Dynamic Programming

The Knapsack Problem

Algorithms: Design and Analysis, Part II

Problem Definition

Input: n items. Each has a value:

- Value v_i (nonnegative)
- Size w_i (nonnegative and integral)
- Capacity W (a nonnegative integer)

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes $\sum_{i \in S} v_i$ subject to $\sum_{i \in S} w_i \leq W$.

Developing a Dynamic Programming Algorithm

Step 1: Formulate recurrence [optimal solution as function of solutions to "smaller subproblems"] based on a structure of an optimal solution.

Let $S =$ a max-value solution to an instance of knapsack.

Case 1: Supose item $n \notin S$.
$\Rightarrow S$ must be optimal with the first $n-1$ items (same capacity W)
[If S^* were better than S with respect to 1st $n-1$ items, then this equally true w.r.t. all n items - contradiction]

Optimal Substructure

Case 2: Suppose item $n \in S$. Then $S - \{n\}$ ...

A) is an optimal solution with respect to the 1st $n-1$ items and capacity W.

B) is an optimal solution with respect to the 1st $n-1$ items and capacity $W - v_n$.

C) is an optimal solution with respect to the 1st $n-1$ items and capacity $W - w_n$.

D) might not be feasible for capacity $W - w_n$.

Proof: If S^* has higher value than $S - \{n\}$ + total size $\leq W - w_n$, then $S^* \cup \{n\}$ has size $\leq W$ and value more than S [contradiction]