A Space Efficient Knapsack Algorithm

Wim Bohm, Sanjay Rajopadhye, CS, CSU

sources: Cormen, Leiserson; Kleinberg, Tardos, Vipin Kumar et.al.
Knapsack Problem

- Given $n$ objects and a "knapsack" of capacity $W$
- Item $i$ has a weight $w_i > 0$ and value $v_i > 0$.
- Goal: fill knapsack so as to maximize total value.
- Is there a Greedy solution?

Greedy approach: repeatedly add item with maximum $v_i / w_i$ profit weight ratio …

Does Greedy work?

Capacity $M = 7$, Number of objects $n = 3$
- $w = [5, 4, 3]$
- $v = [10, 7, 5]$ (ordered by $v_i / w_i$ ratio)
Recursion for Knapsack Problem

Notation: \( \text{OPT}(i, W) = \) optimal value of max weight subset that uses items 1, \( \ldots \), \( i \) with \textbf{weight limit} \( w \).

Case 1: item \( i \) is not included:
- Take best of \( \{1, 2, \ldots, i-1\} \) using weight limit \( W \): \( \text{OPT}(i-1, w) \)

Case 2: item \( i \) with weigh \( w_i \) and value \( v_i \) is included:
- \textbf{only possible if} \( w \geq w_i \)
- \textbf{new weight limit} = \( w - w_i \)
- Take best of \( \{1, 2, \ldots, i-1\} \) using weight limit \( w-w_i \) and add \( v_i \):
  \[
  \text{OPT}(i-1, W-w_i) + v_i
  \]

\[
\text{OPT}(i, w) = \begin{cases} 
0 & \text{if } i = 0 \\
\text{OPT}(i-1, w) & \text{if } w_i > w \\
\max\{\text{OPT}(i-1, w), \ v_i + \text{OPT}(i-1, w-w_i)\} & \text{otherwise}
\end{cases}
\]
Knapsack Problem: Bottom-Up Dynamic Programming

Knapsack. Fill an n-by-W array.

Input: n, W, w_1,…,w_N, v_1,…,v_N

for w = 0 to W
    M[0, w] = 0

for i = 1 to n
    for w = 0 to W
        if w_i > w :
            M[i, w] = M[i-1, w]
        else :
            M[i, w] = max (M[i-1, w], v_i + M[i-1, w-w_i ])

return M[n, W]
Knapsack Algorithm

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

\[ W = 11 \]
## Knapsack Algorithm

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>
Knapsack Algorithm

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

OPT: 40

How do we find the choice vector $x$, in other words the objects picked in the optimum solution?

Walk back through the table!!
**Knapsack Algorithm**

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

W = 11

**OPT:** 40

n=5  Don’t take object 5
## Knapsack Algorithm

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

**W = 11**

**OPT: 40**

n=5  Don’t take object 5
n=4  Take object 4
Knapsack Algorithm

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>${1}$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>${1, 2}$</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>${1, 2, 3}$</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>18</td>
<td>19</td>
<td>24</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>${1, 2, 3, 4}$</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>18</td>
<td>22</td>
<td>24</td>
<td>28</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>${1, 2, 3, 4, 5}$</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>18</td>
<td>22</td>
<td>28</td>
<td>29</td>
<td>34</td>
<td>35</td>
</tr>
</tbody>
</table>

OPT: 40
n=5  Don’t take object 5
n=4  Take object 4
n=3  Take object 3

and now we cannot take anymore, so choice set is \{3,4\}
Time and Space Complexity

Time complexity: $O(n \times W)$ for $n$ objects and knapsack capacity $W$. Does this mean knapsack has a polynomial solution?

NO! WHY NOT? $W$ can be arbitrarily large (e.g. $2^n$)
There are $O(2^n)$ choice vectors

Space complexity: $O(n \times W)$. OK?

NO! This is in practice prohibitively large!

We need a more space efficient algorithm, even if it costs us some recomputation (but not more than a constant factor)
Space inefficiency

What causes it?

The table, why do we need it?

To compute the choice vector

Is there an alternative?

We could only keep rows (n-1 and n), do one step back for the last choice bit, and then recur (re-compute the table for (1..(n-1)). \(O(W)\) Space! What’s the problem?

Table \(n*W\), each sweep gives one bit for \((n-i)\) and takes \((n-i)*W\) time. Time complexity goes up an order of magnitude:

\[O(n^2 * W)\]

Alternative?
Recursive, Divide and Conquer, approach

- Solve two halves of the table, for the first \( n/2 \) objects and for the second. This gives us two solution rows (row \( n/2 \) and row \( n \))

- Combine their solutions somehow to find the cut points

- Now we have two sub tables left, and solve these recursively

- Think about it before you move on! We need to divide the problem in two sub-problems with together half the size (cutting half the remaining table out each time).
Recursive approach

1. Fill two tables, for the first $n/2$ objects and for the second. This gives us two solution rows (row $n/2$)
Recursive approach

1. Fill two tables, say for the first \( n/2 \) objects and for the second. This gives us two solution rows (row \( n/2 \)).

