Advanced Algorithms

Rounding Data

尹一通 Nanjing University, 2022 Fall

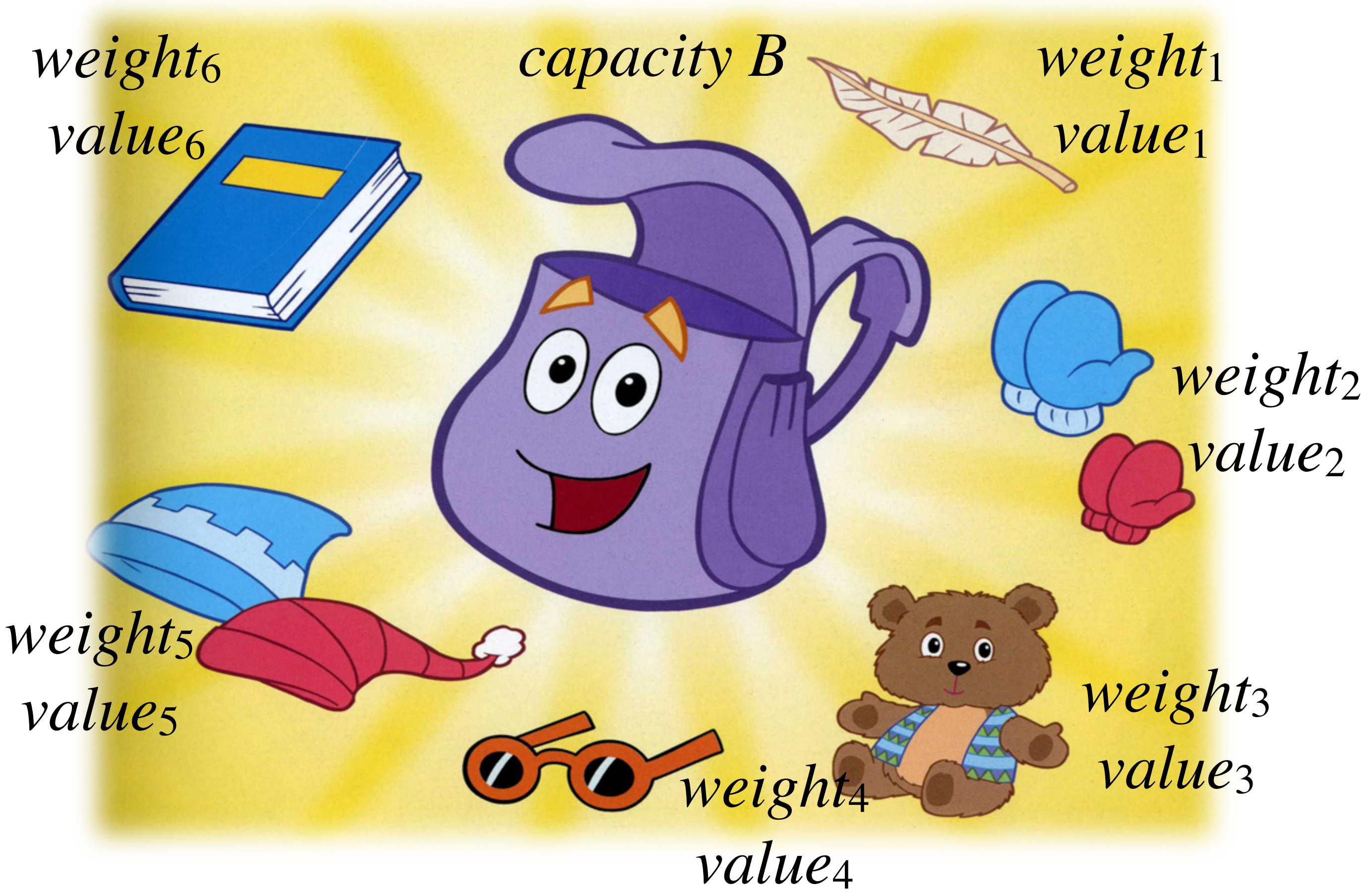
Knapsack Problem

Instance: n items $i = 1,2,\ldots,n$;

weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;

knapsack capacity $B \in \mathbb{Z}^+$;

Find a subset of items whose total weight is bounded by B and total value is maximized.

