0-1 Knapsack Problem

Defining a Subproblem

If items are labeled *1..n*, then a subproblem would be to find an optimal solution for S_k = *{items labeled 1, 2, .. k}*

- This is a reasonable subproblem definition.
- The question is: can we describe the final solution (S_n) in terms of subproblems (S_k)?
- Unfortunately, we can't do that.

Defining a Subproblem

$w_1 = 2$ $b_1 = 3$	$w_2 = 4$ $b_2 = 5$	$w_3 = 5$ $b_3 = 8$	$w_4 = 3$ $b_4 = 4$ **?**	

Max weight: W = 20

For S_4:

Total weight: 14

Maximum benefit: 20

$w_1 = 2$ $b_1 = 3$	$w_2 = 4$ $b_2 = 5$	$w_3 = 5$ $b_3 = 8$	$w_5 = 9$ $b_5 = 10$

For S_5:

Total weight: 20

Maximum benefit: 26

S_4 (items 1–4), S_5 (items 1–5)

Item #	Weight w_i	Benefit b_i
1	2	3
2	4	5
3	5	8
4	3	4
5	9	10

Solution for S_4 is not part of the solution for S_5!!!

Defining a Subproblem

- As we have seen, the solution for S_4 is not part of the solution for S_5

- So our definition of a subproblem is flawed and we need another one!

Defining a Subproblem

- Given a knapsack with maximum capacity W, and a set S consisting of n items
- Each item i has some weight w_i and benefit value b_i (all w_i and W are integer values)
- Problem: How to pack the knapsack to achieve maximum total value of packed items?

Defining a Subproblem

- Let's add another parameter: *w*, which will represent the maximum weight for each subset of items

- The subproblem then will be to compute *V[k,w]*, *i.e.*, to find an optimal solution for S_k = *{items labeled 1, 2, .. k} in a knapsack of size w*

Recursive Formula for subproblems

- The subproblem will then be to compute *V[k,w]*, *i.e.*, to find an optimal solution for S_k = *{items labeled 1, 2, .. k} in a knapsack of size w*

- Assuming knowing V[i, j], where i=0,1, 2, … k-1, j=0,1,2, …w, how to derive V[k,w]?

Recursive Formula for subproblems (continued)

Recursive formula for subproblems:

$$V[k,w] = \begin{cases} V[k-1,w] & \text{if } w_k > w \\ \max\{V[k-1,w], V[k-1,w-w_k]+b_k\} & \text{else} \end{cases}$$

It means, that the best subset of S_k that has total weight w is:

1) the best subset of S_{k-1} that has total weight $\leq w$, **or**

2) the best subset of S_{k-1} that has total weight $\leq w-w_k$ plus the item k

Recursive Formula

$$V[k,w]=\begin{cases} V[k-1,w] & \text{if } w_k > w \\ \max\{V[k-1,w], V[k-1,w-w_k]+b_k\} & \text{else} \end{cases}$$

- The best subset of S_k that has the total weight $\leq w$, either contains item k or not.
- First case: $w_k > w$. Item k can't be part of the solution, since if it was, the total weight would be $> w$, which is unacceptable.
- Second case: $w_k \leq w$. Then the item k can be in the solution, and we choose *the case with greater value*.

0-1 Knapsack Algorithm

```
for w = 0 to W
    V[0,w] = 0
for i = 1 to n
    V[i,0] = 0
for i = 1 to n
    for w = 0 to W
        if w_i <= w // item i can be part of the solution
            if b_i + V[i-1,w-w_i] > V[i-1,w]
                V[i,w] = b_i + V[i-1,w- w_i]
            else
                V[i,w] = V[i-1,w]
        else V[i,w] = V[i-1,w]  // w_i > w
```

Running time

```
for w = 0 to W
    V[0,w] = 0
for i = 1 to n
    V[i,0] = 0
for i = 1 to n
    for w = 0 to W
        < the rest of the code >
```

O(W)

Repeat *n* times

O(W)

What is the running time of this algorithm?

