0-1 背包问题的两种扩展形式及其解法*

刘玉娟[1]，王相海[1,2]

(1. 辽宁师范大学 计算机与信息技术学院，辽宁 大连 116029；2. 中国科学院 研究生院 信息安全国家重点实验室，北京 100039)

摘 要：0-1 背包问题是经典的 NP-HARD 组合优化问题之一，由于其难解性，该问题在信息密码学和数论研究中具有极其重要的应用。首先对 0-1 背包问题及其解法进行了分析，然后提出 0-1 背包问题的两种扩展形式，并给出了基于动态规划和贪心算法的两种有效算法来解决这两类问题。实验结果验证了所提出方法的有效性。

关键词：0-1 背包；扩展形式；动态规划；贪心算法

中图法分类号：TP301.6 **文献标识码**：A **文章编号**：1001-3695(2006)01-0028-03

Two Kinds of Expanding Forms of 0-1 Knapsack Problem and Its Solution Methods

LIU Yu-juan[1], WANG Xiang-hai[1,2]

(1. *College of Computer & Information Technology, Liaoning Normal University, Dalian Lianning* 116029, *China*; 2. *State Key Laboratory of Information Security, Graduate School, Chinese Academy of Science, Beijing* 100039, *China*)

Abstract: The 0-1 knapsack problem is a classic NP-Hard problem in the combinational optimization. Because it is very hard for the solution, it is very important in the research on cryptosystem and number theory. In this paper, the 0-1 knapsack problem and its algorithm is analyzed firstly. And then this paper presents two kinds of expand form, and proposed two efficient algorithms based on dynamic programming and greedy algorithm to solve the proposed problems. Simulation results show it is effective.

Key words: 0-1 Knapsack; Expanding Form; Dynamic Programming; Greedy Algorithm

0-1 背包问题在信息密码学领域和数论研究中具有极其重要的作用，然而由于 0-1 背包问题属于 NP-HARD 问题，直接的枚举解法可能要遍历问题的所有 2^n 个解空间，因此，采用算法设计策略降低求解问题过于庞大的计算量，具有重要的理论和实际意义。

1 0-1 背包问题及其解法讨论

0-1 背包问题：给定 n 种物品和一背包，物品 i 的重量是 w_i，其价值为 v_i，背包的容量为 C，问应如何选择装入背包中的物品，使得装入背包中物品的总价值最大？

在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或不装入背包。不能将物品 i 装入背包多次，也不能只装入部分物品 i。

1.1 0-1 背包问题是一个特殊的整数规划问题

0-1 背包问题可形式化地描述为下列特殊的整数规划问题：

$$\max \sum_{i=1}^{n} v_i x_i, \begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

收稿日期：2004-11-21；修返日期：2005-06-28

基金项目：国家自然科学基金资助项目(60372071)；辽宁省自然基金资助项目(20032125)；大连市科技基金计划资助项目；辽宁省高等学校优秀人才支持计划资助项目

该问题具有有最优子结构性质以及如下递归关系[3]：

$$m(i,j) = \begin{cases} \max\{m(i+1,j), m(i+1,j-w_i)+v_i\} & j \ge w_j \\ m(i+1,j) & 0 \le j < w_i \end{cases}$$

$$m(i,j) = \begin{cases} v_n, & j \ge w_n \\ 0, & 0 \le j < w_n \end{cases}$$

其中 $m(i,j)$ 是背包容量为 j，可选择物品为 $i, i+1, \cdots, n$ 时 0-1 背包问题的最优值。0-1 背包问题可以用动态规划策略求解，其时间复杂度为 $O(nC)$。

1.2 0-1 背包问题又是一个子集选取问题

在定义了 0-1 背包问题的解空间之后，将解空间组织成树或图的形式(图 1 为 $n=3$ 时 0-1 背包问题的解空间)，应用回溯法可方便地搜索整个解空间，搜索结束后找到的最好解是相应 0-1 背包问题的最优解，其时间复杂度为 $O(n2^n)$[3]。

