

Problem analysis of Dynamic Programming: Knapsack problem

YAO ZHAO

Knapsack problem

- ▶ 0-1 Knapsack problem: Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. You cannot break an item, either pick the complete item or don't pick it (0-1 property). This is the most basic Knapsack problem, which is explained in detail in the theory class .

The following are various Knapsack problems that extend from the 01 Knapsack problem :

- ▶ Unbounded Knapsack Problem(完全背包): for each item, you can put as many times as you want.
- ▶ Bounded Knapsack Problem(多重背包): each item has a put times bound.
- ▶ Mixed knapsack problem(混合背包): some items can be taken only once (0-1 Knapsack), some items can take unlimited (Complete Knapsack), the number of some items can be taken with a bound. (Bounded Knapsack)
- ▶ Two-dimensional cost knapsack problem(二维背包): In addition to the capacity dimension, there are additional dimensions, such as volume constraints.
- ▶ Group knapsack problem(分组背包): The items are divided into groups, and there are contradictions between each item in each group

0-1 Knapsack problem

The most common problem being solved is the **0–1 knapsack problem**, which restricts the number x_i of copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n , each with a weight w_i and a value v_i , along with a maximum weight capacity W ,

$$\begin{aligned} &\text{maximize } \sum_{i=1}^n v_i x_i \\ &\text{subject to } \sum_{i=1}^n w_i x_i \leq W \text{ and } x_i \in \{0, 1\}. \end{aligned}$$

Here x_i represents the number of instances of item i to include in the knapsack. Informally, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity.

0-1 Knapsack problem equation:

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

Review:

$W = 11$

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$n + 1$



	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	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

Pseudo-code

```
Input:  $n, W, w_1, \dots, w_N, v_1, \dots, v_N$ 

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

for  $i = 1$  to  $n$ 
    for  $w = 1$  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  $M[n, W]$ 
```

Space complexity optimization of 0-1 knapsack

		$W + 1$											
		0	1	2	3	4	5	6	7	8	9	10	11
$n + 1$	ϕ	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

To calculate the new values of row i , only rows $i - 1$ needed. The data from rows 0 to $i - 2$ do not need storage space.

Input: $n, W, w_1, \dots, w_N, v_1, \dots, v_N$

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

for $i = 1$ to n

for $w = 1$ 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 $M[n, W]$

For W to w_i


$M[w] = \max \{M[w], v_i + M[w-w_i]\}$

For W to w_i

$M[i, w] = \max \{M[i, w], v_i + M[i, w-w_i]\}$

maximize $\sum_{i=1}^n v_i x_i$

subject to $\sum_{i=1}^n w_i x_i \leq W$ and $x_i \in \{0, 1\}$.



Input: $n, W, w_1, \dots, w_N, v_1, \dots, v_N$

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

$M[0] = 0;$
for $w = 1$ to W
 $M[w] = -\infty$

for $i = 1$ to n
 for $w = 1$ 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 $M[n, W]$

Unbounded Knapsack Problem(完全背包)

The **unbounded knapsack problem** (UKP) places no upper bound on the number of copies of each kind of item and can be formulated as above except for that the only restriction on x_i is that it is a non-negative integer.

$$\begin{aligned} & \text{maximize } \sum_{i=1}^n v_i x_i \\ & \text{subject to } \sum_{i=1}^n w_i x_i \leq W \text{ and } x_i \geq 0, x_i \in \mathbb{Z}. \end{aligned}$$

Unbounded Knapsack \longrightarrow 0-1 Knapsack

- ▶ Although the problem indicates that there are an infinite number of items available for each item type, in fact the number of each item cannot exceed $W/w[i]$, so each item type can be expanded to $W/w[i]$ items, each item can be selected or not selected
- ▶ $O(W \sum_{i=0}^{N-1} W/w[i])$.
- ▶ Equation:

$$OPT(i, w) = \begin{cases} 0, & \text{if } i=0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max(OPT(i-1, w), k * v_i + OPT(i-1, w - k * w_i)) & 0 < w_i \leq w, \quad 0 < k \leq W/w_i \end{cases}$$

Example:

weight = 15

Item type	value	weight
0	13	5
1	17	7
2	22	8



Item No.	Item type	value	weight
1	0	13	5
2	0	13	5
3	0	13	5
4	1	17	7
5	1	17	7
6	2	22	8



	0	1	2	...	15
\emptyset					
{1}					
{1,2}					
{1,2,3}					
{1,2,3,4}					
{1,2,3,4,5}					
{1,2,3,4,5,6}					

efficient?

weight = 1500

Item type	value	weight
0	13	5
1	17	7
2	22	8



Item No.	Item type	value	weight
1	0	13	5
..	0	13	5
300	0	13	5
301	1	17	7
..	1	17	7
514	1	17	7
515	2	22	8
...	2	22	8
701	2	22	8



702*1501

$$O(W \sum_{i=0}^{N-1} \log(\frac{w}{w[i]}))$$

Can be optimized to

$O(NW)$?

