Advanced Algorithms

南京大学

尹一通
Knapsack Problem

**Instance:** $n$ items $i=1, 2, \ldots, n$;
- weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$;
- 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 $i=1,2,...,n$;
   weights $w_1, w_2, ..., w_n \in \mathbb{Z}^+$; values $v_1, v_2, ..., v_n \in \mathbb{Z}^+$;
   knapsack capacity $B \in \mathbb{Z}^+$;

Find an $S \subseteq \{1,2, ..., n\}$ that maximizes $\sum_{i \in S} v_i$
subject to $\sum_{i \in S} w_i \leq B$.

• 0-1 Knapsack problem
• one of Karp’s 21 NP-complete problems
Greedy Heuristics

Instance: $n$ items $i=1,2,\ldots, n$;
weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;
Find an $S \subseteq \{1,2,\ldots, n\}$ that maximizes $\sum_{i \in S} v_i$
subject to $\sum_{i \in S} w_i \leq B$.

Sort all items according to the ratio $r_i = v_i/w_i$
so that $r_1 \geq r_2 \geq \cdots \geq r_n$;
for $i=1,2,\ldots, n$
item $i$ joins $S$ if the resulting total weight $\leq B$;

approximation ratio: arbitrarily bad
Dynamic Programming

**Instance:** $n$ items $i=1,2,\ldots,n$;
weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

define:

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

$$A(i, v) = \infty \text{ if no such } S \text{ exists}$$
Dynamic Programming

**Instance:** $n$ items $i=1,2,\ldots,n$;
weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

define:

$$A(i, v) = \begin{cases} 
\min_{S \subseteq \{1,2,\ldots,i\}} \sum_{j \in S} w_j & \text{if } \exists S \subseteq \{1,2,\ldots,i\}, \\
\infty & \text{otherwise}
\end{cases}$$

if $\sum_{j \in S} v_j = v$, $j \in S$
Dynamic Programming

**Instance:**  \( n \) items \( i=1,2, \ldots, n; \)
- weights \( w_1, w_2, \ldots, w_n \in \mathbb{Z}^+; \)
- values \( v_1, v_2, \ldots, v_n \in \mathbb{Z}^+; \)
- knapsack capacity \( B \in \mathbb{Z}^+; \)

\[
A(i, v) = \text{minimum total weight of } S \subseteq \{1,2, \ldots, i\}
\text{ with total value exactly } v
\]

**recursion:**

\[
A(i, v) = \min\{ A(i-1, v), A(i-1, v-v_i) + w_i \} \quad \text{for } i > 1
\]

\[
A(1, v) = \begin{cases} 
w_1 & \text{if } v = v_1 \\
\infty & \text{otherwise}
\end{cases}
\]

\[
1 \leq i \leq n, \quad 1 \leq v \leq V = \sum_i v_i
\]

**Dynamic programming:**
- table size \( O(nV) \)
- time complexity \( O(nV) \)
Dynamic Programming

**Instance:** $n$ items $i=1,2,\ldots,n$;  
weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$;  
knapsack capacity $B \in \mathbb{Z}^+$;

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

$$A(i, v) = \text{minimum total weight of } S \subseteq \{1,2,\ldots,i\}$$

with total value **exactly** $v$

knapsack:

$$\max v \text{ that } A(n,v) \leq B$$

**Dynamic programming:**
- table size $O(nV)$
- time complexity $O(nV)$

Polynomial Time?
Polynomial Time

**Instance:**  $n$ items $i=1,2,\ldots,n$;
weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

**time complexity:**  $O(nV)$ where $V = \sum_i v_i$

- **polynomial-time** Algorithm $A$:
  $\exists$ constant $c$, $\forall$ input $x \in \{0,1\}^*$, $A(x)$ terminates in $|x|^c$ steps
  $|x| = \text{length of input } x$ (in *binary* code)

- **pseudopolynomial-time** Algorithm $A$:
  $\exists$ constant $c$, $\forall$ input $x \in \{0,1\}^*$, $A(x)$ terminates in $|x|^c$ steps
  $|x| = \text{length of input } x$ (in *unary* code)
Dynamic Programming

**Instance:** Let $n$ items $i=1,2, \ldots, n$;
weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$; values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