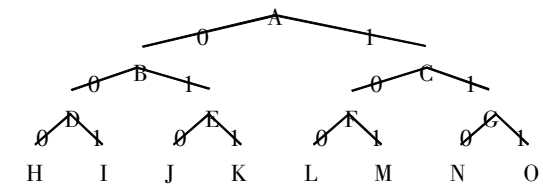

图 1 0-1 背包问题的解空间

由动态规划策略和回溯算法的计算复杂性可知，当背包容量 $C > 2^n$ 时，应用回溯算法求解 0-1 背包问题效率更高一些，反之，应用动态规划策略求解 0-1 背包问题则更好一些。可见对于 0-1 背包问题本身的求解，为了降低求解问题的时间复杂度，也要依据实际问题设计合适的算法策略。

2　0-1 背包问题的扩展形式之一

2.1　问题提出

给定 n 种物品和一背包, 物品的重量是 w_i, 体积为 b_i, 价值为 v_i, 背包的容量为 C, 容积为 D, 问应如何选择装入背包中的物品, 使得装入背包中物品的总价值最大? 在选择装入背包的物品时, 对每种物品 i 只有两种选择, 即装入背包或不装入背包。不能将物品 i 装入背包多次, 也不能只装入部分物品 i。

2.2　问题符号化

此问题的形式化描述是, 给定 $C>0$, $w_i>0$, $b_i>0$ 和 $v_i>0$, 其中 $i\in[1, n]$, 要求找出一个 n 元向量 $(x_1, x_2, \cdots, x_n)$, 其中 $x_i\in\{0, 1\}$, $1\le i\le n$, 使得当 $\sum_{i=1}^{n} w_i x_i\le C$, $\sum_{i=1}^{n} b_i x_i\le D$ 时式子 $\sum_{i=1}^{n} v_i x_i$ 达到最大。因此, 该问题也是一个特殊的整数规划问题: 在约束 $\sum_{i=1}^{n} w_i x_i\le C$ 和 $\sum_{i=1}^{n} b_i x_i\le D$ (其中 $x_i\in\{0, 1\}$, $1\le i\le n$) 下, 求使 $\sum_{i=1}^{n} v_i x_i$ 达到最大的 $\{x_i\}$。

2.3　动态规划解法

(1) 问题的最优子结构性质

命题 1: 设 $(y_1, y_2, \cdots, y_n)$ 是上述问题的一个最优解, 则 $(y_2, \cdots, y_n)$ 是下面子问题的一个最优解: 在约束 $\sum_{i=2}^{n} w_i x_i\le C-w_1 y_1$ 和 $\sum_{i=2}^{n} b_i x_i\le D-b_1 y_1$ (其中 $x_i\in\{0, 1\}$, $1\le i\le n$) 下, $(y_2, \cdots, y_n)$ 可使 $\sum_{i=2}^{n} v_i x_i$ 达到最大。

证明: 反证, 若不然, 设 $(z_2, z_3, \cdots, z_n)$ 是上述子问题的一个最优解, 则有 $\sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i$ 且 $\sum_{i=2}^{n} w_i z_i\le C-w_1 y_1$, $\sum_{i=2}^{n} b_i z_i\le D-b_1 y_1$, 故有 $v_1 y_1+\sum_{i=2}^{n} v_i z_i > \sum_{i=1}^{n} v_i y_i$ 且 $w_1 y_1+\sum_{i=2}^{n} w_i z_i\le C_1$, $b_1 y_1+\sum_{i=2}^{n} b_i z_i\le D$, 这说明 $(y_1, z_2, z_3, \cdots, z_n)$ 是原问题的一个较 $(y_1, y_2, \cdots, y_n)$ 更优的解, 这与 $(y_1, y_2, \cdots, y_n)$ 为原问题的最优解发生矛盾。结论得证。

(2) 递归关系

所给问题的子问题:

$$\max\sum_{k=i}^{n} v_k x_k,\begin{cases}\sum_{k=i}^{n} w_k x_k\le j\\ \sum_{k=i}^{n} b_k x_k\le v\\ x_k\in\{0, 1\},\ i\le k\le n\end{cases}$$

的最优值为 $m(i, j, v)$, 即 $m(i, j, v)$ 表示背包容量为 j, 容积为 v, 可选择物品为 $i, i+1, \cdots, n$ 时此问题的最优值, 由此问题的最优子结构性质, 可以建立 $m(i, j, v)$ 的递归式:

$$m(i, j, v)=\begin{cases}\max\{m(i+1, j, v), m(i+1, j-w_i, v-b_i)+v_i\}, & j\ge w_i \&\& v\ge b_i\\ m(i+1, j, v), & 0\le j<w_i \,\|\, 0\le v\le b_i\end{cases}$$

$$m(n, j, v)=\begin{cases}v_n, & j\ge w_n \&\& v\ge b_n\\ 0, & 0\le j<w_n \,\|\, 0\le v<b_n\end{cases}$$

