

Packing Algorithms

An instance of a packing problem consists of:

1. Items (associated with sizes, weights, profits).
2. Bins with limited capacity.
3. A set of constraints.

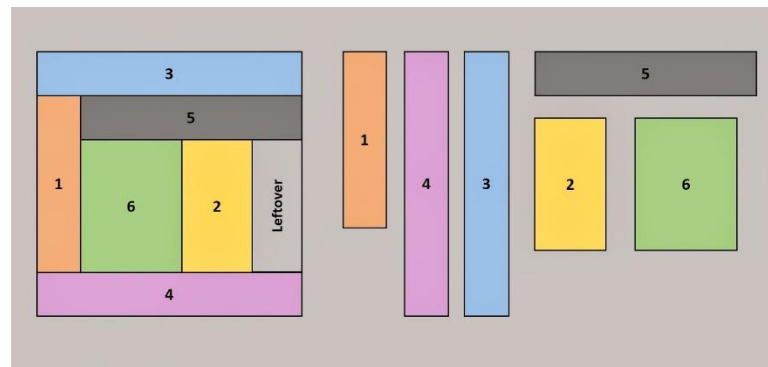
General Goal: Place items in the bin, items must not overlap with each other. Bins may not be filled beyond their capacity.



Packing Problems

Variants:

- Multi-dimensional items.
- Limited number of bins - pack as much as you can.
- Unlimited number of bins - pack all items using minimal number of bins.
- Conflicting items - cannot be placed together.
- Cardinality constraints.
- Cutting stock (minimizing wasted material)
- Class-constraints.
- Online vs. offline
- Variable bins.
- Many more...



Popular Applications

- Physical items → boxes, tracks (shipping, delivery).
- Files → disks.
- Advertisements → commercial break/ magazines / web-pages.
- Orders → limited amount of material
- Jobs → processors.

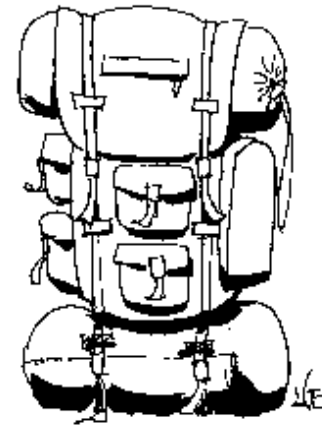


The Knapsack problem

- You are about to go to a camp.
- There are many items you want to take.
- You have one knapsack. The total weight you can carry is at most W .
- Item i in your list has weight w_i , and value (benefit) b_i , that measures how much you really need it.
- You need to pack the knapsack in a way that maximizes the total value of the packed items.

The Knapsack problem

Item #	Weight	Value
1	1	8
2	3	6
3	5	5
4	4	6



Max
weight
=8

A possible packing: Items 2 and 3. Value: 11

An optimal packing: Items 1,2,4. Value: 20

The Knapsack problem is NP-hard.

Greedy Algorithm for Knapsack

1. Consider the items in order of non-increasing b_i/w_i ratio
 $b_1/w_1 \geq b_2/w_2 \geq \dots \geq b_n/w_n$
2. Add items to the knapsack as long as there is space.

Time Complexity:

$O(n \log n)$ (for sorting)

$O(n)$ for packing loop.

→ $O(n \log n)$

Greedy Algorithm for Knapsack

Claim: The approximation ratio of Greedy is not bounded.

Proof: To get ratio c , consider the following instance:

There are two items:

$$b_1 = 2, w_1 = 1$$

$$b_2 = 2c, w_2 = 2c$$

The knapsack has volume $W = 2c$

Greedy packs only the first item, value = 2.

Optimal: Pack the second item, value = $2c$

Ratio = c .

Improved Algorithm for Knapsack

Take the maximum of Greedy and the most valuable item that fits by itself.

Theorem: The above algorithm is 2-approximation.

Proof: We assume w.l.o.g that no single item has weight more than W (these items can be removed in a preprocessing).

