



Knapsack Problem of Algorithm Design and Analysis

Sapna Mittal

Assistant Professor, Computer Science,
Post Graduate Department of Computer Science and Applications,
R.S.D College, Ferozepur, Punjab, India

Abstract: Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. The knapsack problem is in combinatorial optimization problem. It appears as a sub problem in many, more complex mathematical models of real-world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy the ignored constraints.

I. Introduction

In many cases of resource allocation along with some constraint, the problem can be derived in a similar way of Knapsack problem. Following is a set of example.

- Finding the least wasteful way to cut raw materials
- Portfolio optimization
- Cutting stock problems
-

II. Problem Scenario

A thief is robbing a store and can carry a maximal weight of W into his knapsack. There are n items available in the store and weight of i^{th} item is w_i and its profit is p_i . What items should the thief take?

In this context, the items should be selected in such a way that the thief will carry those items for which he will gain maximum profit. Hence, the objective of the thief is to maximize the profit.

Based on the nature of the items, Knapsack problems are categorized as

- Fractional Knapsack
- Knapsack

III. Fractional Knapsack

In this case, items can be broken into smaller pieces; hence the thief can select fractions of items.

According to the problem statement,

- There are n items in the store
- Weight of i^{th} item $w_i > 0$
- Profit for i^{th} item $p_i > 0$ and Capacity of the Knapsack is W

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction x_i of i^{th} item.

$$0 \leq x_i \leq 1$$

The i^{th} item contributes the weight $x_i.w_i$ to the total weight in the knapsack and profit $x_i.p_i$ to the total profit. Hence, the objective of this algorithm is to maximize $\sum_{n=1}^n (x_i.p_i)$ subject to constraint,

$$\sum_{n=1}^n (x_i.w_i) \leq W$$

It is clear that an optimal solution must fill the knapsack exactly; otherwise we could add a fraction of one of the remaining items and increase the overall profit.

Thus, an optimal solution can be obtained by

$$\sum_{n=1}^n (x_i.w_i) = W$$

In this context, first we need to sort those items according to the value of p_i/w_i , so that $p_{i+1}/w_{i+1} \leq p_i/w_i$. Here, x is an array to store the fraction of items.

IV. Analysis

If the provided items are already sorted into a decreasing order of p_i/w_i , then the while loop takes a time in $O(n)$; Therefore, the total time including the sort is in $O(n \log n)$.

Example

Let us consider that the capacity of the knapsack $W = 60$ and the list of provided items are shown in the following table –

Item	A	B	C	D
Profit	280	100	120	120
Weight	40	10	20	24
Ratio (p_i/w_i)	7	10	6	5

As the provided items are not sorted based on p_i/w_i .

After sorting, the items are as shown in the following table.

Item	B	A	C	D
Profit	100	280	120	120
Weight	10	40	20	24
Ratio (p_i/w_i)	10	7	6	5

Solution

After sorting all the items according to p_i/w_i

. First all of **B** is chosen as weight of **B** is less than the capacity of the knapsack. Next, item **A** is chosen, as the available capacity of the knapsack is greater than the weight of **A**. Now, **C** is chosen as the next item. However, the whole item cannot be chosen as the remaining capacity of the knapsack is less than the weight of **C**.

Hence, fraction of **C** (i.e. $(60 - 50)/20$) is chosen.

Now, the capacity of the Knapsack is equal to the selected items. Hence, no more item can be selected.

The total weight of the selected items is $10 + 40 + 20 * (10/20) = 60$

And the total profit is $100 + 280 + 120 * (10/20) = 380 + 60 = 440$

This is the optimal solution. We cannot gain more profit selecting any different combination of items.

V. Dynamic-Programming Approach

In 0-1 Knapsack, items cannot be broken which means the thief should take the item as a whole or should leave it. This is reason behind calling it as 0-1 Knapsack.

Hence, in case of 0-1 Knapsack, the value of x_i can be either **0** or **1**, where other constraints remain the same.

0-1 Knapsack cannot be solved by Greedy approach. Greedy approach does not ensure an optimal solution. In many instances, Greedy approach may give an optimal solution.

The following examples will establish our statement.

Example-1

Let us consider that the capacity of the knapsack is $W = 25$ and the items are as shown in the following table.

Item	A	B	C	D
Profit	24	18	18	10
Weight	24	10	10	7

Without considering the profit per unit weight (p_i/w_i), if we apply Greedy approach to solve this problem, first item **A** will be selected as it will contribute maximum profit among all the elements.

After selecting item **A**, no more item will be selected. Hence, for this given set of items total profit is **24**. Whereas, the optimal solution can be achieved by selecting items, **B** and **C**, where the total profit is $18 + 18 = 36$.

Example-2

Instead of selecting the items based on the overall benefit, in this example the items are selected based on ratio p_i/w_i . Let us consider that the capacity of the knapsack is $W = 30$ and the items are as shown in the following table.

Item	A	B	C
Price	100	280	120
Weight	10	40	20
Ratio	10	7	6

Using the Greedy approach, first item **A** is selected. Then, the next item **B** is chosen. Hence, the total profit is $100 + 280 = 380$. However, selecting items, **B** and **C**, where the total profit is $280 + 120 = 400$, can achieve the optimal solution of this instance.

Hence, it can be concluded that Greedy approach may not give an optimal solution.

To solve 0-1 Knapsack, Dynamic Programming approach is required.

References

- [1]. Mathews, G. B. (25 June 1897). "On the partition of numbers" (PDF). Proceedings of the London Mathematical Society. 28: 486–490. doi:10.1112/plms/s1-28.1.486. Dantzig, Tobias. Numbers: The Language of Science, 1930.
- [2]. Skiena, S. S. (September 1999). "Who is Interested in Algorithms and Why? Lessons from the Stony Brook Algorithm Repository". AGM SIGACT News. 30 (3): 65–74. doi:10.1145/333623.333627. ISSN 0163-5700. Archived from the original on 2016-03-06.
- [3]. Kellerer, Pferschy, and Pisinger 2004, p. 449
- [4]. Kellerer, Pferschy, and Pisinger 2004, p. 461