Advanced Core in Algorithm Design #13
算法設計要論 第13回

Yasushi Kawase
河瀬 康志

Jan. 11th, 2022

last update: 12:21pm, January 11, 2022

Schedule

Lec. #	Date	Topics
1	10/5	Introduction, Stable matching
2	10/12	Basics of Algorithm Analysis, Graphs
3	10/19	Greedy Algorithms (1/2)
4	10/26	Greedy Algorithms (2/2)
5	11/2	Divide and Conquer (1/2)
6	11/9	Divide and Conquer (2/2)
7	11/16	Dynamic Programming (1/2)
8	11/30	Dynamic Programming (2/2)
9	12/7	Network Flow (1/2)
10	12/14	Network Flow (2/2)
11	12/21	NP and Computational Intractability
12	1/4	Approximation Algorithms (1/2)
13	1/11	Approximation Algorithms (2/2)
14	1/18	Final Examination

Guideline of Final Examination

- A password-protected file will be uploaded to ITC-LMS until the day
- The password will be announced at Zoom used in the class
- After the exam., submit your answer file to ICT-LMS (Assignments/課題)

- You may use textbooks and notes
- Discussion with other students are not allowed
- Internet search is not allowed

- If you cannot take the exam, please email me with the reason
- I will send the password after the examination
- Then, submit your answer file by Jan. 25

Outline

1. Set Cover Problem
2. Monotone Submodular Maximization
3. Knapsack Problem

Set Cover Problem

Problem

- Input: $U = \{e_1, \ldots, e_n\}$, $S_1, S_2, \ldots, S_m \subseteq U$
- Goal: minimize $|J|$ s.t. $J \subseteq [m]$ and $\bigcup_{j \in J} S_j = U$

This problem is $\mathbf{NP}$-hard since the decision version is $\mathbf{NP}$-complete

Example

- $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- $S_1 = \{e_1, e_2, e_3\}$
- $S_2 = \{e_1, e_4, e_5, e_7\}$
- $S_3 = \{e_3, e_6, e_7\}$
- $S_4 = \{e_4, e_5, e_6, e_7\}$

Greedy algorithm

Algorithm

$J \leftarrow \emptyset$;
while $\bigcup_{j \in J} S_j \subsetneq U$ **do**
    Select $j^* \in [m]$ that maximizes $|S_{j^*} \setminus \bigcup_{j \in J} S_j|$;
    $J \leftarrow J \cup \{j^*\}$;
Return J;

Example $\quad J = \emptyset$

- $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- $S_1 = \{e_1, e_2, e_3\}$ → $S_1 \setminus \bigcup_{j \in J} S_j = \{e_1, e_2, e_3\}$
- $S_2 = \{e_1, e_4, e_5, e_7\}$ → $S_2 \setminus \bigcup_{j \in J} S_j = \{e_1, e_4, e_5, e_7\}$
- $S_3 = \{e_3, e_6, e_7\}$ → $S_3 \setminus \bigcup_{j \in J} S_j = \{e_3, e_6, e_7\}$
- $S_4 = \{e_4, e_5, e_6, e_7\}$ → $S_4 \setminus \bigcup_{j \in J} S_j = \{e_4, e_5, e_6, e_7\}$

Greedy algorithm

Algorithm

$J \leftarrow \emptyset$;
while $\bigcup_{j\in J} S_j \subsetneq U$ **do**
    Select $j^* \in [m]$ that maximizes $|S_{j^*} \setminus \bigcup_{j\in J} S_j|$;
    $J \leftarrow J \cup \{j^*\}$;
Return J;

Example $J = \{2\}$

- $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- $S_1 = \{e_1, e_2, e_3\}$ → $S_1 \setminus \bigcup_{j\in J} S_j = \{e_2, e_3\}$
- $S_2 = \{e_1, e_4, e_5, e_7\}$
- $S_3 = \{e_3, e_6, e_7\}$ → $S_3 \setminus \bigcup_{j\in J} S_j = \{e_3, e_6\}$
- $S_4 = \{e_4, e_5, e_6, e_7\}$ → $S_4 \setminus \bigcup_{j\in J} S_j = \{e_6\}$