Sort the items such that $b_1/w_1 \geq b_2/w_2 \geq \dots \geq b_n/w_n$.

Let B be the largest value of an item, and let G be the value computed by the greedy algorithm.

Let j be the first item that the greedy algorithm rejects.

Improved Algorithm for Knapsack

$$ALG = \max(B, G) \geq (B + G)/2$$

$$G = \sum_{i=1}^{j-1} b_i \quad (\text{item } j \text{ is the first to be rejected})$$

$$B \geq b_j \quad (B \text{ is the most profitable})$$

$$G+B \geq \sum_{i=1}^j b_i \quad \text{opt} < \sum_{i=1}^j b_i \quad \text{Because the first } j \text{ items have the largest 'profit density'}$$

$$\rightarrow ALG > \text{opt}/2$$

Exact Solution - dynamic programming

Variant 1:

Define a table M of size $(n+1) \times (W+1)$, where the (i, x) entry corresponds to the maximal profit that can be obtained from the first i items and a knapsack having capacity x .

The solution to the knapsack problem lies in $M(n, W)$.

Base cases:

If $i = 0$, then there are no items to pack: $M(0, x) = 0$.

If $x < 0$ then $M(i, x) = -\infty$.

Exact Solution - dynamic programming

The DP recursion:

$$M(i, x) = \max \{ M(i-1, x), M(i-1, x-w_i) + b_i \}.$$

We take the maximum of two options:

1. $M(i-1, x)$: not packing the i -th item.
2. $M(i-1, x-w_i) + b_i$: packing the i -th item.

n								*
...								
i								
i-1								
...								
0	0	0	0	0	...	0	...	0
	1	2	3	4	...	x	...	W

The solution lies here

Knapsack DP Example

Assume $n=4$ and $W=17$.

Weights: {2, 4, 7, 10} Values: {3, 7, 9, 16}

4																		
3																		
2																		
1																		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i/x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

In class exercise - complete the table

Knapsack DP Example

Assume $n=4$ and $W=17$.

Weights: $\{2, 4, 7, 10\}$ Values: $\{3, 7, 9, 16\}$

4	0	3	3	7	7	10	10	10	12	16	16	19	19	23	23	26	26
3	0	3	3	7	7	10	10	10	12	12	12	12	19	19	19	19	19
2	0	3	3	7	7	10	10	10	10	10	10	10	10	10	10	10	10
1	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i/x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

$$M(3,9) = \max\{M(2,9), M(2,2)+9\} = 3+9=12$$

$$M(4,12) = \max\{M(3,12), M(3,2)+16\} = 3+16=19.$$

Exact Solution - dynamic programming

Variant 2:

Define a table M of size $(n+1) \times (\sum_i b_i)$, where the (i, v) entry corresponds to the minimum weight of a combination of the first i items with value at least v .

The solution is in the entry with the maximum v where $M(n, v) < W$. This entry can find by scanning all entries in the line of $i = n$.

Base cases:

If $M(0, 0) = 0$.

If $M(0, v) = \infty$ for all $v > 0$.

Exact Solution - dynamic programming

The DP recursion:

$$M(i, v) = \min \{ M(i-1, v) , M(i-1, v-b_i) + w_i \}.$$

We take the minimum of two options:

1. $M(i-1, v)$: not packing the i -th item.
2. $M(i-1, v-b_i) + w_i$: packing the i -th item.

n								
...								
i								
i-1								
...								
0	0	0	0	0	...	0	...	0
	1	2	3	4	...	v	...	$\sum_i b_i$

← The solution lies in this line

Exact Solution - dynamic programming

The second DP variant is the basis of our next approximation algorithm.

The Knapsack problem is 'easy to approximate' - we can get as close to an optimal solution as required.

