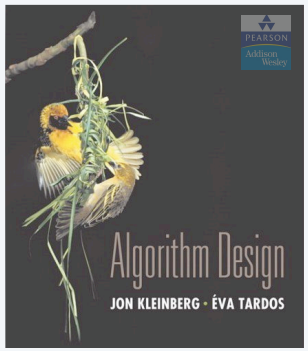
Dynamic Programming

Knapsack problem

Tyler Moore

CS 2123, The University of Tulsa

Some slides created by or adapted from Dr. Kevin Wayne. For more information see http://www.cs.princeton.edu/~wayne/kleinberg-tardos. Some code reused from Python Algorithms by Magnus Lie Hetland.



SECTION 6.4

6. DYNAMIC PROGRAMMING I

Knapsack problem

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ and has value $v_i > 0$.
- Knapsack has capacity of W.
- Goal: fill knapsack so as to maximize total value.

Ex. $\{ 1, 2, 5 \}$ has value 35.
Ex. $\{ 3, 4 \}$ has value 40.
Ex. $\{ 3, 5 \}$ has value 46 (but exceeds weight limit).

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

knapsack instance (weight limit W = 11)

Greedy by value. Repeatedly add item with maximum v_i.
Greedy by weight. Repeatedly add item with minimum w_i.
Greedy by ratio. Repeatedly add item with maximum ratio v_i / w_i.

Observation. None of greedy algorithms is optimal.

Dynamic programming: false start

Def. $OPT(i)$ = max profit subset of items $1, \ldots, i$.

Case 1. OPT does not select item i.
- OPT selects best of $\{ 1, 2, \ldots, i-1 \}$.

optimal substructure property (proof via exchange argument)

Case 2. OPT selects item i.
- Selecting item i does not immediately imply that we will have to reject other items.
- Without knowing what other items were selected before i, we don't even know if we have enough room for i.

Conclusion. Need more subproblems!

Dynamic programming: adding a new variable

Def. $OPT(i, w)$ = max profit subset of items $1, \ldots, i$ with weight limit w.

Case 1. OPT does not select item i.
- OPT selects best of $\{ 1, 2, \ldots, i-1 \}$ using weight limit w.

Case 2. OPT selects item i.
- New weight limit $= w - w_i$.
- OPT selects best of $\{ 1, 2, \ldots, i-1 \}$ using this new weight limit.

optimal substructure property (proof via exchange argument)

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{ OPT(i-1,w),\ \ v_i + OPT(i-1,w-w_i) \} & \text{otherwise} \end{cases}$$

Knapsack problem: bottom-up

KNAPSACK $(n, W, w_1, \ldots, w_n, v_1, \ldots, v_n)$

FOR $w = 0$ TO W
  $M[0, w] \leftarrow 0$.

FOR $i = 1$ TO n
  FOR $w = 1$ TO W
  IF $(w_i > w)$ $M[i, w] \leftarrow M[i-1, w]$.
  ELSE $M[i, w] \leftarrow \max \{ M[i-1, w],\ v_i + M[i-1, w-w_i] \}$.

RETURN $M[n, W]$.

Knapsack problem: bottom-up demo

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{ OPT(i-1,w),\ \ v_i + OPT(i-1,w-w_i) \} & \text{otherwise} \end{cases}$$

weight limit w

subset of items 1, ..., i	0	1	2	3	4	5	6	7	8	9	10	11
{ }	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	34	40

OPT(i, w) = max profit subset of items 1, ..., i with weight limit w.

Knapsack problem: running time

Theorem. There exists an algorithm to solve the knapsack problem with n items and maximum weight W in $\Theta(n\,W)$ time and $\Theta(n\,W)$ space.

Pf. (weights are integers between 1 and W)
- Takes $O(1)$ time per table entry.
- There are $\Theta(n\,W)$ table entries.
- After computing optimal values, can trace back to find solution: take item i in $OPT(i, w)$ iff $M[i, w] < M[i-1, w]$. ▪

Remarks.
- Not polynomial in input size! ← "pseudo-polynomial"
- Decision version of knapsack problem is NP-COMPLETE. [CHAPTER 8]
- There exists a poly-time algorithm that produces a feasible solution that has value within 1% of optimum. [SECTION 11.8]