International journal of Advance Engineering and Research Development (IJAERD)
Volume 1 Issue 1, February 2014, ISSN: 2348 - 4470

Comparision of Dynamic and Greedy Approach for Knapsack Problem

Jay Vala[1], Jaymit Pandya[2], Dhara Monaka[3]

[1]*Assist. Prof. I.T. Department G H Patel College of Engg & Tech jayvala1623@gmail.com*

[2]*Assist. Prof. I.T. Department G H Patel College of Engg & Tech erpandyajaymit@gmail.com*

[3]*Assist.Prof. B.C.A. Department Nandkunvarba BCA Mahila College, dhara.monaka123@gmail.com*

Abstract— The aim of paper is to analyze few algorithms of the 0/1 Knapsack Problem. This problem is a combinatorial optimization problem in which one has to maximize the benefit of objects without exceeding capacity. As it is an NP-complete problem, an exact solution for a large input is not possible. Hence, paper presents a comparative study of the Greedy and dynamic methods. It also gives complexity of each algorithm with respect to time and space requirements. Our experimental results show that the most promising approaches is dynamic programming.

Keywords-knapsack, dynamic programming, greedy programming, NP-Complete, complexity

I. INTRODUCTION

The Knapsack Problem is an example of a combinatorial optimization problem, which seeks for a best solution from among many other solutions. It is concerned with a knapsack that has positive integer volume (or capacity) *V*. There are *n* distinct items that may potentially be placed in the knapsack. Item *i* has a positive integer volume *Vi* and positive integer benefit *Bi*. In addition, there are *Qi* copies of item *i* available, where quantity *Qi* is a positive integer satisfying 1 <= *Qi* <= Infinity. Let *Xi* determines how many copies of item *i* are to be placed into the knapsack. The goal is to:

Maximize

$$\sum_{i=1}^{N} B_i X_i$$

Subject to the constraints

$$\sum_{i=1}^{N} V_i X_i \leq V$$

And

$$0 \leq X_i \leq Q_i.$$

If one or more of the *Qi* is infinite, the KP is *unbounded;* otherwise, the KP is *bounded* [1]. The bounded KP can be either *0-1 KP* or *Multiconstraint KP*. If *Qi* = 1 for *i* = 1, 2, …, *N*, the problem is a *0-1 knapsack problem* In the current paper, we have worked on the bounded *0-1 KP*, where we cannot have more than one copy of an item in the knapsack[1].

II. DIFFERENT APPROACHES TO PROBLEM

1) Greedy Approach

A thief robbing a store and can carry a maximal weight of *w* into their knapsack. There are n items and i^{th} item weigh w_i and is worth v_i dollars. What items should thief take? This version of problem is known as Fractional knapsack problem.
The setup is same, but the thief can take fractions of items, meaning that the items can be broken into smaller pieces so that thief may decide to carry only a fraction of x_i of item *i*, where $0 \leq x_i \leq 1$[2][3].

2) Dynamic Approach

Again a thief robbing a store and can carry a maximal weight of *w* into their knapsack. There are n items and *ith* item weigh w_i and is worth v_i dollars. What items should thief take? This version of problem is known as 0-1 knapsack problem.
The setup is the same, but the items may not be broken into smaller pieces, so thief may decide either to take an item or to leave it (binary choice), but may not take a fraction of an item[2][3].

III. GREEDY ALOGRITHM

George Dantzig proposed a greedy approximation algorithm to solve the unbounded knapsack problem.[1] His version sorts the items in decreasing order of value per unit of weight, v_i/w_i. It then proceeds to insert them into the sack, starting with as many copies as possible of the first kind of item until there is no longer space in the sack for more. Provided that there is an unlimited supply of each kind of item, if m is the maximum value of items that fit into the sack, then the greedy algorithm is guaranteed to achieve at least a value of $m/2$. However, for the bounded problem, where the supply of each kind of item is limited, the algorithm may be far from optimal[4].

Pseudo code for greedy knapsack algorithm is given below.