Greedy algorithm

Algorithm

$J \leftarrow \emptyset$;
while $\bigcup_{j \in J} S_j \subsetneq U$ **do**
    Select $j^* \in [m]$ that maximizes $|S_{j^*} \setminus \bigcup_{j \in J} S_j|$;
    $J \leftarrow J \cup \{j^*\}$;
Return J;

Example $\quad J = \{1, 2\}$

- $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- $S_1 = \{e_1, e_2, e_3\}$
- $S_2 = \{e_1, e_4, e_5, e_7\}$
- $S_3 = \{e_3, e_6, e_7\}$ → $S_3 \setminus \bigcup_{j \in J} S_j = \{e_6\}$
- $S_4 = \{e_4, e_5, e_6, e_7\}$ → $S_4 \setminus \bigcup_{j \in J} S_j = \{e_6\}$

Greedy algorithm

Algorithm

$J \leftarrow \emptyset$;
while $\bigcup_{j \in J} S_j \subsetneq U$ **do**
    Select $j^* \in [m]$ that maximizes $|S_{j^*} \setminus \bigcup_{j \in J} S_j|$;
    $J \leftarrow J \cup \{j^*\}$;
Return J;

Example $\quad J = \{1, 2, 3\}$

- $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- $S_1 = \{e_1, e_2, e_3\}$
- $S_2 = \{e_1, e_4, e_5, e_7\}$
- $S_3 = \{e_3, e_6, e_7\}$
- $S_4 = \{e_4, e_5, e_6, e_7\} \longrightarrow S_4 \setminus \bigcup_{j \in J} S_j = \emptyset$

Analysis of the greedy algorithm

Theorem

The greedy algorithm is an $\mathrm{O}(\log n)$-approximation algorithm

Proof

- $e_1, e_2, \ldots, e_n$: an order in which they were covered by the algorithm

- $p(e_k) :=$ (# of covered elements at the same time as e_k)

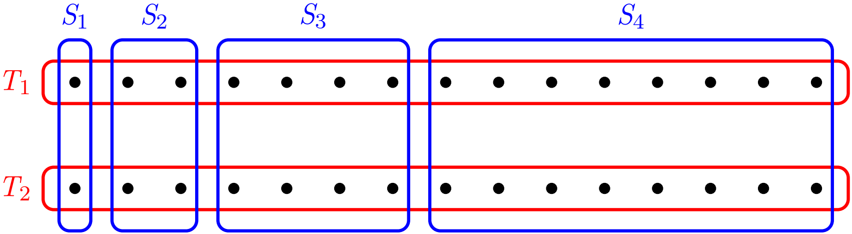
$$\geq \frac{n-k+1}{\mathrm{OPT}}$$

($\because$ at least $n-k+1$ elements are not covered before e_k is covered)

- $$\mathrm{ALG} = \sum_{e_k \in U} \frac{1}{p(e_k)} \leq \sum_{e_k \in U} \frac{\mathrm{OPT}}{n-k+1} = \sum_{t=1}^{n} \frac{\mathrm{OPT}}{t} = H_n \cdot \mathrm{OPT} = O(\log n) \cdot \mathrm{OPT}$$

Worst case example

The greedy algorithm outputs $\Omega(\log n)$-approx. solution for the following



Outline

Definition

A function $f\colon 2^E \to \mathbb{R}$ is called

- monotone if $f(X) \leq f(Y)$ $(\forall X \subseteq \forall Y \subseteq E)$
- submodular if $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ $(\forall X, Y \subseteq E)$
- normalized if $f(\emptyset) = 0$

