5. KNAPSACK PROBLEM

Given n items of known weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$ and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack.

Real time examples:

- A Thief who wants to steal the most valuable loot that fits into his knapsack,
- Atransportplanethathastodeliverthemostvaluablesetofitemstoaremotelocation without exceeding the plane's capacity.

The exhaustive-search approach to this problem leads to generating all the subsets of the set of n items given, computing the total weight of each subset in order to identify feasible subsets (i.e., the ones with the total weight not exceeding the knapsack capacity), and finding a subset of the largest value among them.

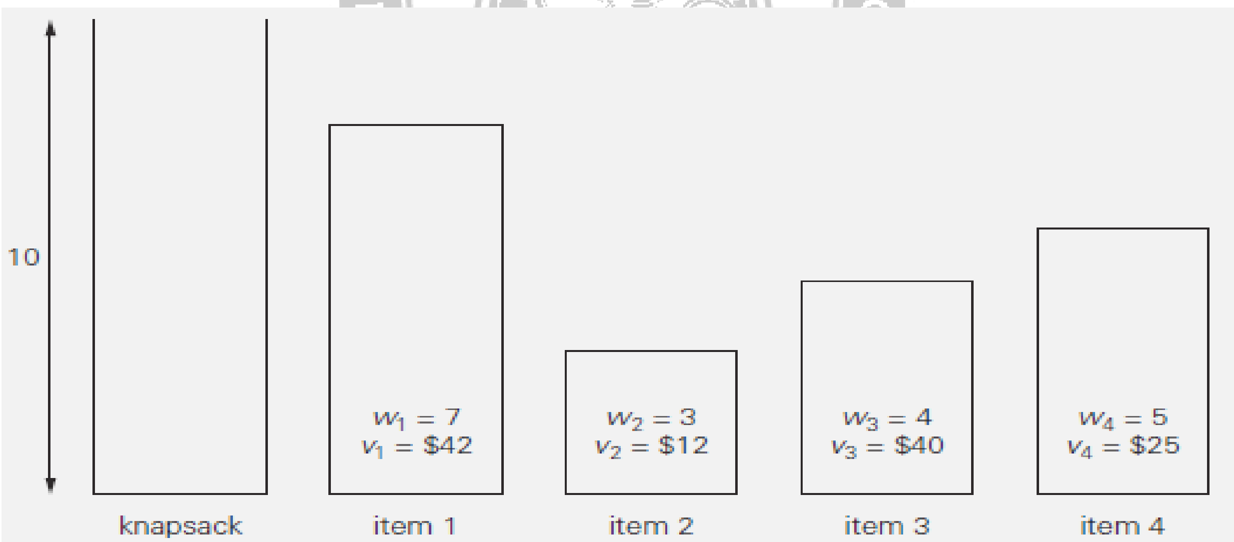

FIGURE 2.5 Instance of the knapsack problem

Subset	Total weight	Total value
Φ	0	\$0
{1}	7	\$42
{2}	3	\$12

{3}	4	$40
{4}	5	$25
{1, 2}	10	$54
{1, 3}	11	not feasible
{1, 4}	12	not feasible
{2, 3}	7	$52
{2, 4}	8	$37
{3, 4}	**9**	**$65 (Maximum-Optimum)**
{1, 2, 3}	14	not feasible
{1, 2, 4}	15	not feasible
{1, 3, 4}	16	not feasible
{ 2, 3, 4}	12	not feasible
{1, 2, 3, 4}	19	not feasible

FIGURE 2.6 knapsack problem's solution by exhaustive search. The information about the optimal selection is in bold.

Time efficiency: As given in the example, the solution to the instance of Figure 2.5 is given in Figure 2.6. Since the *number of subsets of an n-element set is* 2^n, the exhaustive search leads to a $\Omega(2^n)$ algorithm, no matter how efficiently individual subsets are generated.

Note: Exhaustive search of both the traveling salesman and knapsack problems leads to extremely inefficient algorithms on every input. In fact, these two problems are the best-known examples of ***NP-hard problems***. **No polynomial-time** algorithm is known for any *NP*-hard problem. Moreover, most computer scientists believe that such algorithms do not exist. some sophisticated approaches like **backtracking** and **branch-and-bound** enable us to solve some instances but not all instances of these in less than exponential time. Alternatively, we can use one of many **approximation algorithms.**