

We will discuss

→ Polynomial Time Approximation Schemes (PTAS)

Consider knapsack

$O_1(s, v) \ O_2(s, v) \ \dots \ O_n(s, v)$

Capacity = W

Maximize value

→ NP complete

	O_1	O_2	O_3	O_4	O_5	O_6	O_7
Size	3	2	4	5	6	2	1
Value	2	3	5	4	3	5	2

Capacity: 10

Ans: $\{O_2, O_3, O_6, O_7\} = \underline{\underline{15}}$

Knapsack has a pseudopolynomial time algorithm

n objects $(s_1, s_2, \dots, s_n) (v_1, v_2, \dots, v_n)$

$j \rightarrow$

1 2 3 4

$V = (v_1, v_2, \dots, v_n)$

1
↓ 2
↓ 3
.
.
.
n

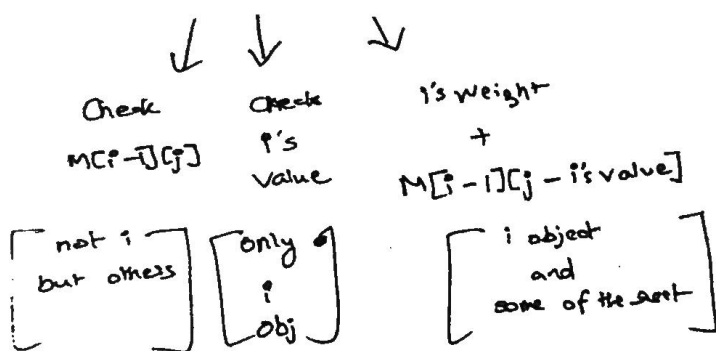
$[i][j]$ ⇒ minimum size reqd to achieve value of exactly j using objects $1 \dots i$
Mark '-' if not possible

$S = [3, 2, 4, 5, 6, 2, 1]$

$V = [2, 3, 5, 4, 3, 5, 2]$

	1	2	3	4	5	...	24
1	-	3	-	-	-		-
2	-	3	2	-	5	...	-
3	-	3	2	-	4	...	-
...							
7							

$M[i][j] =$



take minimum of the 3 ...

Size of the table = $n \cdot V$

time to build the table = $O(n \cdot V)$

V could be exponential function of n .

Idea: Dividing columns by V

$\rightarrow 1, 2, 3, \dots, V \Rightarrow 1/V, 2/V, 3/V, \dots$

\rightarrow RAM can only handle numbers with fixed precision

\rightarrow Also, table size wouldn't change

let's improve the idea. after dividing each column by a factor, let's round them off.

→ RAM can handle these rounded values

→ table size would shrink

0.5 1 1.5 2 2.5 ...

 1 2

by taking minimum of the two

→ causes the soln to be suboptimal

	A	B
S	2	3
V	2	4
	↳ /3	/3
	↳ 0.67	1.34
	↳ 1	1

(loss of info due to round off)

S_1	S_2	S_3	...	S_n	
V_1	V_2	V_3	...	V_n	} V
\downarrow $\left\lfloor \frac{V_1}{x} \right\rfloor$ $\left\lfloor \frac{V_2}{x} \right\rfloor$ $\left\lfloor \frac{V_3}{x} \right\rfloor$... $\left\lfloor \frac{V_n}{x} \right\rfloor$					

what should x be?

$$x = \frac{V}{n} (1-c)$$

$$0 \leq c < 1$$

running time $\Rightarrow O(n \cdot \text{no of cols})$

$$\# \text{ cols} = \cancel{1} \dots V'$$

$$\begin{aligned} V' &= \left\lfloor \frac{V_1}{x} \right\rfloor + \left\lfloor \frac{V_2}{x} \right\rfloor + \dots + \left\lfloor \frac{V_n}{x} \right\rfloor \\ &= \left\lfloor \frac{nV_1}{V(1-c)} \right\rfloor + \left\lfloor \frac{nV_2}{V(1-c)} \right\rfloor + \dots + \left\lfloor \frac{nV_n}{V(1-c)} \right\rfloor \\ &= O\left(\frac{nV}{V(1-c)}\right) = O\left(\frac{n}{1-c}\right) \end{aligned}$$

running time $\Rightarrow O\left(\frac{n^2}{1-c}\right) \quad 0 \leq c < 1$

$$c \Rightarrow 0$$

running time \Rightarrow polynomial

$$c \Rightarrow 1$$

running time \Rightarrow exponential

Quantifying the error

optimum soln $\leq n$ objects

Mistake: selecting A instead of B

$$\text{can only be made if } \left\lfloor \frac{V_A}{x} \right\rfloor = \left\lfloor \frac{V_B}{x} \right\rfloor$$

$$\text{Value lost due to mistake } V_B - V_A < x$$

We can make n such mistakes at maximum

$$\text{Max. absolute value lost / error} = n \cdot x$$

$$\text{Relative error} < \frac{V_{\text{opt}}}{V_{\text{opt}} - n \cdot x} = 1 + \frac{n \cdot x}{V_{\text{opt}} - n \cdot x}$$

$$\leq 1 + \frac{n \cdot x}{V - n \cdot x}$$

$$= 1 + \frac{V(1-c)}{V - V(1-c)}$$

$$= 1 + \frac{1-c}{1-(1-c)} = \frac{1}{c}$$

$c \rightarrow 1$ good approx

$$O\left(\frac{n^2}{n_0}\right)$$

$c \rightarrow 0$ bad approx

$$O\left(\frac{n^2}{n_1}\right)$$