2. Combine the two solution rows somehow to find the cut-points. **HOW?**

3. Given total capacity \( W \), if I use \( C_1 \) for the top table, how much is left for the bottom table?

\[
W - C_1
\]
Recursive approach

1. Fill two tables, say for the first $n/2$ objects and for the second. This gives us two solution rows (row $n/2$)
2. Combine the two solution rows somehow to find optimum cutpoints
3. Now we have two sub-problems left, and solve these recursively
4. Base case: we have 1 object left, if it fits the capacity, it is in.

Runtime: $O(\text{Area}) = n \times W \times (1 + \frac{1}{2} + \frac{1}{4} + \ldots + (\frac{1}{2})^k) < 2 \times n \times W$

Space: $O(W)$ (constant number of rows)
Say we have computed the solution vectors [0..W] of the two sub-tables (step 1 previous slide?)

How do we find the cut points? (step 2)

If sub-problem 1 uses C1 of capacity W, how much is left for sub-problem 2?

What are the two sub-problems we need to solve recursively? (step 3)

- Sp1: objects 1..n/2, capacity C1
- Sp2: objects n/2+1..N, capacity W - C1

Hence, the cut points are C1 and W - C1
Finding the optimum combination

Define $T_1(k)$ the optimal profit that can be achieved for sub-problem 1 ($k = 0..W$), and $T_2(k)$ for sub-problem 2.

Then $P(k) = T_1(k) + T_2(C-k)$, ($k = 0..W$) is the maximum profit for the original problem, and the index of the maximum value of $P(k)$ gets us the optimum cut points!
N=8, W=16, O 1..8
W  2 6 4 3 1 3 2 1
V  2 4 5 2 3 1 1 2

Iterative solution: profit 17, choice vector X = 11101011

{1} 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 1
{1-2} 0 0 2 2 2 2 4 4 6 6 6 6 6 6 6 6 6 1
{1-3} 0 0 2 2 5 5 7 7 7 9 9 11 11 11 11 11 1
{1-4} 0 0 2 2 5 5 7 7 7 9 9 9 11 11 11 13 13 0
{1-5} 0 3 3 5 5 8 8 10 10 10 12 12 14 14 14 16 1
{1-6} 0 3 3 5 5 8 8 10 10 10 12 12 14 14 14 16 0
{1-7} 0 3 3 5 5 8 8 10 10 11 12 12 13 14 14 15 16 1
{1-8} 0 3 5 5 7 8 10 10 12 12 13 14 14 15 16 17 1
example

Solve N=8, W=16, O 1..8
  W  2 6 4 3 1 3 2 1
  V  2 4 5 2 3 1 1 2

N=4,C=16, O 1..4
  0 0 2 2 2 2 2 2 2 2 2 2 2 2 2
  0 0 2 2 2 2 4 4 6 6 6 6 6 6 6
  0 0 2 2 5 5 7 7 7 7 9 9 11 11 11 11
  0 0 2 2 5 5 7 7 7 9 9 11 11 11 11

optimum cut points: C1, C2 = 12, 4

Solve N=4,W=12, O 1..4
  W  2 6 4 3
  V  2 4 5 2
N=2,C=12, O 1..2
  0 0 2 2 2 2 2 2 2 2 2 2
  0 0 2 2 2 2 4 4 6 6 6 6

N=2,W=12, O 3..4
  0 0 0 5 5 5 5 5 5 5 5 5
  0 0 2 5 5 7 7 7 7 7 7 7

optimum cut points: C1.1, C1.2 = 8, 4

Solve N=4,C=4, O 5..8
...

...
example

Solve N=4, C=12, O 1..4
  W 2 6 4 3
  V 2 4 5 2
N=2, C=12, O 1..2
  0 0 2 2 2 2 2 2 2 2 2 2 2
  0 0 2 2 2 4 4 6 6 6 6

N=2, C=12, O 3..4
  0 0 0 0 5 5 5 5 5 5 5 5
  0 0 0 2 5 5 5 7 7 7 7 7

optimum cut points:
  C1.1, C1.2 = 8, 4

Solve N=2, C=4, O 3..4
  W 4 3
  V 5 2
N=1, C=4, O 3
  0 0 0 0 5
N=1, C=4, O 4
  0 0 0 2 2
optimum points
  C2.1 = 4, C2.2 = 0

Solve N=1, C=4, O 3
  W 4, V 5
Base case: fit, TAKE O3, X_3 = 1

Solve N=1, C=4, O 3
  W 4, V 5
Base case: no fit, DON'T TAKE O4: X_4 = 0

Solve N=1, C=0, O 4
  W 3, V 2
Base case: no fit, DON'T TAKE O4: X_4 = 0