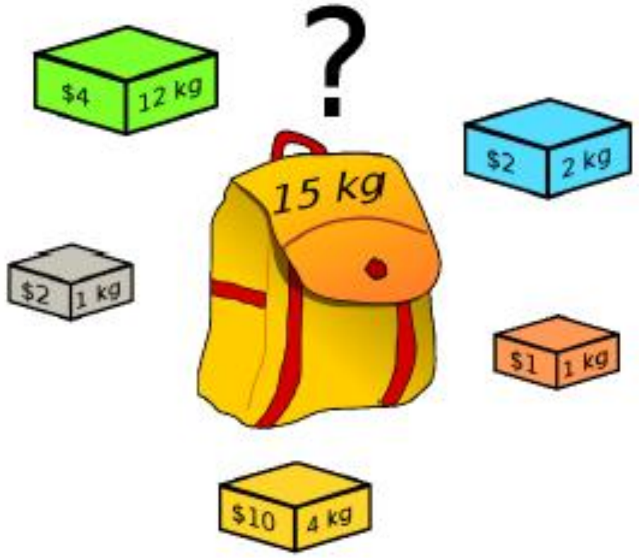
EE8212 Optimization Techniques for Engineers

Knapsack Problem



Introduction

Suppose we are planning a hiking trip and are therefore, interested in filling a knapsack with items that are considered necessary for the trip. There are N different item types that are deemed desirable. These could include bottle of water, apple, orange, sandwich, and so forth. Each item type has two attributes,

- a weight (or volume)
- a value that quantifies the level of importance associated with each unit of that type of item.

Since the knapsack has a limited weight (or volume) capacity, the problem of interest is to figure out how to load the knapsack with a combination of units of the specified types of items that yields the greatest total value

Applications of the Knapsack problem

A large variety of resource allocation problems can be cast in the framework of a knapsack problem. The general idea is to think of the capacity of the knapsack as the available amount of a resource and the item types as activities to which this resource can be allocated. Two quick examples are,

- the allocation of an advertising budget to the promotions of individual products
- the allocation of your effort to the preparation of final exams in different subjects.

Model description

Let us assume the following parameters,

w_k -	the weight of each type-k item, $k = 1, 2, \ldots, N$,
r_k -	the value associated with each type-k item, $k = 1, 2, \ldots, N$,
c -	the weight capacity of the knapsack.

Then, our problem can be formulated as:

Maximise $$\sum_{k=1}^{N} r_k x_k$$

Subject to:

$$\sum_{k=1}^{N} w_k x_k \leq c,$$

Where $x_1, x_2, x_3, \ldots\ldots\ldots, x_N$ are nonnegative integer valued decision variables, defined by

x_k – the number of type k items that are loaded to the knapsack.

Notice that since the x_k's are integer-valued, what we have is not an ordinary linear program, but rather an integer program. As such the Simplex algorithm cannot be applied to solve this problem.

As a simple numerical example, suppose we have: N = 3; $w_1 = 3$, $w_2 = 8$, and $w_3 = 5$; $r_1 = 4$, $r_2 = 6$, and $r_3 = 5$; and finally, c = 8. Observe that of the three item types, the first type has the greatest value per unit of weight. That is, of the three ratios,

$$\frac{r_1}{w_1} = \frac{4}{3}, \frac{r_2}{w_2} = \frac{6}{8}, and \frac{r_3}{w_3} = \frac{5}{5,},$$

The first ratio is the highest. Hence it seems natural to attempt to load as many type 1 items as possible into the knapsack. Since the capacity of the knapsack is 8, such an attempt will then result in the loading combination $x_1 = 2$, $x_2 = 0$, and $x_3 = 0$, with a total value of,

$$r_1x_1 + r_2x_2 + r_3x_3 = 4 \times 2 + 6 \times 0 + 5 \times 0 = 8,$$

Is this loading combination optimal? The fact that this combination leaves a wasted slack of 2 in the knapsack is discomforting. Indeed, it turns out that this combination is not optimal; and that the optimal combination is to let $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$, which achieves a total value of 9.

We now describe how to derive the optimal solution of this problem using dynamic programming

Stages and states

Observe that there is one decision to make for each item type. That is, we can imagine an assignment process in which a sequence of values is specified for the x_k's, one at a time. It follows that we can define one stage for each item type. Furthermore, it should be clear that this definition of stages is independent of the order in which the item types are listed.

To motivate the definition of states, imagine yourself as a consultant who is hired at the beginning of, say, stage k, where 1<= k <=N. This means that decisions for item types 1 through k−1 have already been made and these past decisions cannot be retracted. You are now in charge of making the remaining decisions for item types k through N. So, the question is: What do I need to know in order to make these remaining decisions? A little bit of reflection should convince you that the answer to this consultant question is: the remaining capacity of the knapsack at the beginning of stage k. We should, therefore, define the remaining capacity as the state.

Summery

Stages : Item Type (1<= k <=N)

States : The Remaining Capacity of the Knapsack (0 <= i <=C)

Optimal value function

As noted before, this will be a simple adaptation of the standard definition. In the language of the present problem, let,

$V_k(i)$ - the highest total value that can be achieved from item types k through N, assuming that the knapsack has a remaining capacity of i.

Our goal is to determine $V_1(c)$; in the simple numerical example above, this means that we are interested in $V_1(8)$.