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

**Dynamic programming:**

time complexity $O(nV)$

where $V = \sum_i v_i$

**Knapsack:**

$$A(i, v) = \min\{ A(i-1, v) , A(i-1, v-v_i) + w_i \}$$

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

**Pseudo-Polynomial Time!**
Scaling & Rounding

**Instance:** \( n \) items \( i=1,2, \ldots, n; \)
weights \( w_1, w_2, \ldots, w_n \in \mathbb{Z}^+; \) values \( v_1, v_2, \ldots, v_n \in \mathbb{Z}^+; \)
knapsack capacity \( B \in \mathbb{Z}^+; \)

Set \( k = \) (to be fixed) ;
for \( i=1,2, \ldots, n, \) let \( v'_i = \lfloor v_i / k \rfloor; \)
return the knapsack solution found by
dynamic programming with new values \( v'_i ; \)

\[
 v_i = \begin{cases} 
 0 & \text{if } 0 \leq v_i < k \\
 1 & \text{if } k \leq v_i < 2k \\
 \vdots \\
 1 & \text{if } (n-1)k \leq v_i < nk \\
 v_{\max} = \max_{1 \leq i \leq n} v_i & \text{if } nk \leq v_i < (n+1)k 
\end{cases}
\]
Scaling & Rounding

**Instance:** $n$ items $i=1,2,...,n$; weights $w_1, w_2, ..., w_n \in \mathbb{Z}^+$; values $v_1, v_2, ..., v_n \in \mathbb{Z}^+$; knapsack capacity $B \in \mathbb{Z}^+$;

Set $k = (to be fixed)$;
for $i=1,2,...,n$, let $v'_i = \lfloor v_i / k \rfloor$; return the knapsack solution found by dynamic programming with new values $v'_i$;

time complexity: $O(n \cdot V') = O(nV/k)$

where $V' = \sum_i v'_i = \sum_i \lfloor v_i / k \rfloor = O(V/k)$

and $V = \sum_i v_i$
Instance: $n$ items $i=1,2,...,n$;
weights $w_1, w_2, ..., w_n \in \mathbb{Z}^+$; values $v_1, v_2, ..., v_n \in \mathbb{Z}^+$;
knapsack capacity $B \in \mathbb{Z}^+$;

Set $k = \text{(to be fixed)}$;
for $i=1,2,...,n$, let $v'_i = \lfloor v_i / k \rfloor$;
return the knapsack solution found by dynamic programming with new values $v'_i$;

**Time complexity:** $O(nV/k)$ where $V = \sum_i v_i$

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

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

$S$: the solution returned by the algorithm
(optimal solution of the scaled instance)

$$SOL = \sum_{i \in S} v_i \geq k \sum_{i \in S} \left\lfloor \frac{v_i}{k} \right\rfloor \geq k \sum_{i \in S^*} \left\lfloor \frac{v_i}{k} \right\rfloor \geq OPT - nk$$
**Instance:** \( n \) items \( i=1, 2, \ldots, n; \) 
- weights \( w_1, w_2, \ldots, w_n \in \mathbb{Z}^+; \) 
- values \( v_1, v_2, \ldots, v_n \in \mathbb{Z}^+; \) 
- knapsack capacity \( B \in \mathbb{Z}^+; \)

Set \( k = \) (to be fixed); 
for \( i=1, 2, \ldots, n, \) let \( v'_i = \lfloor v_i / k \rfloor; \) 
return the knapsack solution found by dynamic programming with new values \( v'_i; \)

**time complexity:** \( O(nV/k) \) where \( V = \sum_i v_i \leq nv_{\text{max}} \)

**OPT:** optimal value of the original instance
**SOL:** value of the solution returned by the algorithm

\( SOL \geq OPT - nk \) \[ \quad \Rightarrow \quad \frac{SOL}{OPT} \geq 1 - \frac{nk}{OPT} \geq 1 - \frac{nk}{v_{\text{max}}} \]

**WLOG:** \( OPT \geq v_{\text{max}} = \max_{1 \leq i \leq n} v_i \)
**Instance:** \( n \) items \( i=1,2, \ldots, n; \)
weights \( w_1, w_2, \ldots, w_n \in \mathbb{Z}^+; \) values \( v_1, v_2, \ldots, v_n \in \mathbb{Z}^+; \)
knapsack capacity \( B \in \mathbb{Z}^+; \)

for any \( 0 \leq \varepsilon \leq 1: \)