Example

- linear function: $f(X) = \sum_{i \in X} w_i$
- coverage function: $f(X) = \left|\bigcup_{i \in X} S_i\right|$
- matroid rank function: $f(X) = \max\{|X'| \,:\, X' \subseteq X,\ X' \in \mathcal{I}\}$

Monotone Submodular Maximization

Problem

- Input: monotone submodular function $f\colon 2^E \to \mathbb{R}_+$ and $k \in \{1, \ldots, |E|\}$
- Goal: maximize $f(X)$ subject to $|X| \le k$

Set cover problem $\le_{\mathrm{P}}$ Monotone submodular maximization problem

→ Monotone submodular maximization problem is **NP**-hard

Example

- $U = \{1, 2, \ldots, 13\}$
- $S_1 = \{1, 2, 3\}$
- $S_2 = \{2, 3, 4, 5, 6\}$
- $S_3 = \{4, 5, 6, 7, 8, 9, 10\}$
- $S_4 = \{8, 9, 10, 11, 12, 13\}$

- $E = \{1, 2, 3, 4\}$
- $f(X) = \left|\bigcup_{i \in X} S_i\right|$
- $k = 2$

Greedy algorithm

Initially $S^{(0)} \leftarrow \emptyset$;
for $\ell \leftarrow 1, 2, \ldots, k$ **do**
  Let $e^{(\ell)} \in \arg\max\{f(S^{(\ell-1)} \cup \{e\}) - f(S^{(\ell-1)}) \mid e \in E \setminus S\}$;
  $S^{(\ell)} \leftarrow S^{(\ell-1)} \cup \{e^{(\ell)}\}$;
Return $S^{(k)}$;

Example

- $U = \{1, 2, \ldots, 13\}$
- $S_1 = \{1, 2, 3\}$
- $S_2 = \{2, 3, 4, 5, 6\}$
- $S_3 = \{4, 5, 6, 7, 8, 9, 10\}$
- $S_4 = \{8, 9, 10, 11, 12, 13\}$
- $E = \{1, 2, 3, 4\}$
- $f(X) = \left|\bigcup_{i \in X} S_i\right|$
- $k = 2$

Approximation ratio

Lemma 1

For all $T \subseteq T' \subseteq E$, $f(T') - f(T) \leq \sum_{e \in T' \setminus T}(f(T \cup \{e\}) - f(T))$

Lemma 2

$f(S^{(\ell)}) - f(S^{(\ell-1)}) \geq \frac{1}{|S^* \setminus S^{(\ell-1)}|}(f(S^*) - f(S^{(\ell-1)})) \geq \frac{1}{k}(f(S^*) - f(S^{(\ell-1)}))$

Theorem

$f(S^{(\ell)}) \geq \left(1 - \left(1 - \frac{1}{k}\right)^{\ell}\right) f(S^*)$

The greedy algorithm is a $(1 - 1/e)$-approximation algorithm because $f(S^{(k)}) \geq \left(1 - \left(1 - \frac{1}{k}\right)^{k}\right) f(S^*) \geq \left(1 - \frac{1}{e}\right) f(S^*)$ by the theorem

Proof of Lemma 1

Lemma 1

For all $T \subseteq T' \subseteq E$, $f(T') - f(T) \leq \sum_{e \in T' \setminus T} (f(T \cup \{e\}) - f(T))$

Proof

- Let $T' \setminus T = \{e_1, e_2, \ldots, e_m\}$
- $T_0 = T, T_1 = T \cup \{e_1\}, \ldots, T_\ell = T \cup \{e_1, \ldots, e_\ell\}, \ldots, T_m = T'$
- By submodularity, $f(T_{\ell-1} \cup \{e_\ell\}) - f(T_{\ell-1}) \leq f(T \cup \{e_\ell\}) - f(T)$
- By summing up the inequalities, we get the desired inequality