(3) 算法实现

```
void Knapsack( int* v, int* w, int* b, int c, int d, int n, int*** m)
{   int jMax = min( w[ n] - 1, c) ;
   int vMax = min( b[ n] - 1, d) ;
   for( int j = 0, v = 0; j < = jMax | |v < = vMax; j + +, v + + )
      m[ n] [ j] [ v] = 0;
   for( j = w[ n] , v = b[ n] ; j < = c&&v < = d; j + +, v + + )
      m[ n] [ j] [ v] = v[ n] ;
   for( int i = n; i > 0; i--)
   {   jMax = min( w[ i - 1] - 1, c) ;
      vMax = min( b[ i - 1] - 1, d) ;
      for( int j = 0, v = 0; j < = jMax | |v < = vMax; j + +, v + + )
         m[ i - 1] [ j] = m[ i] [ j] ;
      for( j = w[ i - 1] ; j < = c; j + + )
         m[ i - 1] [ j] [ v] = max( m[ i] [ j] [ v] , m[ i] [ j - w[ i - 1] ]
[ v - b[ i - 1] ] + v[ i - 1] ) ;
   }
   m[ 0] [ c] [ d] = m[ 1] [ c] [ d] ;
   if( c > = w[ 0] &&d > = b[ 0] )
   m[ 0] [ c] [ d] = max( m[ 0] [ c] [ d] , m[ 1] [ c - w[ 0] ] [ d - b[ 0] ] +
v[ 0] ) ;
   }

   void Trackback( int* * * m, int* w, int* b, int c, int d, int n, int* x)
   {   for( int i = 1; i < n; i + + )
      if( m[ i - 1] [ c] [ d] = = m[ i] [ c] [ d] )
         x[ i - 1] = 0;
      else{   x[ i - 1] = 1;
         c - = w[ i - 1] ;
         d - = b[ i - 1] ;
         }
      x[ n - 1] = ( m[ n - 1] [ c] [ d] ) ? 1: 0;
   }
```

(4) 时间复杂度分析

容易得出函数 Knapsack( ) 的复杂度为 O(max(nC, nD)), 函数 Trackback( ) 的复杂度为 O(n), 故所给出算法的时间复杂度为 O(max(nC, nD, n))。

3　0-1 背包问题的扩展形式之二

3.1　问题提出

在 0-1 背包问题中, 各物品依重量递增排列时, 其价值恰好依递减顺序排列, 这是一个特殊的 0-1 背包问题, 设计一个有效算法找出最优解。

3.2　问题符号化

此问题的形式化描述是, 给定 $C>0$, $w_i>0$, $v_i>0$, $1\le i\le n$, 且分别对 w_i, v_i 进行排序后有 $w_i\le w_{i+1}$, $v_i\ge v_{i+1}$, $1\le i\le n-1$, 要求找一个 n 元向量 $(x_1, x_2, \cdots, x_n)$, $x_i\in\{0, 1\}$, $1\le i\le n$, 使得 $\sum_{i=1}^{n} w_i x_i\le C$, 且 $\sum_{i=1}^{n} v_i x_i$ 达到最大。我们用贪心算法求解这个问题。

3.3　贪心解法

(1) 贪心选择性质

命题 2: 上述问题满足贪心选择性质。

证明: 设 $(x_1, x_2, \cdots, x_n)$ 是上述问题的最优解, 又设 $k=\min\{i\,|x_i=1\}$, $1\le i\le n$。这样, 如果给定的问题有解, 则必有 $1\le k\le n$, 当 $k=1$ 时, $(x_1, x_2, \cdots, x_n)$ 即是一个满足贪心选择性质的最优解; 当 $1<k\le n$ 时, 取 $y_1=1$, $y_k=0$, $y_i=x_i$, 其中 $1<i\le n$, $i\ne k$, 则由问题关于价值的单调递减性和关于重量的单调递增性可知有: $\sum_{k=1}^{n} v_i y_i > \sum_{k=1}^{n} v_i x_i$ 且 $\sum_{i=1}^{n} w_i y_i=(w_1-w_k)+\sum_{i=1}^{n} w_i x_i\le C$, 故 $(y_1, y_2, \cdots, y_n)$ 是所给问题的一个可行解, 且 $(y_1, y_2, \cdots, y_n)$ 是一个满足贪心选择性质的最优解, 故此问题具有贪心选择性质。