Set \( k = \left\lfloor \frac{\varepsilon v_{\text{max}}}{n} \right\rfloor \); where \( v_{\text{max}} = \max_{1 \leq i \leq n} v_i \)
for \( i=1,2, \ldots, n; \) let \( v'_i = \lfloor v_i / k \rfloor; \)
return the knapsack solution found by dynamic programming with new values \( v'_i \);

**time complexity:** \( O \left( \frac{n^2 v_{\text{max}}}{k} \right) = O \left( \frac{n^3}{\varepsilon} \right) \)

**OPT:** optimal value of the original instance

**SOL:** value of the solution returned by the algorithm

\[
\frac{\text{SOL}}{\text{OPT}} \geq 1 - \frac{nk}{v_{\text{max}}} \geq 1 - \varepsilon
\]
Approximation Ratio

Optimization problem:

• instance $I$: $\text{OPT}(I) =$ optimum of instance $I$

• algorithm $A$: returns a solution $s$ for every instance $I$

  $\text{SOL}_A(I) =$ value returned by $A$ on instance $I$

minimization: approximation ratio of algorithm $A$ is $\alpha$

  if $\forall$ instance $I : \frac{\text{SOL}_A(I)}{\text{OPT}(I)} \leq \alpha$

maximization: approximation ratio of algorithm $A$ is $\alpha$

  if $\forall$ instance $I : \frac{\text{SOL}_A(I)}{\text{OPT}(I)} \geq \alpha$

$\varepsilon$-approximation: $(1-\varepsilon) \text{OPT}(I) \leq \text{SOL}_A(I) \leq (1+\varepsilon) \text{OPT}(I)$

  (maximization) \hspace{1cm} (minimization)
Approximation Ratio

Optimization problem:

• instance $I$:
  \[ \text{OPT}(I) = \text{optimum of instance } I \]

• algorithm $A$: returns a solution $s$ for every instance $I$ and $0 \leq \varepsilon \leq 1$

\[ \text{SOL}_A(\varepsilon, I) = \text{value returned by } A \text{ on instance } I \text{ and } \varepsilon \]

• $A$ is a Polynomial-Time Approximation Scheme (PTAS) if:
  \[ \forall 0 \leq \varepsilon \leq 1, \text{ } A \text{ returns in polynomial time and } \]

\[ (1-\varepsilon) \text{OPT}(I) \leq \text{SOL}_A(\varepsilon, I) \leq (1+\varepsilon) \text{OPT}(I) \]

  (maximization)    (minimization)

• $A$ is a Fully Polynomial-Time Approximation Scheme (FPTAS) if:
  furthermore, $A$ returns in time $\text{Poly}(1/\varepsilon, n)$ where $n = |I|$

  (in binary code)
Instance: \( n \) items \( i=1,2, \ldots, n; \)
weights \( w_1, w_2, \ldots, w_n \in \mathbb{Z}^+; \) values \( v_1, v_2, \ldots, v_n \in \mathbb{Z}^+; \)
knapsack capacity \( B \in \mathbb{Z}^+; \)

for any \( 0 \leq \epsilon \leq 1: \)

Set \( k = \left\lfloor \frac{\epsilon v_{\text{max}}}{n} \right\rfloor \); where \( v_{\text{max}} = \max_{1 \leq i \leq n} v_i \)
for \( i=1,2, \ldots, n \), let \( v_i' = \lfloor v_i / k \rfloor; \)
return the knapsack solution found by dynamic programming with new values \( v_i' \);

time complexity: \( O\left(\frac{n^3}{\epsilon}\right) \) \( \text{FPTAS} \)
approximation ratio: \( \frac{\text{SOL}}{\text{OPT}} \geq 1 - \epsilon \)

Are FPTASs the “best” approximation algorithms?
Bin Packing

**Instance:** $n$ items $i=1,2, ..., n$;
with sizes $s_1, s_2, ..., s_n \in \mathbb{Z}^+$;

Find a *packing* of the $n$ items into *smallest* number
of *bins* with *capacity* $B \in \mathbb{Z}^+$.  

**items:**

```
...
...
...
```

**bins:**

```
...
...
...
```
**Instance:** \( n \) items \( i=1,2, \ldots, n; \) with sizes \( s_1, s_2, \ldots, s_n \in (0, 1] \);

Find a *packing* of the \( n \) items into *smallest* number of *unit-sized bins*.
Bin Packing

**Instance:** $n$ items $i=1,2,\ldots,n$; with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$; Find a $\phi: [n] \rightarrow [m]$ with smallest $m$ such that $
exists j \in [m], \sum_{i: \phi(i) = j} s_i \leq 1$. 