Proof of Lemma 2

Lemma 1

For all $T \subseteq T' \subseteq E$, $f(T') - f(T) \leq \sum_{e \in T' \setminus T}(f(T \cup \{e\}) - f(T))$

Lemma 2

$f(S^{(\ell)}) - f(S^{(\ell-1)}) \geq \frac{1}{|S^* \setminus S^{(\ell-1)}|}(f(S^*) - f(S^{(\ell-1)})) \geq \frac{1}{k}(f(S^*) - f(S^{(\ell-1)}))$

Proof

- For each $\ell \in \{1, \ldots, k\}$, we have

$$\begin{aligned} f(S^*) - f(S^{(\ell-1)}) &\leq f(S^* \cup S^{(\ell-1)}) - f(S^{(\ell-1)}) \\ &\leq \sum_{e \in S^* \setminus S^{(\ell-1)}} (f(S^{(\ell-1)} \cup \{e\}) - f(S^{(\ell-1)})) \\ &\leq |S^* \setminus S^{(\ell-1)}| \cdot \max_{e \in S^* \setminus S^{(\ell-1)}} (f(S^{(\ell-1)} \cup \{e\}) - f(S^{(\ell-1)})) \\ &\leq |S^* \setminus S^{(\ell-1)}| \cdot (f(S^{(\ell)}) - f(S^{(\ell-1)})) \end{aligned}$$

by Lemma 1

- The second inequality of Lemma 2 holds by $|S^* \setminus S^{(\ell-1)}| \leq k$

Proof of Theorem

Theorem

$f(S^{(\ell)}) \geq \left(1 - \left(1 - \frac{1}{k}\right)^{\ell}\right) f(S^*)$

Proof by induction

- Base step $(\ell = 0)$: $f(S^{(0)}) = 0 = \left(1 - (1 - 1/k)^0\right) f(S^*)$
- Induction step $(\ell \geq 1)$:

$$
\begin{aligned}
f(S^{(\ell+1)}) &\geq f(S^{(\ell)}) + \frac{1}{k}\left(f(S^*) - f(S^{(\ell)})\right) \\
&= \left(1 - \frac{1}{k}\right) \cdot f(S^{(\ell)}) + \frac{1}{k} \cdot f(S^*) \\
&\geq \left(1 - \frac{1}{k}\right)\left(1 - \left(1 - \frac{1}{k}\right)^{\ell}\right) \cdot f(S^*) + \frac{1}{k} \cdot f(S^*) \\
&= \left(1 - \left(1 - \frac{1}{k}\right)^{\ell+1}\right) \cdot f(S^*)
\end{aligned}
$$

by Lemma 2

induction hypothesis

Outline

1. Set Cover Problem
2. Monotone Submodular Maximization
3. Knapsack Problem

Knpsack problem

Problem

- Input: items $E = \{1, 2, \dots, n\}$ and a capacity $W \in \mathbb{Z}_+$
  item i has value $v_i \in \mathbb{Z}_+$ and size $w_i \in \mathbb{Z}_+$
- Goal: maximize $\sum_{i \in I} v_i$ subject to $I \subseteq \{1, 2, \dots, n\}$ $\sum_{i \in I} w_i \leq W$

Recap: knapsack problem is **NP**-hard

Examples

$W = 16$

i	v_i	w_i
1	55	4
2	61	2
3	82	9
4	38	1
5	63	3

Knpsack problem

Problem

- Input: items $E = \{1, 2, \ldots, n\}$ and a capacity $W \in \mathbb{Z}_+$
  item i has value $v_i \in \mathbb{Z}_+$ and size $w_i \in \mathbb{Z}_+$
- Goal: maximize $\sum_{i \in I} v_i$ subject to $I \subseteq \{1, 2, \ldots, n\}$ $\sum_{i \in I} w_i \leq W$