Input: v- array of values, w-array of weights, c-capacity
Output: Profit of Knapsack

1. Load <- 0
2. i<-1
3. while load < c and I <= n
4. do if w_i < c – load
5. then take all item i
6. else take (c – load) /w_i of item i
7. i<-i+1

III. DYNAMIC ALGORITHM

A similar dynamic programming solution for the 0/1 knapsack problem also runs in pseudo-polynomial time. Assume $w_1, w_2, \ldots, w_n, W$ are strictly positive integers.
Define $m[i, w]$ to be the maximum value that can be attained with weight less than or equal to w using items up to i[5].

We can define $m[i, w]$ recursively as follows:

- $m[i, w] = m[i-1, w]$ if $w_i > w$ (the new item is more than the current weight limit)
- $m[i, w] = \max(m[i-1, w], m[i-1, w-w_i] + v_i)$ if $w_i \leqslant w$.

The solution can then be found by calculating $m[n, W]$. To do this efficiently we can use a table to store previous computations.

Pseudo code for dynamic knapsack algorithm is given below.

Input: $\{w_1, w_2, \cdots w_n\}$, W, $\{b_1, b_2, \cdots b_n\}$
Output: $B[n, W]$

1. for $w \leftarrow 0$ to W do // *row* 0
2. $B[0,w] \leftarrow 0$
3. for $k \leftarrow 1$ to n do // rows 1 to n
4. $B[k, 0] \leftarrow 0$ // element in column 0
5. for $w \leftarrow 1$ to W do // elements in columns 1 to W
6. if ($w_k \leq w$) and ($B[k\text{-}1, w\text{-} w_k] + b_k > B[k\text{-}1, w]$)
7. then $B[k, w] \leftarrow B[k\text{-}1, w\text{-} w_k] + b_k$
8. else $B[k, w] \leftarrow B[k\text{-}1, w]$

IV. RESULTS

Implemented knapsack problem with different values of weight and profit or value in Turbo C.
If we consider a data value is w={1, 2, 5, 6, 7}, v={1, 6, 18, 22, 28} and Carrying capacity W= 11 then output of greedy is:

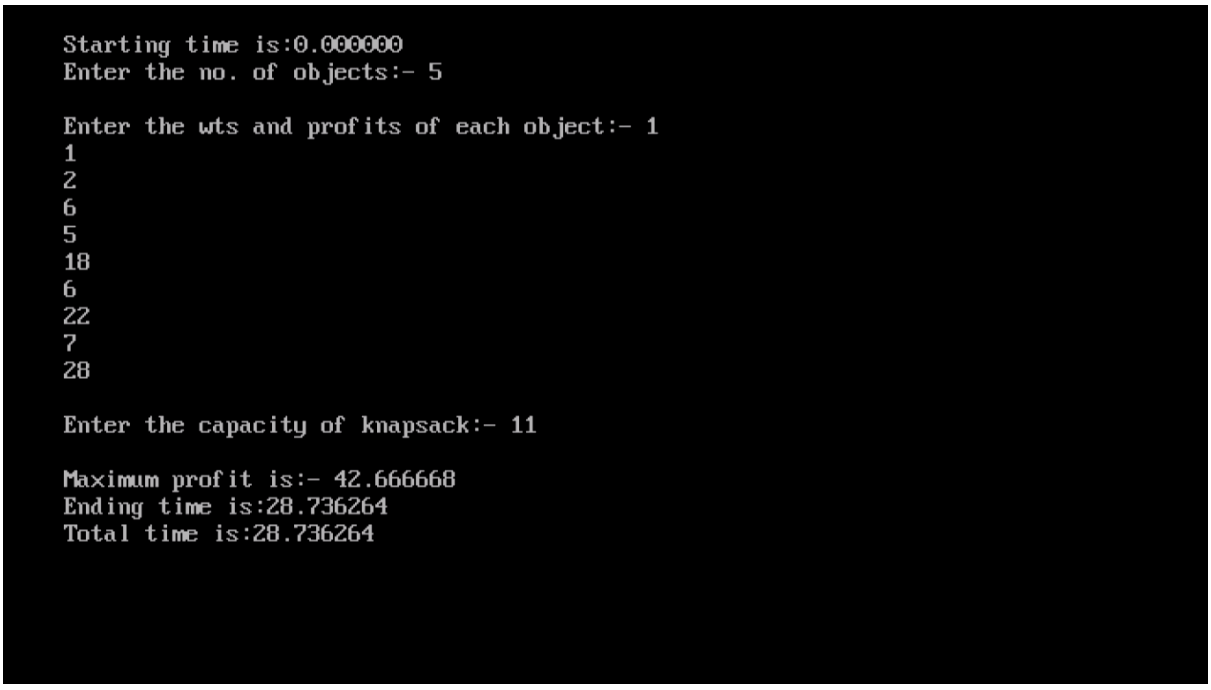

Fig. 1 Solved by Greedy approach

Same data values and solving by Dynamic programming.