**items:**

---

**bins:**

---
First Fit

Instance: \( n \) items \( i=1,2, \ldots, n; \)
with sizes \( s_1, s_2, \ldots, s_n \in (0, 1] \);

Find a packing of the \( n \) items into smallest number of unit-sized bins.

- NP-hard.

FirstFit

Initially \( k=1; \)
for \( i=1,2, \ldots, n \)
item \( i \) joins the first bin among \( 1,2, \ldots, k \) in which it fits;
if item \( i \) can fit into none of these \( k \) bins
open a new bin \( k++ \) and item \( i \) joins it;
FirstFit

Initially $k=1$;
for $i=1,2,...,n$
    item $i$ joins the first bin among $1,2,...,k$ in which it fits;
    if item $i$ can fit into none of these $k$ bins
        open a new bin $k++$ and item $i$ joins it;

items:
**Instance:** \( n \) items with sizes \( s_1, s_2, \ldots, s_n \in (0, 1] \);
Pack them into smallest number of unit-sized bins.

**FirstFit**

Initially \( k = 1 \);
for \( i = 1, 2, \ldots, n \)
item \( i \) joins the first bin among \( 1, 2, \ldots, k \) in which it fits;
if item \( i \) can fit into none of these \( k \) bins
open a new bin \( k++ \) and item \( i \) joins it;

**Observation:** All but at most one bin are more than half full.

\[ \sum_i s_i > (SOL - 1) / 2 \]
\[ OPT \geq \sum_i s_i \]
\[ SOL - 1 < 2 \sum_i s_i \leq 2 OPT \]
\[ SOL \leq 2 OPT \]
**Instance:** $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$; Pack them into smallest number of unit-sized bins.

**FirstFit**

Initially $k=1$;

for $i=1,2,\ldots,n$

- item $i$ joins the first bin among $1,2,\ldots,k$ in which it fits;
- if item $i$ can fit into none of these $k$ bins, open a new bin $k++$ and item $i$ joins it;

**Assumption:** If all items are small, $s_i < \gamma < 0.5$

**Observation:** All but at most one bin are more than $(1-\gamma)$ full.

\[ \sum_i s_i > (1-\gamma)(SOL - 1) \quad \Rightarrow \quad SOL \leq OPT / (1-\gamma) + 1 \]
**Instance:** $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$; Pack them into smallest number of unit-sized bins.

**Theorem**
Unless $P = NP$, there is no poly-time approximation algorithm for bin packing with approximation ratio $<3/2$.

Reduction from the **partition** problem:

**Input:** $n$ numbers $x_1, x_2, \ldots, x_n \in \mathbb{Z}^+$. Determine whether $\exists$ a partition of $\{1, 2, \ldots, n\}$ into $A$ and $B$ such that $\sum_{i \in A} x_i = \sum_{i \in B} x_i$.

Reduction: $n$ items with sizes $s_1, s_2, \ldots, s_n$ where $s_i = 2x_i / \sum_j x_j$.

- $\exists$ a packing into 2 unit-sized bins $\Rightarrow$ “yes” $\Rightarrow$ partition problem
- all packings use $\geq 3$ unit-sized bins $\Rightarrow$ “no” $\Rightarrow$ partition problem
**Instance:** \( n \) items with sizes \( s_1, s_2, ..., s_n \in (0, 1]; \)
Pack them into smallest number of unit-sized bins.

**Theorem**
Unless \( P=NP \), there is no poly-time approximation algorithm for bin packing with approximation ratio <\( 3/2 \).

It is **NP-hard** to distinguish between:

- the instances with \( OPT = 2 \);
- the instances with \( OPT \geq 3 \).

**FirstFitDecreasing (FFD)**
Sort items in non-increasing order of sizes; run **FirstFit**;

\( \text{FFD returns a packing into} \leq 11/9 \times OPT + 1 \text{ bins} \)

\( (1+\varepsilon) \times OPT + 1? \)
**Dynamic Programming**

**Instance:** $n$ items with sizes $s_1, s_2, ..., s_n \in (0, 1]$; Pack them into *smallest* number of *unit-sized* bins.