O(n*W)

Remember that the brute-force algorithm takes $O(2^n)$

Example

Let's run our algorithm on the following data:

n = 4 (# of elements)
W = 5 (max weight)
Elements (weight, benefit):
(2,3), (3,4), (4,5), (5,6)

Example (2)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

for w = 0 to W

V[0,w] = 0

Example (3)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

for i = 1 to n

V[i,0] = 0

Example (4)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=1

w-w_i =-1

if w_i <= w // item i can be part of the solution

    if b_i + V[i-1,w-w_i] > V[i-1,w]

        V[i,w] = b_i + V[i-1,w- w_i]

    else

        V[i,w] = V[i-1,w]

else **V[i,w] = V[i-1,w]** // w_i > w

Example (5)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	**3**			
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=2$

$w-w_i=0$

if $w_i \le w$ // item i can be part of the solution

  if $b_i + V[i-1,w-w_i] > V[i-1,w]$

    $V[i,w] = b_i + V[i-1,w-w_i]$

  else

    $V[i,w] = V[i-1,w]$

else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (6)

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	**3**		
2	0					
3	0					
4	0					

i=1
b_i=3
w_i=2
w=3
$w-w_i$ =1

if $w_i <= w$ // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w- w_i]$
  else
    $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (7)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	**3**	
2	0					
3	0					
4	0					

i=1
$b_i=3$
$w_i=2$
w=4
$w-w_i=2$

if $w_i <= w$ // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w-w_i]$
  else
    V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // $w_i > w$

Example (8)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	**3**
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=5$

$w-w_i=3$

if $w_i <= w$ // item i can be part of the solution

if $b_i + V[i-1,w-w_i] > V[i-1,w]$

$V[i,w] = b_i + V[i-1,w- w_i]$

else

$V[i,w] = V[i-1,w]$

else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (9)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	**0**				
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=1$

$w-w_i = -2$

```
if wi <= w // item i can be part of the solution
        if bi + V[i-1,w-wi] > V[i-1,w]
            V[i,w] = bi + V[i-1,w- wi]
        else
            V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w]  // wi > w
```

Example (10)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	**3**			
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=2$

$w-w_i=-1$

if w_i <= w // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w-w_i]$
  else
    $V[i,w] = V[i-1,w]$
else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (11)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	**4**		
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=3$

$w-w_i=0$

if $w_i <= w$ // item i can be part of the solution

    if $b_i + V[i-1,w-w_i] > V[i-1,w]$

        $V[i,w] = b_i + V[i-1,w- w_i]$

    else

        $V[i,w] = V[i-1,w]$

else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (12)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	**4**	
3	0					
4	0					

$i=2$
$b_i=4$
$w_i=3$
$w=4$
$w-w_i=1$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w-w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (13)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	**7**
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=5$

$w-w_i=2$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (14)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	**0**	**3**	**4**		
4	0					

i=3

b_i=5

w_i=4

w= 1..3

if w_i <= w // item i can be part of the solution

if b_i + V[i-1,w-w_i] > V[i-1,w]

V[i,w] = b_i + V[i-1,w- w_i]

else

V[i,w] = V[i-1,w]

else **V[i,w] = V[i-1,w]** // w_i > w

Example (15)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	**5**	
4	0					

$i=3$

$b_i=5$

$w_i=4$

$w= 4$

$w- w_i=0$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (16)

Items:

1: (2,3)
2: (3,4)
3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	**7**
4	0					

i=3

b_i=5

w_i=4

w= 5

w- w_i=1

if w_i <= w // item i can be part of the solution

  if b_i + V[i-1,w-w_i] > V[i-1,w]

    V[i,w] = b_i + V[i-1,w- w_i]

  else

    V[i,w] = V[i-1,w]

else V[i,w] = V[i-1,w] // w_i > w

Example (17)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	**0**	**3**	**4**	**5**	

i=4

b_i=6

w_i=5

w= 1..4

if w_i <= w // item i can be part of the solution

    if b_i + V[i-1,w-w_i] > V[i-1,w]

