基于 0-1 背包问题的两种算法

李强，蓝雯飞

（中南民族大学计算机科学学院，武汉 430074）

摘 要：0-1 背包问题是计算机科学中一个经典问题，0-1 背包问题是一个最优化问题。因其结构简单，可扩展性强，可作为其他问题的子问题，因此通过对其研究可以解决更为复杂的优化问题。本文概述了两种求解 0-1 背包问题的算法设计方法，并对两种算法进行了分析和比较。

关键字：0-1 背包问题；回溯；动态规划

中图分类号：TP301.6 **文献标识码：**A DOI: 10.3969/j.issn.1003-6970.2014.03.032

本文著录格式：[1] 李强，蓝雯飞 . 浅析 0-1 背包问题的两种算法 [J]. 软件 ,2014,35(3)：105-106

Two Algorithms Based on 0-1 Knapsack Problem

LI Qiang, LAN Wen-fei

(South-Central University for Nationalities, Wuhan 430074*,China)*

【Abstract】The 0-1 knapsack problem is a classical problem in Computer Science, 0-1 knapsack problem is an optimization problem. Because of its simple structure, strong scalability, it can be used as sub problems of other problems. Therefore the research can solve more complex optimization problems. This paper summarizes the two kinds of algorithms to solve 0-1 knapsack problem，finally two algorithms are compared and analyzed.

【Key words】0-1 knapsack problem；backtrack；dynamic programming

0 引言

背包问题是运筹学中一个优化难题，最早由 Dantzing 于 20 世纪 50 年代首次提出，现已成为计算机学科中一个经典的 NP 问题，它在很多领域都有广泛的应用，很多实际问题都可转换为 0-1 背包问题。例如货物优化装载，存储优化分配等都可转换为 0-1 背包问题。因此，对 0-1 背包问题的研究无论是在理论上还是在实践中都具有一定的意义 [1]。

1 0-1 背包问题

0-1 背包问题描述为，有 n 种物品，第 i 物品的重量为 w_i，价值为 v_i，选一些物品装到一个容量为 C 的背包中，使的背包内的物品在总重量不超过 C 的前提下总价值最大？在选择装入背包的物品时，只有两种选择，要么装入，要么不装入，不能只装入部分物品。因此，该问题被称为 0-1 背包问题。

到目前为止，求解 0-1 背包问题的方法很多。本文主要针对常见的求解 0-1 背包问题的算法设计方法进行阐述。

2 动态规划算法

2.1 动态规划算法的基本思想

动态规划算法在 20 世纪 50 年代初提出，是为了解决一类多阶段决策问题而诞生的。其通常应用于最优化问题，即要做出一组决策以达到问题的最优解。

用动态规划求解问题的基本过程是将所求解的问题分解成若干个相互联系的子问题。然后从子问题出发，逆向求解，最后得到原问题的解。在求解的过程中，依据问题的当前状态做出决策，所做决策引起问题当前状态的变化，然后在进一步做出决策，最终形成决策序列。求解子问题的过程中，将子问题的状态填入表中，从而避免每次遇到各个子问题时在重新计算，这样就提高了算法的效率。

2.2 用动态规划解 0-1 背包问题

0-1 背包问题的子问题定义为：F[i][w] 表示将前 i 件物品放入一个重量为 w 的背包中可以获得的最大价值。其决策过程为：

$F[i][w] = \max\{ F[i-1][w], F[i-1][w-w_i] + v_i \};$

这个过程可以这样理解，“将前 i 件物品放入重量为 w 的背包中”这个子问题，若当前考虑第 i 件物品的策略 (那只有放与不放两种情况），那么可以转化为只和前 i-1 件物品相关的问题。若在当前的重量限制下，不放第 i 件物品，那么问题就转化为“将前 i-1 件物品放入容量为 w 的背包中”，所获得的价值为 F[i-1][w]，如果放第 i 件物品（即当前背包重量 w 大于物品重量 w_i），那么问题转化为“前 i-1 个物品放入剩下重量为 $w - w_i$ 的背包中”，此时能获得的最大价值就是 $F[i-1][w-w_i]$ 加上通过放入第 i 件物品获得的价值 v_i。

作者简介：李强（1989- ），男，硕士研究生，数据库，中南民族大学。

通信作者：蓝雯飞，教授，硕士生导师，主要研究方向：数据库，软件工程。

伪代码如下：

$$F[0,1,2\cdots n][0,1,2\cdots C]\leftarrow 0$$
$$for\ \ i\leftarrow 1\ \ to\ \ n$$
$$\quad for\ w\leftarrow 1\ \ to\ \ C$$
$$\quad\quad if\ (\ w<w_i\ )$$
$$\quad\quad\quad F[i][w]=F[i-1][w]$$
$$\quad\quad else$$
$$\quad\quad\quad F[i][w]=\max\{F[i-1][w],F[i-1][w-w_i]+v_i\}$$