Recap: knapsack problem is $\mathbf{NP}$-hard

Examples

$W = 16$

i	v_i	w_i
1	55	4
2	61	2
3	82	9
4	38	1
5	63	3

optimal value is $61 + 82 + 38 + 63 = 244$

Continuous knapsack problem

(Integral) Knapsack problem

$$\max \quad \sum_{i\in E} v_i x_i \quad \text{s.t.} \quad \sum_{i\in E} w_i x_i \le W, \quad x_i \in \{0,1\} \ (\forall i \in E)$$

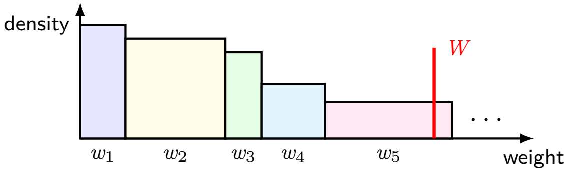
Continuous Knapsack problem

$$\max \quad \sum_{i\in E} v_i x_i \quad \text{s.t.} \quad \sum_{i\in E} w_i x_i \le W, \quad x_i \in [0,1] \ (\forall i \in E)$$

Observations

- $\mathrm{OPT}^{\text{int}} \le \mathrm{OPT}^{\text{cont}}$
- fractional knapsack problem can be solved by a greedy algorithm

  descending order of their density (values per unit weight v_i/w_i)



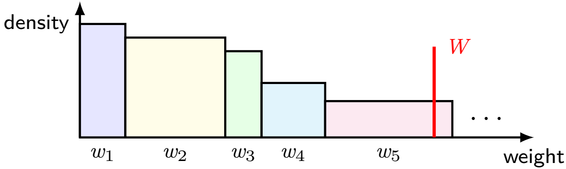
1/2-approximation algorithm

1. Sort items (with size $\leq W$) and relabel so that $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$;
2. Let k be the index such that $\sum_{i=1}^{k-1} w_i \leq W < \sum_{i=1}^{k} w_i$;
3. Pick the better of $\{1, 2, \ldots, k-1\}$ and $\{k\}$;

Theorem

The above algorithm is $1/2$-approximation

- $\boldsymbol{x}^* = (\underbrace{1}_{1}, \ldots, \underbrace{1}_{k-1}, \underbrace{\frac{W - \sum_{i=1}^{k-1} w_i}{w_k}}_{k}, \underbrace{0}_{k+1}, \ldots, \underbrace{0}_{n})$ is optimal for cont. ver.
- $2 \max\left\{\sum_{i=1}^{k-1} v_i, v_k\right\} \geq \sum_{i=1}^{k} v_i \geq \sum_{i=1}^{n} v_i x_i^* \geq \mathrm{OPT}^{\text{cont}} \geq \mathrm{OPT}^{\text{int}}$

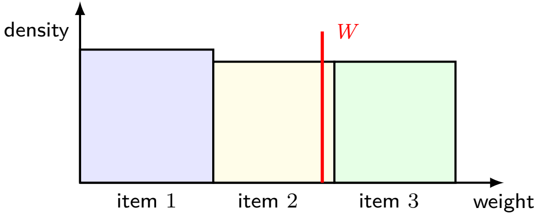


Worst case

Instance

$n = 3$ and $W = 2$

- item 1: $v_1 = (1+\epsilon)^2$, $w_1 = 1+\epsilon$
- item 2: $v_2 = 1$, $w_2 = 1$
- item 3: $v_3 = 1$, $w_3 = 1$



Analysis $\frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{(1+\epsilon)^2}{2} \to \frac{1}{2} \quad (\epsilon \to 0)$

PTAS and FPTAS

Definition: Polynomial-time approximation scheme (PTAS)

A PTAS is a $(1-\epsilon)$-approximation algorithm that runs in time polynomial in the problem size for any constant $\epsilon > 0$

Definition: Fully polynomial-time approximation scheme (FPTAS)

A FPTAS is a $(1-\epsilon)$-approximation algorithm that runs in time polynomial in both the problem size and $1/\epsilon$ for any $\epsilon > 0$

