Approximation Algorithms for NP-Complete Problems

A Greedy Knapsack Heuristic

Algorithms: Design and Analysis, Part II

Strategies for NP-Complete Problems

(1) Identify computationally tractable special cases.

Example: Knapsack instances with small capacity [i.e., knapsack capacity W = polynomial in number of items n]

(2) Heuristics $\rightarrow$ today

- Pretty good greedy heuristic
- Excellent dynamic programming heuristic

$\}$ $\rightarrow$ For Knapsack

(3) Exponential time but better than brute-force search

Example: $O(nW)$-time dynamic programming vs. $O(2^n)$ brute-force search.

Ideally: Should provide a performance guarantee (i.e., "almost correct") for all (or at least many) instances.

Knapsack Revisited

Input: n items. Each has a positive value v_i and a size w_i. Also, knapsack capacity is W.

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that

$$\begin{array}{ll} \text{Maximizes} & \sum_{i \in S} v_i \\ \text{Subject to} & \sum_{i \in S} w_i \leq W \end{array}$$

A Greedy Heuristic

Motivation: Ideal items have big value, small size.

Step 1: Sort and reindex item so that

$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$ [i.e., nondecreasing "bang-per-buck"]

Step 2: Pack items in this order until one doesn't fit, then halt.

Example:

$$\text{W=5} \quad \begin{array}{ll} v_1 = 2 & w_1 = 1 \\ v_2 = 4 & w_2 = 3 \\ v_3 = 3 & w_3 = 3 \end{array} \quad \Rightarrow \text{Greedy gives } \{1, 2\} \text{ [also optimal]}$$

Quiz

Consider a Knapsack instance with $W = 1000$ and

$v_1 = 2 \quad w_1 = 1$

$v_2 = 1000 \quad w_2 = 1000$

Question: What is the value of the greedy solution and the optimal solution, respectively?

A) 2 and 1000

B) 2 and 1002

C) 1000 and 1002

D) 1002 and 1002

A Refined Greedy Heuristic

Upshot: Greedy solution can be arbitrarily bad relative to an optimal solution.

Fix: Add:

Step 3: Return either the Step 2 solution, or the maximum valuable item, whichever is better.

Theorem: Value of the 3-step greedy solution is always $\geq 50\% \cdot$ value of an optimal solution. [Also, runs in $O(n \log n)$ time]

[i.e., a "$\frac{1}{2}$-approximation algorithm"]