Recurrence relation

Suppose the values of x_1 through x_{k-1} have all been assigned, and we are ready to make an assignment to x_k; that is, we are now in stage k. Suppose further that the knapsack at this point has a remaining capacity of i, where 0 <= i <= c; that is, we are in state i. Since each type-k item has a weight of w_k, we cannot assign a value greater than i/w_k to x_k. Moreover, observe that i/w_k is not necessarily an integer. It follows that the feasible range for x_k is, $0 \leq x_k \leq \lfloor i/w_k \rfloor$ where the notation $\lfloor x \rfloor$ is, for any given x, defined as the greatest integer less than or equal to x.

For any integer x_k in the range specified above, the immediate one-stage contribution to the total value in the knapsack is given by $r_k x_k$. Observe that as a result of such an assignment, or action, the capacity of the knapsack will be further reduced by $w_k x_k$. It follows that the "new" state at the next stage, namely stage k+1, will be $i - w_k x_k$. Therefore, the best possible "future" contribution to the total value in the knapsack is given by $V_{k+1}(i - w_k x_k)$.

Combining the discussions in the above two paragraphs now yields the following recurrence relation:

$$V_k(i) = \max_{0 \leq x_k \leq \lfloor i/w_k \rfloor} \left[r_k x_k + V_{k+1}(i - w_k x_k) \right]$$

With the recurrence relation in place, the final step of the solution procedure consists of the recursive computation of the $V_k(i)$'s.

Computation

We will illustrate the computation with the numerical example specified above. Recall that with three item types, the total number of stages is 3.

Since the specified capacity of the knapsack is 8, the highest possible state in any stage is 8. Suppose we are now in stage 3, which means that we are considering an allocation of type-3 items. The fact that every type-3 item has a weight of 5 implies that the state, or the remaining capacity of the knapsack, must be at least 5 to make a positive assignment to x_3 feasible. It follows that $V_3(0) = V_3(1) = V_3(2) = V_3(3) = V_3(4) = 0$. Next, a similar argument shows that for any state between 5 and 9, there is only enough capacity to accommodate a single type-3 item. Since $r_3 = 5$, this implies that $V_3(5) = V_3(6) = V_3(7) = V_3(8) = 5$. In summary, the boundary condition for our problem is given by the table below.

Stage 3:

i	$V_3(i)$	x_3^*
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	5	1
6	5	1
7	5	1
8	5	1

Note that the last column lists the optimal allocation, denoted by x_3^*, for every state in stage 3.

We now consider stage 2. Since every type-2 item has a weight of 8, the state, or the remaining capacity of the knapsack, must be at least 8 to make a positive assignment to x_2 feasible. Therefore, for any i from 0 through 7, the recurrence relation evaluates to,

$$\begin{aligned} V_2(i) &= \max_{0\le x_2\le 0}\left[r_2x_2 + V_3(i - w_2x_2)\right] \\ &= 6\times 0 + V_3(i - 8\times 0) \\ &= 0 + V_3(i), \end{aligned}$$

Where $V_3(i)$, is given in the previous table. For state 8, where x_2 can be either 0 or 1. The recurrence relation evaluates to,

$$\begin{aligned} V_2(8) &= \max_{0\le x_2\le 1}\left[r_2x_2 + V_3(8 - w_2x_2)\right] \\ &= \max\left[6X0 + V_3(8-8X0),\ 6X1 + V_3(8-8X1)\right] \\ &= \max\left[0 + V_3(8),\ 6 + V_3(0)\right] \\ &= \max\left[0+5,\ 6+0\right] \\ &= 6. \end{aligned}$$

These calculations are summarised in the table below.

Stage 2:

	Actions			x_2^*
I	x_2=0	x_2=1	V_2(i)	
0	0+0	-	0	0
1	0+0	-	0	0
2	0+0	-	0	0
3	0+0	-	0	0
4	0+0	-	0	0
5	0+5	-	5	0
6	0+5	-	5	0
7	0+5	-	5	0
8	0+5	6+0	6	1

(The dashes in the x_2 = 1 column indicate the infeasibility of that action.)

Finally, in stage 1, the only state is 8. Since $\lfloor 8/w_1 \rfloor = \lfloor 8/3 \rfloor = 2,$ it is feasible to allocate 0, 1,or 2 amount of type-1 items. Therefore the recurrence relation evaluates to,

$V_1(8) = \max_{0 \le x_2 \le 2} \left[r_1 x_1 + V_2(8 - w_1 x_1) \right]$

$= \max [4X0 + V_2(8\text{-}3X0), 4X1 + V_2(8\text{-}3X1), 4X2 + V_2(8\text{-}3X2)]$

$= \max [0 + V_2(8) , 4 + V_2(5), 8 + V_2(2)]$

$= \max [0+6, 4+5, 8+0]$

$= 9.$

These calculations are summarised in the table below.

Stage 1:

	Actions				
i	x_1=0	x_1=1	x_1=2	V_1(i)	x_1^*
8	0+6	4+5	8+0	9	1

Since V_1(8) =9, the highest total value the knapsack can hold is 9.

The sequence of optimal actions can be read from the above tables sequentially.
From the stage-1 table, we have x_1^* = 1.
With x_1^* = 1, the remaining capacity, or the state, at stage 2 will be 8 − 3 × 1 = 5;
therefore, from the row with i = 5 in the stage-2 table, we pick up x_2^* = 0.
This implies that the remaining capacity, or the state, at stage 3 will be 5 − 8 × 0 = 5;
and a reading of the row with i = 5 in the stage-3 table shows that x_3^* = 1.
Thus, the optimal policy prescribes a knapsack that is loaded with one type-1 item, no type-2 item, and one type-3 item.
This completes the solution of the numerical example.