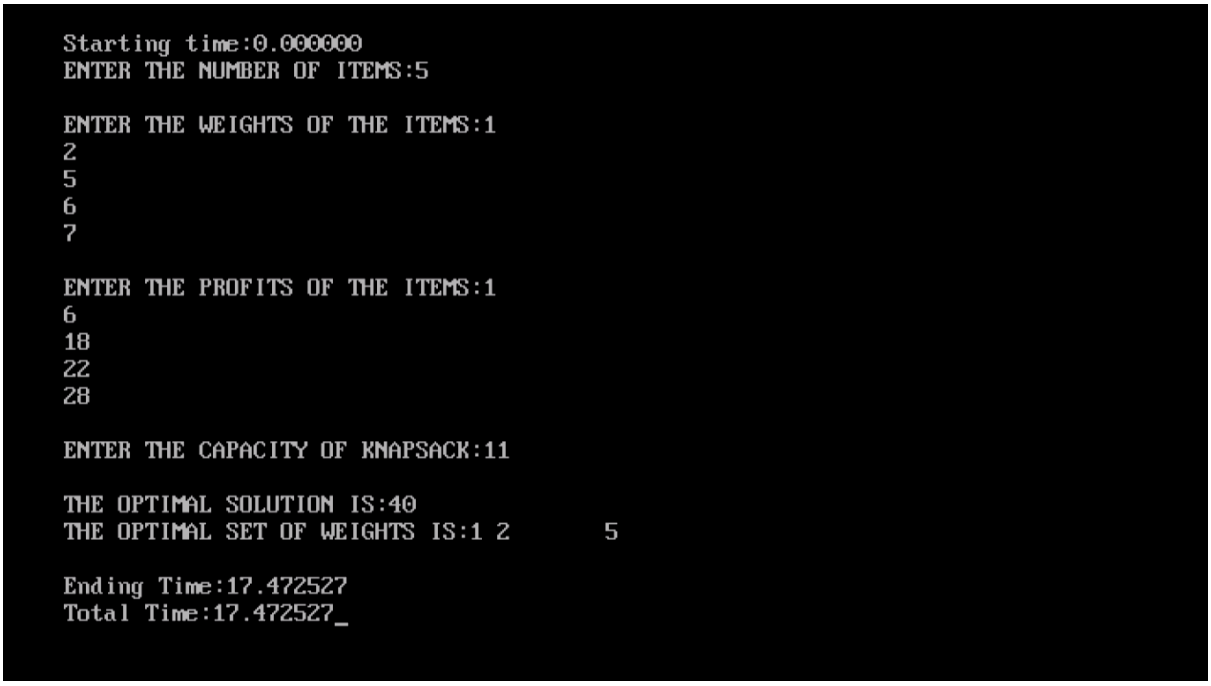

Fig. 2 Solved by Dynamic programming

After implemented knapsack problem in c programming for different values of weight and profit. Result of both methods gives optimal solution and time .

Method	Input Data	Capacity	Profit	Time
Greedy	W={2,3,4,5} V={3,4,5,6}	9	12	16.86
	W={1,2,5,6,7} , V={1,6,18,22,28}	11	42.67	28.73
	W={10,20,30,40,50} V={10,30,66,50,60}	100	158	26.68
Dynamic	W={2,3,4,5} V={3,4,5,6}	9	12	12.69
	W={1,2,5,6,7} , V={1,6,18,22,28}	11	40	17.47
	W={10,20,30,40,50} V={10,30,66,50,60}	100	156	19.65

Table 1. Comparison of Greedy and Dynamic with different values.

