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 intergral)
- capacity W (a nonnegative integer)

Output: a subset $S \subseteq \{1,2,3,\ldots,n\}$

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 structure of an optimal solution.

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

Case 1: Suppose 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 w.r.t. the 1st $(n-1)$ items and capacity $W - v_n$

(C) is an optimal solution w.r.t. 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 $\le W - w_n$, then $S^* \cup \{n\}$ has size $\le W$ and value more than S [contradiction]

Tim Roughgarden