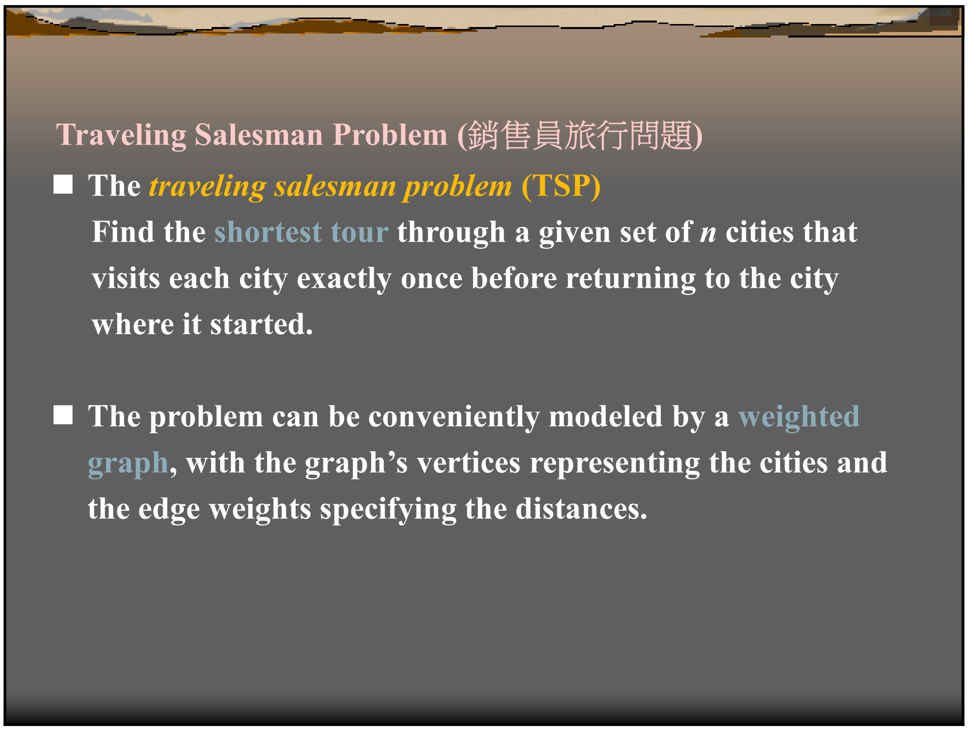
3.4 Exhaustive Search

Exhaustive search **is simply a brute-force approach to combinatorial problems. It suggests generating each and every element of the problem domain, selecting those of them that satisfy all the constraints, and then finding a desired element (e.g., the one that optimizes some objective function).**
(窮舉搜尋法只是一種解決組合問題的暴力法。 它建議生成問題定義域的每個元素，選擇滿足所有限制條件的元素，然後找到所需的元素（例如，最佳化某些目標函數的元素））。

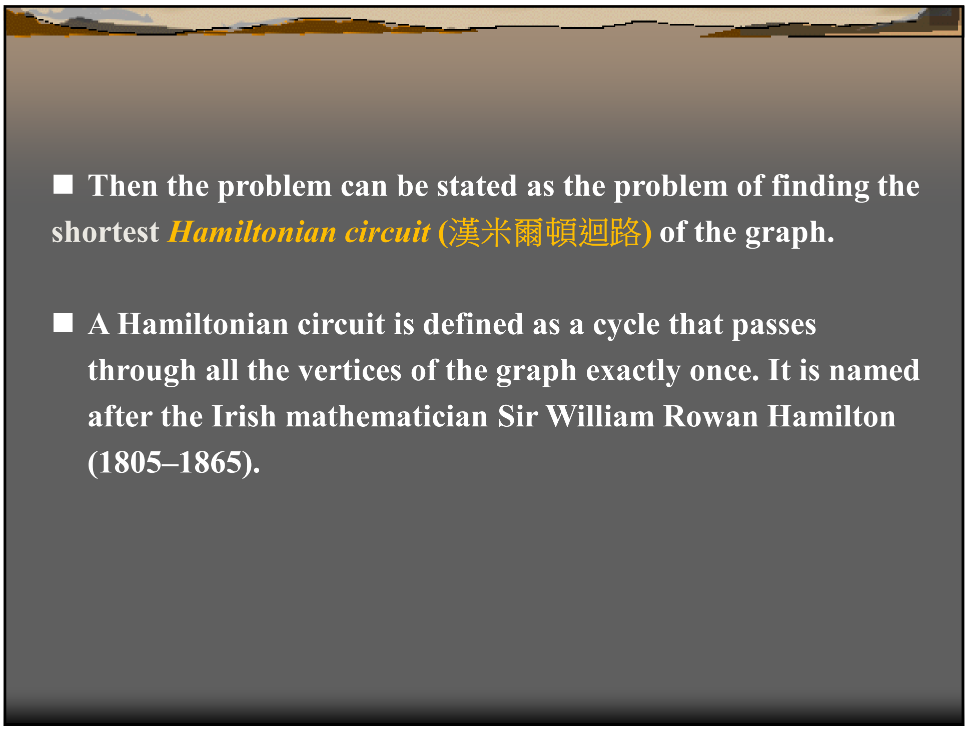
Note that although the idea of exhaustive search is quite straightforward, its implementation typically requires an algorithm for generating certain combinatorial objects.
(請注意，儘管窮舉搜尋法的想法非常簡單，但其實現通常需要一種演算法來生成某些組合物件。)