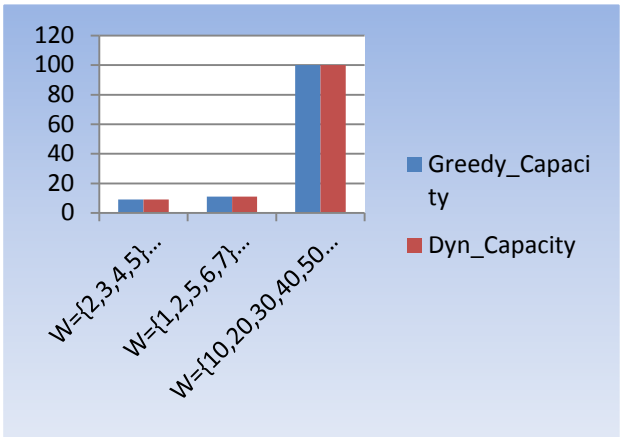

Fig 3(a)

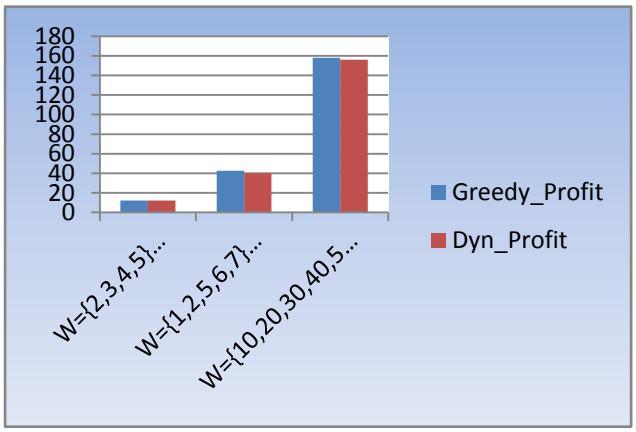

Fig 3(b)

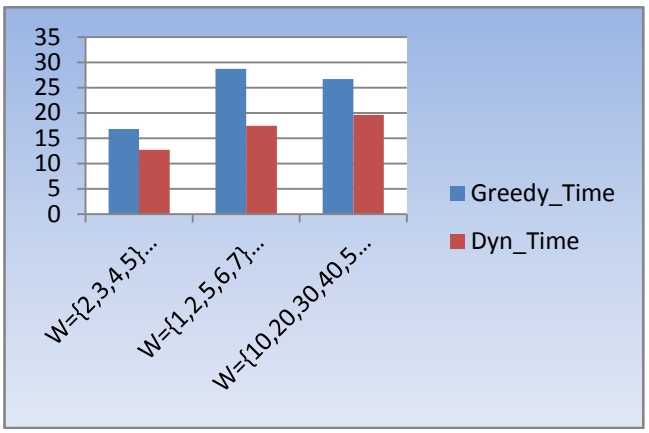

Fig 3(c)

Fig 3 (a, b, c) Comparison of greedy and dynamic

V. CONCLUSION

In this paper we conclude that for particular one knapsack problem we can implement two methods greedy and dynamic. But when we implemented both method for different dataset values then we notice something is like, we consider comparison parameter as optimal profit or total value for filling knapsack using available weight then greedy is better than dynamic. If we consider time then dynamic take less amount of time compare with greedy. so we can say that dynamic is better than greedy with respect to time.

REFERENCES

[1]. George B. Dantzig, Discrete-Variable Extremum Problems, Operations Research Vol. 5, No. 2, April 1957, pp. 266–288,doi:10.1287/opre.5.2.266
[2] Gossett, Eric. Discreet Mathematics with Proof. New Jersey: Pearson Education Inc., 2003.
[3] Levitin, Anany. The Design and Analysis of Algorithms. New Jersey: Pearson Education Inc., 2003.
[4] Mitchell, Melanie. An Introduction to Genetic Algorithms. Massachusettss: The MIT Press, 1998.
[5] Obitko, Marek. "Basic Description." IV. Genetic Algorithm. Czech Technical University (CTU). http://cs.felk.cvut.cz/~xobitko/ga/gaintro.html
[6] Different Approaches to Solve the 0/1 Knapsack Problem. Maya Hristakeva, Dipti Shrestha; Simpson Colleges