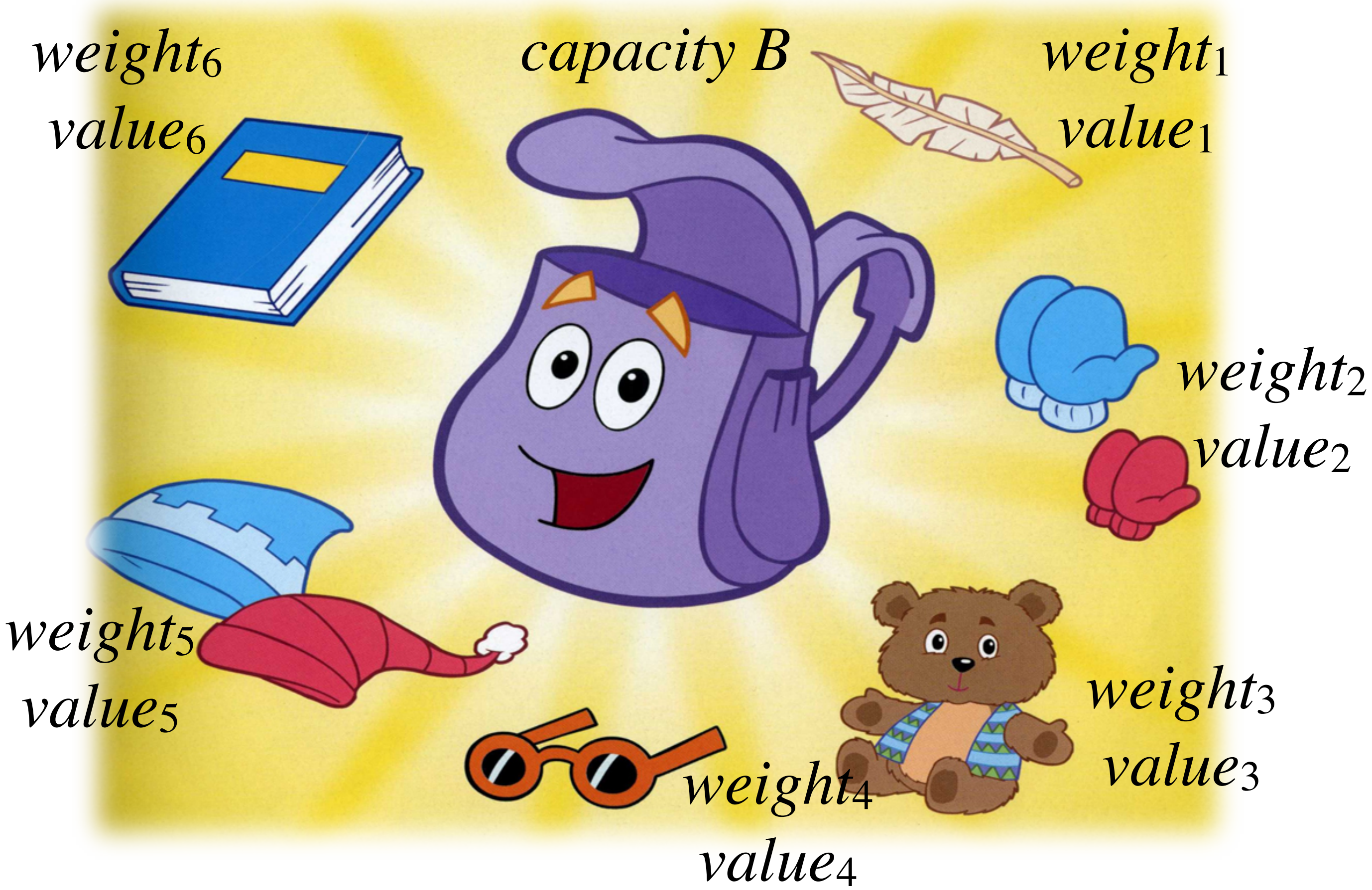


Knapsack Problem

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

Find an $S \subseteq \{1,\ldots,n\}$ that maximizes $\sum_{i\in S} v_i$
subject to $\sum_{i\in S} w_i \le B$.