**Assumption:**

- $|\{s_1, s_2, ..., s_n\}| = k$
  
  There are $k$ distinct sizes: $s^{(1)}, s^{(2)}, ..., s^{(k)}$

  $n_1, n_2, ..., n_k$ where $\sum_j n_j = n$

  There are exactly $n_j$ items of size $s^{(j)}$ for $j = 1, 2, ..., k$.

  Let $\text{Bins}(i_1, i_2, ..., i_k) = \text{minimum # of bins to pack}$:

  $\text{OPT} = \text{Bins}(n_1, n_2, ..., n_k)$

  $i_1 \times$ items of size $s^{(1)}$

  $i_2 \times$ items of size $s^{(2)}$

  $\vdots$

  $i_k \times$ items of size $s^{(k)}$
Dynamic Programming

**Instance**: $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$; Pack them into smallest number of unit-sized bins.

**Assumption**:  
- $|\{s_1, s_2, \ldots, s_n\}| = k$  
- $k$ distinct sizes: $s^{(1)}, s^{(2)}, \ldots, s^{(k)}$

$$OPT = \text{Bins}(n_1, n_2, \ldots, n_k) = \text{minimum # of bins to pack:}$$

$$n_j \times \text{items of size } s^{(j)}, 1 \leq j \leq k$$

**Recursion**:  
$$\text{Bins}(i_1, i_2, \ldots, i_k) = 1 + \min_{\bar{x} = (x_1, \ldots, x_k)} \text{Bins}(x_1, \ldots, x_k) = 1 \circ \text{enumerable in time } O(n^k)$$

**Dynamic programming**:  
- table size $O(n^k)$  
- time complexity $O(n^{2k})$
Grouping & Rounding

**Instance \( I \):** \( n \) items with sizes \( s_1, s_2, \ldots, s_n \in (0, 1] \);

**linear grouping:**

- Sort items in non-decreasing order: \( s_1 \leq s_2 \leq \cdots \leq s_n \)
- **Partition** them into \( k \) groups, each with \( \leq \lceil n/k \rceil \) items.
- **Round up** the size of each item to the size of the largest item in its group.

**Instance \( I_{\text{finite}} \):** \( n \) items with sizes \( s_1, s_2, \ldots, s_n \in (0, 1] \)
where \( |\{s_1, s_2, \ldots, s_n\}| = k \)
**Instance** \( I \): \( n \) items with sizes \( s_1, s_2, \ldots, s_n \in (0, 1] \);

**Lemma:** \( \text{OPT}(I) \leq \text{OPT}(I_{\text{finite}}) \leq \text{OPT}(I) + \lceil n/k \rceil \)

- any packing of \( I_{\text{finite}} \) must be a feasible packing of \( I \): \( \text{OPT}(I) \leq \text{OPT}(I_{\text{finite}}) \)
- consider **rounding down** version \( J \) of \( I_{\text{finite}} \): \( \text{OPT}(J) \leq \text{OPT}(I) \)

any packing of \( J \) corresponds to a packing of \( I_{\text{finite}} \) except for the last group

\( \text{OPT}(I_{\text{finite}}) \leq \text{OPT}(J) + \lceil n/k \rceil \)
**Instance** $I$: $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$;

**Lemma:** $OPT(I) \leq OPT(I_{finite}) \leq OPT(I) + \lceil n/k \rceil$

= SOL of DP in time $O(n^{2k})$

$$\frac{SOL}{OPT} \leq 1 + \frac{\lceil n/k \rceil}{OPT} \leq 1 + \epsilon ?$$
Instance $I$: $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$;

Assumption:

- $\forall 1 \leq i \leq n$: $s_i \geq \gamma$

  - each bin contains $\leq 1/\gamma$ items
  - $OPT \geq \gamma n$

$s_i$: 0 \quad \quad \quad \quad \quad \quad \quad 1 \quad k$ groups

$\leq \lfloor n/k \rfloor$ items

$$\frac{SOL}{OPT} \leq 1 + \frac{\lfloor n/k \rfloor}{OPT} \leq 1 + \frac{2}{\gamma k}$$
**Instance $I$:** $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$;

**Instance $I^{\text{big}}$:** $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$ where $s_i \geq \gamma$ for all $1 \leq i \leq n$.

