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

$\Big\}\rightarrow$ 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:

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

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) $\boxed{2 \text{ and } 1000}$

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"]

Approximation Algorithms for NP-Complete Problems

Analysis of a Greedy Knapsack Heuristic

Algorithms: Design and Analysis, Part II

Performance Guarantee

Theorem: Value of the 3-step greedy algorithm's solution is always $\geq 50\% \cdot$ value of an optimal solution.

Thought experiment: What if we were allowed to fill fully the knapsack using a suitable "fraction" (like 70%) of item (k+1)? [The value of which is "pro-rated"]

$\Rightarrow$ Will call this the "greedy fractional solution"

Example: $W = 3$, $v_1 = 3$, $v_2 = 2$, $w_1 = w_2 = 2$

get 100% get 50%

$\Rightarrow$ Greedy fractional solution has value 4

Quiz

Question: Let F = value of greedy fractional solution and OPT = value of optimal (non-fractional) solution. Which of the following is true?

A) $F = OPT$ for every knapsack instance

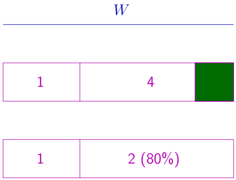
B) $F > OPT$ for every knapsack instance

C) $F \leq OPT$ for every instance, and can be strict

C) $F \geq OPT$ for every instance, and can be strict

Proof Sketch

Claim: Greedy fractional solution at least as good as every non-fractional feasible solution.



(1) Let S = an arbitrary feasible solution

(2) Suppose l units of knapsack filled by S with items not packed by the greedy fractional solution

(3) Must be at least l units of knapsack filled by greedy fractional solution not packed by S

(4) By greedy criterion, items in (3) have larger bang-per-buck v_i/w_i than those in (2) [i.e., more valuable use of space]

(5) Total value of greedy fractional solution at least that of S

Analysis of Greedy Heuristic

In Step 2, suppose our greedy algorithm picks the 1st k items (sorted by v_i/w_i).

by step 3

Value of 3-step greedy algorithm	$\geq$	total value of 1st k items
also is	$\geq$	value of $(k+1)$th item

$\Rightarrow$ 2$\cdot$(value of 3-step greedy)	$\geq$	total value of 1st $(k+1)$ items
	$\geq$	total value of greedy fractional soln
	$\geq$	optimal knapsack solution

QED!

Analysis is Tight

Example: $W = 1000$

$v_1 = 502 \quad v_2 = v_3 = 500$

$w_1 = 501 \quad w_2 = w_3 = 500$

$\Rightarrow$ 3-step greedy solution has value 502

$\Rightarrow$ optimal solution has value 1000

A Refined Analysis

Suppose: Every item i has size $w_i \leq 10\% \cdot$knapsack capacity W.

Consequence: If greedy algorithm fails to pack all items in Step 2, then the knapsack is $\geq 90\%$ full.

$\Rightarrow$ Value of 2-step greedy algorithm

$\geq 90\% \cdot$ value of greedy fractional solution

$\geq 90\% \cdot$ value of an optimal solution.

[In general, if $\max_i w_i \leq \delta W$, then 2-step greedy value is $\geq (1-\delta) \cdot$optimal]

Algorithms: Design and Analysis, Part II

Approximation Algorithms for NP-Complete Problems

A Dynamic Programming Heuristic for Knapsack

Arbitrarily Good Approximation

Goal: For a user-specified parameter $\epsilon > 0$ (e.g., $\epsilon = 0.01$) guarantee a $(1 - \epsilon)$-approximation.

Catch: Running time will increase as ϵ decreases.
(i.e., algorithm exports a running time vs. accuracy trade-off).

[Best-case scenario for NP-complete problems]

The Approach: Rounding Item Values

High-level idea: Exactly solve a slightly incorrect, but easier, knapsack instance.

Recall: If the w_i's and W are integers, can solve the knapsack problem via dynamic programming in $O(nW)$ time.

Alternative: If v_i's are integers, can solve knapsack via dynamic programming in $O(n^2 v_{\max})$ time, where $v_{\max} = \max_i\{v_i\}$.
(See separate video)

Upshot: If all v_i's are small integers (polynomial in n) then we already know a poly-time algorithm.

Plan: Throw out lower-order bits of the v_i's!

A Dynamic Programming Heuristic

Step 1 of algorithm:

Round each v_i down to the nearest multiple of m
[larger $m \Rightarrow$ throw out more info $\Rightarrow$ less accuracy]
[Where m depends on ϵ, exact value to be determined later]

Divide the results by m to get $\hat{v}_i$'s (integers). (i.e., $\hat{v}_i = \lfloor \frac{v_i}{m} \rfloor$)

Step 2 of algorithm: Use dynamic programming to solve the knapsack instance with values $\hat{v}_1, \ldots, \hat{v}_n$, sizes $w_1, \ldots, w_n$, capacity W.

Running time $= O(n^2 \max_i \hat{v}_i)$

Note: Computes a feasible solution to the original Knapsack instance.