- 0-1 Knapsack problem
- one of Karp's 21 **NP**-complete problems



Greedy Can Fail Badly

Instance: n items $i = 1,2,\ldots,n$;

weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;

knapsack capacity $B \in \mathbb{Z}^+$;

Find an $S \subseteq \{1,\ldots,n\}$ that maximizes $\sum_{i\in S} v_i$

subject to $\sum_{i\in S} w_i \le B$.

Greedy Fit:

Sort items non-decreasingly in v_i/w_i;

scan items one-by-one in that order, for each item:

include the item in the knapsack if it fits;

approximation ratio: arbitrarily bad

Dynamic Programming

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

- **Define:**

$A(i, v) \triangleq$ minimum total weight of $S \subseteq \{1,2,\ldots,i\}$
with total value *exactly* v

$A(i, v) \triangleq \infty$ if no such S exists

- Answer to the knapsack problem:

$$\max \left\{ v \mid A(n, v) \leq B \right\}$$

Dynamic Programming

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

- **Define:**

$$A(i, v) = \begin{cases} \min\limits_{\substack{S \subseteq \{1,\ldots,i\} \\ \sum_{j\in S} v_j = v}} \sum_{j\in S} w_j & \text{if } \exists S \subseteq \{1,\ldots,i\} \text{ s.t. } v = \sum\limits_{j\in S} v_j \\ \infty & \text{otherwise} \end{cases}$$

- Answer to the knapsack problem:

$$\max \left\{ v \mid A(n, v) \leq B \right\}$$

Dynamic Programming

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

$A(i, v) \triangleq$ minimum total weight of $S \subseteq \{1,2,\ldots,i\}$
with total value *exactly* v

- **Recursion**: for $1 \le i \le n$ and $1 \le v \le V = \sum_i v_i$

$$A(i,v) = \min\left\{A(i-1,v), A(i-1,v-v_i) + w_i\right\} \text{ for } i > 1$$

$$A(1,v) = \begin{cases} w_1 & \text{if } v = v_1 \\ \infty & \text{o.w.} \end{cases}$$

- **Output**: $\max\left\{v \mid A(n,v) \le B\right\}$

Dynamic Programming:

Table size: $n \times V$.

Total time cost: $O(nV)$.

Polynomial Time?

Computational Complexity

- **decision problem** $f : \{0,1\}^* \to \{0,1\}$
- **formal language** $L \subseteq \{0,1\}^*$ $\quad L = \left\{x \in \{0,1\}^* \mid f(x) = 1\right\}$
- poly-time **Turing machine** (*algorithm*) M: $\quad \exists c > 0$
  - $\forall x \in \{0,1\}^*$, $M(x)$ terminates within $O(|x|^c)$ steps

    length of x (in bits)
- $\mathbf{P}, \mathbf{NP}$: classes of formal languages (*decision problems*)
- $L \in \mathbf{P}$: $\exists$ poly-time TM M ***decides*** L
  - $M(x) = 1$ iff $x \in L$
- $L \in \mathbf{NP}$: $\exists$ polynomial $p(\cdot)$ and poly-time TM M that ***verifies*** L
  - $x \in L \Longrightarrow \exists$ **certificate** $y \in \{0,1\}^*$ of length $p(|x|)$, $M(x, y) = 1$;
  - $x \notin L \Longrightarrow \forall y \in \{0,1\}^*$ of length $p(|x|)$, $M(x, y) = 0$;

Dynamic Programming

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

- Polynomial-time Algorithm A:

  $\exists c > 0$, $\forall$ input x, $A(x)$ terminates within $O(|x|^c)$ steps

  $|x| =$ length of input x (in *binary* code)

- Pseudo-polynomial-time Algorithm A:

  above definition (except
  $|x| =$ length in *unary* code)

  $V = \sum_i v_i$

Dynamic Programming:

Table size: $n \times V$.