- A $(1-\epsilon)$-approximation algorithm that runs in $\mathrm{O}(n^{1/\epsilon})$ or $\mathrm{O}(n^{(1/\epsilon)^{1/\epsilon}})$ is PTAS but not FPTAS
- We will see PTAS and FPTAS for the knapsack problem

PTAS

Idea: guess the top-ℓ for value in the optimal solution

Algorithm

```
Sort items (with size ≤ W) and relabel so that v_1/w_1 ≥ v_2/w_2 ≥ ··· ≥ v_n/w_n;
foreach nonempty X ⊆ E with |X| ≤ ℓ and Σ_{i∈X} w_i ≤ W do
    S_X ← X and v* ← min_{i∈X} v_i;
    for i ← 1, 2, ..., n do
        if i ∉ X, v_i ≤ v*, and w(S_X) + w_i ≤ W then
            S_X ← S_X ∪ {i};
Return the optimal solution among S_X;
```

Theorem

The above algorithm is $(1 - \frac{1}{\ell+1})$-approximation and runs in $\mathrm{O}(n^{\ell+1})$ time

- $\mathrm{O}(n^{\ell})$ possibilities of X imply $\mathrm{O}(n^{\ell+1})$ time $+\mathrm{O}(n \log n)$ time for sort
- By setting $\ell = \lceil 1/\epsilon \rceil - 1$, it is $(1-\epsilon)$-approx. alg. that runs in $\mathrm{O}(n^{\lceil 1/\epsilon \rceil})$ time

Proof of approximation ratio

Theorem

The algorithm is $(1 - \frac{1}{\ell+1})$-approximation

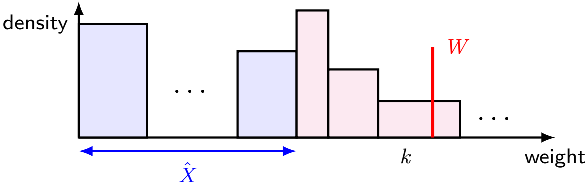
Proof

- Assumption: the optimal solution contains at least $\ell + 1$ items

  since otherwise the algorithm outputs the optimal solution

- Let X^* be the optimal solution and let $\hat{X}$ be the top-ℓ items in it
- $v(X^*) \leq v(S_{\hat{X}}) + \sum_{i=1}^{k} v_i \leq v(S_{\hat{X}}) + \sum_{i=1}^{k-1} v_i + \frac{v(X^*)}{\ell+1}$

  ⟶ $\mathrm{ALG} \geq v(S_{X^*}) - v_k \geq (1 - \frac{1}{\ell+1}) v(X^*) = (1 - \frac{1}{\ell+1})\mathrm{OPT}$



Recap: Dynamic programming for knapsack problem

$$\mathrm{OPT}(k,w) = \max\bigl\{v(X) \mid X \subseteq \{1,2,\dots,k\},\ w(X) \le w\bigr\}$$

Recursive formula

$$\mathrm{OPT}(k,w) = \begin{cases} 0 & \text{if } k=0, \\ \mathrm{OPT}(k-1,w) & \text{if } w_k > w \\ \max\{\mathrm{OPT}(k-1,w),\ \mathrm{OPT}(k-1,b-w_k)+v_k\} & \text{otherwise} \end{cases}$$

Compute for $k = 0,1,\dots,n$ and $w = 0,1,\dots,W$ ⟶ $\mathrm{O}(nW)$ time

Example $(w_i, v_i) = (4,55),(2,61),(9,82),(1,38),(3,63),\ W = 16$

$k\backslash w$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	55	55	55	55	55	55	55	55	55	55	55	55	55
2	0	0	61	61	61	61	116	116	116	116	116	116	116	116	116	116	116
3	0	0	61	61	61	61	116	116	116	116	116	143	143	143	143	198	198
4	0	38	61	99	99	99	116	154	154	154	154	154	181	181	181	198	236
5	0	38	61	99	101	124	162	162	162	179	217	217	217	217	217	244	244