**Lemma:** Given *any* packing of $I^{\text{big}} \subseteq I$ into $m$ bins, using FirstFit to pack items in $I$ of sizes $< \gamma$, altogether uses $m'$ bins:

$$m' \leq \max\{m, \text{OPT}(I)/(1-\gamma) + 1\}$$

- **Case.1:** FirstFit does not open a new bin: $m' = m$
- **Case.2:** FirstFit opens a new bin:

  All but at most one bin are more than $(1-\gamma)$ full.

$$\sum_i s_i > (1-\gamma)(m' - 1) \quad m' \leq \sum_i s_i / (1-\gamma) + 1 \leq \text{OPT}(I)/(1-\gamma) + 1$$
**Instance $I$:** $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$;

**Instance $I^{\text{big}}$:** $n$ items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$ where $s_i \geq \gamma$ for all $1 \leq i \leq n$.

**Algorithm**

Remove all items of size $< \gamma$: $I \Rightarrow I^{\text{big}}$;
apply linear grouping with $k$ groups & round up: $I^{\text{big}} \Rightarrow I^{\text{finite}}$;
find $\text{OPT}(I^{\text{finite}})$ by dynamic programming in $O(n^{2k})$ time;
pack items of size $< \gamma$ by **FirstFit**;
Algorithm

Remove all items of size \( < \gamma \): \( I \Rightarrow I'' \);
apply linear grouping with \( k \) groups & round up: \( I'' \Rightarrow I' \);
find \( \text{OPT}(I') \) by dynamic programming in \( O(n^{2k}) \) time;
pack items of size \( < \gamma \) by FirstFit;

Lemma: Given \textit{any} packing of \( I'' \) into \( m \) bins, using FirstFit
to pack items in \( I \) of sizes \( < \gamma \), altogether uses \( m' \) bins:

\[
m' \leq \max\{ m, \ \text{OPT}(I)/(1-\gamma) + 1 \}
\]

\[
\text{SOL} \leq \max\{ \text{OPT}(I'), \ \text{OPT}(I)/(1-\gamma) + 1 \}
\]

Lemma:

\[
\text{OPT}(I'') \leq \text{OPT}(I') \leq \text{OPT}(I'') + \left\lceil \frac{n(I'')}{k} \right\rceil
\]

\[
\text{SOL} \leq \max\{\text{OPT}(I'') + \left\lceil \frac{n(I'')}{k} \right\rceil, \ \text{OPT}(I)/(1-\gamma) + 1 \}
\]
Algorithm
Remove all items of size < $\gamma$ : $I \Rightarrow I''$;
apply linear grouping with $k$ groups & round up: $I'' \Rightarrow I'$;
find $\text{OPT}(I')$ by dynamic programming in $O(n^{2k})$ time;
pack items of size < $\gamma$ by FirstFit;

Lemma: Given any packing of $I''$ into $m$ bins, using FirstFit to pack items in $I$ of sizes < $\gamma$, altogether uses $m'$ bins:

$$m' \leq \max\{ m, \text{OPT}(I)/(1-\gamma) + 1 \}$$

$$\text{SOL} \leq \max\{ \text{OPT}(I'), \text{OPT}(I)/(1-\gamma) + 1 \}$$

Lemma: $\text{OPT}(I'') \leq \text{OPT}(I') \leq \text{OPT}(I'') + \left\lceil \frac{n(I'')}{k} \right\rceil$

$$\text{SOL} \leq \max\{ (1 + \frac{2}{\gamma k}) \text{OPT}(I''), \text{OPT}(I)/(1-\gamma) + 1 \}$$

large items: $\text{OPT}(I'') \geq \gamma n(I'')$
**Algorithm**

Remove all items of size < \( \gamma \) : \( I \Rightarrow I'' \);
apply linear grouping with \( k \) groups & round up: \( I'' \Rightarrow I' \);
find \( \text{OPT}(I') \) by dynamic programming in \( O(n^{2k}) \) time;
pack items of size < \( \gamma \) by \textbf{FirstFit};

**Lemma:** Given \textit{any} packing of \( I'' \) into \( m \) bins, using \textbf{FirstFit} to pack items in \( I \) of sizes < \( \gamma \), altogether uses \( m' \) bins:

\[
m' \leq \max \{ m, \text{OPT}(I)/(1-\gamma) + 1 \}
\]

\[
\text{SOL} \leq \max \{ \text{OPT}(I'), \text{OPT}(I)/(1-\gamma) + 1 \}
\]