Total time cost: $O(nV)$.

Pseudo-Polynomial Time!

Scaling & Rounding

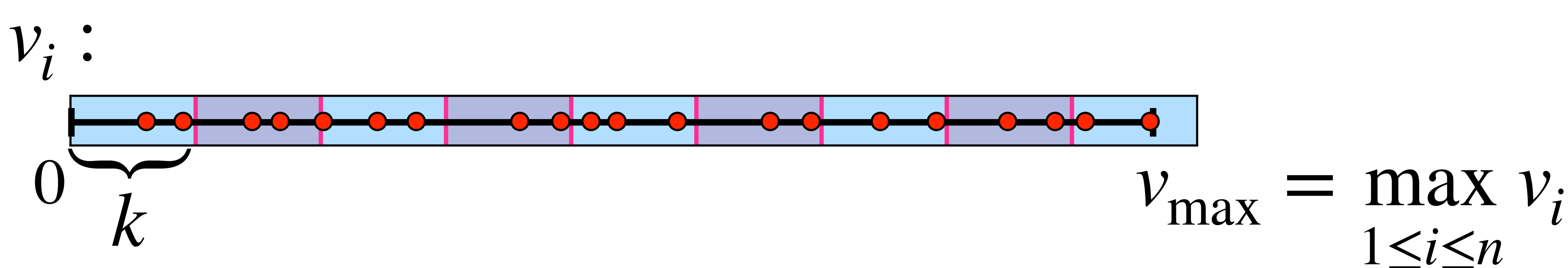
Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

DP with truncated precision:

Set $k =$ (to be determined);

for $i = 1,2,\ldots,n$: let $v_i' = \lfloor v_i / k \rfloor$;

return the knapsack solution found by DP using new values v_i' (but old weights w_i and capacity B);



Scaling & Rounding

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

DP with truncated precision:

Set $k =$ (to be determined);

for $i = 1,2,\ldots,n$: let $v_i' = \lfloor v_i/k \rfloor$;

return the knapsack solution found by DP using new values v_i' (but old weights w_i and capacity B);

- **Complexity:** $O(nV') = O(nV/k)$ $V' = \sum_i v_i'$
- **Soundness:** the output must be a *feasible* solution of the original instance since weights and capacity are unchanged.

Instance: n items $i = 1,2,\ldots,n$;

weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;

knapsack capacity $B \in \mathbb{Z}^+$;

DP with truncated precision:

Set $k =$ (to be determined);

for $i = 1,2,\ldots,n$: let $v_i' = \lfloor v_i / k \rfloor$;

return the knapsack solution found by DP using new values v_i' (but old weights w_i and capacity B);

- S^* : optimal knapsack solution of the original instance

$$OPT = \sum_{i \in S^*} v_i = k \sum_{i \in S^*} \frac{v_i}{k} \le k \sum_{i \in S^*} \left(\left\lfloor \frac{v_i}{k} \right\rfloor + 1 \right) \le k \sum_{i \in S^*} v_i' + nk$$

- S : solution returned by the algorithm (optimal solution of the scaled instance) $\Longrightarrow \sum_{i \in S} v_i' \ge \sum_{i \in S^*} v_i'$

$$SOL = \sum_{i \in S} v_i \ge k \sum_{i \in S} \left\lfloor \frac{v_i}{k} \right\rfloor = k \sum_{i \in S} v_i' \ge k \sum_{i \in S^*} v_i' \ge OPT - nk$$

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

DP with truncated precision:

Set $k =$ (to be determined);

for $i = 1,2,\ldots,n$: let $v_i' = \lfloor v_i/k \rfloor$;

return the knapsack solution found by DP using new values v_i' (but old weights w_i and capacity B);