Another dynamic programming for knapsack problem

$$\mathrm{OPT}(k, v) = \min\{w(X) \mid X \subseteq \{1, 2, \dots, k\},\ v(X) = v\}$$

Recursive formula

$$\mathrm{OPT}(k, v) = \begin{cases} 0 & \text{if } v = 0, \\ +\infty & \text{else if } k = 0, \\ \min\{\mathrm{OPT}(k, v),\ \mathrm{OPT}(k-1, v - v_k) + w_k\} & \text{else if } v \geq v_k, \\ \mathrm{OPT}(k-1, v) & \text{otherwise} \end{cases}$$

Compute for $k = 0, 1, \dots, n$ and $v = 0, 1, \dots, \sum_{i=1}^{n} v_i$ ⟶ $\mathrm{O}(n^2 \max_i v_i)$ time

Example $(w_i, v_i) = (55, 4), (61, 2), (82, 6), (38, 1), (63, 3),\ W = 160$

$k \backslash v$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	55	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	61	∞	55	∞	116	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	0	∞	61	∞	55	∞	82	∞	143	∞	137	∞	198	∞	∞	∞	∞
4	0	38	61	99	55	93	82	120	143	181	137	175	198	236	∞	∞	∞
5	0	38	61	63	55	93	82	118	143	145	137	175	198	200	238	261	299

FPTAS

Idea: scale down the value of every item into $0, 1, \dots, \ell$

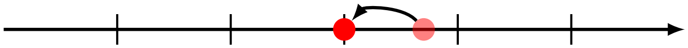
Algorithm

1. Let $v_{\max} = \max_{i'=1}^{n} v_{i'}$;
2. For each item i, let $\hat{v}_i = \left\lfloor \frac{v_i}{v_{\max}} \cdot \ell \right\rfloor$ (delete i with $w_i > W$);
3. Compute the optimal solution for $\big((w_1, \hat{v}_1), \dots, (w_n, \hat{v}_n); W\big)$ by the dynamic programming and output it;

Theorem

The above algorithm is $(1 - \frac{n}{\ell})$-approximation and runs in $\mathrm{O}(n^2\ell)$ time

- The running time is $\mathrm{O}(n^2\ell)$ since $\max_i \hat{v}_i = \ell$
- By setting $\ell = \left\lfloor \frac{n}{\epsilon} \right\rfloor$, it is $(1 - \epsilon)$-approx. alg. that runs in $\mathrm{O}(\frac{n^3}{\epsilon})$ time



Proof of approximation ratio

Theorem

The algorithm is $(1-\frac{n}{\ell})$-approximation

Proof

- Let X^* be the optimal solution and let X be the output of the algorithm
- $\frac{v_i}{v_{\max}}\cdot\ell - 1 < \hat{v}_i = \left\lfloor \frac{v_i}{v_{\max}}\cdot\ell \right\rfloor \le \frac{v_i}{v_{\max}}\cdot\ell$
- Thus, we have

X is optimal for $\hat{v}_i$

$$\sum_{i\in X} v_i \ge \sum_{i\in X} \hat{v}_i \cdot \frac{v_{\max}}{\ell} \ge \frac{v_{\max}}{\ell}\sum_{i\in X^*}\hat{v}_i \ge \frac{v_{\max}}{\ell}\sum_{i\in X^*}\left(\frac{v_i}{v_{\max}}\ell - 1\right)$$

$$= \sum_{i\in X^*} v_i - \frac{v_{\max}}{\ell}|X^*| \ge \left(1-\frac{|X^*|}{\ell}\right)\sum_{i\in X^*} v_i \ge \left(1-\frac{n}{\ell}\right)\sum_{i\in X^*} v_i$$

$v_{\max} \le \sum_{i\in X^*} v_i$