1,1]，其总价值为 30．

另一种方案是重量贪婪准则：从剩下的物品中选择可装入背包的重量最小的物品．虽然这种规则对于前面的例子能产生最优解，但在一般情况下则不一定能得到最优解．考虑 n = 2，w = [10, 20]，p = [5,100]，c = 25．当利用重量贪婪策略时，获得的解为 x = [1,0]，比最优解[0,1]要差．

还可以利用另一方案，价值密度 pi/wi 贪婪算法，这种选择准则为：从剩余物品中选择可装入包的 pi/wi 值最大的物品，这种策略也不能保证得到最优解．利用此策略试解 n = 3，w = [20,15, 15]，p = [40,25,25]，c = 30 时的最优解，获得的解为 x = [1,0,0]，比最优解[0,1,1]要差．

有序组合树法的优点在于原理简单，操作简便，不用大量穷举，只是隐含地考虑了变量组合的取值，也不用回溯，可以一直向前执行下去，直到找到最优解为止；缺点就是对于约束条件中的“约束信息”利用不够，解的收敛速度有时不够理想，但对于某些算例，用该方法很容易就能找到最优解，有时甚至树根就是最优解．作为一种搜索技术，所给算法能够找到最优解是肯定的，但在运算效率方面有必要进一步提高，需尽可能缩小搜索范围，减少搜索步数，提高收敛速度，使组合树的生长能够得到更好的控制．解决 0-1 背包问题，该法是一种有效的新解题方法．

5 结束语

本文采用有序组合树法对背包问题进行研究，该算法原理简单，操作简便．对许多算例，特别是对中小规模的背包问题，采用有序组合树法能很容易地找到最优解，但对于求解规模太大的背包问题，该算法的优势不是很明显，有待于进一步改进．

参考文献

[1] 王晓东．计算机算法设计与分析[M]．北京：电子工业出版社, 2003: 68-70.

[2] 朱松年．有序组合树法[J]．西南交通大学学报, 1985, (2): 15-25.

[3] 焦永兰．管理运筹学[M]．北京：中国铁道出版社, 2002: 130-132.

[4] 王慈光．对有序组合树法的改进[J]．西南交通大学学报, 2006, 41(5): 561-565.

[5] 周颖．零担货物配装模型与算法研究[J]．铁道运输与经济, 2006, 28(8): 81-82.

On the Sequential Combination Tree Algorithm for 0-1 Knapsack Problem

AN Chen, FU Yongjun

(School of Traffic and Transportation, Lanzhou Jiaotong University, Lanzhou, China 730070)

Abstract: With 0-1 knapsack problem being research object, this paper has set up a mathematical model for 0-1knapsack problem. Planning to solve 0-1 knapsack problem for medium-sized and small scale by the sequential combination tree algorithm, through analysis and computation for the concrete example, it is proved that the sequential combination tree algorithm is feasible and effective for solving 0-1 knapsack problem for medium and small scale, compared to traditional greedy algorithm, the optimal solution is very easy to be found.

Key words: Knapsack problem; Sequential combination tree; Algorithm

（编辑：王一芳）