动态规划算法是一种经典的 0-1 背包问题求解算法，算法思路清晰，易于实现。由于动态规划算法通过缓存子问题的解，从而避免了大量的重复计算，在实际的运行中效率较高。但是对于规模较大的背包问题时，此时计算所需的存储量和计算量会很大。这样就限制了动态规划所解决问题的规模 [1]。

3 回溯算法

3.1 回溯算法的基本思想

回溯算法有“通用的解题法”之称，它通过搜索一个问题的所有解而得到最优解。回溯算法采用“试探”的思想，逐步尝试去解决一个问题。在解决问题的过程中，若当它判断从该状态出发不能得到正确有效的解时，它就跳过对该状态的试探，并逐步向先前的状态回溯， 通过其它的状态再次尝试寻找问题的解。否则，则继续搜索，直达搜索出问题所要求的解。这种实质是以深度优先方式搜索问题解的算法称为回溯法。

3.2 回溯法解 0-1 背包问题

0-1 背包问题的解空间可以形象的用二叉树的形式表示出来，树中从第 i 层到 i+1 层表示对第 i 个物品取舍的选择，用左子树表示将物品装入背包 [2]，右子树表示不装入 (如图 1 所示)，这样从根节点到叶子结点的任一路径都是问题一个可能的解。

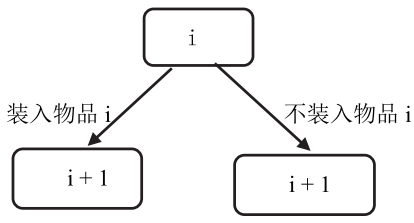

图 1 0-1 背包问题的搜索过程

根据上面的分析，具体算法如下：

对于每一个待选择的物品，对该物品只有选或者不选两个决策，顺序考虑所有的物品，这样就形成了一颗解的空间树。由当前结点向下搜索时，左子树表示装入物品，并将该物品的重量和价值加到当前所得的总重量和总价值中，右子树则表示不加入该物品，当到达树的叶子结点时，表示所有的物品都已考虑，这时就可以更新最优解了。

这种通过尝试所有可能的方式，会导致指数时间的时间复杂度，在问题规模较大时，则难以求出问题的解。回溯法在搜索解空间树时，为了避免无效的搜索，通常会在待扩展的结点处剪去不满足条件的子树，这样会大大提高回溯法的搜索效率。

分析可得 0-1 背包问题的两种“剪枝”优化为：

(1) 当前物品若不能装入背包，则可“剪枝”当前结点的左子树。

(2) 如果当前所得的价值和剩下所有物品的价值总和都不如目前已经得到的最优价值，则可进行“剪枝”。

伪代码如下：

```
void BackTrack( int i, int currentValue, int currentWeight )
    if ( i > n )
        bestValue = currentValue
        return
    if ( currentWeight + w[i] < C )
        BackTrack( i +1, currentValue + v[i], currentWeight + w[i] );
    remainValue  -= v[i];
    if ( currentValue + remainValue > best )
        BackTrack( i + 1, currentValue , currentWeight )
    remainValue  += v[i];
```

4 动态规划法与回溯法的分析

通过上面的分析可以看出，两种算法在处理 0-1 背包问题都有一定的可行性。动态规划算法思路清晰，易于实现。算法时间复杂度和空间复杂度均为 O(Cn)，当物品重量很大时，所需要的物理空间也会很大，当物品的重量不是整数时也较难处理。回溯算法通过搜索问题的解空间而得到最优解，最坏情况下时间复杂度为 $O(2^n)$，在搜索过程中，仅保留从开始结点到当前可扩展结点的路径，因此其空间复杂度为 O(n)。可以预见，当问题规模稍大时，此时回溯法就不具备求出问题解的可能。

总结：

本文首先对 0-1 背包问题做了简单的描述。在此基础上，介绍了运用在 0-1 背包问题上的两种算法。求解 0-1 背包问题，除了用动态规划算法和回溯算法外，还可以用暴力枚举法，近似算法，启发式算法等。求解一个问题有不同的算法，在程序设计中，多数情况下是将几种算法结合起来使用最优的方式解决问题。

参考文献

[1] 田烽楠，王于 . 求解 0-1 背包问题算法综述 [J]. 软件导刊，2009, 8（1）：59-61.

[2] 赵健英 . 0/1 背包问题的非线性降维近似算法 [J]. 内蒙古师范大学学报（自然科学汉文版），2007(1).

[3] 赵娟，樊超 . 动态规划方法的应用研究 [J]. 计算机时代 , 2014(2):28-30.

[4] 姜宇，苏中滨，郑萍 . 求解 0/1 背包问题的算法综述 [J]. 黑龙江大学自然科学学报 , 2009(26):92-99.

[5] 王晓东 . 计算机算法设计与分析 [M]. 北京 : 电子工业出版社 ,2012.

[6] 刘汝佳 算法竞赛入门经典 [M]. 北京：清华大学出版社，2009.