**Lemma:** \( \text{OPT}(I'') \leq \text{OPT}(I') \leq \text{OPT}(I'') + \lceil n(I'')/k \rceil \)

\[
\text{SOL} \leq \max \{ (1 + 2/\gamma k) \text{OPT}(I), \text{OPT}(I)/(1-\gamma) + 1 \}
\]

**large items:** \( \text{OPT}(I'') \geq \gamma n(I'') \) \textbf{trivially:} \( \text{OPT}(I'') \leq \text{OPT}(I) \)
Algorithm

Remove all items of size $< \gamma : I \Rightarrow I^{\text{big}}$;
apply linear grouping with $k$ groups & round up: $I^{\text{big}} \Rightarrow I^{\text{finite}}$;
find $\text{OPT}(I^{\text{finite}})$ by dynamic programming in $O(n^{2k})$ time;
pack items of size $< \gamma$ by FirstFit;

$$SOL \leq \max\{\frac{1+2/\gamma k}{\text{OPT}(I)}, \frac{\text{OPT}(I)}{1-\gamma} + 1\}$$
$$\leq (1 + \epsilon)OPT + 1$$
choose $\gamma = \epsilon/2$ and $k = 4/\epsilon^2$

time complexity: $n^{O(1/\epsilon^2)}$

approximation: $SOL \leq (1 + \epsilon)OPT + 1$

Asymptotic PTAS
Scheduling

$m$ machines

$n$ jobs

makespan
Scheduling

$m$ machines

$n$ jobs with processing time $p_j$

makespan

**Instance:** $n$ jobs $j=1, 2, ..., n$ each with processing time $p_j \in \mathbb{Z}^+$. Find a schedule of $n$ jobs to $m$ machines that minimizes the makespan $C_{\text{max}}$. 
Scheduling:

**Instance:** $n$ jobs $j=1, 2, \ldots, n$ each with processing time $p_j \in \mathbb{Z}^+$. Find a schedule of $n$ jobs to $m$ machines that minimizes the makespan $C_{\text{max}}$.

- The *List* algorithm has approximation ratio 2.
- The *LPT (Longest Processing Time)* algorithm has approximation ratio $4/3$.
- The problem of minimum makespan on identical machines has a *PTAS (Polynomial Time Approximation Scheme)*.
Scheduling:

**Instance**: $n$ jobs $j=1,2,...,n$
each with processing time $p_j \in \mathbb{Z}^+$.
Find a schedule of $n$ jobs to $m$ machines that minimizes the *makespan* $C_{\text{max}}$.

Bin packing:

**Instance**: $n$ items $i=1,2,...,n$;
with sizes $p_1, p_2, ..., p_n \in \mathbb{Z}^+$;
Find a *packing* of the $n$ items into *smallest* number of *bins* with *capacity* $B \in \mathbb{Z}^+$.

Given an instance $I$: $p_1, p_2, ..., p_n$

$$C_{\text{max}}(I) = \min\{ B : \text{Bins}(I, B) \leq m \}$$
Scheduling:

**Instance:** \( n \) jobs \( j=1, 2, ..., n \) each with processing time \( p_j \in \mathbb{Z}^+ \).

Find a schedule of \( n \) jobs to \( m \) machines that minimizes the makespan \( C_{\text{max}} \).

Given an instance \( I: p_1, p_2, ..., p_n \)

\[
C_{\text{max}}(I) = \min \{ B : \text{Bins}(I, B) \leq m \}
\]

\[
L = \max \left\{ \max_{1 \leq j \leq n} p_j, \frac{1}{m} \sum_{j=1}^{n} p_j \right\} \leq \text{OPT} \leq 2 \cdot L
\]

**Idea for algorithm:**

binary search for \( B \) between \([L, 2L]\) to find the minimum \( B \) such that \( \text{Bins}(I, B) \leq m \);
Given an instance $I$: $p_1, p_2, \ldots, p_n$

$$C_{\text{max}}(I) = \min\{B : \text{Bins}(I, B) \leq m\}$$

**idea for algorithm:**

Binary search for $B$ between $[L, 2L]$ to find the minimum $B$ such that $\text{Bins}(I^{\text{finite}}, B) \leq m$;

$I^{\text{finite}}$: • group jobs into $k$ intervals $[B/(1+\varepsilon)^{t+1}, B/(1+\varepsilon)^t]$;
  • round down;

To further control the time complexity:

• deal with small jobs ($p_j < B/(1+\varepsilon)^{k+1}$) separately;

This will give a PTAS for scheduling.