Approximation Algorithms for NP-Complete Problems

Dynamic Programming for Knapsack, Revisited

Algorithms: Design and Analysis, Part II

Two Dynamic Programming Algorithms

Dynamic programming algorithm #1: (See earlier videos)

(1) Assume sizes w_i and capacity W are integers

(2) Running time $= O(nW)$

Dynamic programming algorithm #2: (This video)

(1) Assume values v_i are integers

(2) Running time $= O(n^2 v_{\max})$, where $v_{\max} = \max_i v_i$

The Subproblems and Recurrence

Subprolems: For $i = 0, 1, \ldots, n$ and $x = 0, 1, \ldots, nv_{\max}$ define $S_{i,x}$ = minimum total size needed to achieve value $\geq x$ while using only the first i items. (Or $+\infty$ if impossible)

Recurrence: ($i \geq 1$)

$$S_{i,x} =$$

$$\min \begin{cases} S_{(i-1),x} & \text{Case 1, item } i \text{ not used in optimal solution} \\ w_i + S_{(i-1),(x-v_i)} & \text{Case 2, item } i \text{ used in optimal solution} \end{cases}$$

Interpret as 0 if $v_i > x$

The Algorithm

Let A = 2-D array

[indexed by $i = 0, 1, \ldots, n$ and $x = 0, 1, \ldots, nv_{\max}$]

Base case: $A[0, x] = \begin{cases} 0 \text{ if } x = 0 \\ +\infty \text{ otherwise} \end{cases}$

For $i = 1, 2, \ldots, n$ $\longrightarrow$ $n^2 v_{\max}$ iterations

For $x = 0, 1, \ldots, nv_{\max}$

Interpret as 0 if $v_i > x$

$$A[i, x] = \min\{A[i-1, x], w_i + \boxed{A[i-1, x - v_i]}\}$$

$O(1)$ work per iteration

Return the largest x such that $A[n, x] \leq W \leftarrow O(nv_{\max})$

Running time: $O(n^2 v_{\max})$

Approximation Algorithms for NP-Complete Problems

Algorithms: Design and Analysis, Part II

Analysis of a Dynamic Programming Heuristic for Knapsack

The Dynamic Programming Heuristic

Step 1: Set $\hat{v}_i = \lfloor \frac{v_i}{m} \rfloor$ for every item i.

Step 2: Compute optimal solution with respect to the $\hat{v}_i$'s using dynamic programming.

Plan for analysis:

(1) Figure out how big we can take m, subject to achieving a $(1-\epsilon)$-approximation

(2) Plug in this value of m to determine running time

Quiz

Question: Suppose we round v_i to the value $\hat{v}_i$. Which of the following is true?

A) $\hat{v}_i$ is between $v_i - m$ and v_i

B) $\hat{v}_i$ is between v_i and $v_i + m$

C) $m\hat{v}_i$ is between $v_i - m$ and v_i

D) $m\hat{v}_i$ is between $v_i - m$ and v_i

Accuracy Analysis I

From quiz: Since we rounded down to the nearest multiple of m, $m\hat{v}_i \in [v_i - m, v_i]$ for each item i.

Thus: (1) $v_i \geq m\hat{v}_i$, (2) $m\hat{v}_i \geq v_i - m$

Also: If $S^* =$ optimal solution to the original problem (with the original v_i's), and $S =$ our heuristic's solution, then

$$(3) \quad \sum_{i \in S} \hat{v}_i \geq \sum_{i \in S^*} \hat{v}_i$$

[Since S is optimal for the $\hat{v}_i$'s] (recall Step 2)

Accuracy Analysis II

S = our solution, S^*= optimal solution

$$\sum_{i \in S} v_i \overset{(1)}{\geq} m \sum_{i \in S} \hat{v}_i \overset{(3)}{\geq} m \sum_{i \in S^*} \hat{v}_i \overset{(2)}{\geq} \sum_{i \in S^*} (v_i - m)$$

contains at most n items

Thus: $\sum_{i \in S} v_i \geq (\sum_{i \in S^*} v_i) - mn$

Constraint: $\sum_{i \in S} v_i \geq (1 - \epsilon) \sum_{i \in S^*} v_i$

To achieve above constraint: Choose m small enough that $mn \leq \epsilon \boxed{\sum_{i \in S^*} v_i}$

unknown to algorithm, but definitely$\geq v_{\max}$

Sufficient: Set m so that $mn = \epsilon v_{\max}$, i.e., heuristic uses $m = \frac{\epsilon v_{\max}}{n}$

Tim Roughgarden

Running Time Analysis

Point: Setting $m = \frac{\epsilon v_{\max}}{n}$ guarantees that value of our solution is $\geq (1-\epsilon) \cdot$value of optimal solution.

Recall: Running time is $O(n^2 \hat{v}_{\max})$

Note: For every item i, $\hat{v}_i \leq \frac{v_i}{m} \leq \frac{v_{\max}}{m} = v_{\max} \frac{n}{\epsilon v_{\max}} = \frac{n}{\epsilon}$

Running time $= O(n^3/\epsilon)$