(2) 最优子结构性质

命题 3: 上述问题具有最优子结构性质。

证明: 设 $(y_1, y_2, \cdots, y_n)$ 是所给 0-1 背包问题的一个最优解, 则 $(y_2, y_3, \cdots, y_n)$ 是下面相应子问题的一个最优解:

$$\max\sum_{i=2}^{n} v_i x_i,\begin{cases}\sum_{i=2}^{n} w_i x_i\le C-w_1 y_1\\ x_i\in\{0, 1\},\ 2\le i\le n\end{cases}$$

如若不然,设$(z_2, z_3, \cdots, z_n)$是上述子问题的一个最优解,则有:$\sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i$,且$\sum_{i=2}^{n} w_i z_i \le C - w_1 y_1$,从而有:$v_1 y_1 + \sum_{i=2}^{n} v_i z_i > \sum_{i=1}^{n} v_i y_i$且$w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le C$。这说明$(y_1, z_2, z_3, \cdots, z_n)$是一个较$(y_1, y_2, \cdots, y_n)$更优的解,这与$(y_1, y_2, \cdots, y_n)$是问题的最优解发生矛盾。故上述问题具有最优子结构性质。

(3) 算法实现

此问题采用以重量最轻者先装的贪心选择策略,具体算法如下:

```
int Partion( int* w, int l, int r)
{   int i, j;
    i = l; j = r;
    int temp = w[ i] ;
    do{
      while( ( w[ j] > = temp) &&( i < j) )   j--;
      if( i < j)   w[ i + + ] = w[ j] ;
      while( ( w[ i] < = temp) &&( i < j) )   i + + ;
      if( i < j)   w[ j - - ] = w[ i] ;
    } while( i! = j) ;
    w[ i] = temp;
    return i;
}
void QuickSort( int* w, int l, int r)
{   int i;
    if( l < r) {
      i = Partion( w, l, r) ;
      QuickSort( w, l, i - 1) ;
      QuickSort( w, i + 1, r) ;
      }
}
void Loading ( int* x, int* w, int c, int n)
{   int* t = new int [ n + 1] ;
  QuickSort( w, t, n) ;
  for( int i = 1; i < = n; i + + )
    x[ i] = 0;
  for( int i = 1; i < = n&&w[ t[ i] ] < = c; i + + )
{   x[ t[ i] ] = 1;
    c - = w[ t[ i] ] ;
  }
}
```

(4) 时间复杂度分析

算法的时间复杂度为 $O(n\log_2 n)$。

4 分析与讨论

前面的两个扩展问题均可采用用动态规划策略求解,但第二个问题采用贪心解法更简单,效率更高一些,而第一个扩展问题用贪心算法却不能获得最优解。例如:若背包的容量为 50kg,容积为 100,物品 1 的重量为 10kg,体积为 10,价值为 60 元,即 6 元/kg;物品 2 的重量为 20kg,体积为 20,价值为 100 元,即 5 元/kg;物品 3 的重量为 30kg,体积为 30,价值为 120 元,即 4 元/kg,因为物品 1,2,3 的体积之和为 60,小于背包的总容积 100,故该问题是第一个扩展问题的实例。若采用贪心选择策略解决,应首选物品 1 装入背包,可此问题的最优选择方案是选择物品 2 和物品 3 装入背包,首选物品 1 不是最优的,因此不能获得最优解。

参考文献:

[1] E Bellman. Dynamic Programming[M]. Princeton University Press, 1957.

[2] Greedy Algorithm[EB/OL]. http://www.trentu.ca/~bpatrick/cosc300/notes/greedy.pdf, 2004.

[3] 王晓东. 计算机算法设计与分析[M]. 北京:电子工业出版社, 2001.

