Knapsack Problem

Manish Purohit

April 25, 2013

1 Problem Definition

You are given n items, where each item has size s_i ("weight") and profit p_i ("benefit"). You have a knapsack (bag) of capacity W, and the objective is to choose items so that the total benefit of the chosen items is maximized. There is only one copy of each item.

2 Main Idea

Let the items be ordered arbitrarily. Notice the following "optimal substructure"property. Consider the n^{th} item - an optimal solution to the knapsack problem either chooses this item or it doesn't. Suppose this item is chosen, then you gain profit p_n and your bag (knapsack) has remaining capacity $W - s_n$ and the only job left is to fill this remaining capacity with as much benefit as possible from among the first $n-1$ items. This is simply another (smaller) knapsack problem! Also, if you suppose that the n^{th} item is not chosen, then in fact you still have a knapsack of capacity W which must be filled using the first $n-1$ items, which too is a smaller knapsack problem. The optimal solution is clearly the better of these two choices.

Let us formalize this notion. Suppose $OPT(i, w)$ denotes the optimal way to pack a knapsack of capacity w using the first i items. Clearly, the solution we are interested in is $OPT(n, W)$. Using the two choices described above, we get the following recurrence -

$$OPT(i, w) = \max \begin{cases} p_i + OPT(i-1, w - b_i) \\ OPT(i-1, w) \end{cases}$$

One little detail which we missed above is that the i^{th} item can be chosen only if its size s_i is less than the allowed budget w.

$$OPT(i, w) = \max \begin{cases} p_i + OPT(i-1, w - b_i) & \text{, If } b_i < w \\ OPT(i-1, w) \end{cases}$$

To compute $OPT(n, W)$, we use above recurrences to get smaller and smaller knapsack problems. In the base case, $OPT(i, 0) = 0$ since one cannot earn any benefit if the knapsack has zero capacity.

3 Implementation Issues

Above algorithm is a classic example of an algorithm paradigm called "Dynamic Programming". Typically, we can think of constructing a $n \times W$ matrix where the entry in the i^{th} row and w^{th} column corresponds to $OPT(i, w)$. Since each entry of this matrix only depends on previous values (using the recurrence described above), we can populate the entire table in $O(nW)$ time. Finally, the solution is present in the last cell, i.e. $OPT(n, W)$.

You might have noticed, that the above approach would only give us the "value" of the optimal solution, i.e. the maximum benefit that can be obtained and NOT the actual items that must be selected. These

actual items can be obtained by retracing your steps back the matrix, and observing the decisions you made at each step, i.e. of the two possible options in the recurrence, which one did you choose?

4 Example

Consider a knapsack of total capacity, $W = 7$. There are a total of 6 items with sizes (weights) and profits as given below. Find the best selection of items to gain maximum profit subject to the capacity constraints.

Object	O1	O2	O3	O4	O5	O6
Weight	1	2	4	3	2	4
Profit	1	5	8	10	20	25

To solve above problem, we build a matrix to store all $OPT(i, w)$ values, and use the recurrence to fill up the matrix entries. We use the base cases $OPT(i, 0) = 0$ and $OPT(0, w) = 0$ to fill up initial entries.

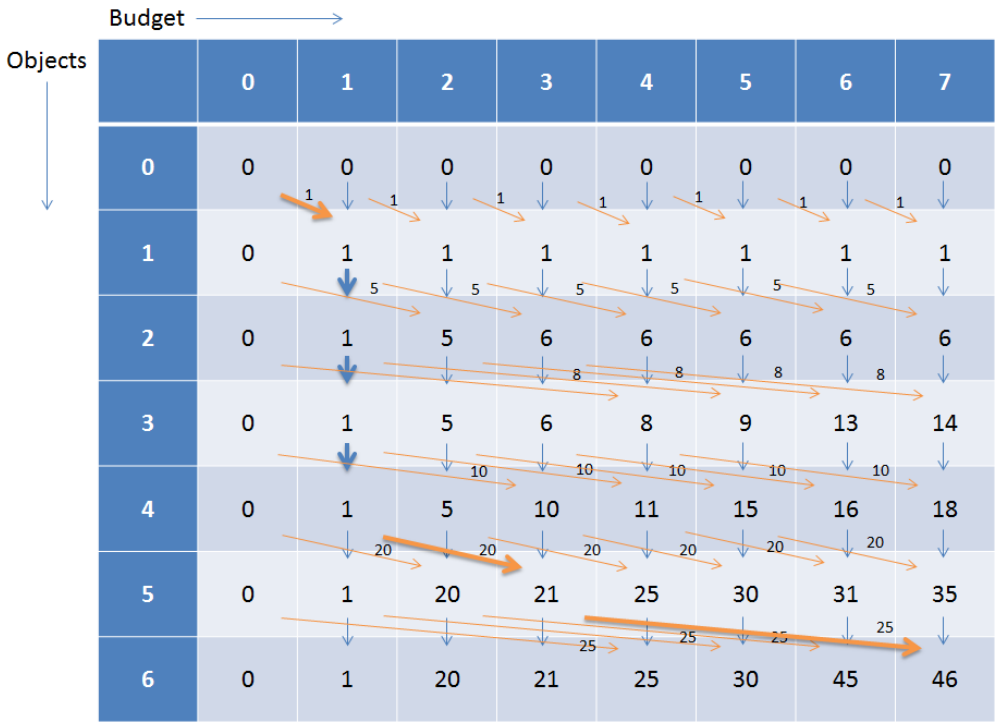

Figure 1: Solution Matrix. The arrows entering a cell indicate the two possible choices at that cell. The numbers over the red arrows indicate the profit of the i^{th} element. The bold arrows indicate the path to $OPT(n, W)$ showing the elements we finally select (Items 1,5, and 6 for a total profit of 46).