Unbounded Knapsack problem equation:

$$OPT(i, w) = \max(OPT(i-1, w), v_i + OPT(i, w-w_i))$$

Item type	value	weight
0	13	5
1	17	7
2	22	8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
\emptyset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{0}	0	0	0	0	0	13	13	13	13	13	26	26	26	26	26	39
{0,1}	0	0	0	0	0	13	13	17	17	17	26	26	30	30	34	39
{0,1,2}	0	0	0	0	0	13	13	17	22	22	26	26	30	35	35	39

VS

0-1 Knapsack problem equation:

$$OPT(i, w) = \max(OPT(i-1, w), v_i + OPT(i-1, w-w_i))$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
\emptyset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{0}	0	0	0	0	0	13	13	13	13	13	13	13	13	13	13	13
{0,1}	0	0	0	0	0	13	13	17	17	17	17	17	30	30	30	30
{0,1,2}	0	0	0	0	0	13	13	17	22	22	22	22	30	35	35	39

0-1 Knapsack **VS** Unbounded Knapsack

- ▶ **0-1 Knapsack:** To ensure that each item is selected only once, make sure that the “select the i th item” strategy is considered in terms of a sub result that has no i th item: $f[i-1][w-w[i]]$
- ▶ **Unbounded Knapsack:** The characteristic of a complete backpack is that each item has an infinite number of items to choose from, so when considering the strategy of “select the i -th item”, you need a sub result that might already be selected into the i -th item: $f[i][w-w[i]]$

Bounded Knapsack Problem(多重背包)

The **bounded knapsack problem (BKP)** removes the restriction that there is only one of each item, but restricts the number x_i of copies of each kind of item to a maximum non-negative integer value c_i :

$$\begin{aligned} &\text{maximize } \sum_{i=1}^n v_i x_i \\ &\text{subject to } \sum_{i=1}^n w_i x_i \leq W \text{ and } x_i \in \{0, 1, 2, \dots, c_i\}. \end{aligned}$$

- ▶ *Solution see P.13*
- ▶ **Time complexity:** $O(W \sum_{i=1}^n \log(c_i))$

Two-dimensional knapsack problem(二维背包)

- ▶ There are N items and a knapsack with the capacity W and the volume D . The weight of item i is $w[i]$, volume is $d[i]$, value is $v[i]$. Figure out which items can be put into the knapsack so that the total capacity of these items does not exceed the knapsack's capacity, the total volume does not exceed the knapsack's volume, and the total value is maximum.

Two-dimensional knapsack problem(二维背包)

- ▶ In fact, two-dimensional knapsack is equivalent to two dimensions of cost consumption. We just need to add one dimension to the state equation.
- ▶ The following is the 0-1 two-dimensional knapsack equation:

$$OPT(i, w, d) = \begin{cases} 0, & \text{if } i=0 \\ OPT(i-1, w, d) & \text{if } w_i > w > 0 \text{ or } d_i > d > 0 \\ \max(OPT(i-1, w, d), v_i + OPT(i-1, w-w_i, d-d_i)) & 0 < w_i \leq w, \quad 0 < d_i \leq d \end{cases}$$

- ▶ You can extend the 0-1 two-dimensional knapsack problem to unbounded and bounded cases.

Group knapsack problem(分组背包)

- ▶ There are N items and a knapsack with capacity W . The weight of item i is $w[i]$ and the value is $v[i]$. The items were divided into groups of conflicting items, with no more than one item can be selected from the same group. Figure out which items to put into the knapsack so that the total cost of the items does not exceed the knapsack capacity and the total value of the items is maximum.

Example

- ▶ Using the logic of the 01 backpack, the problem becomes that there are several strategies for each group of items: choose one item in the group, or choose none.
- ▶ $OPT[k][w]$ is used to represent the maximum value that can be obtained from the first k groups of items under the limit of weight W
- ▶ Equation: $OPT[k][w] = \max\{OPT[k-1][w], OPT[k-1][w-w[i]] + v[i]\}$ (item i is in group k)

weight = 15

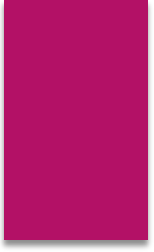
Group			w	v
toothpaste	0	Crest(佳洁士)	7	8
		Colgate(高露洁)	3	7
toothbrush	1	manual toothbrush	3	5
		electric toothbrush	5	8
clothes	2	jumpsuit	5	10
		protective suit	7	15

$$OPT(k, w) = \max(OPT(k-1, w), v_{k1} + OPT(k-1, w-w_{k1}), v_{k2} + OPT(k-1, w-w_{k2}), \dots)$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
\emptyset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{0}	0	0	0	7	7	7	7	8	8	8	8	8	8	8	8	8
{0,1}	0	0	0	7	7	8	12	12	15	15	15	15	16	16	16	16
{0,1,2}	0	0	0	7	7	10	12	15	17	17	22	22	23	27	27	30

A Task-scheduling Problem

- ▶ The problem of **scheduling non-unit time tasks with deadlines and penalties for a single processor** has the following inputs:
 - a set $S = \{a_1, a_2, \dots, a_n\}$ of n non-unit time tasks;
 - A set of n integer **time** t_1, t_2, \dots, t_n , to finish a_i need t_i time;
 - A set of n integer **deadlines** d_1, d_2, \dots, d_n , such that each d_i satisfies $1 \leq d_i \leq n$ and task a_i is supposed to finish by time d_i ; and
 - a set of n nonnegative weights or **penalties** w_1, w_2, \dots, w_n , such that we incur a penalty of w_i if task a_i is not finished by time d_i , and we incur no penalty if a task finishes by its deadline.
- ▶ We wish to find a schedule for S that minimizes the total penalty incurred for missed deadlines.

- 
- ▶ It is a 0-1 Knapsack problem
 - ▶ sorting tasks in non-descending order by deadline d_i
 - ▶ $p(i,d) = \min\{p(i-1,d)+w[i], p(i-1,\min\{d, d[i]\}-t[i])\}$

$$p(1,d)=\begin{cases} 0 & t[1] \leq d \\ w[1] & t[1] > d \end{cases}$$