Traveling Salesman Problem (銷售員旅行問題)

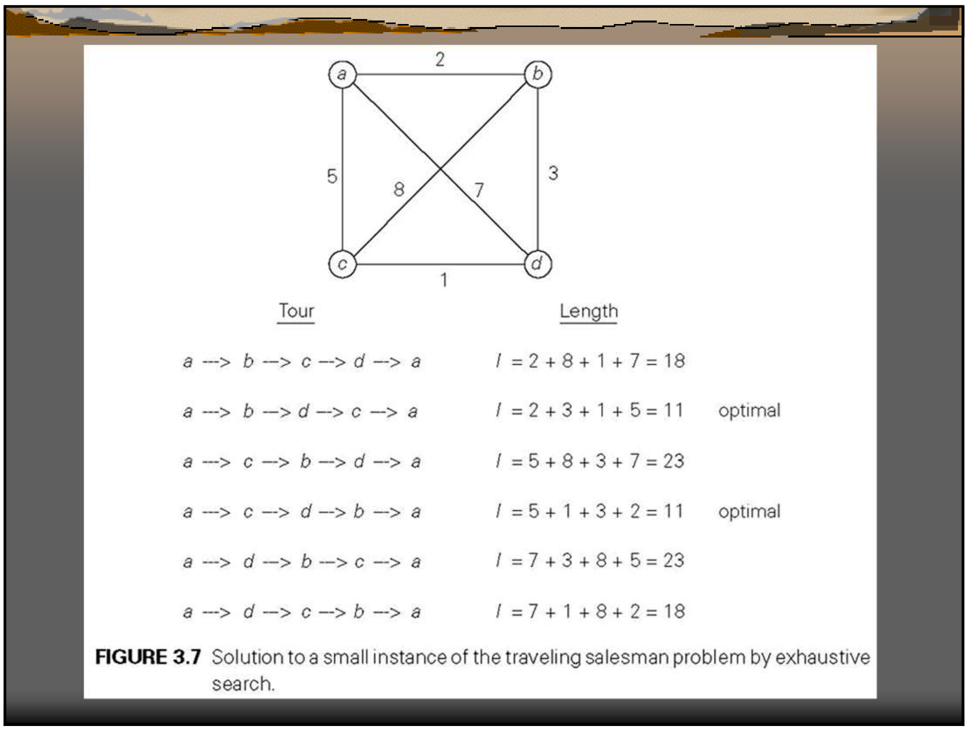
- The *traveling salesman problem* (TSP)
  Find the shortest tour through a given set of *n* cities that visits each city exactly once before returning to the city where it started.

- The problem can be conveniently modeled by a weighted graph, with the graph's vertices representing the cities and the edge weights specifying the distances.



- Then the problem can be stated as the problem of finding the shortest *Hamiltonian circuit* (漢米爾頓迴路) of the graph.

- A Hamiltonian circuit is defined as a cycle that passes through all the vertices of the graph exactly once. It is named after the Irish mathematician Sir William Rowan Hamilton (1805–1865).