作者简介:

刘玉娟(1982-),辽宁人,学士,研究方向为算法设计与分析;王相海(1965-),吉林人,教授,博士,研究方向为计算机图形学、多媒体技术和算法设计与分析。

(上接第 27 页)

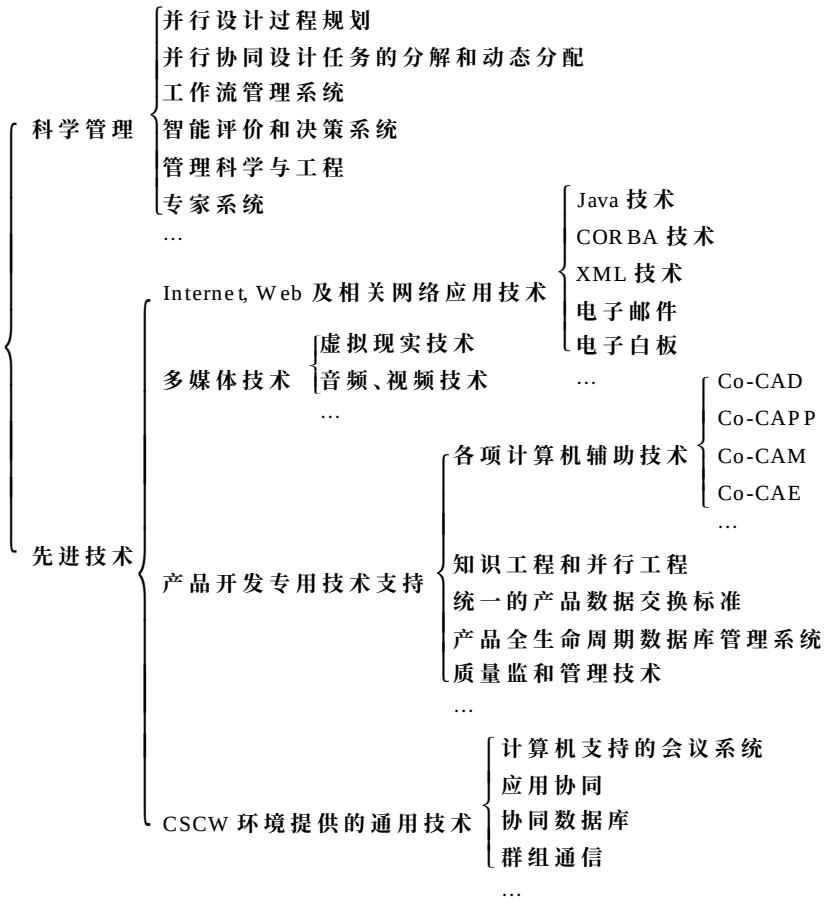

图 6 系统开发所需技术支持

图 7 为系统最终经过协作形成的 AutoCAD 2004 输出的大小轮二维零件图、小轮为轴齿轮时的零件图以及由 I-deas 三维软件进行实体建模后交给 SolidWorks 产生的大小轮啮合时的虚拟现实图形(WRL 文件)在浏览器中的交互输出。

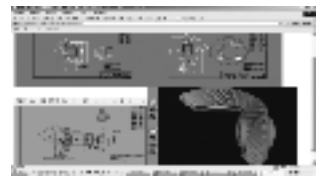

图 7 协同设计后二维和三维零件图的交互输出

3 结束语

本文将网络协同设计技术和方法的研究与实际齿轮产品应用背景相结合,开发的齿轮网络协同设计系统非常符合当前齿轮设计发展的要求,具有一定的理论价值和现实意义。

参考文献:

[1] 史美林,向勇,等. 计算机支持的协同工作理论与应用[M]. 北京:电子工业出版社, 2000.

[2] 任东锋,黄广君,方宗德. 基于网络面向并行工程的螺旋锥齿轮 CSCW 系统研究[J]. 计算机应用研究, 2003, 20(11):18-21.

[3] 任东锋,方宗德. 面向并行工程的螺旋锥齿轮网络协同设计系统研究[J]. 计算机工程与应用, 2003, 39(14):17-21.

[4] 任东锋,方宗德,黄广君. XML 在网络协同设计系统中的应用[J]. 计算机工程, 2003, 29(12):92-94.

[5] 任东锋,张金良,方宗德. 基于 ASP 模式的螺旋锥齿轮远程设计分析服务系统[J]. 机械设计, 2004, 21(2):7-10.

[6] 任东锋,杨玉婷,方宗德. 基于移动代理的分布式工作流管理系统[J]. 制造业自动化, 2003, (8):8-13.

作者简介:

任东锋(1976-),男,河南焦作人,博士研究生,主要研究方向为并行设计、计算机支持的协同设计;丰树谦,副教授,研究方向为软件;李体红,副教授,研究方向为软件;方宗德,男,教授,博士生导师。