Formally, it has a fully polynomial time approximation scheme (FPTAS)

Polynomial Time Approximation Scheme

A polynomial time approximation scheme is an algorithm which takes as input an additional parameter, ϵ , which determines the desired approximation ratio. This ratio can be arbitrarily close to **1**, when ϵ approaches **0**.

The time complexity of the scheme is polynomial in the input size but may be exponential in $1/\epsilon$.

For example, the following running times are acceptable for a PTAS:

- $O(n^2/\epsilon)$
- $O(n^{100}2^{1/\epsilon})$
- $O(n^{2^{2^{1/\epsilon}}})$

A Fully Polynomial Time Approximation Scheme

A fully polynomial time approximation scheme (FPTAS) is a PTAS with running time polynomial in both n and $1/\epsilon$.

The good news: The Knapsack problem has a fully polynomial time approximation scheme.

FPTAS for Knapsack

The DP has size $n \cdot \sum_i b_i$

$\sum_i b_i$ is loosely bounded by $n \cdot B$, where $B = \max(b_i)$.

Since we must compute every cell in the table, the DP running time is $O(n^2 B)$, which is pseudo-polynomial.

To construct the FPTAS, we will reduce the number of distinct value categories (columns).

We will have only $\text{poly}(n)/\epsilon$ value categories by scaling and rounding each b_i

This way, the DP table will have size $\text{poly}(n, 1/\epsilon)$. As a result the running time will also be $\text{poly}(n, 1/\epsilon)$.

FPTAS for Knapsack

1. Given $\varepsilon > 0$, let $k = \varepsilon B/n$ be the scaling parameter.

2. Replace every item value b_i by $b'_i = \left\lceil \frac{b_i}{k} \right\rceil k$.

The number of columns (different values of b'_i) is now no more than $n \frac{B}{k} = \frac{n^2}{\varepsilon}$.

3. Apply the DP (variant 2) with values b'_i . the running time is $O(n^3/\varepsilon) = O(\text{poly}(n, 1/\varepsilon))$.

Theorem: The profit achieved by the algorithm is at least $(1 - \varepsilon)OPT$.

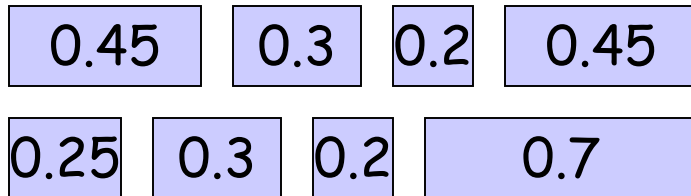
Proof: In class.

The Bin Packing Problem

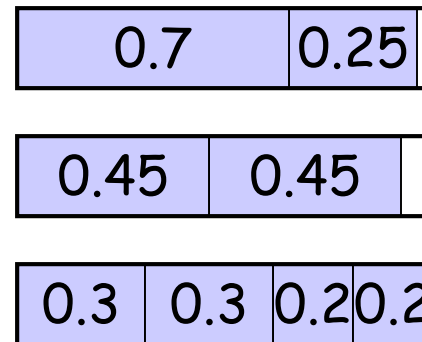
- **Input:** Items of sizes $0 < s_i < 1$
- **Output:** A feasible packing in bins of size 1
- **Goal:** minimize number of bins used.

Example:

Input:



A packing in 3 bins:



Approximating Bin Packing

Next-fit Algorithm:

1. Open an *active* bin.
2. For all $i=1,2,\dots,n$:
 - If possible, place a_i in the current active bin;
 - Otherwise, open a new active bin and place a_i in it.

Example: The input: $\{0.3, 0.9, 0.2\}$.

Next-fit packing (three bins): $(0.3), (0.9), (0.2)$.

Theorem: Next-fit is 2-approximation to BP

Proof: An optimal algorithm must use at least $\sum_i a_i$ bins (why?).

Approximating Bin Packing

Analysis of Next Fit (cont'): Assume that Next Fit uses h bins. The sum of items sizes in two consecutive bins is greater than 1 (otherwise, we can put them together).