        V[i,w] = b_i + V[i-1,w- w_i]

    else

        V[i,w] = V[i-1,w]

else **V[i,w] = V[i-1,w]** // w_i > w

Example (18)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	**7**

i=4

$b_i=6$

$w_i=5$

w= 5

$w- w_i=0$

if $w_i <= w$ // item i can be part of the solution

if $b_i + V[i-1,w-w_i] > V[i-1,w]$

$V[i,w] = b_i + V[i-1,w- w_i]$

else

V[i,w] = V[i-1,w]

else V[i,w] = V[i-1,w] // $w_i > w$

How to find actual Knapsack Items

- All of the information we need is in the table.
- $V[n,W]$ is the maximal value of items that can be placed in the Knapsack.
- Let i=n and k=W

if $V[i,k] \neq V[i-1,k]$ then

mark the i^{th} item as in the knapsack

$i = i-1$, $k = k-w_i$

else

$i = i-1$ // Assume the i^{th} item is <u>not</u> in the knapsack

// Could it be in the optimally packed knapsack?

Finding the Items

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=4

k= 5

b_i=6

w_i=5

V[i,k] = 7

V[i–1,k] =7

i=n, k=W

while i,k > 0

 if *V[i,k]* ≠ *V[i–1,k]* then

  mark the *i*th item as in the knapsack

  i = *i–1*, *k* = *k*-w_i

 else

  i = *i–1*

Finding the Items (2)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=4
k= 5
b_i=6
w_i=5
$V[i,k] = 7$
$V[i-1,k] = 7$

i=n, k=W
while i,k > 0
  if $V[i,k] \neq V[i-1,k]$ then
    mark the i^{th} item as in the knapsack
    $i = i-1$, $k = k-w_i$
  else
    i = i–1

Finding the Items (3)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=3

k= 5

b_i=5

w_i=4

$V[i,k] = 7$

$V[i-1,k] = 7$

i=n, k=W

while i,k > 0

if $V[i,k] \neq V[i-1,k]$ then

mark the i^{th} item as in the knapsack

$i = i-1$, $k = k-w_i$

else

i = i–1

Finding the Items (4)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=2
k= 5
b_i=4
w_i=3
V[*i*,*k*] = 7
V[*i*–*1*,*k*] =3
k – w_i=2

i=n, k=W
while i,k > 0
  if *V*[*i*,*k*] ≠ *V*[*i*–*1*,*k*] then
    mark the i^{th} item as in the knapsack
    i = *i*–*1*, *k* = *k*-w_i
  else
    i* = *i*–*1

Finding the Items (5)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

$i=1$

$k=2$

$b_i=3$

$w_i=2$

$V[i,k] = 3$

$V[i-1,k] = 0$

$k - w_i = 0$

i=n, k=W
while i,k > 0
    if $V[i,k] \neq V[i-1,k]$ then
        mark the i^{th} item as in the knapsack
        $i = i-1,\ k = k-w_i$
    else
        i = i–1

Finding the Items (6)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=0
k= 0

The optimal knapsack should contain {1, 2}

i=n, k=W
while i,k > 0
  if $V[i,k] \neq V[i-1,k]$ then
    mark the n^{th} item as in the knapsack
    $i = i-1$, $k = k-w_i$
  else
    $i = i-1$

Finding the Items (7)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

The optimal knapsack should contain {1, 2}

i=n, k=W
while i,k > 0
    if $V[i,k] \neq V[i-1,k]$ then
        mark the n^{th} item as in the knapsack
        $i = i-1$, $k = k-w_i$
    else
        $i = i-1$

Assignment

- Q.1)What is 0-1 knapsack problem
- Q.2)How 0-1 knapsack problem is solved using Dynamic prpgramming.