Tour	Length	
a —> b —> c —> d —> a	$l = 2 + 8 + 1 + 7 = 18$	
a —> b —> d —> c —> a	$l = 2 + 3 + 1 + 5 = 11$	optimal
a —> c —> b —> d —> a	$l = 5 + 8 + 3 + 7 = 23$	
a —> c —> d —> b —> a	$l = 5 + 1 + 3 + 2 = 11$	optimal
a —> d —> b —> c —> a	$l = 7 + 3 + 8 + 5 = 23$	
a —> d —> c —> b —> a	$l = 7 + 1 + 8 + 2 = 18$	

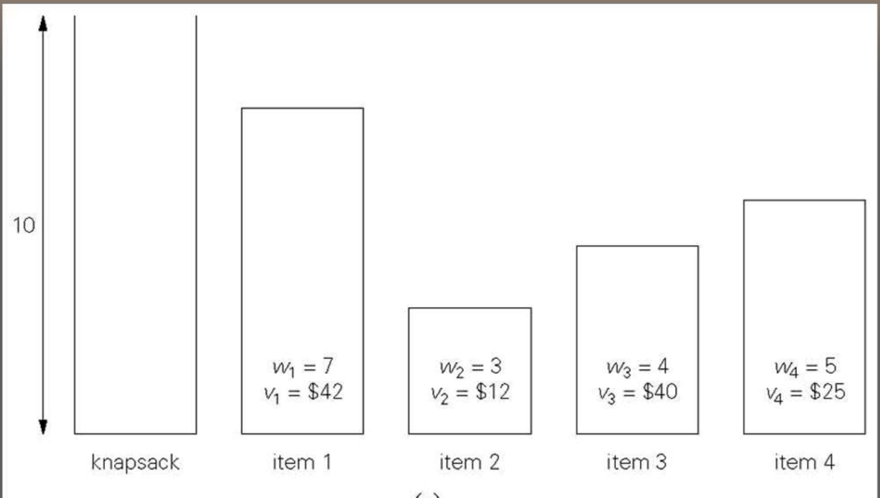
FIGURE 3.7 Solution to a small instance of the traveling salesman problem by exhaustive search.

Knapsack Problem

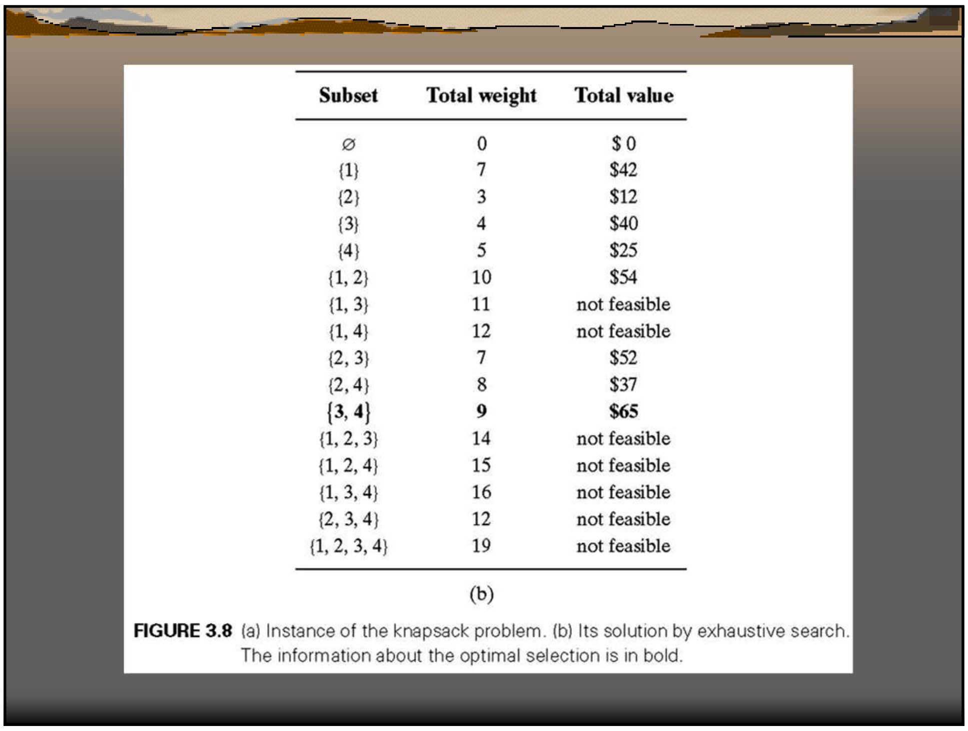
Given n items of known weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$ and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack.

For both the traveling salesman and knapsack problems considered above, exhaustive search leads to algorithms that are extremely inefficient on every input. In fact, these two problems are the best-known examples of so called ***NP-hard problems***.

No polynomial-time algorithm is known for any *NP*-hard problem. Alternatively, we can use one of many approximation algorithms, such as those described in Section 12.3.