Case 1: h is even:

$$c(B_1) + c(B_2) > 1$$

$$c(B_3) + c(B_4) > 1$$

$$c(B_{h-1}) + c(B_h) > 1$$

$$\Sigma_i a_i > h/2$$

Case 1: h is odd:

$$c(B_1) + c(B_2) > 1$$

$$c(B_3) + c(B_4) > 1$$

$$c(B_{h-2}) + c(B_{h-1}) > 1$$

$$\Sigma_i a_i > (h-1)/2 + c(B_h)$$

In both cases, we can obtain $h \leq \lceil 2\Sigma_i a_i \rceil \leq 2opt$

Remark: it can be shown that $h \leq 2opt-1$

Approximating Bin Packing

Is the analysis tight? consider an instance with $4n$ items $\{1/2, 1/2n, 1/2, 1/2n, \dots\}$.

Next-fit will put any two consecutive items in a bin.

Total number of bin used: $2n$.

An optimal packing in $n+1$ bins: n bins, each with $1/2+1/2$, one bin for the tiny items.

The ratio: $2n/(n+1) \rightarrow 2$ as n grows.

Approximating Bin Packing

First fit algorithm: place the next item in the first open bin that can accommodate it. Open a new bin only if no open bin has enough room.

Theorem: $h_{ff} \leq 1.7\text{opt} + 2$ (proof not here)

First fit Decreasing: sort the items from largest to smallest. Run FF according to the resulting order.

Theorem: $h_{ffd} \leq 1.222\text{opt} + 3$ (proof not here)

▪ No additive-error approximation is known for bin packing. That is, the best known is $(1+\delta)\text{opt}$.

Unit Fractions Bin Packing

- **A Unit Fraction:** A fraction of the form $1/i$ for an integer i .
- **Input:** integers w_1, w_2, \dots, w_n .
- **Goal:** Bin packing of the unit fractions $\{1/w_1, 1/w_2, \dots, 1/w_n\}$.
- Let $H(W) = \left\lceil \sum_{i \in W} \frac{1}{w_i} \right\rceil$. Clearly, $\text{OPT}(W) \geq H(W)$.
- **We will see:** An algorithm that uses at most $H(W)+1$ bins (additive error of one for any input).

Any-fit Decreasing for UFBB

1. Sort the items such that $1/w_1 \geq 1/w_2 \geq \dots \geq 1/w_n$
2. Pack the items in this order, each item is placed in any open bin that can accommodate it, or in a new bin, if none exists.

Theorem: The number of bins used is at most

$$1 + \left\lceil \sum_i \frac{1}{w_i} \right\rceil \leq 1 + \text{OPT}$$

Proof idea: After packing all the items of size at least $1/k$:

- (i) There are at most $k-1$ non-full bins, and
- (ii) Each of the non-full bins is at least $1-1/k$ full.

Details: In Class

Any-fit Decreasing for UFBP

Remark: The analysis is tight (the alg. is not optimal)

Example: $\frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}$ - in decreasing order.

- Will be packed in three bins:

$$\left\{ \frac{1}{2}, \frac{1}{3} \right\} \quad \left\{ \frac{1}{3}, \frac{1}{4}, \frac{1}{4} \right\} \quad \left\{ \frac{1}{4} \right\}$$

- Can be packed in two bins:

$$\left\{ \frac{1}{2}, \frac{1}{4}, \frac{1}{4} \right\} \quad \left\{ \frac{1}{3}, \frac{1}{3}, \frac{1}{4} \right\}$$

Online Bin Packing

The input: A sequence of items (numbers), a_1, a_2, \dots, a_n , such that for all i , $0 < a_i < 1$

The goal: 'pack' the items in bins of size 1. Use as few bins as possible.

Example: The input: $1/2, 1/3, 2/5, 1/6, 1/5, 2/5$.

Optimal packing in two bins:

$(1/2, 1/3, 1/6), (2/5, 2/5, 1/5)$.