- **Complexity**: $O(nV/k) = O\left(n^2 v_{\max}/k\right)$ $\quad V = \sum_i v_i \le n v_{\max}$
- **Approximation ratio:**

$$SOL \ge OPT - nk \implies \frac{SOL}{OPT} \ge 1 - \frac{nk}{OPT} \ge 1 - \frac{nk}{v_{\max}}$$

- **WLOG, assume**: $OPT \ge v_{\max} = \max_i v_i$

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

- For arbitrary $0 < \epsilon < 1$:

DP with truncated precision:

Set $k = \lfloor \epsilon \cdot v_{\max}/n \rfloor$ where $v_{\max} = \max_i v_i$

for $i = 1,2,\ldots,n$: let $v_i' = \lfloor v_i/k \rfloor$;

return the knapsack solution found by DP using new values v_i' (but old weights w_i and capacity B);

- **Complexity:** $O\left(n^2 v_{\max}/k\right) = O\left(\frac{n^3}{\epsilon}\right)$
- **Approximation ratio:** $\frac{SOL}{OPT} \geq 1 - \frac{nk}{v_{\max}} \geq 1 - \epsilon$

Approximation Ratio

- **Optimization** problem:
  - instance I: $OPT(I)$ = optimum of instance I
  - algorithm $\mathscr{A}$: $SOL_A(I)$ = output of $\mathscr{A}$ on instance I
- **Minimization:** algorithm $\mathscr{A}$ has approximation ratio $\alpha \geq 1$

  if ∀ instance I : $\dfrac{SOL_{\mathscr{A}}(I)}{OPT(I)} \leq \alpha$

- **Maximization:** algorithm $\mathscr{A}$ has approximation ratio $\alpha \leq 1$

  if ∀ instance I : $\dfrac{SOL_{\mathscr{A}}(I)}{OPT(I)} \geq \alpha$

- ϵ-approximation:

$$(1-\epsilon)OPT(I) \leq SOL_{\mathscr{A}}(I) \leq (1+\epsilon)OPT(I)$$

(maximization) (minimization)

PTAS and FPTAS

- **Optimization** problem:
  - instance I: $OPT(I)$ = optimum of instance I
  - algorithm $\mathscr{A}$: $SOL_{\mathscr{A}}(\epsilon, I)$ = output of A on I and $0 < \epsilon < 1$
- ϵ**-approximation**:

$$(1-\epsilon)OPT(I) \le SOL_{\mathscr{A}}(\epsilon, I) \le (1+\epsilon)OPT(I)$$

(maximization) (minimization)

Algorithm $\mathscr{A}$ is a **PTAS** (**Polynomial-Time Approximation Scheme**) if for every $0 < \epsilon < 1$ and instance I, it returns an ϵ**-approximation** in $\text{poly}(|I|)$ time, where $|I|$ is the length of I in binary code.

Algorithm $\mathscr{A}$ is a **FPTAS** (**Fully Polynomial-Time Approximation Scheme**) if for every $0 < \epsilon < 1$ and every instance I, it returns an ϵ**-approximation** in $\text{poly}(1/\epsilon, |I|)$ time.

Instance: n items $i = 1,2,\ldots,n$;
weights $w_1,\ldots,w_n \in \mathbb{Z}^+$; values $v_1,\ldots,v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

- For arbitrary $0 < \epsilon < 1$:

DP with truncated precision:

Set $k = \lfloor \epsilon \cdot v_{\max}/n \rfloor$ where $v_{\max} = \max_i v_i$

for $i = 1,2,\ldots,n$: let $v_i' = \lfloor v_i/k \rfloor$;

return the knapsack solution found by DP using new values v_i' (but old weights w_i and capacity B);

- **Complexity:** $O\left(n^2 v_{\max}/k\right) = O\left(\frac{n^3}{\epsilon}\right)$
- **Approximation ratio:** $\frac{SOL}{OPT} \geq 1 - \frac{nk}{v_{\max}} \geq 1-\epsilon$

} **FPTAS**

Are FPTASs the "best" approximation algorithms?