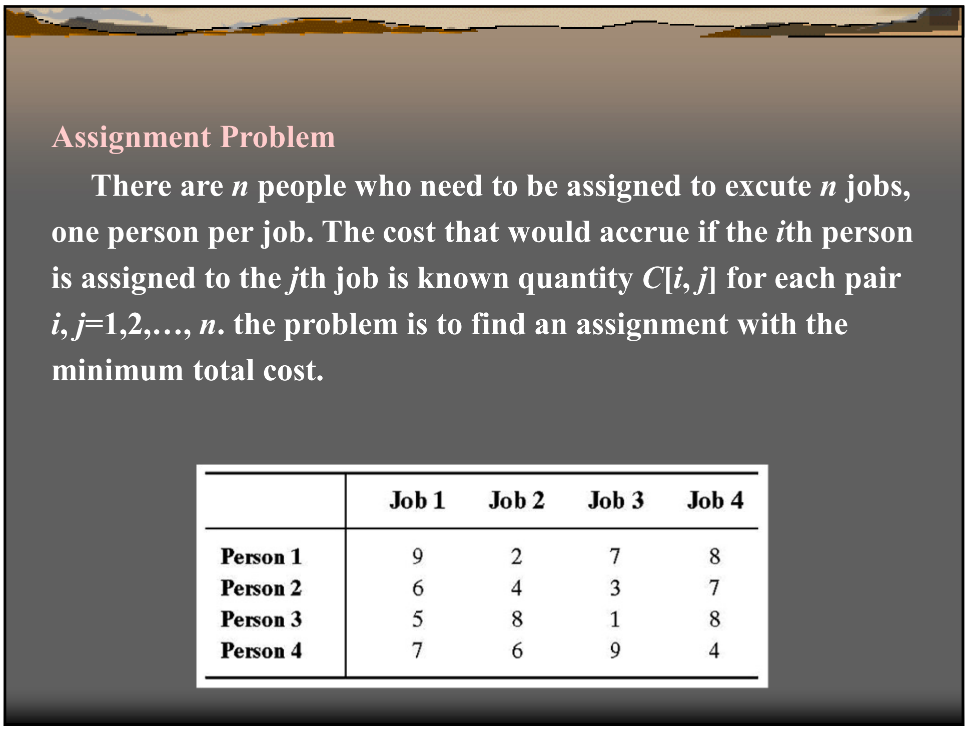
(a)



Subset	Total weight	Total value
∅	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**
{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

(b)

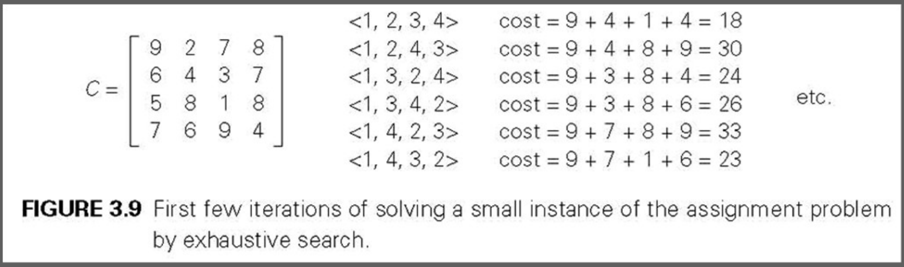
FIGURE 3.8 (a) Instance of the knapsack problem. (b) Its solution by exhaustive search. The information about the optimal selection is in bold.



Assignment Problem

There are *n* people who need to be assigned to excute *n* jobs, one person per job. The cost that would accrue if the *i*th person is assigned to the *j*th job is known quantity *C*[*i*, *j*] for each pair *i*, *j*=1,2,…, *n*. the problem is to find an assignment with the minimum total cost.

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4



$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

<1, 2, 3, 4>	cost = 9 + 4 + 1 + 4 = 18	
<1, 2, 4, 3>	cost = 9 + 4 + 8 + 9 = 30	
<1, 3, 2, 4>	cost = 9 + 3 + 8 + 4 = 24	
<1, 3, 4, 2>	cost = 9 + 3 + 8 + 6 = 26	etc.
<1, 4, 2, 3>	cost = 9 + 7 + 8 + 9 = 33	
<1, 4, 3, 2>	cost = 9 + 7 + 1 + 6 = 23	

FIGURE 3.9 First few iterations of solving a small instance of the assignment problem by exhaustive search.

Since the number of permutations to be considered for the general case of the assignment problem is *n*!, exhaustive search is impractical for all but very small instances of the problem. Fortunately, there is a much more efficient algorithm for this problem called the *Hungarian method* (匈牙利法) after the Hungarian Mathematicians König and Egerváry, **whose work underlies the method (see, e.g., [Kol95]).**