Legal packing in three bins:

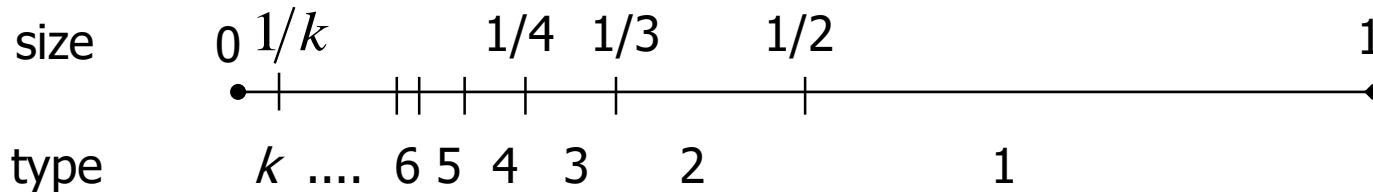
$(1/2, 1/3), (2/5, 1/6, 1/5), (2/5)$

Online BP: a_i must be packed before we know a_{i+1}, \dots, a_n

The HARMONIC-k Algorithm

Classify items into k intervals according to size

$(1/2, 1]$	one item per bin
$(1/3, 1/2]$	two items per bin
...	
$(1/k, 1/(k-1)]$	$k-1$ items per bin
$(0, 1/k]$	use NF



The HARMONIC Algorithm

- Each bin contains items from only one class: i items of type i per bin
- Items of **last type** are packed using **NEXT FIT**: use one bin until next item does not fit, then start a new bin
- Keeps $k-1$ bins open

Analysis of HARMONIC-3

- Let X be the number of bins for $(1/2, 1]$
 - Those bins are full by more than $1/2$
- Let Y be the number of bins for $(1/3, 1/2]$
 - Those bins are full by more than $2/3$
- Let T be the number of bins for $(0, 1/3]$
 - Those bins are full by more than $2/3$

Let W be the total size of all items

Then $W > X/2 + 2Y/3 + 2T/3$

Analysis of HARMONIC-3

Other bounds:

- $OPT \geq X$ (items larger than $1/2$)
- $OPT \geq (X+2Y)/2$ (items larger than $1/3$)
- $H3 \leq X+Y+T (+2) \leq (3(W+X/6))/2(+2)$
 $\leq 3W/2+X/4 (+2) \leq 1.75OPT (+2)$

Asymptotically, this is neglected.

Analysis of HARMONIC-4

- Let X be the number of bins for $(1/2, 1]$
 - Those bins are full by more than $1/2$
- Let Y be the number of bins for $(1/3, 1/2]$
 - Those bins are full by more than $2/3$
- Let Z be the number of bins for $(1/4, 1/3]$
 - Those bins are full by more than $3/4$
- Let T be the number of bins for $(0, 1/4]$
 - Those bins are full by more than $3/4$
- Let W be the total size of all items
Then $W > X/2 + 2Y/3 + 3Z/4 + 3T/4$

Analysis of HARMONIC-4

Other bounds:

- $OPT \geq X$ (items larger than $1/2$)
- $OPT \geq (X+2Y)/2$ (items larger than $1/3$)
- $H4 \leq X+Y+Z+T (+3) \leq$
 $(4(W+X/4+Y/12))/3(+3)$
 $\leq 4 \cdot W/3+X/3+Y/9 (+3) =$
 $=4 \cdot W/3+(x/18+Y/9)+5 \cdot X/18 (+3)$
 $\leq 31 \cdot OPT /18 (+3) \approx 1.7222 \cdot OPT (+3)$

Analysis of HARMONIC

- **Theorem:** For any k , **Harmonic- k** is at most **1.691** competitive.
- **Proof:** C. C. Lee and D. T. Lee. **A simple on-line bin-packing algorithm.** *Journal of the ACM* 32 (3) July 1985. (beyond our scope. Available in the course web-page).